MIRA

Multiplayer Interactive World Models with Representation Autoencoders

General Intuition
Anthony Hu*, Chris Mulder*, Aditya Makkar°, Adam Jelley, Eloi Alonso, Florian Laurent, Fredrik Norén, James Swingos, Jan Hünermann, Kent Rollins, Lucas Hosseini, Matthieu Le Cauchois, Maxim Peter, Pim de Witte, Tim Brown, Vincent Micheli

Kyutai
Václav Volhejn*, Adrien Ramanana Rahary*[1], Amélie Royer°, Alyx Liao°, Manu Orsini°, Moritz Böhle, Gabriel de Marmiesse, Patrick Pérez

Epic Games
Viktoriia Sharmanska, Lucia Specia, Michael Black

* *equal contribution* ° *core contributors*

We introduce the first multiplayer world model for highly dynamic environments governed by complex physical interactions. Whereas single-player world models treat the other agents as part of the environment, ours conditions on the action streams of multiple agents, learning to attribute changes in the scene to the correct player and to stay coherent under arbitrary combinations of their actions. We study this problem in the game of Rocket League, where players compete and cooperate under fast, tightly coupled dynamics. Trained on 10,000 hours of gameplay collected with publicly available bots, our 5-billion-parameter latent diffusion model generates four-player matches in real time, producing 20 frames per second on a single Nvidia B200 GPU. Although trained only on short clips, its rollouts stay stable far beyond the training horizon: distributional quality holds steady out to five minutes, the longest horizon we measure, and in practice we observe rollouts continuing for hours with no sign of collapse. We systematically investigate the central design choices: the video codec, the generative objective, and the multiplayer conditioning scheme. In addition, we characterize how behavior changes with model and data scale, including the capabilities that emerge and the failure modes that persist. We further develop targeted evaluations that probe the model's physical understanding rather than visual appearance alone. To support continued research on multiplayer world models, we release our dataset, our full training and inference codebase, and a live demo.

Live demo: https://mira-wm.com
Blog: https://mira-wm.com/blog-post
Code: https://github.com/mira-wm/mira
Dataset: https://huggingface.co/datasets/kyutai/rocket-science

1 Introduction

Predicting how a scene will evolve in response to actions is a core ability of any embodied agent (LeCun, 2022; Silver and Sutton, 2025). A world model learns this ability directly from experience, typically by building an internal latent representation of the environment and predicting future latent states conditioned on actions. Once learned, the world model can act as a controllable simulator (Ha and Schmidhuber, 2018; Micheli et al., 2023; Russell et al., 2025; Hafner et al., 2025b): agents can be trained inside it, evaluated against it, and used to imagine the consequences of candidate actions before committing to them in the real environment.

However, for such imagined rollouts to be useful, the world model must be faithful to the environment it

[1] École nationale des ponts et chaussées

Figure 1 World model imagination. Rows are the four players' viewpoints (Players 1–4) and columns show three timesteps in the trajectory ($t_0 \to t_0 + 6\,\mathrm{s} \to t_0 + 8\,\mathrm{s}$). From the initial state at t_0 and the players' action streams, the model imagines a dynamic game scene that captures the physical interplay between the players, the ball, and the scene. The rollout is temporally consistent: the in-game clock counts down with elapsed time (4:41 → 4:35 → 4:33), and the players' views stay mutually coherent. For instance, Players 2 and 4 at $t_0 + 6\,\mathrm{s}$, who are contesting the ball near the goal, each render the ball and the cars in a matching spatial configuration. This is best experienced interactively: try our live demo at `mira-wm.com`.

represents (Alonso et al., 2024). As we ultimately aim to build agents that operate in real environments, limitations in world model fidelity can directly hinder downstream agent performance. In particular, the model must capture the causal relationships between actions and their consequences, and preserve the dynamics that determine how the environment evolves. A model that produces visually plausible futures but fails to respond correctly to interventions may be useful as a video predictor, but is insufficient as a substrate for planning and control (Quevedo et al., 2025; Gemini Robotics Team, 2025).

This challenge becomes even more important in multi-agent environments. While most real-world environments are inherently multi-agent, the majority of existing world models adopt a single-agent perspective, representing other agents merely as part of the environment. This design limits their applicability to learning paradigms that depend on explicit control over multiple participants, such as self-play reinforcement learning (Schrittwieser et al., 2020), multi-agent training (Albrecht et al., 2024), counterfactual policy evaluation (Wayve, 2025), and human-in-the-loop interaction (Abramson et al., 2022). A multi-agent world model should instead condition on the action streams of multiple agents and predict how their joint behavior shapes the future state of the environment.

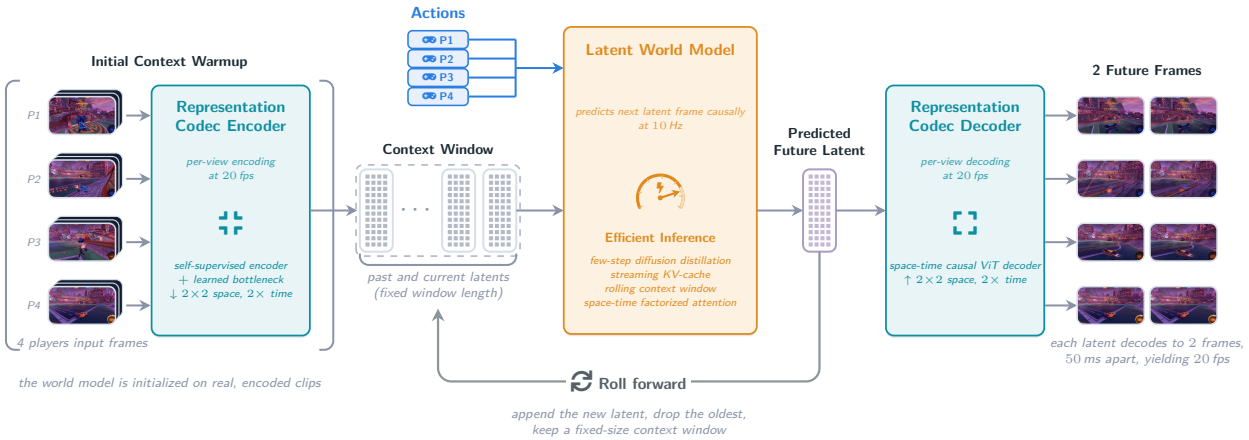


Figure 2 Method overview. MIRA simulates four-player Rocket League in the latent space of a video representation codec. During an initial context warmup, the codec encoder maps a short clip of each player's view into a stack of latent frames that seed a fixed-length context window (Section 4.2). Conditioned on the per-player action streams (Section 4.5) and the latents currently in the context window, the latent world model predicts the next latent frame causally at 10 Hz (Sections 4.3 and 4.4). Few-step diffusion distillation, a streaming KV-cache, and a rolling context window keep inference fast enough to generate 20 frames per second on a single Nvidia B200 GPU (Section 5). The codec decoder then renders each predicted latent back into the four player views at 20 fps (Section 4.2). Appending the predicted latent to the context window and dropping the oldest (*roll forward*) autoregressively extends the rollout over long horizons (Section 5).

Extending world models from the single-agent to the multi-agent setting takes more than adding inputs. Conditioning on multiple action streams is strictly more informative than conditioning on a single action stream, but it also imposes a stronger requirement on the model: it must learn to attribute changes in the scene to the correct agent, represent interactions between agents, and remain coherent under arbitrary combinations of actions. This setting introduces potential failure modes beyond inaccurate dynamics: the model can mis-assign agency, ignore the actions of one player, or entangle the effects of multiple players. Faithful multi-agent prediction therefore requires both accurate visual reconstruction and correct action-conditioned dynamics.

In this technical report, we study multi-agent world modeling in Rocket League, a visually rich, physically grounded, and strategically multi-agent environment. Although Rocket League is a video game and thus itself simulated, it serves as a useful proxy for world modeling research by making large-scale data collection and precise action logging feasible in a setting with fast, interactive dynamics that remain difficult to predict from pixels alone. Its appeal as a testbed lies in the combination of discrete high-frequency actions, rich object interactions, partial observability from rendered observations, and tightly coupled coordination and competition between players. We collect 10,000 hours of multiplayer data generated by reinforcement learning policies (Braaten and Necto contributors, 2022) in private matches. Each recording pairs the rendered video stream with the controller actions of every player; we also log privileged game state, such as ball and car physics, for downstream evaluation of the model's ability to learn physically meaningful representations from pixels and actions alone. On this data, we train a multiplayer diffusion world model with 5 billion parameters. The model operates in the latent space of a video representation codec and runs in real time, generating 20 frames per second on a single Nvidia B200 GPU.

A systematic study of our design choices yields several observations. For the codec, building the latent space on a pretrained feature extractor makes it markedly better suited to world modeling, improving both the quality and the temporal consistency of rollouts, while matching the reconstruction quality of a feature extractor trained from scratch (Section 6.3). For world modeling, diffusion forcing and few-step distillation together yield stable long-horizon predictions and fast inference (Sections 5 and 6.4). Multiplayer world models substantially outperform their single-player counterparts, indicating that the additional views and action streams reduce uncertainty about the future and yield more accurate predictions; warm-starting a multiplayer model from a single-player model works better than training on multiplayer data directly

(Section 6.6). Across training runs, we observe a consistent progression: the model reaches visually plausible frames early and then spends most of the remaining training refining how actions map to their consequences. We also examine how performance scales as we vary model size from 100M to 5B parameters and training data from 100 to 10,000 hours, uncovering a range of emergent capabilities and characteristic failure modes (Sections 6.7 to 6.9). In summary, our contributions are as follows.

- We present the first interactive multiplayer world model for a highly dynamic environment with complex physical interactions. The model is controllable by four concurrent agents, can be rolled out over long horizons, and runs in real time at 20 fps (Sections 4 and 5).
- We conduct a systematic study of the latent space (Section 6.3), the generative objective (Section 6.4), and the multiplayer conditioning scheme (Section 6.6), and we characterize scaling behavior over model and data size, together with the emergent capabilities and failure cases that arise (Sections 6.7 to 6.9).
- We propose a comprehensive evaluation protocol that measures physical understanding, visual fidelity, and human preference across the models we develop (Section 6.2).
- To foster future research on multiplayer world models, we release our training data (Section 3) at `https://huggingface.co/datasets/kyutai/rocket-science`, together with our full training and inference codebase. A live demo of our model is also available at `https://mira-wm.com`.

2 Related Work

We model Rocket League, a fast, physically dynamic multiplayer game, as a real-time generative simulator that predicts future video frames from several players' actions. This places our work in the world-modeling literature, which spans two formulations: an abstract latent dynamics model for control that never renders the world (Section 2.1), and a controllable video generator whose frames are themselves the objective (Section 2.2). Our approach is the latter, in one of its most challenging cases: the multiplayer setting where several players continually reshape a shared world (Section 2.3). We then review the techniques these models build on: generative video backbones (Section 2.4), the choice of prediction space (Section 2.5), objectives for stable real-time rollouts (Section 2.6), and the evaluation of generative world models (Section 2.7).

2.1 World models for control

A world model predicts how an environment evolves under actions. One line uses this for control: planning or training an agent inside the model. Ha and Schmidhuber (2018) train a recurrent latent model and evolve a controller within its imagined rollouts. The Dreamer family learns recurrent state-space models and optimizes behavior through imagination, progressing from latent-space planning (Hafner et al., 2019, 2020) to discrete latent representations (Hafner et al., 2021), a single configuration that masters diverse domains (Hafner et al., 2025a), and a scalable transformer world model that trains agents inside it (Hafner et al., 2025c). Model-based imitation learning trains a driving policy inside a learned latent world model (Hu et al., 2022). Transformer-based world models improve sample efficiency (Micheli et al., 2023, 2024; Robine et al., 2023; Zhang et al., 2023), and latent model-predictive control operates in decoder-free latent spaces (Hansen et al., 2022, 2024). Joint-embedding predictive architectures (JEPA) predict in representation space rather than pixels (LeCun, 2022; Assran et al., 2023; Bardes et al., 2024; Maes et al., 2026), including action-conditioned variants for robotics (Assran et al., 2025) and approaches built on frozen self-supervised features (Zhou et al., 2025). These methods prioritize accurate dynamics in compact state spaces over visual fidelity; pixels, when reconstructed at all, only probe or learn the representation, and JEPA omits rendering entirely. Our model also favors a compact prediction space, but is generative, decoding that latent into the video frames on which it is evaluated.

2.2 Interactive and playable game world models

Generative world models instead predict observations directly. At the least interactive end, they keep the dynamics fixed while synthesising novel views as the camera moves: navigation world models (Koh et al., 2021; Bar et al., 2025), monocular novel-view synthesis (Liu et al., 2021; Li et al., 2022; Liu et al., 2023a;

Sargent et al., 2024; Ramanana Rahary et al., 2026), camera-controlled video diffusion (He et al., 2024), and persistent explorable worlds generated from a single image (World Labs, 2024, 2025) or a text prompt (Wang et al., 2026). By producing 3D scenes that are traversable and support navigation and interaction, these approaches emphasize perceptual and geometric consistency without modeling scene dynamics.

Interactive game world models go further, generating frames whose contents change with each action, requiring both visual realism and dynamical correctness. Valevski et al. (2025) simulate DOOM in real time from past frames and player actions, while Bruce et al. (2024) learn a latent action space from unlabeled video to generate controllable worlds, later scaled to 3D scenes from a single prompt image (Google DeepMind, 2024, 2025). Alonso et al. (2024) train agents inside diffusion world models and deploy them as standalone engines for Atari and CS:GO. Further systems generate playable game video in Minecraft (Decart and Etched, 2024; Guo et al., 2025; Zhang et al., 2025), in open-world games (Che et al., 2025; Feng et al., 2024), and in AAA titles such as Grand Theft Auto (He et al., 2025; Li et al., 2025b). These build on earlier learned simulators that rendered the next screen directly from controller input (Kim et al., 2020; Menapace et al., 2021), which Yu et al. (2025) frame as a path toward generative game engines. These systems are driven by a single action stream, that of one player or camera. The same recipe extends to driving: GAIA generates driving video conditioned on action and text (Hu et al., 2023; Russell et al., 2025; Wayve, 2025), related diffusion models produce controllable future driving video for prediction and planning (Wang et al., 2024; Gao et al., 2024; Yang et al., 2024), and autoregressive video pretraining yields driving representations for a downstream action model (Bartoccioni et al., 2025). Games and driving alike are modeled from a single agent's controls; our model instead conditions on the simultaneous actions of multiple players.

2.3 Multiplayer and multi-agent world models

The dynamics are hardest to model when multiple agents jointly reshape a shared environment, the regime our work targets. The World and Human Action Model (Kanervisto et al., 2025), trained on four-versus-four gameplay, conditions on a single player's controls and treats all other players as part of the environment, learning multiplayer dynamics through one control interface. More recent generative approaches condition on several agents at once: in two-dimensional grid worlds with up to seven agents (Pondaven et al., 2026); in Minecraft with two players or with per-agent first-person views generalized beyond two players (Savva et al., 2026; Liu et al., 2026); across multiple camera views of cooperative play and robot manipulation (Wu et al., 2026); in DOOM levels anchored to editable two-dimensional maps (Po et al., 2026); from precomputed trajectories in real-world video (Hu et al., 2026); and through per-entity language prompts over shared combat views (Zhu et al., 2026). These works condition generative world models on multiple agents, but focus on grid worlds, voxel environments, planar navigation, or combat animation, where rigid-body interaction is limited, and many treat scaling the agent count as the central challenge. Our setting, Rocket League, is among the most dynamic and interactive: a continuous-physics, fully three-dimensional sports game where cars and a ball collide at speed. We model it in real time, jointly predicting each of the four players' own view of the shared match and conditioning on their simultaneous low-level controls.

2.4 Generative video models and diffusion

The visual backbone of these world models comes from generative video modeling. Denoising diffusion made high-fidelity image synthesis practical (Ho et al., 2020), and latent diffusion moved generation into a learned latent (Rombach et al., 2022); flow matching and rectified flow recast this as a continuous transport between noise and data, often with a simpler objective (Lipman et al., 2023; Liu et al., 2023b). The transformer is the scalable architecture for both: the diffusion transformer (Peebles and Xie, 2023) and its interpolant counterpart (Ma et al., 2024). Extending these processes across time yields video diffusion (Ho et al., 2022b,a; Singer et al., 2023), and large latent video models now reach strong fidelity over long durations (Blattmann et al., 2023; Yang et al., 2025b; Bar-Tal et al., 2024; Gupta et al., 2024; Kong et al., 2024; HaCohen et al., 2025, 2026); at scale such models have been described as general simulators of the visual world (Brooks et al., 2024), while token-autoregressive models offer an alternative (Kondratyuk et al., 2024). These models produce realistic video, but are neither causal nor action-controllable by default, the two properties a world model needs. We therefore adopt a latent video diffusion backbone and make it causal and action-conditioned.

2.5 Visual representations for generation

Beyond the choice of backbone, two design questions cut across these world models. The first concerns the representation space in which prediction occurs. Predicting in a compact learned latent rather than in pixels makes the problem lower-dimensional, removes the burden of synthesizing unpredictable high-frequency texture, and, when the latent is well-structured, keeps predictions stable over long rollouts. Latent diffusion established this paradigm by compressing images into a latent through an autoencoder and training the diffusion process in that latent (Rombach et al., 2022). Generation quality then depends on the structure of the latent, motivating work to improve it. One direction distills a pretrained self-supervised encoder such as DINO (Caron et al., 2021; Oquab et al., 2024; Siméoni et al., 2025) into the latent space of an autoencoder: VA-VAE and the GAIA driving models both align that latent to DINO features, in an image autoencoder and a video tokenizer respectively (Yao et al., 2025; Hu et al., 2023; Russell et al., 2025), while earlier BEiT v2 distills a pretrained teacher into a vector-quantized tokenizer (Peng et al., 2022). This principle carries over to other modalities: in audio, the Mimi codec (Défossez et al., 2024) distills a pretrained WavLM encoder (Chen et al., 2022) into its latent, and SpeechTokenizer (Zhang et al., 2024) distills a self-supervised speech teacher into its tokens. A related direction regularizes the latent through self-supervised objectives that encourage semantic structure, denoising, or equivariance (Chen et al., 2025a,b; Yang et al., 2025a; Kouzelis et al., 2025). At the extreme, some approaches remove the learned encoder entirely and generate in the feature space of frozen self-supervised encoders such as DINO (Caron et al., 2021) or SigLIP (Zhai et al., 2023; Tschannen et al., 2025), training only the decoder (Zheng et al., 2025; Bi et al., 2025; Gui et al., 2025). This representation-autoencoder paradigm has since been extended to text-to-image generation (Tong et al., 2026), simplified (Singh et al., 2026), and reduced to lightweight adaptation layers over frozen encoders (Gao et al., 2025). Our codec follows the principle these works share, that representation quality matters for prediction, and builds its prediction space on DINOv3, a strong self-supervised encoder (Caron et al., 2021; Oquab et al., 2024; He et al., 2022; Siméoni et al., 2025).

Latent prediction is also long established for world models. The recurrent world model of Ha and Schmidhuber (2018), the Dreamer family (Hafner et al., 2020, 2025a), and transformer world models (Micheli et al., 2023) all predict within the latent of a learned variational or discrete autoencoder, while Zhou et al. (2025) predict future frozen DINO features for planning. Prediction in pixel space remains competitive in some domains, for image generation with pixel-space transformers (Li and He, 2025) and in some world models used to train policies, where fine visual detail matters (Alonso et al., 2024), so the best prediction space remains an open question. We therefore treat the prediction space as a design choice and study it empirically in Section 6.3.

2.6 Long-horizon, real-time rollout and the training objective

The second design question concerns keeping a rollout coherent over a long horizon while producing it in real time. Rolling out autoregressively exposes a mismatch between training, where the model conditions on clean ground-truth frames, and inference, where it conditions on its own predictions, so small errors compound into drift. This exposure bias has long been studied in sequence modeling (Bengio et al., 2015; Lamb et al., 2016). Several schemes reduce it for video: diffusion forcing assigns each frame an independent noise level so the model trains under partially corrupted context (Chen et al., 2024), history guidance conditions on a variable number of past frames (Song et al., 2025), self-forcing unrolls the model on its own samples during training (Huang et al., 2025), and sliding-window noise schedules limit accumulation over long videos (Ruhe et al., 2024; Kim et al., 2024; Liu et al., 2025). Few-step training objectives such as consistency and shortcut models (Song et al., 2023; Song and Dhariwal, 2024; Luo et al., 2023; Boffi et al., 2025; Frans et al., 2025) reduce the number of sampling steps a model needs, which matters because an interactive simulator must render each frame before the next action arrives; our model predicts at 10 Hz in latent space and renders video at 20 fps. We compare diffusion forcing and teacher forcing within our world model and measure their long-horizon stability in Section 6.4.

2.7 Evaluating generative world models

Evaluating generative world models requires more than visual fidelity. The Fréchet Inception Distance and Inception Score capture per-frame realism (Heusel et al., 2017; Salimans et al., 2016), and the Fréchet Video Distance extends it to short clips (Unterthiner et al., 2018), all distributional metrics that need no aligned

reference. Reference-based metrics such as PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) instead score each frame against a ground-truth target, which suits short rollouts with a known reference but not the many plausible futures of a stochastic world model. Recent benchmarks score video generation along human-aligned axes such as temporal consistency and motion quality (Huang et al., 2024), probe physical plausibility (Motamed et al., 2025; Bansal et al., 2024; Meng et al., 2025), and judge video models directly as world models (Duan et al., 2025; Li et al., 2025a), though they target text-to-video generation rather than action-conditioned simulation. A standard alternative probes frozen features with lightweight readout models to recover interpretable state variables (Alain and Bengio, 2017; Li et al., 2023; Maes et al., 2026); we adapt this to recover game-state quantities and measure physical consistency. The same probing idea extends to actions: playable video generation predicts the action from generated motion (Menapace et al., 2021), and we build on this to introduce the Action Recoverability Ratio (ARR), a metric that measures how faithfully the rollout obeys the commanded actions (Section 6.2). None of these metrics captures slow temporal drift over long autoregressive rollouts, a failure mode highlighted by recent work on video generation (Chen et al., 2024; Yin et al., 2025; Liu et al., 2025). We therefore complement visual fidelity with metrics for physical consistency and long-horizon stability, validated against human judgment (Section 6.2).

3 Data

We train on 10,000 match-hours of recorded 2v2 Rocket League matches, generated entirely by self-play between game-playing bots. We did not use any human data to build this dataset. Each match yields four synchronized first-person recordings, one per player. Every recording pairs the gameplay video with the player's action stream and the underlying physics state, all aligned to the image frame boundaries on a shared timeline. We first describe how the data is produced and recorded (Section 3.1) and then the physics game state recorded alongside each frame (Section 3.2); the processing that turns the raw per-player recordings into the frame-aligned chunks the model trains on is detailed in Section B.1.

3.1 Data collection

Game environment. We restrict the game to its 2v2 mode and to three arenas chosen for visual and layout diversity: Champions Field, Forbidden Temple, and Deadeye Canyon. Every car in every match is driven by Nexto (Braaten and Necto contributors, 2022), the highest-skilled publicly available Rocket League bot. Nexto is a state-based policy trained by self-play reinforcement learning in RLGym (RLGym contributors, 2021). It observes the privileged game state, meaning all of the game's internal parameters rather than the pixels. Fifteen times per second, once every 8 game ticks, it emits three-valued axis commands $\{-1, 0, 1\}$ for steer, throttle, yaw, pitch, and roll, together with binary jump, boost, and handbrake actions. Using a single bot to drive all four cars makes the matches reproducible and removes any dependence on human players, at the cost of behavioral diversity, which we discuss later in the section.

Match generation. Each game instance runs a custom BakkesMod (Mulder and BakkesMod contributors, 2016) plugin that both records the session and bridges the game to the bot: it extracts the game state, passes it to the Nexto process, and applies the actions the bot returns. A centralized orchestrator pools virtual machines (VMs) into matches and assigns teams. One machine is designated host and starts a LAN match; the next three machines in the queue are told to join it, and once all four players are present the match begins and runs to completion. A single match is the atomic unit of data: matches in which any VM had problems (network lag, recording hiccups, or a failure to play to completion) are discarded before they reach the dataset.

Recorded signals. For each player we record three time-synchronized streams, summarized in Table 1: the gameplay video, the game state (ball and per-car physics, scores), and the bot's actions. Actions are captured at three layers, so downstream consumers can choose the level they need: (a) the policy's intended action (the raw Nexto output); (b) the mapped keypresses we actually issue to the game; and (c) the action the engine ultimately consumed. The world model trains on layer (b); the other two serve as sanity checks. Nexto's outputs are translated to a canonical nine-key keyboard vocabulary: its three-valued ($\{-1, 0, 1\}$) axis commands drive the six directional keys (`W`, `A`, `S`, `D`, `Q`, `E`), and its binary outputs drive `SPACE`, `LSHIFT`, and `LCTRL`,

whose in-game meaning depends on whether the car is grounded or airborne; the keybindings and the full policy-to-keyboard mapping are given in Section B. All four per-player recordings from a match share a common match identifier and are aligned on the shared kickoff-countdown anchor, so they stitch together cleanly. Processing resamples these raw streams into the 20 fps video and per-frame action labels the model trains on (Section B.1).

Table 1 Recorded streams (per player). Each match produces four of these streams, one per player.

Stream	Contents	Sample rate
Video	Per-player gameplay (720p)	30 fps
Game state	Ball + per-car physics, scores, events	120 Hz (physics tick)
Actions	Nexto output: 3-valued axes + buttons	15 Hz (every 8 ticks)

Scale. After processing, the training set comprises ≈ 10,000 match-hours of clean gameplay (82,983 matches, or 331,932 per-view recordings), evenly split across the three arenas. The full per-map breakdown is given in Section B (Table 15).

Limitations. Every car is driven by its own independent instance of the same bot policy: each VM runs its own game instance with its own Nexto process controlling a single car, so the four cars in a match are four separate instances of the same policy rather than one multi-agent controller. This limits behavioral diversity, as the dataset reflects one policy's style of play rather than the full range of human strategies. Environment diversity is likewise limited: all matches are played on three fixed maps, whereas games with procedurally generated worlds such as Minecraft expose a model to far more varied scenes. The setting also carries little stochasticity, since the game is nearly deterministic once all four players' actions are known, leaving less predictive uncertainty than partially observed environments.

3.2 Physics game state

Alongside video and actions, we log the game's privileged physics state: the exact internal configuration of the simulation rather than what is visible on screen. This signal is not consumed by the world model, which trains on pixels and actions alone; instead it serves as ground truth for evaluation, letting us probe whether the learned representations encode physically meaningful quantities such as ball and car dynamics (Section 6.2).

Extraction. The same BakkesMod plugin that bridges the game to the bot (Section 3.1) reads the physics state directly from the game's internal memory. Rocket League advances its physics at 120 steps per second, and we hook the physics step so that, immediately after every tick, we record the state the engine just computed. Collection is therefore exactly in lockstep with the simulation: one physics game state per tick at 120 Hz, with no interpolation or resampling at capture time. This is the same extraction that supplies the game state to Nexto.

Contents. Each state comprises match-level information (score, time remaining, overtime flag), the ball's physics (position, velocity, orientation, and spin), and one entry per car for all four cars on the field (pose, velocity, boost, and gameplay flags such as ground/wall contact, jump and dodge status). Positions are in Unreal units in the game's Z-up world frame. The full schema, units, and physical value ranges are given in Section C.

4 Method

We model 2v2 Rocket League with a world model that operates in the latent space of a representation codec (Figure 2). After formalizing the prediction problem (Section 4.1), we describe the codec and then the world model that runs in its latent. The codec is a video representation autoencoder built on a frozen self-supervised

feature extractor (Section 4.2): it compresses frames spatially and temporally into a compact latent, and decodes that latent back to video. Within this latent space, the world model, a flow-matching transformer (Section 4.4) predicts the future autoregressively, one latent frame at a time, and the codec decodes the predicted latents into the video the players see and act on. Training this transformer for stable long-horizon rollouts (Section 4.3), modeling all four players jointly (Section 4.5), and running it in real time (Section 5) each raise a distinct problem.

4.1 Problem setup

We consider a game with P players; the two-versus-two matches we study have $P = 4$. Each player p observes their own video stream of the shared match and issues a stream of actions encoding their keyboard inputs, meaning one video stream and one action stream per player. A codec compresses each player's video, both spatially and temporally, into a sequence of latent frames at a lower rate than the video (Section 4.2); we index these latent frames by t, writing z_t^p for player p's latent at step t and a_t^p for that player's actions over the same step. The world model defines a distribution over all players' next latents given the past latents and every player's actions,

$$p_\theta\left(z_{t+1}^{1:P} \mid z_{\le t}^{1:P},\, a_{\le t}^{1:P}\right),$$

which is rolled out autoregressively and decoded back to frames. We learn p_θ from recorded play alone, without access to the game engine or to any privileged state.

4.2 Codec

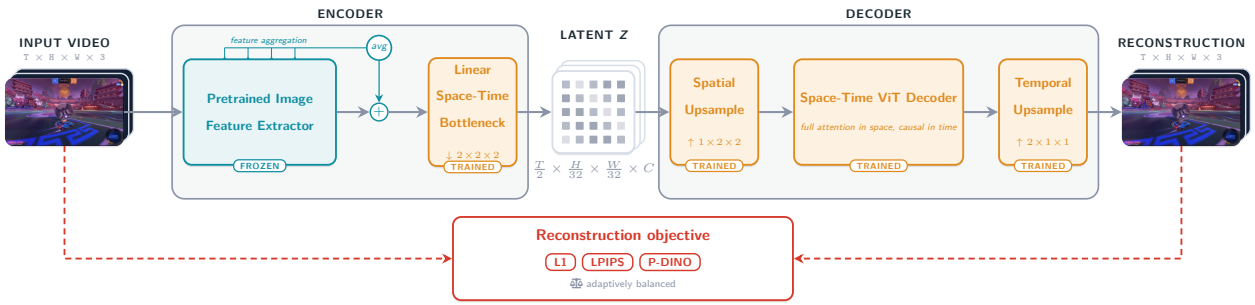


Figure 3 Codec. A frozen pretrained feature extractor (DINOv3-L) extracts per-frame features, mixed across several intermediate layers, and a learned linear bottleneck downsamples them by 2×2 in space and $2\times$ in time and projects to a latent z with C channels. The decoder reverses these steps: a spatial upsampling restores the spatial resolution, a causal spatio-temporal vision transformer reconstructs the video, and a temporal upsampling restores the frame rate.

Predicting in pixel space is expensive, spends capacity on high-frequency detail unrelated to dynamics, and is too slow to roll out in real time. We instead predict in the latent space of a *representation codec*: an autoencoder whose latent is built on a frozen self-supervised feature extractor, so the prediction space inherits strong, semantically meaningful features instead of being learned from scratch, following the representation-autoencoder idea (Zheng et al., 2025). The codec must preserve the feature extractor's representation, stay compact enough for real-time prediction, and decode back to sharp video.

Pretrained feature extractor. The feature extractor is a frozen DINOv3-L/16 (Siméoni et al., 2025), applied per frame. Following RAEv2 (Singh et al., 2026), the codec reads several of DINOv3's intermediate blocks, not its final layer alone. Writing f_ℓ for the patch features at block ℓ and $\mathcal{S}$ for the chosen blocks, we aggregate

$$\bar{f} = \frac{1}{|\mathcal{S}|} \sum_{\ell \in \mathcal{S}} f_\ell + f_{\ell_{\max}},$$

the mean over the selected blocks plus the deepest of them ($\ell_{\max} = \max \mathcal{S}$): the earlier blocks supply spatial and low-level detail that the deepest layer has abstracted away, while $f_{\ell_{\max}}$ retains its semantics. The feature extractor stays frozen; we train only the bottleneck and the decoder.

Bottleneck. A learned *linear* bottleneck maps $\bar{f}$ to the latent with a single $2 \times 2 \times 2$ patchifier and reduces the channel dimension from 1024 to 32. Building on DINOv3's 16×16 patches, the bottleneck coarsens the spatial grid by a further $2\times$ per side, so each latent token covers a 32×32 pixel region; its temporal patchifier halves the rate, so the bottleneck emits latents at 10 Hz from the 20 fps video, about a $192\times$ reduction in the number of values relative to the raw RGB frames. This departs from a standard representation autoencoder, which keeps the feature extractor's grid, frame rate, and dimension: our single learned linear bottleneck instead compresses along two axes at once. It downsamples in space and time, so the shorter latent sequence makes real-time rollout feasible, and it reduces the latent dimension, staying linear so it compresses the features without distorting them.

Decoder. A vision transformer decoder with factorized spatial (bidirectional attention within a frame) and temporal (causal: each token attends only to the same spatial position in earlier frames) attention reconstructs the video. Its first layer is an unpatchify layer that upsamples the latent by 2×2 spatially, from the coarse latent grid (one token per 32×32 pixels) back to the feature extractor's finer patch grid (one token per 16×16 pixels), so the attention blocks decode at that finer grid rather than at the compressed latent; this single step is important for reconstruction quality, and we ablate it in Section 6.3. Time is upsampled only at the end, in the unpatchification that expands each latent frame into two video frames, so the attention runs at the latent's 10 Hz rate.

Loss. We train the codec to reconstruct frames with an L1 loss, an LPIPS perceptual loss (Zhang et al., 2018), and a feature-consistency loss that re-encodes the decoded frames and matches their DINOv3 features, read at several intermediate layers, to those of the originals,

$$\mathcal{L} = \|x - \hat{x}\|_1 + \lambda_p \,\mathrm{LPIPS}(x, \hat{x}) + \lambda_d \,\|\phi(x) - \phi(\hat{x})\|_2^2,$$

with ϕ the frozen DINOv3-L features stacked over those layers and $\hat{x}$ the reconstruction. This mirrors LPIPS, which compares reweighted intermediate VGG activations; using DINOv3 features in this role improves reconstruction and generation quality in recent autoencoders and pixel-space generators (Hansen-Estruch et al., 2026; Ma et al., 2026; Ramanana Rahary et al., 2026). We set the perceptual weights λ_p and λ_d adaptively, reusing the gradient-norm balancing that VQ-GAN (Esser et al., 2021) applies to its single adversarial term. Our codec has no discriminator; we apply the same rule to each of our two perceptual losses, giving LPIPS (λ_p) and the feature-consistency loss (λ_d) their own weights and rescaling each every step so its gradient magnitude matches that of the reconstruction loss at the decoder's last layer θ,

$$\lambda = \frac{\|\nabla_\theta \mathcal{L}_{\mathrm{rec}}\|}{\|\nabla_\theta \mathcal{L}_{\mathrm{perc}}\| + \epsilon},$$

with $\mathcal{L}_{\mathrm{rec}} = \|x - \hat{x}\|_1$ the reconstruction loss, $\mathcal{L}_{\mathrm{perc}}$ either perceptual term, and ϵ a small constant, so neither perceptual term dominates training regardless of its raw scale. We ablate this rule against fixed weights in Section 6.3. Two ingredients common in latent autoencoders are deliberately absent. We use *no adversarial loss*: Hansen-Estruch et al. (2026) show that a feature-space objective can replace the discriminator and train stably at scale, and our codec follows suit. We also inject *no noise* into the latent and impose no KL penalty: where representation autoencoders and variational autoencoders regularize the latent with added noise or a Gaussian prior (Zheng et al., 2025; Yang et al., 2025a), our latent is deterministic, kept faithful by the linear bottleneck and recoverable through the consistency loss.

We evaluate the codec's design choices, from the feature extractor it builds on to the scale of the decoder, in Section 6.3.

4.3 Generative objective

We train the world model with flow matching (Lipman et al., 2023; Liu et al., 2023b). For a target latent z_1 and a Gaussian noise sample z_0, we interpolate $z_\tau = \tau z_1 + (1 - \tau)\, z_0$ and regress the velocity that carries noise to data,

$$\mathcal{L} = \mathbb{E}_{\tau, z_0, z_1} \left\| v_\theta(z_\tau, \tau) - (z_1 - z_0) \right\|_2^2,$$

with flow time $\tau \in [0, 1]$. A rollout integrates v_θ from noise to a latent over a few steps, conditioned on a bounded window of past latents and the players' actions; Section 5 details how the rollout is run in practice.

Diffusion forcing. Each frame receives its own flow time τ, drawn independently (Chen et al., 2024), so a training sequence mixes clean, lightly noised, and heavily noised frames. This emulates what the model meets at rollout, where it conditions on its own imperfect past predictions, and keeps small per-step errors from compounding over a long horizon. We ablate it against teacher forcing, which conditions on fully clean context, in Section 6.4.

Few-step distillation. Sampling a frame with standard flow matching takes many sequential integration steps, which caps the rollout speed, so we distill the pretrained model for few-step sampling with a progressive self-distillation objective (PSD), following shortcut and consistency models (Frans et al., 2025; Song et al., 2023; Boffi et al., 2025). We condition the velocity network on a step size Δ alongside the noise level s, so that $v_\theta(z_s, s, \Delta)$ predicts the *average* velocity of the probability-flow ODE over the interval $[s, s{+}\Delta]$, and a single evaluation advances the sample across the whole interval; setting $\Delta{=}0$ recovers the instantaneous velocity of the standard flow-matching loss. To keep this average-velocity field self-consistent, the model maps one large step to two smaller ones: for a sampled pair $s < t$ with midpoint $u = (s{+}t)/2$, the student $v_\theta(z_s, s, t{-}s)$ regresses onto a stop-gradient two-hop target,

$$\tfrac{1}{2}\big[\, v_\theta(z_s, s, u{-}s) + v_\theta(\hat{z}_u, u, t{-}u) \,\big], \qquad \hat{z}_u = z_s + (u{-}s)\, v_\theta(z_s, s, u{-}s),$$

which replaces one step with two half-size steps. We mix this term into training either stochastically or deterministically with a fixed weight, and the distilled model then matches many-step sampling quality in one or two function evaluations, making real-time rollouts feasible.

4.4 World model architecture

This subsection specifies the velocity network v_θ trained by the objective of Section 4.3. We describe it for a single player and defer the tiling that handles all four players to Section 4.5.

The world model is a diffusion transformer (Peebles and Xie, 2023; Ma et al., 2024) adapted to video with factorized space-time attention (Bertasius et al., 2021; Arnab et al., 2021) (Figure 4). During training it denoises a whole chunk of latent frames jointly, each frame at its own flow time (Section 4.3); at rollout it generates one latent frame at a time, autoregressively (Section 5).

Tokenization. It operates on the grid of continuous latent vectors: a linear layer embeds each latent vector to the model width; with a patch size of one, every latent vector is one token. In the multiplayer setting the per-player views are tiled into a single grid before this embedding (Section 4.5).

Positional encoding. We encode position with factorized rotary embeddings (RoPE) (Su et al., 2024): a two-dimensional axial encoding over the spatial grid and a separate temporal encoding indexed in seconds. Because RoPE encodes position through relative rotations rather than a learned table of absolute positions, the transformer carries no positional parameters tied to a fixed grid size; the same weights run whether a single view or several tiled views fill the grid, which we use when warm-starting the multiplayer model from the single-player one (Section 4.5).

Space-time attention. Within each block, attention runs in two stages. A spatial layer attends bidirectionally within a frame, over all token positions, so every part of the scene informs every other; a temporal layer then attends causally across frames at the same spatial position, so a token attends only to that same location in earlier frames, seeing the past but not the future. The blocks otherwise use components that help large transformers train stably at scale: grouped-query attention (Ainslie et al., 2023), query-key normalization (Henry et al., 2020; Dehghani et al., 2023), a sigmoid gate on the attention output (Qiu et al., 2025), SwiGLU feed-forward layers (Shazeer, 2020), and learnable spatial and temporal register tokens (Darcet et al., 2024). The model is a stack of these blocks.

Action conditioning. The model is action-conditioned through adaptive layer normalization (Peebles and Xie, 2023). At each step we form a conditioning vector c (dimension of the transformer block) by summing an embedding of the flow time τ (Section 4.3) and an embedding of all players' actions (Section 4.5), and

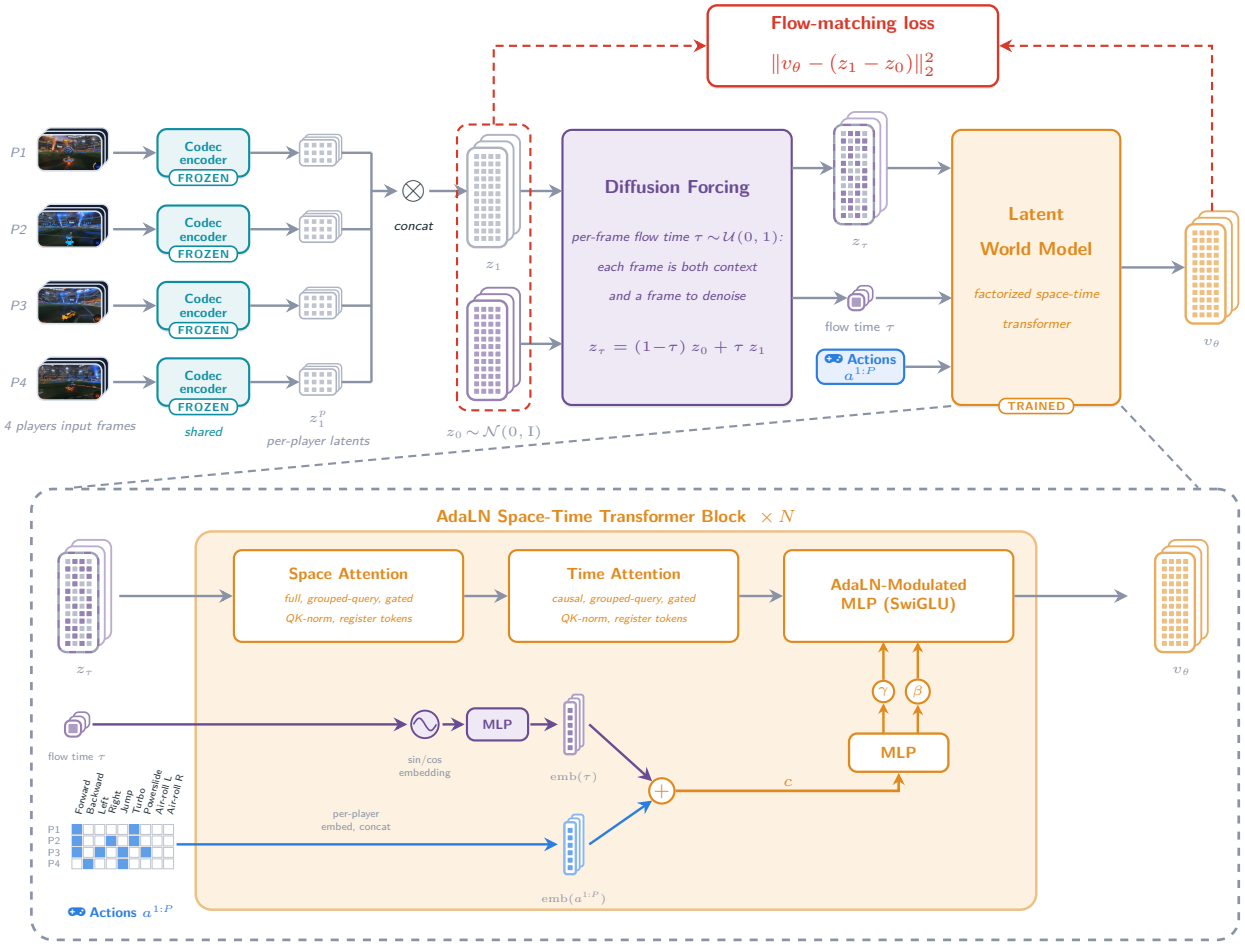


Figure 4 World model training and architecture. *Top:* the training pipeline, read left to right. Each of the P players' frames is mapped by a shared, frozen codec encoder (Section 4.2) to a per-player latent z_1^p, and the P latents are concatenated into a single clean joint latent z_1. Together with Gaussian noise $z_0 \sim \mathcal{N}(0, \mathrm{I})$ of the same shape, *diffusion forcing* (Section 4.3) forms the noised latent $z_\tau = (1-\tau)\, z_0 + \tau\, z_1$, drawing an independent flow time $\tau \in [0, 1]$ for each frame so that every frame is at once clean context and a target to denoise. The trained latent world model, conditioned on all players' actions $a^{1:P}$ and the flow time τ, predicts the flow velocity v_θ, supervised against $z_1 - z_0$ by the flow-matching loss $\|v_\theta - (z_1 - z_0)\|_2^2$ (Section 4.3). *Bottom:* one of the N stacked AdaLN space-time transformer blocks inside the world model. The noised latent z_τ passes through *space attention* (bidirectional, within a frame), *time attention* (causal, across frames), and an *AdaLN-modulated* SwiGLU feed-forward layer. Conditioning enters once per block: the action embedding $\mathrm{emb}(a^{1:P})$ (a per-player embedding of the keyboard controls, concatenated over players) and the flow-time embedding $\mathrm{emb}(\tau)$ (a sinusoidal encoding followed by an MLP) are summed into a single vector c, from which an MLP predicts the scale γ and shift β that modulate the feed-forward layer normalization, $(1+\gamma)\,\mathrm{LN}(x) + \beta$; the same c is broadcast over every spatial position. Here p indexes the P players (P=4 in our 2v2 setting).

we broadcast c over every spatial position. Because actions are recorded once per video frame, at twice the latent rate, a learned linear layer pools the two action frames spanned by each latent frame into one action embedding per latent frame. The conditioning sets the scale and shift of the layer normalization before each feed-forward layer,

$$\mathrm{AdaLN}(x, c) = \big(1 + \gamma(c)\big)\,\mathrm{LN}(x) + \beta(c),$$

with γ and β linear in c, so the actions and the noise level modulate the network in every block. A final linear layer maps each token to the flow velocity the model predicts (Section 4.3), which we split back into the per-player views. Figure 4 illustrates this conditioning path: the action and flow-time embeddings are summed into c, which produces the AdaLN scale and shift γ, β. The past latents enter the model through the causal temporal attention (the space-time attention above), while AdaLN carries only the actions and the flow time.

4.5 Multiplayer conditioning

Tiled views. We stack the four players' latent views into one grid along its height and predict them together. Spatial attention then spans all the views, so the model renders the shared world, the ball, the field, and the other cars, consistently across the four perspectives in a single pass. We find this consistency reflected in the model's cross-view attention (Figure 17).

Per-player actions. Each player's key presses are embedded and passed through a per-player network; the four embeddings are concatenated height-wise in a fixed player order and mixed into the conditioning vector that drives the model through AdaLN (Section 4.4). This order, aligned with the tiling order of the views, binds each action stream to its player. The model conditions on key presses only. Because this conditioning vector is broadcast to every latent position, each player's view is conditioned on all four players' actions, and the model learns which actions affect a given view and which leave it unchanged, rather than being told the mapping.

Action dropout. During training we randomly replace a player's action embedding with a learned absent token, so the model must predict that player from the scene alone and learns to drive the cars whose actions are withheld. At inference this lets the world model itself act as the policy for any player the user does not control, a limited form of agent modeling that needs no separately trained agent.

Two-stage training. We train the world model in two stages: we first pretrain on single-player views, then continue on all four players' views jointly, initialized from the single-player checkpoint, so the model keeps the dynamics it has learned and only adapts to the joint layout and the per-player conditioning. Section 6.6 isolates the ingredients of this recipe: a single player with a single action stream, a single player with the full four-action representation, four players each with their own action stream, and the warm start from the single-player model. The codec is trained in one stage, purely on single-player clips; for the multi-player world model, each player stream is encoded and decoded independently.

5 Streaming inference

We serve the world model as an interactive online demo: a user plays the game from a browser with the keyboard, and the model generates the resulting video on the fly. This imposes a fixed time constraint on the generation of each frame, with a streaming loop that conditions each new frame on the player's most recent actions and on the previously generated frames. Inference runs at the rate of play: the model produces a 10 Hz latent stream that the codec temporally upsamples and decodes to 20 fps video, and a single Nvidia B200 GPU keeps up with this rate. This section condenses the design choices that make real-time, interactive generation possible.

Autoregressive rollout with a streaming cache. We generate the latent stream autoregressively with a key-value cache (Shazeer, 2019): each new latent frame attends to the stored keys and values of the past instead of recomputing them, so the per-frame cost remains constant as the game progresses. Concretely, the

model holds a fixed-size window of $T = 20$ latents, rolling the window by 1 each step to append a fresh noise latent which is denoised into the next frame. Similarly the codec's space-time decoder is conditioned on the three most recent latent frames, enabling constant-cost frame decoding.

Priming the rollout. A session starts from a short clip of real gameplay: we encode its frames to latents and prefill the cache, allowing the model to roll out from a coherent, in-distribution context. During prefill, the demo displays the decoded ground-truth latents; afterwards, during the decode stage, every frame is generated by the model.

Few-step sampling. The model denoises each new frame in a few steps on a linear-quadratic schedule (Polyak et al., 2024), using the distilled few-step sampler from Section 4.3.

Decoupled production and display. Producing frames and displaying them run as a producer/consumer pair. The producer advances the model on the GPU worker thread, copies the decoded frames to the host, JPEG-encodes them off the GPU thread, and pushes them onto a small bounded queue; a separate sender drains that queue at a steady per-frame cadence. Because each world-model step emits two video frames at once, this decoupling turns bursts of two frames per step into evenly spaced delivery. The bounded queue also provides backpressure: when it is full the producer blocks, which paces generation to the display rate and keeps the model from running ahead of what the client can show.

System optimizations. We further incorporate system optimizations for real-time serving. We use the TorchInductor backend in PyTorch (Ansel et al., 2024) for operator fusion and CUDA graphs to reduce host overhead. We also use different attention backends depending on the stage and type of factorized attention. CuDNN (Chetlur et al., 2014) is used for spatial attention and during prefill for temporal attention, while custom Triton (Tillet et al., 2019) kernels are used during the decode stage.

Real-time rollout. We aim to serve at 20 fps on a single Nvidia B200 GPU for the 5B model with a latent window size of 20. Combining all the optimizations presented above, one full step, from the world-model update through decoding, takes roughly 70 ms end to end and produces two video frames (about 35 ms per frame), comfortably inside the 50 ms budget of a 20 fps stream. Running faster than real time leaves headroom to absorb scheduling jitter and to pace delivery smoothly (below).

6 Experiments

Our experiments cover three design axes. The first is the prediction space. We predict in a learned latent; Section 6.3 tests this against pixels, then studies each part of the codec that produces the latent: the feature extractor, the bottleneck, and the decoder. The second is the world model's training objective; Section 6.4 compares diffusion forcing (Chen et al., 2024) against teacher forcing for long-rollout stability, and Section 6.5 checks whether the trained model actually executes the actions it is conditioned on. The third is conditioning on four simultaneous players (Section 6.6). We then measure how performance scales with model and data size (Section 6.7), and close with the properties that emerge in the trained model (Section 6.8) and the failure modes that remain (Section 6.9).

6.1 The flagship model

The model behind our live demo is a 5-billion-parameter multiplayer world model, the largest configuration in this paper and the culmination of the design choices the rest of this section validates: a 600M-parameter video representation codec built on a DINOv3-L feature extractor with a trained bottleneck and decoder (Section 4.2), flow matching with diffusion forcing (Section 4.3), and tiled four-player conditioning (Section 4.5). It is warm-started from a single-player model and then trained on multiplayer data over a dataset comprising ∼10,000 hours of gameplay; its full configuration is given in Table 11.

It simulates two-versus-two Rocket League interactively and in real time, rendering all four players' views at 20 frames per second on a single Nvidia B200 GPU, and stays controllable under four simultaneous action

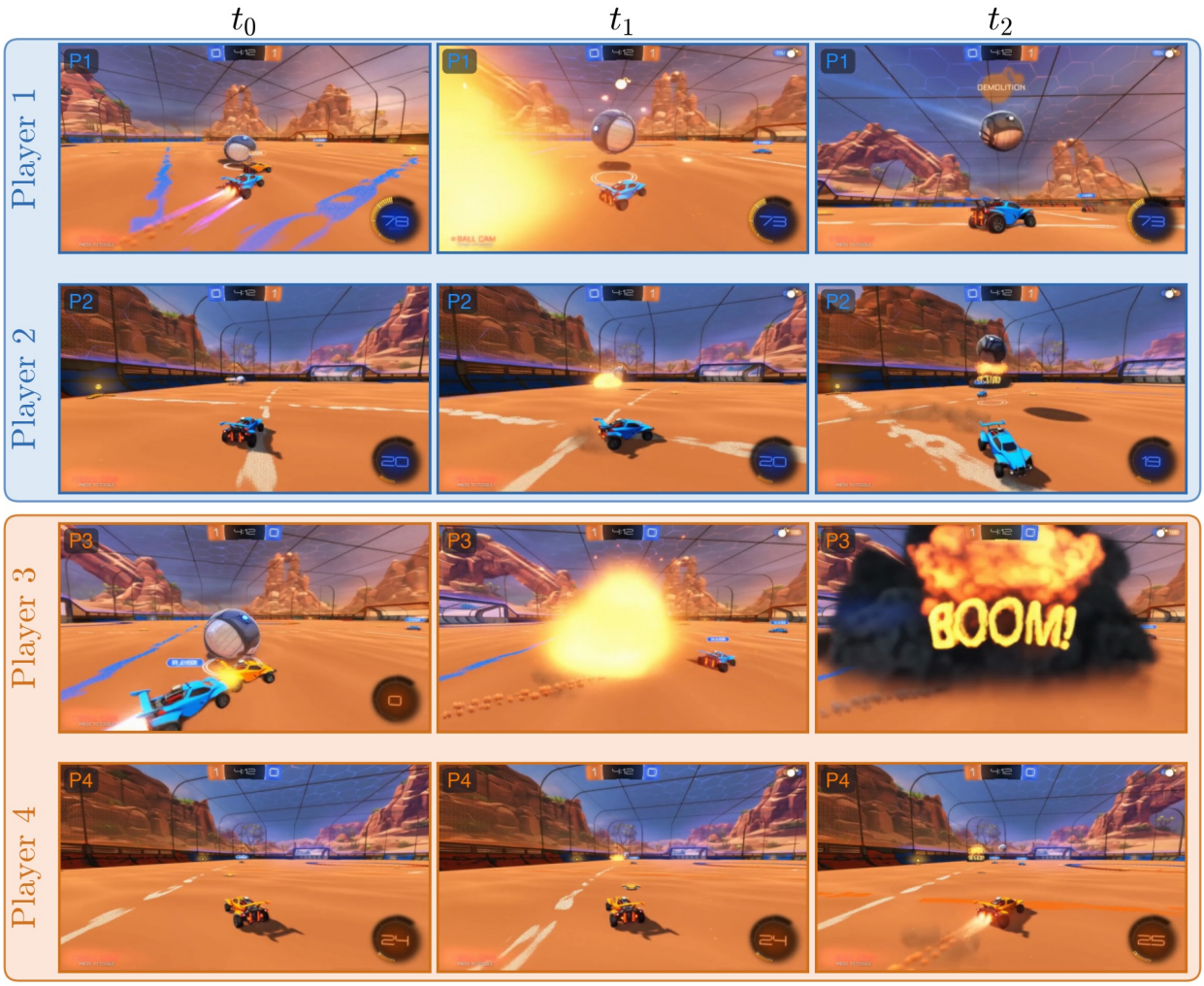


Figure 5 World model imagination of a demolition event. Rows are the four players' viewpoints (Players 1–4) and columns show three timesteps ($t_0 \to t_1 \to t_2$) of an imagined rollout in the desert arena. From the initial state at t_0 and the players' action streams, the model imagines a fast, sub-second collision: a blue car boosts into an opponent and demolishes it. The event stays mutually coherent across viewpoints. At t_2, Player 1 renders the "DEMOLITION" callout for its successful hit, while Player 3 (the demolished car) sees the "BOOM!" explosion fill its own view, and Player 2 renders the very same blast as a distant burst of smoke on the pitch. The in-game clock (4:12) is consistent across all players and barely advances, reflecting how brief the event is.

streams, including under live human play. Its rollouts stay coherent far beyond the training window, running without collapse for far longer than they are trained on (Section 6.8). Most importantly, it reproduces the game's dynamics rather than only its appearance: cars are knocked off course by contact and bump one another, demolitions are rendered consistently from every camera (Figure 5), and a shot into the net sets off the goal explosion and the on-screen SCORED! callout across all four views (Figure 6). A game-state probe further confirms that the rolled-out car and ball positions track the true physics (Figure 21).

6.2 Evaluation metrics

Because the world model generates seconds of video whose exact future is unknowable, we judge a rollout by its realism rather than by an exact match to the reference. We evaluate the codec and the world model along five complementary axes: distributional distances that compare generated or reconstructed video against real play, tracked against the rollout horizon to reveal drift; pixel-level and perceptual distances that measure frame-by-frame fidelity; a probe of physical state; a controllability metric that tests whether the model

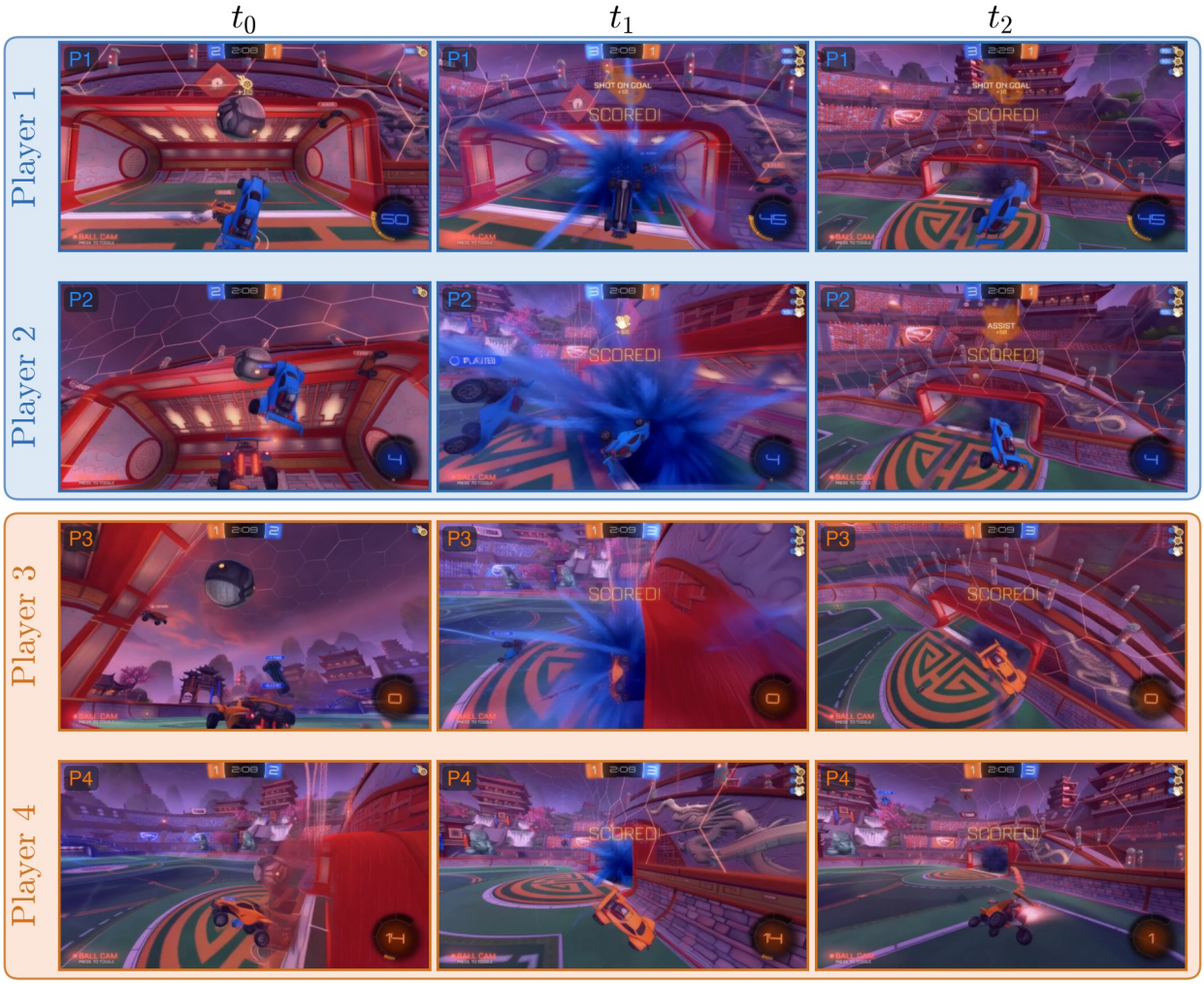


Figure 6 World model imagination of a goal. Rows are the four players' viewpoints (Players 1–4) and columns show three timesteps ($t_0 \to t_1 \to t_2$) of an imagined rollout in the temple arena. From the initial state at t_0 and the players' action streams, the model imagines the ball being driven into the net for a goal. The event stays mutually coherent across viewpoints: at t_1 the "SCORED!" callout and the blue goal-explosion appear simultaneously in all four players' views, each rendering the same blast from its own camera, and by t_2 every player sees the aftermath on the pitch. The rollout is also temporally consistent: the scoreline updates from 2–1 to 3–1 for the blue team as the goal registers.

executes the actions it is given; and human evaluation. Distributional distances compare statistics over a set of clips and capture global realism, while pixel-level metrics are computed per frame and capture local fidelity. We prefix each distributional metric with r or g to indicate whether it is applied to codec *reconstructions* (rFID, rFVD, rFDD) or to world-model *generations* (gFID, gFVD, gFDD).

Distributional distances. The Fréchet Inception Distance (FID) (Heusel et al., 2017) measures the distance between the feature distributions of real and generated frames, extracted by an Inception-V3 network trained on ImageNet. The Fréchet Video Distance (FVD) (Unterthiner et al., 2018) extends this to short video clips using a 3D feature extractor, making it sensitive to temporal artefacts that per-frame FID misses. Because both backbones are trained on natural images, they may transfer imperfectly to Rocket League's synthetic domain, so we complement them with the Fréchet DINO Distance (FDD), a Fréchet distance computed in the feature space of the DINOv3-B encoder (Siméoni et al., 2025), which better supports out-of-domain generalization and has richer features. All three distances are lower for better quality. For reconstruction we apply them to 40-frame codec outputs (2 s windows) against real clips; for generation we apply them to

world-model rollouts conditioned on ground-truth actions, scored in 1 s windows. Unless stated otherwise, we sample each world model with 10 flow-matching steps and re-noise the past context to level 0.2 (its effect on drift is studied in Figure 10), and generation distances are reported at the 4 s horizon of the rollout.

Pixel and perceptual distances. Reconstruction quality is measured per frame. PSNR and SSIM (Wang et al., 2004) compare pixel-level fidelity and structural similarity against the ground truth. LPIPS (Zhang et al., 2018) measures perceptual distance in AlexNet feature space, and P-DINO does the same in DINOv3-B feature space. These metrics apply only to the codec, not to the world model, whose exact future is unknowable. PSNR and SSIM are higher for better reconstruction; LPIPS and P-DINO are lower.

Game-state probing. To test whether generated rollouts respect the physics and rules of the game, we train a probe that reads out interpretable game-state quantities from the world model's internal representations. The probe hooks the last layer of world-model activations, spatially pools them per player, concatenates the four player vectors, and passes the result through a three-layer MLP with hidden dimension 1024. The prediction layer maps to the game state: position, quaternion, and linear velocity for each of the four players and the ball, 50 dimensions in total. We train the probe with an L2 loss against the ground-truth states, reading the world model's activations as it processes the real, encoded latent sequence rather than its own generated rollout. At evaluation time, we instead apply the probe to a generated rollout: we encode a video, roll out the world model conditioned on the ground-truth actions, run the probe on the rollout activations, and compare the predicted states against the corresponding ground-truth physical states. Learning the readout on real latents and testing it on the model's own rollout isolates whether the generated trajectory preserves the same physical information. This metric targets correctness of dynamics rather than appearance.

Action Recoverability Ratio. Distributional and pixel metrics judge how a rollout looks, not whether it obeys the actions it is given. To measure controllability, we test whether each commanded action is recoverable from the generated video. We train an action probe, a frozen DINOv3-B encoder (Siméoni et al., 2025) with a small attention-pooling head, to detect which of the nine controls are pressed within an 8-frame (0.4 s) window, supervised by the logged key presses with a per-action binary cross-entropy. Only the head is trained, so the probe reads domain-general features and transfers across real, reconstructed, and generated frames. We score its detections with *average precision* (AP), the area under a per-action precision–recall curve; on real held-out video the probe reaches 0.84 mean AP (mAP, averaged over the nine actions), which makes it a reliable instrument for this calibration (Section D). For a rollout conditioned on the real actions, we slide the probe over both the generated video and the codec reconstruction of the same clip and compute each action's AP against the commanded actions, $\mathrm{AP}_{\mathrm{gen}}$ and $\mathrm{AP}_{\mathrm{recon}}$. The action recoverability ratio for action a is

$$\mathrm{ARR}(a) = \frac{\mathrm{AP}_{\mathrm{gen}}(a)}{\mathrm{AP}_{\mathrm{recon}}(a)},$$

averaged over the nine actions and reported with bootstrap confidence intervals over clips. The reconstruction is the achievable ceiling, since a perfect world model given the real actions would reproduce the real clip's latents; dividing by it cancels the probe's own imperfection and the codec's visual domain, leaving only how faithfully the world model renders the action. An ARR of 1 means actions are as recoverable from the generation as from a faithful reconstruction, below 1 means the model under-renders them.

Human evaluation. Automatic metrics are proxies, so we corroborate them with human preference studies scored by a Bayesian Elo. Two protocols mirror our two main axes. For *quality*, raters compare a pair of models rollout-for-rollout from the same initial frame and action sequence and pick the video that better matches the ground-truth reference, both in appearance and in the end state of the play. For *action adherence*, we build a hybrid clip by applying one reference's actions to another's initial frame, holding appearance and starting point fixed while the actions differ; raters then judge which clip follows the reference actions more faithfully, the human analog of ARR (Action Recoverability Ratio, defined above). Sample and rater counts vary by comparison. We report the full studies in Section I; the action-adherence study validates ARR in Section 6.5.

6.3 Designing the codec's latent space

The codec learns to reconstruct video, but the world model must generate in its latent space, and good reconstruction does not guarantee an easy-to-generate latent (Yao et al., 2025; Xu et al., 2026). We therefore judge each codec by the world model trained on it, and treat reconstruction quality as secondary evidence.

All codec ablations run in a single-player setting, on one player's 288×512, 20 fps view; the codec operates per view, so one view is enough to compare codecs. The baseline is a representation codec: a frozen DINOv3-L feature extractor (Siméoni et al., 2025) extracts per-frame features, a learned linear bottleneck compresses them from 1024 to 32 channels and downsamples them to one latent cell per 32×32 pixels at 10 Hz, half the input rate. A spatio-temporal ViT-XL decoder, causal in time, reconstructs the full-rate video. The feature extractor and bottleneck shape the latent; the decoder governs only how faithfully it is rendered back to pixels. The ablation runs use a shorter schedule than the flagship: each codec is taken at its 125k-step checkpoint on 40-frame clips (Table 9 gives the baseline codec's full configuration), and a 1B-parameter world model then trains for 100k steps at global batch 16 on 80-frame clips on that codec. Both stages process 1280 frames per step, so every ablated codec sees 160M frames and every world model 128M frames.

For each codec we report reconstruction quality, with pixel and perceptual metrics (PSNR, SSIM, LPIPS, and P-DINO, a perceptual distance in DINOv3 feature space) and reconstruction Fréchet distances (rFID, rFVD, rFDD). For the world model trained on that codec, we roll out 12 s conditioned on the ground-truth actions and measure distributional distances to real play (gFID, gFVD, gFDD) in 1 s windows, and summarize each rollout by its distance at the 4 s horizon: 4 s is the model's training window, long enough to exercise multi-step generation yet short enough to keep models discriminable. We score reconstruction on 40-frame clips and every distance over a frozen set of 2048 clips (Section 6.2). In every table below, the best value in each column is bold and the second best underlined, generation columns are taken at the 4 s horizon, and dashes mark runs not yet scored.

Designing the latent space, in brief.

We judge every codec by the world model trained on it and select for an easy-to-generate latent, treating reconstruction quality as secondary.

Presented in the main text:

- **A latent is essential.** Pixel-space world models trail ours by an order of magnitude on generation and controllability (Table 2).
- **The feature extractor must be pretrained.** A frozen DINOv3 extractor makes the latent easier to generate and far more stable over long rollouts, whereas a from-scratch extractor reconstructs more sharply yet drifts (Table 3, Figure 7).
- **The bottleneck is best learned, but need not be.** A learned linear bottleneck adds a small, consistent margin; a random or PCA projection of the pretrained features is already predictable (Table 4).
- **Temporal downsampling is free.** A 10 Hz latent leaves generation unchanged while halving the world model's sequence length, buying the real-time 20 fps rollout at no cost in quality (Table 5).
- **Perceptual losses are essential and sufficient.** LPIPS and P-DINO are complementary and removing both collapses generation; together they reach enough fidelity that no adversarial (GAN) loss is needed (Table 6).
- **Decoder quality saturates with scale.** Reconstruction improves as the decoder grows but with diminishing returns beyond the Large (0.3B) decoder (Table 7).

Detailed in Section F:

- **Any reasonable pretrained extractor works.** Smaller DINOv3 variants and EUPE-B trail DINOv3-L only slightly, and all resist drift (Table 21).
- **Read several feature-extractor layers, not just the last.** Averaging a spread of the feature extractor's intermediate blocks keeps spatial detail that the deepest block discards (Table 22).
- **Compression can go in the codec or the world model.** The 2× reduction works in either, so we do all of it in the codec (Table 23).
- **Adaptive loss balancing helps.** Gradient-matched weights beat fixed weights on nearly all reconstruction and generation metrics (Table 24).
- **Upsample before decoding, not after.** Expanding the compact latent to the feature-extractor grid at the decoder's input, so the decoder runs on more tokens, far outperforms upsampling only at the output (Table 25).

Is pixel-space world modeling enough? Before designing the latent, we ask whether a codec is needed at all. Pixel-space generation has drawn renewed interest, with transformers that flow-match directly on pixels (Li and He, 2025; Ma et al., 2026). We test two pixel-space world models against our latent-space baseline. The first changes only the world model's input and output, replacing the codec latent with patchified pixels while keeping the architecture, flow-matching velocity-based objective, and training unchanged; the second also adopts the JiT recipe (Li and He, 2025), which feeds the pixel patches through a linear bottleneck before the transformer and predicts the clean signal instead of the velocity. Both pixel-space models train for 225k steps, matching the total budget of the codec-plus-world-model pipeline (125k + 100k). The pixel-space models have no codec and thus no reconstruction to score, so we compare on generation quality and controllability (Table 2).

Table 2 Latent vs pixel space. Our latent world model against two pixel-space world models. All runs match the total training budget (225k steps); the latent splits this into codec (125k) plus world model (100k). The pixel models have no autoencoder, so their controllability (ARR) is calibrated against real frames rather than a reconstruction.

	Steps	gFID↓	gFVD↓	gFDD↓	ARR↑
Latent (ours)	125k + 100k	**10.7**	**163.1**	**0.55**	**0.91**
Pixels, plain	225k	104.9	1456.3	<u>16.19</u>	<u>0.61</u>
Pixels, JiT recipe	225k	<u>81.0</u>	<u>961.2</u>	17.05	0.49

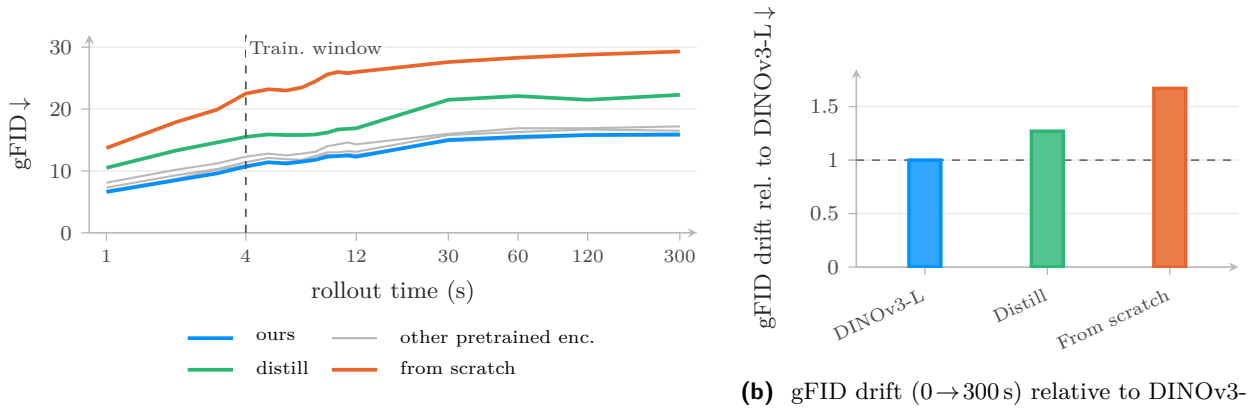
Is pixel-space world modeling enough?

No. Both pixel models trail the latent baseline by an order of magnitude on generation quality, even with the JiT bottleneck and a matched total training budget, and their rollouts are markedly less controllable, with commanded actions far harder to recover; the pixel-space rollout also drifts into warped, unstructured texture within a second (Figure 23). A learned latent is essential.

Must the feature extractor be pretrained? The baseline feature extractor is a frozen, pretrained DINOv3-L. To test whether the pretrained features are needed, we replace it with a feature extractor of the same architecture trained from scratch (randomly initialized, learned end-to-end with the rest of the codec), in two forms: on its own, and with DINO distillation, where a learned linear map W projects the bottleneck latent z to match a frozen DINOv3-L teacher's features f, minimizing the squared error $\|Wz - f\|^2$, so the latent inherits DINO's structure even though the feature extractor is learned (Table 3). Beyond generation quality at the 4 s horizon, we also roll the world model out to five minutes to measure drift (Figure 7).

Table 3 Feature-extractor adaptation. Frozen pretrained DINOv3-L against a feature extractor trained from scratch, with and without DINO distillation.

	Codec (reconstruction)							World model (generation)		
Feature extractor	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
Frozen DINOv3-L (ours)	29.7	0.891	**0.051**	**0.021**	**5.4**	38.6	**0.17**	**10.7**	**163.1**	**0.55**
From scratch	**32.2**	**0.923**	0.068	0.046	13.1	**23.2**	1.39	22.5	375.0	1.93
From scratch + DINO distillation	31.4	0.916	0.056	0.037	8.8	26.2	0.98	15.7	271.5	1.35



(a) gFID over a five-minute rollout (log time) for each codec.

(b) gFID drift (0→300 s) relative to DINOv3-L; the dashed line marks the baseline.

Figure 7 Rollout drift. Each variant uses the same codec architecture and training, differing only in the feature extractor: ours uses a frozen DINOv3-L, "distill" trains the feature extractor from scratch with DINO distillation, and "from scratch" trains it from scratch without distillation. (a) gFID over a five-minute rollout: the codecs with a pretrained feature extractor stay low and nearly flat on long horizons, the distilled codec sits higher, and the from-scratch codec is highest. (b) gFID drift over the five-minute rollout relative to DINOv3-L: the distilled and from-scratch codecs drift 1.3 and 1.7× as much as the pretrained baseline.

We show qualitative examples of drift elsewhere: the pixel-space rollout in Figure 23 and the out-of-distribution excursion in Figure 29.

Must the feature extractor be pretrained?

Yes, and pretraining helps generation twice over.

- **Quality:** a from-scratch feature extractor reconstructs more sharply, yet its latent is far harder to generate, inflating every generation distance at the 4 s horizon.
- **Stability:** over a sustained rollout the from-scratch and distilled codecs drift substantially more than the pretrained baseline (Figure 7); ours stays visibly stable far longer.

We attribute this to the smoothness of the pretrained features: nearby states map to nearby latents, so a wrong prediction stays valid and is absorbed.

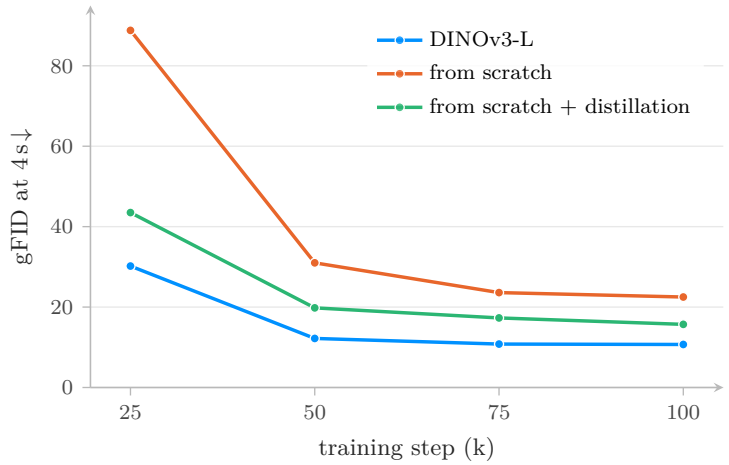


Figure 8 A pretrained feature extractor accelerates learning. gFID at the 4 s horizon against training step, from 25k steps. The latent on a pretrained DINOv3-L feature extractor reaches low gFID far sooner than a feature extractor trained from scratch. Distilling DINO into a from-scratch feature extractor recovers much of the gap.

Does the bottleneck need to be learned? To compress the aggregated DINOv3-L features to the 32-channel latent (one cell per 32×32 pixels, 10 Hz; Section 4.2), our baseline uses a learned linear $2 \times 2 \times 2$ strided convolution. We replace it with progressively simpler maps, holding the feature extractor and decoder fixed: a random frozen projection, a fixed PCA projection with pooling, and pooling alone with no channel reduction (keeping all 1024 channels). The pooling-only latent is high-dimensional, so for that variant we also shift the world model's flow-matching noise schedule toward higher noise, following the RAE recipe (Zheng et al., 2025) (Table 4).

Table 4 Bottleneck. Replacing the learned channel compression with simpler, parameter-free maps. The shifted-noise row reuses the pooling-only codec, so only its world model differs.

	Codec (reconstruction)							World model (generation)		
Bottleneck	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
Learned convolution (ours)	**29.7**	**0.891**	**0.051**	**0.021**	**5.4**	**38.6**	**0.17**	**10.7**	**163.1**	**0.55**
Random frozen projection	28.3	0.869	0.067	0.025	6.6	65.9	0.20	11.9	219.9	0.67
PCA + pooling	28.4	0.869	0.068	0.025	6.8	67.0	0.21	11.7	182.4	**0.55**
Pooling only (no channel reduction)	29.2	0.882	0.055	**0.021**	6.0	46.8	0.18	13.1	243.4	0.60
+ shifted noise schedule	–	–	–	–	–	–	–	12.7	238.1	0.62

Does the bottleneck need to be learned?

Not strictly, but learning helps. A learned linear bottleneck adds a small, consistent margin over a random or PCA projection of the DINO features, yet even those parameter-free maps already give a predictable latent. The pretrained features do most of the work.

Does temporal downsampling hurt quality? The baseline halves the latent's frame rate in the codec (20 fps video to a 10 Hz latent), which shortens the world model's sequence but discards temporal detail. We compare against a codec that keeps the full 20 Hz latent (Table 5); the two are matched in wall-clock context, so the full-rate variant carries twice the world model's latent sequence length.

Table 5 Temporal downsampling. A 10 Hz latent (the codec halves the frame rate) against a 20 Hz latent (no temporal downsampling).

	Codec (reconstruction)							World model (generation)		
Latent rate	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
10 Hz (ours)	29.7	0.891	0.051	0.021	5.4	38.6	0.17	10.7	**163.1**	**0.55**
20 Hz (no temporal ds.)	**30.6**	**0.909**	**0.039**	**0.016**	**4.3**	**23.0**	**0.13**	**10.5**	166.3	0.65

Does latent temporal downsampling hurt quality?

No, it is essentially free. Halving the latent frame rate to 10 Hz only softens reconstruction and leaves generation unchanged, while halving the world model's sequence length, which buys the real-time 20 fps rollout at no cost in generation quality.

What perceptual losses does the codec need? Beyond the L1 reconstruction loss, the codec uses two perceptual terms, an LPIPS term (Zhang et al., 2018) and a P-DINO term (the perceptual DINO distance used as a metric above). We toggle which are active: both, LPIPS alone, P-DINO alone, or neither (Table 6).

Table 6 Perceptual terms. Toggling the LPIPS and P-DINO perceptual losses.

	Codec (reconstruction)							World model (generation)		
Perceptual loss	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
LPIPS + P-DINO (ours)	29.7	0.891	**0.051**	**0.021**	**5.4**	**38.6**	**0.17**	**10.7**	**163.1**	**0.55**
LPIPS only	30.1	0.899	0.067	0.041	10.6	49.9	1.16	14.4	175.6	1.36
P-DINO only	29.8	0.890	0.080	0.028	11.9	61.4	0.43	16.6	196.6	0.83
No perceptual loss	**30.3**	**0.903**	0.104	0.069	25.0	72.3	2.57	26.8	194.1	2.58

What perceptual losses does the codec need?

Both, and only these. LPIPS and P-DINO are complementary, LPIPS helping gFID and gFVD more and P-DINO helping gFDD more; removing either hurts generation and removing both collapses it. Together they reach enough fidelity that we add no adversarial (GAN) loss.

How large must the decoder be? We scale the ViT decoder (Dosovitskiy et al., 2021) across three sizes, Base (12 layers, width 768; ≈85M parameters), Large (24 layers, width 1024; ≈0.3B), and XL (ours; 28 layers, width 1152; ≈0.45B), holding the rest of the codec fixed (Table 7).

Table 7 Decoder size. Base, Large, and XL (ours) decoders, all upsampling before the ViT.

	Codec (reconstruction)							World model (generation)		
Decoder size	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
XL (ours)	**29.7**	**0.891**	**0.051**	**0.021**	**5.4**	**38.6**	**0.17**	10.7	**163.1**	**0.55**
Base	27.6	0.842	0.082	0.029	8.2	89.8	0.27	12.5	201.3	0.66
Large	29.3	0.882	0.055	0.022	5.6	43.1	**0.17**	**10.5**	169.9	**0.55**

How large must the decoder be?

Large is enough. Reconstruction improves with capacity but saturates with diminishing returns from the Large decoder onward, which already matches XL, while a Base decoder clearly degrades reconstruction.

6.4 World model objectives

We train the world model with diffusion forcing (Section 4.3), and test whether it improves on teacher forcing, a long-established sequence-modeling scheme that earlier diffusion world models adopt. We study its effect on sample quality and on robustness to drift over long rollouts, ask whether the context must be noised at inference, and whether sampling can be accelerated by distilling the model into a few-step sampler. All three studies use 1B-parameter world models on the baseline codec, with the same batch size and schedule as the codec ablations (Section 6.3).

How do teacher and diffusion forcing compare? Earlier diffusion world models (Alonso et al., 2024; Valevski et al., 2025) are trained by teacher forcing: at each step the model denoises the next frame from clean, ground-truth context, so it is supervised on a single frame and never trains on the imperfect context it must build on at rollout time. Diffusion forcing (Chen et al., 2024), introduced to narrow this train/inference gap, instead noises every frame of the clip independently, supervising all frames at once and exposing the model to noised context that stands in for the imperfect predictions it conditions on at rollout. We compare the two on the baseline codec, both on sample quality at the 4 s horizon (Table 8) and on drift across a five-minute rollout (Figure 9).

Table 8 Training objective. Teacher forcing against diffusion forcing on the baseline codec; generation quality at the 4 s horizon.

Objective	gFID ↓	gFVD ↓	gFDD ↓
Teacher forcing	32.5	944.1	2.21
Diffusion forcing (ours)	**10.7**	**163.1**	**0.55**

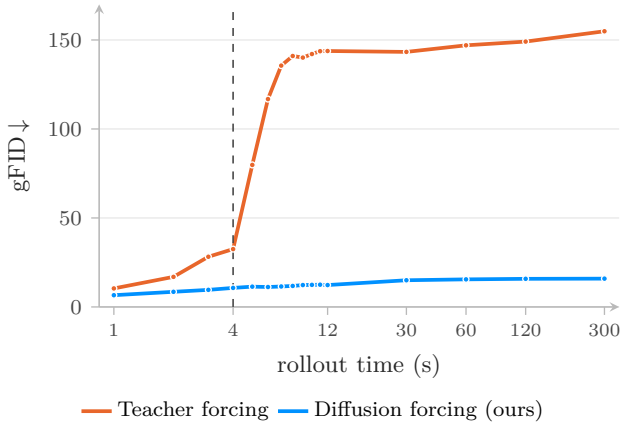


Figure 9 Objective and drift. gFID across a 5-minute rollout (log time) on the baseline codec. Past the 4 s training window (dashed), teacher forcing degrades sharply and stays about 10× higher, while diffusion forcing stays low and flat.

How do teacher and diffusion forcing compare?

Diffusion forcing wins clearly. It is markedly better across the generation metrics at the 4 s horizon, and over a long rollout it limits drift far better: teacher forcing collapses to a much higher gFID while diffusion forcing stays flat (Figure 9).

Should the context be noised at inference? Diffusion forcing trains on partially noised context, so at inference the past latents can be fed clean or re-noised to a chosen level; adding noise to the context is a common way to curb drift in autoregressive diffusion models. We roll out while injecting Gaussian noise into the past latents at standard deviations from 0 to 0.5, and track gFID across the rollout for every codec, so any instability from rolling out on clean latents becomes visible as it compounds over time (Figure 10).

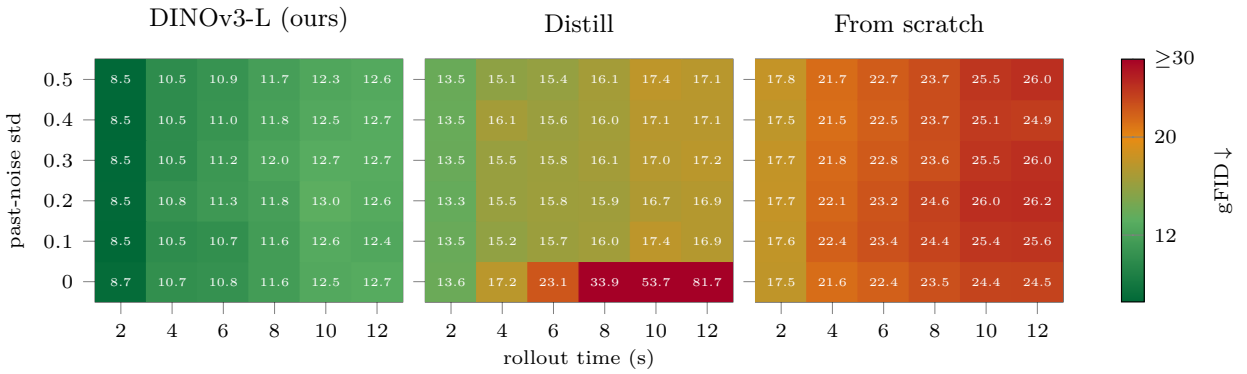
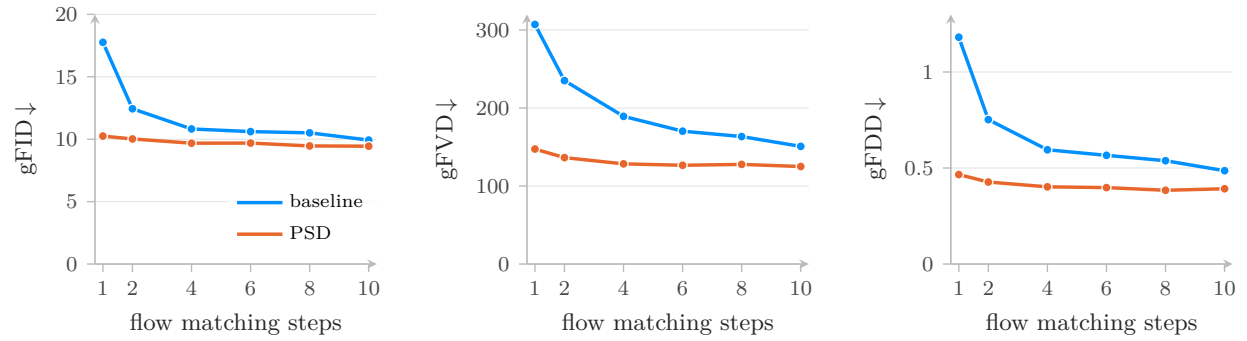


Figure 10 Context noise at inference. gFID as a function of the past-noise standard deviation (vertical) and rollout time (horizontal), for three representative codecs; cell values are gFID and color runs green (low) to red (high, clipped at 30). The distilled codec's quality collapses over the rollout when the past is fed clean (noise 0) and is rescued by a small amount of noise, while the pretrained and from-scratch codecs are flat in noise. The other pretrained feature extractors and the gFVD/gFDD metrics show the same pattern (Section H).

Should the context be noised at inference?

For most codecs, no, but it rescues the fragile one. The pretrained and from-scratch codecs are flat across noise levels even at the 12 s horizon (Figure 10), so they roll out on clean past frames just as well. The distilled codec is the exception: fed a clean past its quality collapses at the long horizon, and a little noise restores it. A light default therefore costs the stable codecs nothing and keeps the fragile one from diverging.

How does few-step distillation affect quality and inference cost? Sampling integrates the flow-matching field over many steps, and this step count sets the inference cost. Self-distillation can shorten it: the model is distilled so that one large step replaces two smaller ones, an idea introduced concurrently as progressive self-distillation (Boffi et al., 2025) and shortcut models (Frans et al., 2025). We compare two settings on the baseline codec: the standard world model, and a model trained from scratch with the self-distillation objective (PSD), applied stochastically on a random 10% of the training updates. Figure 11 traces each generation metric against the number of flow-matching steps.



(a) gFID **(b)** gFVD **(c)** gFDD

Figure 11 Few-step sampling. Generation quality against the number of flow-matching steps, for the baseline and the PSD self-distilled model (trained from scratch).

How does few-step distillation affect quality and inference cost?

Yes. PSD outperforms the baseline at every step count, by a wide margin in the few-step regime, while staying stable even when using far fewer flow-matching steps.

6.5 Controllability

Conditioning the world model on actions does not guarantee that it obeys them. We measure how faithfully the model renders the commanded actions with the action recoverability ratio (ARR, Section 6.2), evaluated on the single-player model.

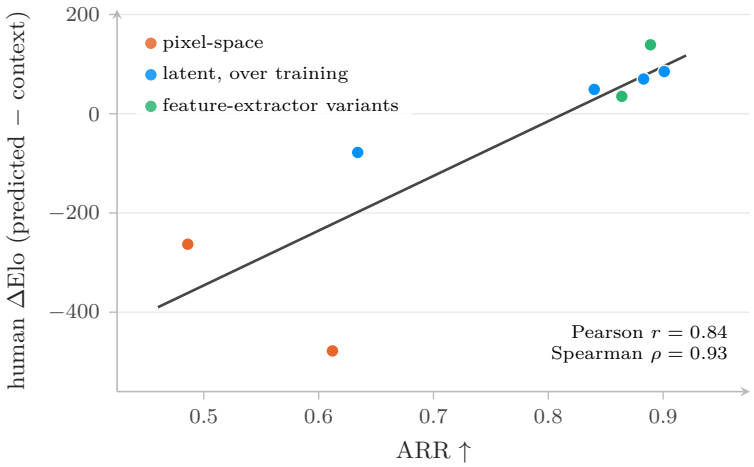


Figure 12 ARR tracks human judgment of controllability. Automatic ARR against the human action-adherence preference (an Elo delta, predicted vs. context) for the models rated in both, spanning pixel-space, feature-extractor, and training-checkpoint comparisons. The two are strongly correlated (Pearson $r = 0.84$) and rank the models almost identically (Spearman $\rho = 0.93$); the line is a linear least-squares fit.

Does the Action Recoverability Ratio (ARR) agree with human judgment? Before using ARR to study the model, we check that it reflects how people perceive controllability: we run a human action-adherence study (Section I) and plot ARR against the human preference for the models rated in both (Figure 12).

Does the Action Recoverability Ratio (ARR) agree with human judgment?

Closely. ARR and human action-adherence preference rank the models almost identically, so ARR captures controllability as human raters perceive it.

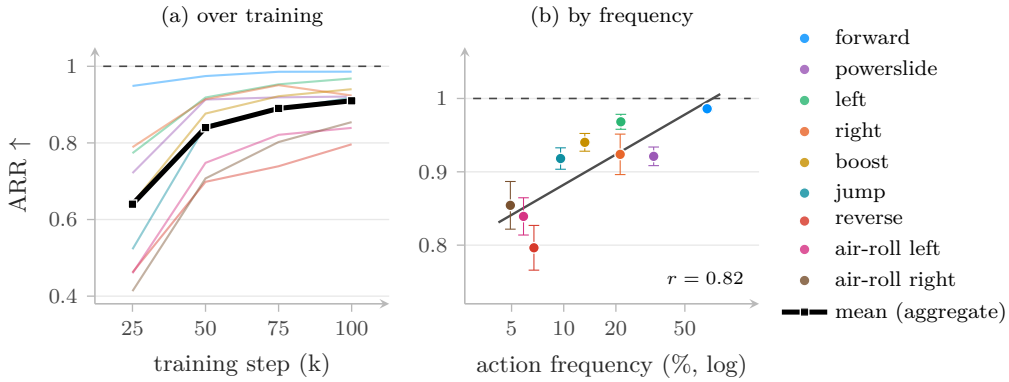


Figure 13 Controllability. (a) Per-action ARR (faint) and their aggregate (bold black) against training step for the single-player model. **(b)** Per-action ARR at the final checkpoint against training frequency, with bootstrap 95% confidence intervals and a log-linear fit ($r = 0.82$). Colors identify the nine controls across both panels (shared legend); the dashed line marks a faithful reconstruction (ARR = 1).

How does controllability improve over training? We measure controllability over the course of training, computing aggregate ARR at successive checkpoints of the single-player model together with the nine per-action curves (Figure 13a), and read it against how generation quality converges over the same run (Figure 8).

How does controllability improve over training?

Gradually, and it keeps improving after image quality has saturated. Aggregate ARR rises steadily and every action becomes more recoverable, approaching the reconstruction ceiling. Generation quality converges far earlier (Figure 8): the model first learns to render realistic frames, then continues to refine how faithfully it obeys the commanded actions.

Are rarer actions harder to control? We test whether infrequent actions are modeled worse, plotting each control's ARR at the final checkpoint against how often it appears in the training data, across the nine actions (Figure 13b).

Are rarer actions harder to control?

Yes. ARR rises with an action's training frequency: common controls are recovered near-perfectly while the rarest air-roll and reverse inputs lag behind, so the model under-renders actions it rarely saw.

6.6 Going from single-player to multiplayer

How should a fixed budget be split between single- and multiplayer training? The world model must ultimately drive four interacting players (Section 4.5); Figure 21 shows a rollout from the multiplayer model, with the game-state probe's predicted ball and car positions overlaid. A multiplayer step tiles the four player views and ingests about 4× as many views as a single-player step (baseline configurations in Table 10), so at a fixed budget, matched to the single-player model's compute, multiplayer training runs proportionally fewer steps. We ask how best to spend that budget to reach a multiplayer model: on multiplayer alone, on single-player alone, or on a mix that pretrains on single-player and then trains into multiplayer.

We fix the budget at the single-player model's own (1.6M player-views, 100k single-player steps) and sweep the single-player share from 0% (multiplayer from scratch) to 100% (pure single-player); each intermediate split pretrains on single-player for that share of the budget, then continues on multiplayer with the remainder, warm-starting from the single-player checkpoint. The x-axis reports the single-player and multiplayer optimization-step counts at each split. The 100% point is the single-player model itself, a per-view reference evaluated with the single-player pipeline. All runs share the same codec, world model, and data, and we score one player's view at the 4 s horizon (Figure 14).

How should a fixed budget be split between single- and multiplayer training?

Pretrain on single-player, then continue on multiplayer. Training multiplayer from scratch collapses at this budget; pretraining on single-player first and then continuing on multiplayer recovers most of the gap, and a modest single-player share works best. Pure single-player is stronger on per-view quality here, but it never models the four-player interaction, so it serves only as a reference.

At a larger budget the collapse disappears: with twice the clips (3.2M player-views), multiplayer trains adequately even from scratch (gFID 9.9 at 4 s), and a mostly-multiplayer mix (75% multiplayer, 25% single-player) does better still (gFID 9.4; Figure 22 in Section E).

6.7 Scaling behavior

We study how the single-player world model scales along two axes: the amount of training data and the size of the model.

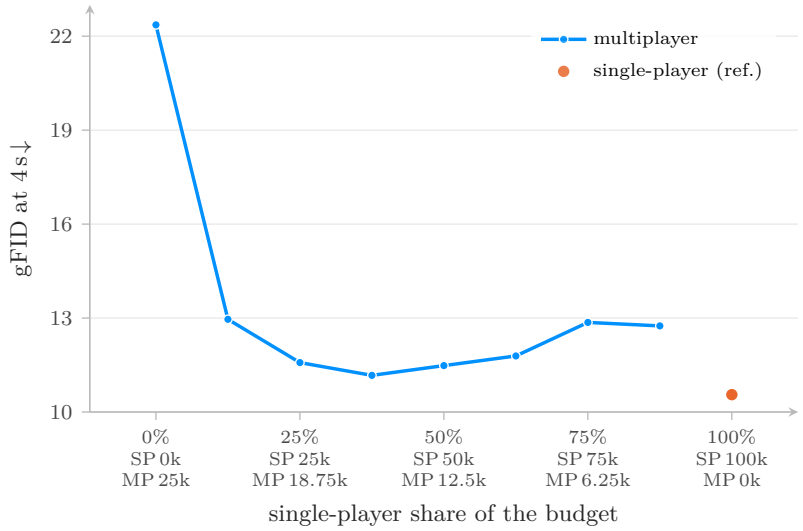


Figure 14 Allocating a fixed budget between single- and multiplayer training. gFID of one player's view at the 4 s horizon at the single-player model's budget (1.6M player-views). The single-player share sets how long we pretrain on single-player before continuing on multiplayer; each tick gives the resulting single-player and multiplayer optimization-step counts. Blue marks the multiplayer runs (0% is multiplayer from scratch); orange marks the pure single-player reference. Multiplayer from scratch collapses; pretraining on single-player first rescues it, and a modest single-player share is best, though at this budget pure single-player still edges ahead.

How much does more data help? We isolate the value of unique data at a fixed compute budget. Holding the model, batch size, optimizer, and total number of steps fixed at 100k updates, we train the 1B world model on nested random subsets of the single-player data. At this budget a run sees about 1.6M clips, so the unique data it covers is capped at roughly 2,000 hours; smaller subsets are therefore revisited more often, up to about 18× at 100 hours and down to a single pass for the full data. Compute is identical across points; only the amount of unique data, and with it the degree of repetition, changes. We report gFID and controllability (ARR) at the 4 s horizon (Figure 15).

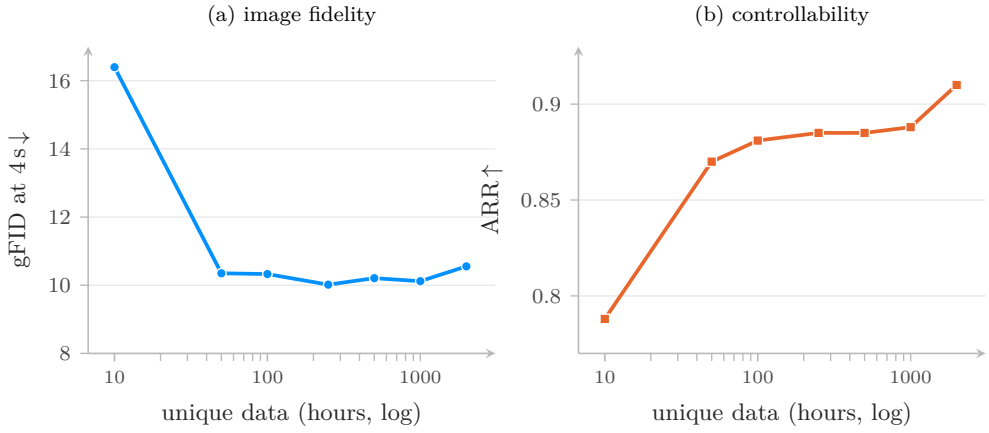
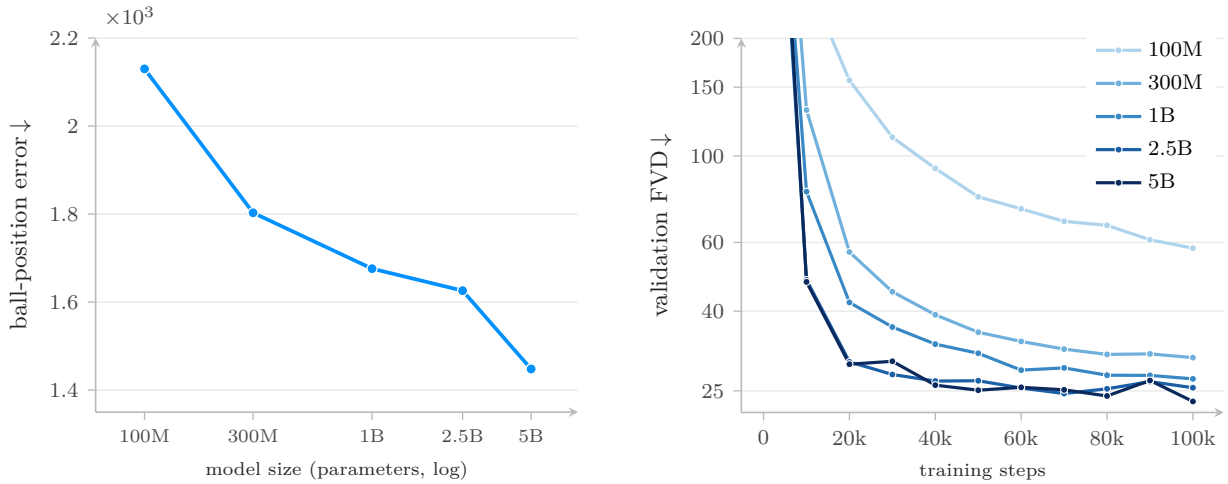


Figure 15 Data scaling at fixed compute. **(a)** gFID and **(b)** controllability ARR of the single-player model against the amount of unique training data seen, at a fixed 100k-step budget (smaller subsets are revisited more, so unique data and repetition vary together). Below about 50 hours both metrics collapse as distinct data runs short. Above it gFID saturates while ARR keeps climbing, so more unique data buys action fidelity that gFID no longer registers.

How much does more data help?

It depends on what is measured, and on whether the model has enough data to begin with. When unique data is very scarce, both per-frame fidelity (gFID) and controllability (ARR) collapse. Beyond the data-starved regime, more data leaves per-frame appearance unchanged at this budget while steadily improving how faithfully the model follows the commanded actions.

How does quality scale with model size? Following standard scaling practice, we grow the multiplayer world model across five sizes from 100M to 5B parameters (100M, 300M, 1B, 2.5B, 5B), holding the data and training recipe fixed so that any difference in quality is attributable to capacity alone; the architecture dimensions and training schedules for each size are given in Section A. Throughout training we track validation FID (image quality) and FVD (temporal coherence), and observe a clear and consistent scaling trend: larger models train to lower error and converge faster, and their ordering by size is preserved for essentially the entire run (Figure 16). The gains are steepest from 100M to 1B (the 100M model plateaus at a markedly higher error), after which returns diminish, with the 2.5B and 5B models converging to comparable quality at this fixed data budget. As a complementary, capacity-sensitive probe of physical understanding, we also measure how well a linear classifier recovers the game's physical state (ball and car positions and velocities) from the model's frozen features.



(a) Physical state (ball-position error)

(b) Temporal quality (FVD)

Figure 16 Larger models improve generation quality and encode physical state more faithfully. (a) Ball-position readout error of the linear game-state probe (L2, in Unreal units) against model size, and **(b)** validation FVD throughout training, for five multiplayer world models spanning 100M to 5B parameters (light to dark), with the data and training recipe held fixed. Lower is better on both axes; the FVD y-axis is logarithmic, cropped to the converged regime. Larger models converge faster and to lower error, their ordering by size holds for almost the entire run, and the probe error decreases monotonically with model size. Returns diminish beyond 2.5B, where the two largest models reach comparable quality.

The probe tells the same story from the side of physical understanding: the ball-position readout error falls monotonically with capacity, from 2130 at 100M to 1448 at 5B, with the largest drop between the 100M and 300M models (Figure 16). Larger models therefore not only look better but encode the game's physical state more faithfully in their features.

How does quality scale with model size?

Generation quality improves monotonically with model size: larger models reach lower FID and FVD and converge faster, with the steepest gains at smaller scales and diminishing returns for the largest models at this fixed data budget. The game-state probe corroborates this, with ball-position readout error decreasing as capacity grows.

6.8 Emergent properties

The model is trained only to predict future frames from past frames and actions, yet its rollouts show behavior it was never explicitly supervised on. We highlight four such properties.

The model emulates a theory of mind for players whose actions are withheld. During training we randomly drop some players' action streams, so the model must predict those players' behavior without actions to condition on. At rollout time it fills the gap itself: an uncommanded car keeps playing plausibly, moving and contesting the ball as a real player would, with no action supervision. By predicting how each unconditioned player would act, the model learns a policy for these agents on top of the environment's dynamics. As discussed in the data generation pipeline (Section 3.2), policies used to generate the action-annotated clips have access to the full game state, far more than the world model receives, yet the model recovers their complex decisions from pixels alone.

The model keeps the four views mutually consistent. The multiplayer model renders all four players' views jointly and keeps them agreeing on the shared world: a car that leaves one view stays present in the others, and when it re-enters that view it reappears where those views place it. The same coherence holds for salient game events: when the ball is driven into the net, the "SCORED!" callout and the goal explosion appear together across all four views (Figure 6), and a demolition is rendered consistently from every camera: the "DEMOLITION" callout for the attacker and the "BOOM!" blast filling the victim's own view, seen as a distant burst by the others (Figure 5). A single-player model sees only one view, so it has no cross-view signal to fall back on and loses or merges off-screen cars, a limitation we discuss in Section 6.9. This consistency is visible in the model's attention: a query token placed on an object in one view attends to the same object in the other three views (Figure 17).

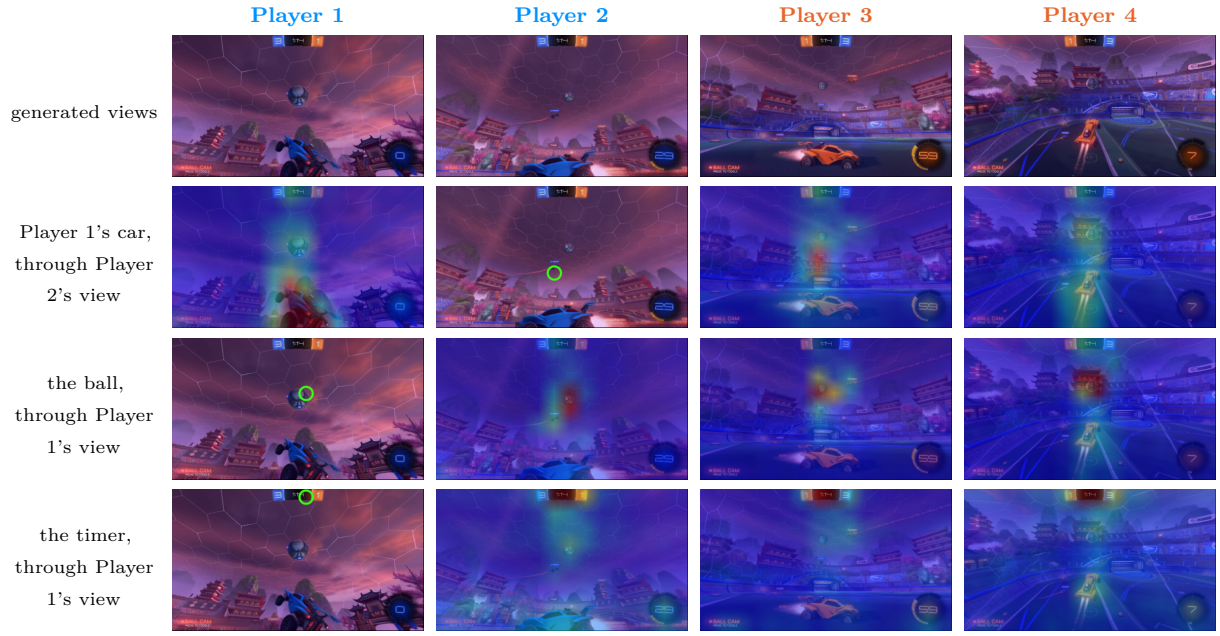


Figure 17 The multiplayer model attends across views to shared objects and state. Cross-view spatial attention on a rollout generated by the multiplayer world model. Each column is one player's generated view. After the generated views, each row queries the token named on the left and shows where its attention lands in the other views, warm colors marking strong attention and cool colors weak. Player 1's car, queried through Player 2's view, is picked out in the other views that see it and stays cooler where it is off-screen. The ball, queried through Player 1's view, activates in every view. The timer, queried through Player 1's view, attends to the shared top-center HUD band of every view. Cross-view attention is spatially structured; averaged over layers 4–7.

The model generalizes beyond the training action distribution. At rollout the model is routinely driven in ways whose statistics differ from training, and it stays stable under the shift. For instance, a scene in which every car sits still never appears in the data, since recorded play is always in motion; the model nonetheless holds such a scene stable, with cars that neither drift nor accelerate on their own. It is equally robust to who is at the controls: the automated policies that generate the training clips (Section 3.2) act fast and uniformly, concentrating near 390 actions per minute, while a person plays far more slowly and variably, from near-idle to bursts and spread broadly below 350 (Figure 18); in live play we observe that the model stays responsive and coherent under human control all the same. Driven well outside the training action distribution, it still obeys the underlying rule that a car moves only when acted on, evidence that it has captured the game's dynamics themselves and transfers across playing styles.

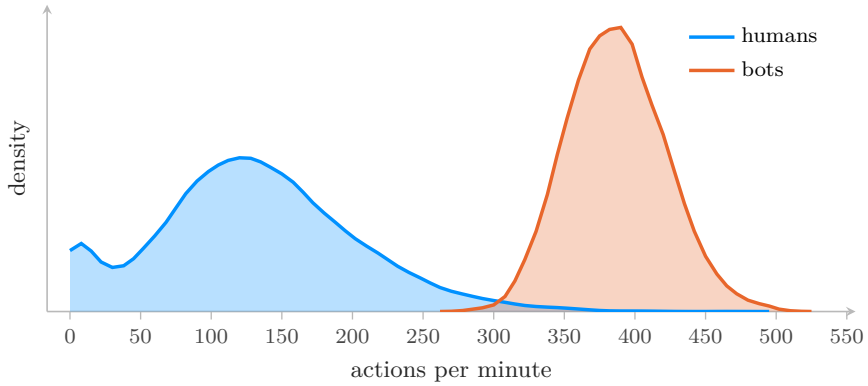


Figure 18 Humans play unlike the training policies. Distribution of actions per minute across clips for the nine controls (forward, backward, left, right, jump, boost, powerslide, air-roll left and right). Both curves come from recorded gameplay, and neither is produced by the world model: the automated policies behind the training data (*bots*) act fast and consistently, in a narrow band near 390 actions per minute, while *humans* act more slowly and far more variably, from near-idle to bursts and spread broadly below 350. The human curve comes from a separate corpus of publicly shared human gameplay clips on the Medal.tv platform (Medal.tv, 2024) (roughly 5,000 hours). The plot documents the distribution shift the model must handle; in our live demo it stays coherent under human control.

Desynchronized views recover on their own. The multiplayer model renders the four players' views jointly. On rare occasions one view gradually desynchronizes from the others and degrades into noise, typically when a player enters a regime the bots never produce, such as staying still inside the goal, which is far out of distribution. The model then pulls it back, within moments, to a view consistent with the other three and with the player's actions. We do not train for this recovery; it points to a joint representation that stays self-consistent across views (Figure 29).

6.9 Failure cases

The model is stable over long rollouts and follows the player's actions closely, so the failures we describe here are rare and localized. Most stem from two limits: a context too short to hold state across a match, and imbalances in the training data, where rare events are under-modeled and a few very common ones over-learned.

The ball, left untouched, tends to move on its own. While in play the ball behaves correctly, taking impulses from the cars and bouncing off the walls. When it is left untouched for a while, however, it tends to gain speed and roll toward a goal. A still ball is a rare event in the data, since play rarely stops, so the model's learned prior favors continued motion and does not keep a resting ball still.

The player's car sometimes acts uncommanded. Occasionally, it jumps or boosts without the corresponding key being pressed: over roughly forty minutes of human play (about 48,000 frames at 20 fps) we counted on the order of 80 uncommanded boosts and 30 jumps. The kickoff is the clearest example: the bots

open almost every match by boosting toward the ball in much the same way, so the kickoff phase carries little behavioral diversity and the model reproduces that boost even when the player holds back.

The clock and score lose track. For most of a match the clock and score are correct. They slip mainly at transitions, the score updating after a goal or the clock crossing a milestone, each of which happens once per game and is thinly represented. The errors stay internally consistent, though: when the model's clock reaches thirty seconds it still triggers the on-screen "thirty seconds remaining" prompt, in step with its own count even when that count is wrong (Figure 19).

Figure 19 The clock drifts from real time. A generated rollout (single view), sampled once per second, with the score-and-clock HUD boxed in red and magnified at the lower left of each frame. Over five seconds of real rollout the clock should count down five seconds, yet it barely moves, reading 4:54, 4:53, 4:53, 4:52, 4:54, 4:53: it advances far too slowly and even ticks back up, one instance of the clock losing track.

Goal replays diverge. After a goal the game cuts to a scripted replay. The replay never reproduces the goal that just happened, since the action sequence it shows is always a different one. Occasionally the replayed ball never reaches the net, and because a full replay outlasts the context, the rollout then stalls. In single-player the frames stay clean and only the content is wrong, whereas in multiplayer the replay also breaks visually.

Single-player rollouts mishandle the other cars. A single-player model is conditioned on one player's actions alone, so to render the other three cars it must model their behavior even while they are off-screen, a form of theory of mind, so that they re-enter the frame correctly. It slips in two distinct ways. A car that leaves the view is often gone when the camera returns to it, a context-length limit: once a car stays off-screen longer than the rolling window, the model has no trace of it left to bring back. Separately, cars sometimes merge into the ball, a modeling error that conflates the two (Figure 20). The multiplayer model, conditioned on all four cars, largely avoids the first failure, and we do not observe the car–ball merging in its rollouts.

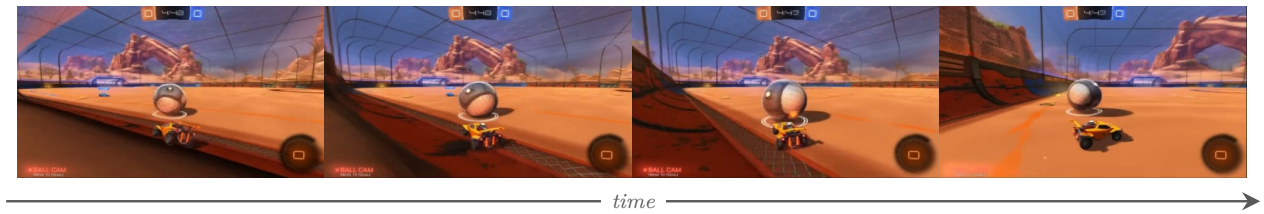


Figure 20 A car merges into the ball in a single-player world model. A rollout from a 1B single-player model, shown left to right. A non-playable car (blue) sits beside the ball and is drawn into it as the two overlap; once the ball moves on, the car does not re-emerge as a distinct object. We do not observe this failure in the multiplayer model.

7 Conclusion

We introduced MIRA, a real-time world model of 2v2 Rocket League for four simultaneous players. It is a diffusion model that predicts in the latent space of a video representation codec. Conditioned on every player's actions and past video context, it renders each player's own view, keeps the four views consistent, stays coherent over hour-long rollouts, and generates 20 frames per second on a single Nvidia B200 GPU.

Three design choices carried the result. Predicting in a latent space worked best when that latent was built on a frozen, pretrained feature extractor: it bought nothing in reconstruction, yet it was what stopped long rollouts from drifting and kept the four views consistent. For the objective, diffusion forcing with few-step distillation gave long-horizon stability and interactive speed where teacher forcing collapsed. And conditioning on all four action streams beat the single-player model, as the extra views resolved uncertainty about the shared state. Since visual quality hides all of this, we evaluated on action adherence, physical understanding, and rollout stationarity alongside image fidelity and human preference, and traced how each scales with data and model size.

Trained only to predict the next frame, the model develops behavior it was never supervised on: it keeps goals, demolitions, and the HUD agreeing across all four views, stays coherent under human control unlike the bots it learned from, and recovers on its own when a view is pushed out of distribution.

MIRA has clear limitations. We study a single game with data from one family of bot policies, so the model inherits that style of play. Its rare failures trace mostly to a context window too short to carry match state, which lets the clock and score slip and off-screen cars be forgotten, and to data imbalance, which leaves rare events such as a resting ball under-modeled. We leave to future work extending MIRA to more varied, human-like behavior, longer memory, and real environments.

We hope MIRA is a useful step toward world models faithful enough to train and evaluate interacting agents. To that end, we release our dataset, our full training and inference code, and a live demo at https://mira-wm.com.

Acknowledgments

We thank David Picard, Nicolas Dufour, Paula Wehmeyer, and Trevor Gravely for insightful discussions and support throughout the course of this project.

References

Josh Abramson, Arun Ahuja, Federico Carnevale, Petko Georgiev, Alex Goldin, Alden Hung, Jessica Landon, Jirka Lhotka, Timothy Lillicrap, Alistair Muldal, George Powell, Adam Santoro, Guy Scully, Sanjana Srivastava, Tamara von Glehn, Greg Wayne, Nathaniel Wong, Chen Yan, and Rui Zhu. Improving multimodal interactive agents with reinforcement learning from human feedback, 2022.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *International Conference on Learning Representations (ICLR) Workshop*, 2017.

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in Atari. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640366. https://doi.org/10.1145/3620665.3640366.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision transformer. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Mahmoud Assran et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Hritik Bansal, Zongyu Lin, Tianyi Xie, Zeshun Zong, Michal Yarom, Yonatan Bitton, Chenfanfu Jiang, Yizhou Sun, Kai-Wei Chang, and Aditya Grover. VideoPhy: Evaluating physical commonsense for video generation. *arXiv preprint arXiv:2406.03520*, 2024.

Amir Bar, Gaoyue Zhou, Danny Tran, Trevor Darrell, and Yann LeCun. Navigation world models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Omer Bar-Tal, Hila Chefer, Omer Tov, Charles Herrmann, Roni Paiss, Shiran Zada, Ariel Ephrat, Junhwa Hur, Guanghui Liu, Amit Raj, et al. Lumiere: A space-time diffusion model for video generation. In *SIGGRAPH Asia*, 2024.

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. *Transactions on Machine Learning Research (TMLR)*, 2024. arXiv:2404.08471.

Florent Bartoccioni, Elias Ramzi, Victor Besnier, et al. VaViM and VaVAM: Autonomous driving through video generative modeling. *arXiv preprint arXiv:2502.15672*, 2025.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *International Conference on Machine Learning (ICML)*, 2021.

Tianci Bi, Xiaoyi Zhang, Yan Lu, and Nanning Zheng. VFM-VAE: Vision foundation models can be good tokenizers for latent diffusion models. *arXiv preprint arXiv:2510.18457*, 2025.

Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.

Nicholas M. Boffi, Michael S. Albergo, and Eric Vanden-Eijnden. How to build a consistency model: Learning flow maps via self-distillation. *arXiv preprint arXiv:2505.18825*, 2025.

Rolv-Arild Braaten and Necto contributors. Necto/nexto: A rocket league bot trained with deep reinforcement learning. https://github.com/Rolv-Arild/Necto, 2022.

Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators, 2024. OpenAI technical report, https://openai.com/index/video-generation-models-as-world-simulators/.

Jake Bruce, Michael 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 *International Conference on Machine Learning (ICML)*, 2024.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. GameGen-X: Interactive open-world game video generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Hao Chen, Yujin Han, Fangyi Chen, Xiang Li, Yidong Wang, Jindong Wang, Ze Wang, Zicheng Liu, Difan Zou, and Bhiksha Raj. Masked autoencoders are effective tokenizers for diffusion models. In *International Conference on Machine Learning (ICML)*, 2025a.

Junyu Chen, Dongyun Zou, Wenkun He, Junsong Chen, Enze Xie, Song Han, and Han Cai. DC-AE 1.5: Accelerating diffusion model convergence with structured latent space. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025b.

Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, et al. WavLM: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, 2022.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. http://arxiv.org/abs/1410.0759.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *International Conference on Learning Representations (ICLR)*, 2024.

Decart and Etched. Oasis: A universe in a transformer, 2024. https://oasis-model.github.io/.

Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue. *arXiv preprint arXiv:2410.00037*, 2024.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning (ICML)*, 2023.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Haoyi Duan, Hong-Xing Guo, Xiaoshuai Zhao, Jiajun Wu, et al. WorldScore: A unified evaluation benchmark for world generation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021.

Ruili Feng, Han Zhang, Zhantao Yang, Jie Xiao, Zhilei Shu, Zhiheng Liu, Andy Zheng, Yukun Huang, Yu Liu, and Hongyang Zhang. The Matrix: Infinite-horizon world generation with real-time moving control. *arXiv preprint arXiv:2412.03568*, 2024.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *International Conference on Learning Representations (ICLR)*, 2025.

Shenyuan Gao, Jiazhi Yang, Li Chen, Kashyap Chitta, Yihang Qiu, Andreas Geiger, Jun Zhang, and Hongyang Li. Vista: A generalizable driving world model with high fidelity and versatile controllability. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yuan Gao, Chen Chen, Tianrong Chen, and Jiatao Gu. One layer is enough: Adapting pretrained visual encoders for image generation. *arXiv preprint arXiv:2512.07829*, 2025.

Gemini Robotics Team. Evaluating Gemini robotics policies in a Veo world simulator. 2025.

Google DeepMind. Genie 2: A large-scale foundation world model, 2024. https://deepmind.google/blog/genie-2-a-large-scale-foundation-world-model/.

Google DeepMind. Genie 3: A new frontier for world models, 2025. https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/.

Ming Gui, Johannes Schusterbauer, Timy Phan, Felix Krause, Josh Susskind, Miguel Angel Bautista, and Björn Ommer. Adapting self-supervised representations as a latent space for efficient generation. *arXiv preprint arXiv:2510.14630*, 2025.

Junliang Guo, Yang Ye, Tianyu He, Haoyu Wu, Yushu Jiang, Tim Pearce, and Jiang Bian. MineWorld: A real-time and open-source interactive world model on Minecraft. *arXiv preprint arXiv:2504.08388*, 2025.

Agrim Gupta, Lijun Yu, Kihyuk Sohn, Xiuye Gu, Meera Hahn, Li Fei-Fei, Irfan Essa, Lu Jiang, and José Lezama. Photorealistic video generation with diffusion models. In *European Conference on Computer Vision (ECCV)*, 2024.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. arXiv:1803.10122.

Yoav HaCohen et al. LTX-Video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2025.

Yoav HaCohen et al. LTX-2: Efficient joint audio-visual foundation model. *arXiv preprint arXiv:2601.03233*, 2026.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *Nature*, 2025a. arXiv:2301.04104.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025b.

Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025c.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *International Conference on Machine Learning (ICML)*, 2022.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

Philippe Hansen-Estruch, Jiahui Chen, Vivek Ramanujan, Orr Zohar, Yan Ping, Animesh Sinha, Markos Georgopoulos, Edgar Schoenfeld, Ji Hou, Felix Juefei-Xu, Sriram Vishwanath, and Ali Thabet. ViTok-v2: Scaling native resolution auto-encoders to 5 billion parameters. *arXiv preprint arXiv:2605.05331*, 2026.

Hao He, Yinghao Xu, Yuwei Guo, Gordon Wetzstein, Bo Dai, Hongsheng Li, and Ceyuan Yang. CameraCtrl: Enabling camera control for text-to-video generation. *arXiv preprint arXiv:2404.02101*, 2024.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Xianglong He, Chunli Zhang, Dongdong Wu, Yifan Zhang, Yiqun Xu, et al. Matrix-Game 2.0: An open-source, real-time, and streaming interactive world model. *arXiv preprint arXiv:2508.13009*, 2025.

Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P. Kingma, Ben Poole, Mohammad Norouzi, David J. Fleet, and Tim Salimans. Imagen Video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022a.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022b.

Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zak Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. GAIA-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023.

Teng Hu et al. MetaWorld: Scaling multi-agent video world model from single-view video data. *arXiv preprint arXiv:2606.02753*, 2026.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. VBench: Comprehensive benchmark suite for video generative models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Anssi Kanervisto, Dave Bignell, Linda Yilin Wen, Martin Grayson, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Tabish Rashid, Tim Pearce, Cristian Cantón Ferrer, et al. World and human action models towards gameplay ideation. *Nature*, 638:656–663, 2025.

Jihwan Kim, Junoh Kang, Jinyoung Choi, and Bohyung Han. FIFO-Diffusion: Generating infinite videos from text without training. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with GameGAN. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Jing Yu Koh, Honglak Lee, Yinfei Yang, Jason Baldridge, and Peter Anderson. Pathdreamer: A world model for indoor navigation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Dan Kondratyuk, Lijun Yu, Xiuye Gu, José Lezama, Jonathan Huang, Grant Schindler, Rachel Hornung, Vighnesh Birodkar, Jimmy Yan, Ming-Chang Chiu, et al. VideoPoet: A large language model for zero-shot video generation. In *International Conference on Machine Learning (ICML)*, 2024.

Weijie Kong et al. HunyuanVideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Theodoros Kouzelis, Ioannis Kakogeorgiou, Spyros Gidaris, and Nikos Komodakis. EQ-VAE: Equivariance regularized latent space for improved generative image modeling. In *International Conference on Machine Learning (ICML)*, 2025.

Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Yann LeCun. A path towards autonomous machine intelligence, 2022. Position paper, version 0.9.2, OpenReview.

Dacheng Li, Yunhao Zhang, Ji Lin, Enze Xie, et al. WorldModelBench: Judging video generation models as world models. *arXiv preprint arXiv:2502.20694*, 2025a.

Jiaqi Li, Junshu Zhang, Boyuan Jiang, Yuxuan Wang, Yujie Zhao, et al. Hunyuan-GameCraft: High-dynamic interactive game video generation with hybrid history condition. *arXiv preprint arXiv:2506.17201*, 2025b.

Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *International Conference on Learning Representations (ICLR)*, 2023.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. *arXiv preprint arXiv:2511.13720*, 2025.

Zhengqi Li, Qianqian Wang, Noah Snavely, and Angjoo Kanazawa. InfiniteNature-Zero: Learning perpetual view generation of natural scenes from single images. In *European Conference on Computer Vision (ECCV)*, 2022.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Andrew Liu, Richard Tucker, Varun Jampani, Ameesh Makadia, Noah Snavely, and Angjoo Kanazawa. Infinite Nature: Perpetual view generation of natural scenes from a single image. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Fangfu Liu, Kai He, Xuanchi Ren, et al. γ-World: Generative multi-agent world modeling beyond two players. *arXiv preprint arXiv:2605.28816*, 2026.

Kunhao Liu, Wenbo Li, Jiale Zhao, Ziwei Liu, Shijian Lu, et al. Rolling forcing: Autoregressive long video diffusion in real time. *arXiv preprint arXiv:2509.25161*, 2025.

Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023a.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations (ICLR)*, 2023b.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hongsheng Li. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

Nanye Ma, Mark Goldstein, Michael S. Albergo, Nicholas M. Boffi, Eric Vanden-Eijnden, and Saining Xie. SiT: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *European Conference on Computer Vision (ECCV)*, 2024.

Zehong Ma et al. PixelGen: Improving pixel diffusion with perceptual supervision. *arXiv preprint arXiv:2602.02493*, 2026.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. LeWorldModel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

Medal.tv. Medal: A gameplay clip capture and sharing platform. https://medal.tv, 2024.

Willi Menapace, Stéphane Lathuilière, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. Playable video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Fanqing Meng, Jiaqi Liao, Xinyu Tan, Wenqi Shao, Quanfeng Lu, Kaipeng Zhang, Yu Cheng, Dianqi Li, Yu Qiao, and Ping Luo. Towards world simulator: Crafting physical commonsense-based benchmark for video generation. In *International Conference on Machine Learning (ICML)*, 2025.

Meta AI. EUPE: Efficient universal perception encoder. *arXiv preprint arXiv:2603.22387*, 2026.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. In *International Conference on Learning Representations (ICLR)*, 2023.

Vincent Micheli, Eloi Alonso, and François Fleuret. Efficient world models with context-aware tokenization. In *International Conference on Machine Learning (ICML)*, 2024.

Saman Motamed, Laura Culp, Kevin Swersky, Priyank Jaini, and Robert Geirhos. Do generative video models understand physical principles? *arXiv preprint arXiv:2501.09038*, 2025.

Chris Mulder and BakkesMod contributors. BakkesMod: A rocket league modding framework. https://bakkesmod.com, 2016.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Zhiliang Peng, Li Dong, Hangbo Bao, Qixiang Ye, and Furu Wei. BEiT v2: Masked image modeling with vector-quantized visual tokenizers. *arXiv preprint arXiv:2208.06366*, 2022.

Ryan Po, Kai Zhang, Amir Hertz, Gordon Wetzstein, Neal Wadhwa, and Nataniel Ruiz. MultiGen: Level-design for editable multiplayer worlds in diffusion game engines. *arXiv preprint arXiv:2603.06679*, 2026.

Adam Polyak et al. Movie Gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024.

Alexander Pondaven, Haoyu Wu, Igor Gilitschenski, Philip Torr, Sergey Tulyakov, Fabio Pizzati, and Aliaksandr Siarohin. ActionParty: Multi-subject action binding in generative video games. *arXiv preprint arXiv:2604.02330*, 2026.

Zihan Qiu et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Julian Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. Worldgym: World model as an environment for policy evaluation, 2025.

Adrien Ramanana Rahary, Nicolas Dufour, Patrick Pérez, and David Picard. One View Is Enough! monocular training for in-the-wild novel view generation. *arXiv preprint arXiv:2603.23488*, 2026.

RLGym contributors. RLGym: A python api for reinforcement learning in rocket league. https://rlgym.org, 2021.

Jan Robine, Marc Höftmann, Tobias Uelwer, and Stefan Harmeling. Transformer-based world models are happy with 100k interactions. In *International Conference on Learning Representations (ICLR)*, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogeboom. Rolling diffusion models. In *International Conference on Machine Learning (ICML)*, 2024.

Lloyd Russell, Anthony Hu, Lorenzo Bertoni, George Fedoseev, Jamie Shotton, Elahe Arani, and Gianluca Corrado. GAIA-2: A controllable multi-view generative world model for autonomous driving. *arXiv preprint arXiv:2503.20523*, 2025.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Kyle Sargent, Zizhang Li, Tanmay Shah, Charles Herrmann, Hong-Xing Yu, Yunzhi Zhang, Eric Ryan Chan, Dmitry Lagun, Li Fei-Fei, Deqing Sun, and Jiajun Wu. ZeroNVS: Zero-shot 360-degree view synthesis from a single image. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Georgy Savva, Oscar Michel, Saining Xie, et al. Solaris: Building a multiplayer video world model in Minecraft. *arXiv preprint arXiv:2602.22208*, 2026.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 2020.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 2025.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. DINOv3. *arXiv preprint arXiv:2508.10104*, 2025.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-A-Video: Text-to-video generation without text-video data. In *International Conference on Learning Representations (ICLR)*, 2023.

Jaskirat Singh, Boyang Zheng, Zongze Wu, Richard Zhang, Eli Shechtman, and Saining Xie. Improved baselines with representation autoencoders. *arXiv preprint arXiv:2605.18324*, 2026.

Kiwhan Song, Boyuan Chen, Max Simchowitz, Yilun Du, Russ Tedrake, and Vincent Sitzmann. History-guided video diffusion. In *International Conference on Machine Learning (ICML)*, 2025.

Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models. In *International Conference on Learning Representations (ICLR)*, 2024.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning (ICML)*, 2023.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. https://doi.org/10.1145/3315508.3329973.

Shengbang Tong, Boyang Zheng, Ziteng Wang, Bingda Tang, Nanye Ma, Ellis Brown, Jihan Yang, Rob Fergus, Yann LeCun, and Saining Xie. Scaling text-to-image diffusion transformers with representation autoencoders. *arXiv preprint arXiv:2601.16208*, 2026.

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, et al. SigLIP 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*, 2025.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric and challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. In *International Conference on Learning Representations (ICLR)*, 2025.

Dilin Wang, Hyunyoung Jung, Tom Monnier, Kihyuk Sohn, Chuhang Zou, Xiaoyu Xiang, Yu-Ying Yeh, Di Liu, Zixuan Huang, Thu Nguyen-Phuoc, et al. Worldgen: From text to traversable and interactive 3d worlds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Xiaofeng Wang, Zheng Zhu, Guan Huang, Xinze Chen, Jiagang Zhu, and Jiwen Lu. DriveDreamer: Towards real-world-driven world models for autonomous driving. In *European Conference on Computer Vision (ECCV)*, 2024.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 2004.

Wayve. GAIA-3: Advancing world models from simulation to evaluation, 2025. https://wayve.ai/press/wayve-launches-gaia3/.

World Labs. Generating worlds, 2024. https://www.worldlabs.ai/blog/generating-worlds.

World Labs. RTFM: A real-time frame model, 2025. https://www.worldlabs.ai/blog/rtfm.

Haoyu Wu, Jiwen Yu, Yingtian Zou, and Xihui Liu. MultiWorld: Scalable multi-agent multi-view video world models. *arXiv preprint arXiv:2604.18564*, 2026.

Tongda Xu et al. Making reconstruction FID predictive of diffusion generation FID. *arXiv preprint arXiv:2603.05630*, 2026.

Jiawei Yang, Tianhong Li, Lijie Fan, Yonglong Tian, and Yue Wang. Latent denoising makes good tokenizers. *arXiv preprint arXiv:2507.15856*, 2025a.

Jiazhi Yang, Shenyuan Gao, Yihang Qiu, Li Chen, et al. Generalized predictive model for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. CogVideoX: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations (ICLR)*, 2025b.

Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Frédo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Jiwen Yu, Yiran Qin, Haoxuan Liu, Xiao Yu, Bowen Zhang, Xihui Wang, et al. Position: Interactive generative video as next-generation game engine. *arXiv preprint arXiv:2503.17359*, 2025.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Weipu Zhang, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang. STORM: Efficient stochastic transformer based world models for reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Xin Zhang, Dong Zhang, Shimin Li, Yaqian Zhou, and Xipeng Qiu. SpeechTokenizer: Unified speech tokenizer for speech language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Yifan Zhang, Chunli Wei, Dongdong Wu, Xianglong He, Yiqun Xu, Xinjie Lyu, Yongchao Zhang, Hao Liu, Yang Chen, et al. Matrix-Game: Interactive world foundation model. *arXiv preprint arXiv:2506.18701*, 2025.

Boyang Zheng, Nanye Ma, Shengbang Tong, and Saining Xie. Diffusion transformers with representation autoencoders. *arXiv preprint arXiv:2510.11690*, 2025.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Learning Representations (ICLR)*, 2025.

Shangwen Zhu, Yiran Peng, Ruili Feng, et al. Incantation: Natural language as the action interface for multi-entity video world models. *arXiv preprint arXiv:2605.18601*, 2026.

A Implementation Details

Codec. Table 9 gives the full training, architecture, and loss configuration of the baseline representation codec (Section 4.2) used in the codec ablations, the frozen temporally-downsampled DINOv3 RAE.

Table 9 Codec training configuration. Optimization, architecture, and loss hyperparameters for the frozen tdown RAE used as the baseline codec in the ablations (Section 6.3). The codec is trained to 250k steps; the ablations use its 125k-step checkpoint.

Hyperparameter	Value
Compute and schedule	
GPUs (DDP)	8 H100 (1 node)
Global batch size	32 clips (4 per device)
Training steps	250k (ablations use the 125k checkpoint)
`torch.compile`	enabled
Optimizer (AdamW)	
Learning rate	2×10^{-4}
(β_1, β_2)	(0.9, 0.95)
Weight decay	0
Learning-rate schedule	1k-step warmup, then decay to 1×10^{-6} over 249k steps
Weight EMA	none
Latent-statistics EMA	0.99
Encoder (feature extractor + bottleneck)	
Feature extractor	DINOv3-L, frozen
Aggregated blocks	$\{11, 13, 15, 17, 19, 21, 23\}$
Latent channels	32
Spatial / temporal downsampling	$\times 2$ / $\times 2$ (net /32 spatial, 10 Hz latent)
Input	288×512, 40 frames, 20 fps
Decoder (space-time ViT)	
Width / depth / heads	1152 / 28 / 16
MLP multiplier	4
QK normalization	LayerNorm
Patch size (space / time)	16 / 2
Loss (adaptive gradient-norm weighting)	
L_1 reconstruction	1.0
LPIPS perceptual	1.0 (on 25% of frames)
DINO latent consistency	1.0 (on 25% of frames)
Perceptual DINO backbone	DINOv3-B
Adversarial (GAN) loss	none
Data	
Players	4

World model. Table 10 gives the training and architecture configuration of the baseline single-player and multiplayer world models (Section 6.6). Both share the same latent world model, optimizer, and frozen codec (Table 9); they differ only in the number of players and the multiplayer conditioning.

Table 10 World model training configuration. Baseline single-player and multiplayer world models. Values that are shared span both columns; rows that differ are split. Both decode through the frozen codec of Table 9.

Hyperparameter	Single-player	Multiplayer
Compute and schedule		
GPUs (DDP)	16 H100 (8×2 nodes)	32 H100 (8×4 nodes)
Global batch	16 views	32 tiled (128 player-views)
Training steps	150k	
`torch.compile`	enabled	
Optimizer (AdamW)		
Learning rate	1×10^{-4}	
(β_1, β_2)	(0.9, 0.99)	
Weight decay	0.1	
Schedule	1k-step warmup, then constant	
Weight EMA decay	0.9999	
Architecture (latent world model)		
Hidden dimension	2048	
Layers	16	
Attention heads (query / KV)	16 / 4 (GQA)	
Temporal attention	every 4 layers	
Positional encoding	RoPE (space and time)	
Patch size	1	
Context length (training)	78 latent frames	
Attention gating / AdaLN	yes / yes	
Objective	flow matching, causal (diffusion forcing)	
Input	288×512, 80 frames, 20 fps	
Multiplayer conditioning		
Players	1	4
View tiling	—	4 views tiled, joint spatial attention
Per-player action dropout	—	yes (0.5 subset-drop)
Action dropout (per step)	0.1	

A.1 Live-demo model configuration

Table 11 lists the full set of hyperparameters used to train the 5B-parameter multiplayer world model served in the live demo (`https://mira-wm.com`). This is the largest configuration in the paper: a 4-player, action-conditioned flow-matching transformer (Section 4.4) trained with diffusion forcing (Section 4.3) in the latent space of the RAE codec (Section 4.2), on the $\approx$ 10,000 clean match-hours of the multiplayer corpus (Section B). The model runs in real time, emitting 20 frames per second on a single Nvidia B200 GPU (Section 5). The demo checkpoint is warm-started from a single-player model pretrained for 30,000 steps and then trained for a further 100,000 steps on multiplayer data; this is well short of the 2M-step schedule ceiling in the configuration.

Live-demo codec. The demo model decodes through a temporally-downsampled DINOv3 RAE (Section 4.2) that shares the architecture and loss of the baseline codec in Table 9: 288×512 input at 20 fps, temporal patch size 2 for a 10 Hz latent, and 32 latent channels. Relative to the ablation codec it is trained longer and on more compute: 400k steps on 4 nodes (32 B200, DDP) at a global batch of 32 clips, with `torch.compile` enabled and the same AdamW optimizer and warmup schedule as Table 9.

Table 11 Live-demo model hyperparameters. Full training configuration of the 5B multiplayer world model served at https://mira-wm.com.

Hyperparameter	Value
Transformer architecture	
Parameters	$\approx 5\,\mathrm{B}$
Hidden dimension	4096
Layers	16
Attention heads	32
Key/value heads (GQA)	8
Patch size	1 (one token per latent vector)
Positional encoding	factorized spatio-temporal RoPE
Attention	QK-norm, sigmoid output gate
Feed-forward	SwiGLU
Register tokens	spatial + temporal
Action conditioning	AdaLN (flow time + action embedding)
Multiplayer	
Players	4
Per-player embedding	yes
Action dropout	per-player, $p = 0.1$
Inputs	
Resolution	$288 \times 512 \times 3$
Frame rate	20 fps
Training clip length	80 frames (4 s)
Context window (inference)	20 latent frames
Codec	RAE (Section 4.2)
Latent mean / std	0.457 / 10.688
Actions (kbm)	`W A S D Q E SPACE LSHIFT LCTRL`
Optimization	
Optimizer	AdamW
Learning rate	1×10^{-4}
Betas	$(0.9, 0.99)$
Weight decay	0.1
Gradient clipping	1.0
LR schedule	1000 warmup steps, then constant
Minimum LR	1×10^{-6}
EMA decay (γ)	0.999
Training	
Single-player pretraining	30 000 steps
Multiplayer training	100 000 steps
Batch size (per device)	1
Train-set proportion	0.995
Training data	10,000 match-hours

B Data Details

This appendix gives the action representation used during collection (Section 3.1), the cross-player synchronization signals, the action-noise injection procedure and full noise-composition breakdown, the processed dataset scale, and the collection-time quality control.

Keybindings. The nine-key vocabulary maps to different in-game actions depending on whether the car is grounded or airborne (Table 12).

Table 12 Keybindings. In-game meaning of each key on the ground and in the air.

Key	In-game action (ground)	In-game action (air)
`W`	Throttle forward (accelerate)	Pitch down (nose down)
`S`	Throttle back (reverse)	Pitch up (nose up)
`A`	Steer left	Yaw left
`D`	Steer right	Yaw right
`SPACE`	Jump	Jump / flip / double-jump
`LSHIFT`	Boost	Boost
`LCTRL`	Handbrake (powerslide)	—
`Q`	—	Air-roll left
`E`	—	Air-roll right

Policy-to-keyboard mapping. Nexto's three-valued axis commands and binary buttons are translated to keypresses by thresholding each axis at ± 0.5 and conditioning on whether the car is grounded or airborne (Table 13).

Table 13 Policy-to-keyboard mapping. Translation from Nexto output to the nine-key vocabulary. Axes are thresholded at ± 0.5; buttons fire on any non-zero value.

Nexto output	Condition	On ground	In air
Throttle	$\geq +0.5$	`W`	—
	≤ -0.5	`S`	—
Steer	≤ -0.5	`A`	—
	$\geq +0.5$	`D`	—
Pitch	≤ -0.5	—	`W`
	$\geq +0.5$	—	`S`
Yaw	≤ -0.5	—	`A`
	$\geq +0.5$	—	`D`
Roll	≤ -0.5	—	`Q`
	$\geq +0.5$	—	`E`
Jump	$\neq 0$	`SPACE`	`SPACE`
Boost	$\neq 0$	`LSHIFT`	`LSHIFT`
Handbrake	$\neq 0$	`LCTRL`	`LCTRL`

Synchronization signals. Each VM records on its own local clock, so raw timestamps are not directly comparable across the four players. To make cross-player alignment possible later, we record the raw signals it needs: a shared match identifier, per-frame video timestamps on the VM's clock, and networked game events (kickoff countdown start, kickoff countdown end, goal scored, goal replay end) that fire at the same logical instant in every player's recording. The alignment procedure itself is described in Section B.1.

Action-noise injection. To widen the behavioral distribution beyond what a single deterministic policy produces, we optionally inject noise into a player's actions. Noise was off for the first 21 days of collection and on for the final 5 days. When it is on, each of the four players independently has a 50% chance of being made noisy; the rest play the bot policy unchanged. A noisy player alternates between policy-driven and noise spans within the match, with each round starting policy-driven so the car re-positions before noise resumes. The noise type is fixed per player per match and is either no-op (zero input) or uniformly random ($\{-1, 0, 1\}$ for axes, $\{0, 1\}$ for buttons). Across the full corpus, 83.48% of matches are fully clean (all four views play the unaltered policy) and 16.52% contain at least one noisy view. Every recording is tagged with its noise condition (mode and per-view) so the dataset can be filtered or reweighted. All models in this paper train on the clean, unaltered-policy recordings only ($\approx$ 10,000 match-hours); the full collected corpus is larger ($\approx$ 12,500 match-hours, Table 15) and additionally includes the noisy views, which we release in full but do not train on.

Noise-composition breakdown. Table 14 gives the full distribution of matches over per-view noise compositions. A match is described by how many of its four views are no-op (zero input) and how many are uniformly random; the remaining views play the unaltered policy.

Table 14 Noise composition. Match counts by per-view noise composition, out of 99,408 matches. "Perfect" means all four views play the unaltered policy. The 16,425 noisy matches (16.52%) split across the remaining rows.

Composition	Matches	%
Perfect	82,983	83.48
1 random, 1 noop	3,309	3.33
1 random	2,184	2.20
1 noop	2,150	2.16
2 random	1,672	1.68
1 random, 2 noop	1,672	1.68
2 noop	1,630	1.64
2 random, 1 noop	1,583	1.59
3 noop	583	0.59
3 random	573	0.58
2 random, 2 noop	402	0.40
3 random, 1 noop	272	0.27
1 random, 3 noop	256	0.26
4 noop	73	0.07
4 random	66	0.07

Scale. The figures below describe the processed 720p / 20 fps dataset (Section B.1), not the larger raw recordings; counts are taken after the collection-time quality gates. The corpus comprises 99,408 matches, or 397,632 per-view recordings (4 views per match), totalling roughly 12,454 match-hours of gameplay ($\approx$ 49,816 view-hours), with a median match length of about 7.5 minutes ($\approx$ 451 s). Of the match-hours, 9,998 are fully clean and 2,456 are noisy. The three maps are near-uniformly represented, at roughly 33,100 matches and 4,150 hours each (Table 15). The data was collected over about 26 days (2026-05-08 to 2026-06-02) across 120 concurrent VMs, with some downtime and restarts.

Quality control. Capture stalls, mid-match player dropouts, and network lag cause the entire match to be discarded rather than shipped as partial data. Long-running game instances are periodically restarted to prevent a slow drift in capture quality.

Table 15 Dataset scale, per map. Matches and gameplay hours after processing, split by noise condition. A match is *clean* when all four views play the unaltered policy and *noisy* otherwise.

	Matches			Hours		
Map	Clean	Noisy	Total	Clean	Noisy	Total
Champions Field	27,670	5,448	33,118	3,314.7	814.6	4,129.2
Forbidden Temple	27,658	5,481	33,139	3,364.5	821.5	4,186.0
Deadeye Canyon	27,655	5,496	33,151	3,318.9	820.3	4,139.2
Total	82,983	16,425	99,408	9,998.1	2,456.3	12,454.5

B.1 Data processing

Processing turns the raw per-player recordings into the frame-aligned chunks the model consumes. It aligns the four views onto a shared timeline, selects the usable gameplay window, tiles it into fixed-length chunks, and emits frame-aligned video, action, and physics files for each.

Cross-player alignment. Because each VM records on its own clock, raw timestamps are not comparable across players. We align on the end of the first kickoff countdown, a networked game event that fires at the same logical instant in all four recordings, which places every view on a shared master timeline used by all downstream steps. The anchor is tight in practice: all VMs run in the same network region, and the game's client-server networking keeps networked-event presentation synchronized across clients even though one VM acts as host.

Gameplay-window selection. We decode the per-match event log for the match anchors (kickoff start and end, goal scored, goal-replay end). We then drop the initial recorder warm-up window (about the first 8 s of each recording), which covers the asset-streaming artifacts at the start of a fresh game process (the opening video is slightly degraded while game assets are still streaming in), and keep the remainder end-to-end, including kickoff countdowns and goal replays.

Chunking. The kept gameplay interval is tiled into fixed-length 4-second chunks, identified by (`match_id`, `player_id`, `chunk_idx`). The same `chunk_idx` across the four players covers the same interval on the master timeline, so the four views of a chunk are frame-aligned.

Video transcoding. Per-player video is re-encoded into standardized 720p / 20 fps chunks, downsampled from the 720p / 30 fps source. We encode with H.264 and disable B-frames to avoid any non-causal temporal leakage, and we seek frame-accurately per chunk with a two-stage approach (coarse to the nearest keyframe, then fine to the target frame). No audio is kept.

Per-frame action labels. The keypress event stream defines a step function over the nine-key vocabulary (Section 3.1). At each video-frame instant on the master timeline we emit the set of keys held at that moment, reading the step function with a hold-last-seen rule and no interpolation between events. This produces one action row per video frame, written as one action file per chunk and aligned with the corresponding video chunk.

Output format. Each chunk ships as a triple of frame-aligned files: the video (`mp4`), the action labels (one row per video frame), and the physics state (one row per video frame, where the 120 Hz source telemetry is hold-last-seen sampled at each frame timestamp, the same step-function reconstruction used for actions). The three files share a (match, player, chunk_idx) key, so every video frame has exactly one action row and one physics row. Fixing `match_id` and `chunk_idx` and varying `player_id` recovers the four frame-aligned views of the same interval. Each chunk also carries per-(match, player) metadata: which chunks were kept, their timestamps on the master timeline, the bot identifier, the decoded match anchors, and the noise condition propagated from Section 3.1.

C Physics Game State Schema

This appendix gives the full schema of the physics game state logged during collection (Section 3.2): one record per physics tick (120 Hz), comprising match-level state (Table 16), ball state (Table 17), and one car state per car on the field (Table 18; four cars in 2v2).

Conventions. `Vec3` is (x, y, z) floats and `Quat` is (x, y, z, w) floats, both in the game's world frame; Rocket League uses a Z-up coordinate system. Positions are in Unreal units (uu, with 1 uu = 1 cm), velocities in uu/s, angular velocities in rad/s, and times in seconds. Boost is normalized to $[0, 1]$ in the game code (shown as 0–100 to players during gameplay). A "–" in the unit column denotes a dimensionless field.

Table 16 Match-level game state.

Field	Type	Unit	Description
`time_remaining`	float	s	seconds left in the match
`score_blue`	int	–	blue team score
`score_orange`	int	–	orange team score
`is_overtime`	bool	–	overtime active

Table 17 Ball state.

Field	Type	Unit	Description
`location`	Vec3	uu	world position
`velocity`	Vec3	uu/s	linear velocity
`rotation`	Quat	–	orientation (unit quaternion)
`angular_velocity`	Vec3	rad/s	spin

Demolitions. The `is_demolished` flag is never populated (it is always false) due to a broken game hook during data collection. A demolition is instead recoverable from `attacker_player_id`, which holds the demolishing player's id while the car is demoed and reverts to -1 shortly afterwards.

Physical value ranges. Table 19 lists engine caps and arena geometry for the main physical quantities, useful for normalization and sanity-checking by downstream users. These are engine limits and arena bounds for vanilla 2v2 with no mutators, not empirical per-field minima and maxima of our recordings. Values are drawn from the RLBot community wiki ("Useful Game Values", `wiki.rlbot.org`).

Authority and per-view differences. Every player's recording stores the entire game state from that client's point of view. Because clients apply client-side prediction to hide network latency, predicted values can differ slightly across the four views; the effect is minor as matches run on a LAN. The host machine holds the authoritative state and can be identified as the player with the lowest `player_id`. During goal replays the physics values are frozen at the last state before the replay began. When we downsample the 120 Hz telemetry to frame timestamps during processing (Section B.1), we use the same hold-last-seen rule applied to actions.

D Action Probe for Controllability

The action recoverability ratio (Section 6.2) relies on a learned action probe. We report its architecture and its accuracy on real held-out video so its calibration can be judged.

Table 18 Car state (one entry per car).

Field	Type	Unit	Description
`player_id`	int	–	player identifier
`team`	ubyte	–	0 = blue, 1 = orange
`is_local`	bool	–	this is the recording player's car
`location`	Vec3	uu	world position
`velocity`	Vec3	uu/s	linear velocity
`rotation`	Quat	–	orientation (unit quaternion)
`angular_velocity`	Vec3	rad/s	rotational velocity
`boost_amount`	float	–	current boost, normalized (0–1)
`is_on_ground`	bool	–	car is on the ground
`is_on_wall`	bool	–	car is driving on a wall
`is_supersonic`	bool	–	car is at supersonic speed
`is_demolished`	bool	–	always false (not populated; see below)
`attacker_player_id`	int	–	demolishing player while demoed, else -1
`has_jumped`	bool	–	first jump performed
`has_double_jumped`	bool	–	second jump used
`has_flip`	bool	–	flip/dodge still available
`is_dodging`	bool	–	currently in a flip/dodge
`jump_time`	float	s	time since first jump (up to 0.2 s while jump held)
`dodge_time`	float	s	seconds into current dodge
`max_dodge_time`	float	s	flip-window duration (static 1.25 s)

Table 19 Physical value ranges (vanilla 2v2, no mutators). Engine caps and arena geometry, not empirical ranges of the recordings.

Quantity	Range / cap	Unit	Notes
position x	$[-4096, +4096]$	uu	side walls
position y	$[-5120, +5120]$	uu	back walls; goal mouth extends to $\sim \pm 6000$
position z	$[0, 2044]$	uu	floor to ceiling
car speed	≤ 2300	uu/s	supersonic ≥ 2200; no-boost drive ≤ 1410
ball speed	≤ 6000	uu/s	
car ang. vel.	≤ 5.5	rad/s	per-axis
ball ang. vel.	≤ 6.0	rad/s	
`boost_amount`	$[0, 1]$	–	in-game 0–100; drains 33.3/s
rotation	unit quaternion	–	$\lvert q \rvert = 1$
`jump_time`	$[0, 0.2]$	s	accrues while jump held
`max_dodge_time`	1.25	s	constant (flip window)
ball radius	91.25	uu	rest center $z \approx 93.15$
gravity	650	uu/s^2	downward ($-z$)

Figure 21 Multiplayer rollout with game-state probe. A rollout from the multiplayer world model; the game-state probe (Section 6.2) reads ball and car positions and velocities from the model's pre-head activations.

Architecture and training. The probe is a frozen DINOv3-B (ViT-B/16) encoder with a small (≈4M-parameter) attention-pooling head. Each 8-frame (0.4 s at 20 fps) chunk is resized to 224 px and encoded frame by frame into a 14×14 grid of patch tokens; learnable queries cross-attend over all tokens, with temporal and spatial positional embeddings, and an MLP maps the pooled output to nine action logits. Training is multi-label: a control is positive if it is pressed in any frame of the chunk, and the head is optimized with per-action binary cross-entropy on the single-player training split. The DINOv3-B backbone stays frozen so the probe transfers across real, reconstructed, and generated frames.

Accuracy on real data. On 35,000 held-out windows the probe reaches 0.838 mAP and 0.896 mean AUROC (Table 20). Accuracy is high for common controls and lower for rare ones such as reverse, mirroring the action-frequency trend that ARR reveals in the world model.

Table 20 Action-probe accuracy on real held-out video (35,000 windows, nine controls). Positive rate is the fraction of windows in which the control is pressed.

Action	Positive rate	F1	AP	AUROC
Forward	82.6%	0.903	0.965	0.850
Powerslide	47.2%	0.850	0.941	0.940
Right	46.9%	0.758	0.856	0.861
Left	44.7%	0.711	0.829	0.843
Boost	34.5%	0.718	0.831	0.875
Jump	22.2%	0.800	0.894	0.956
Air-roll Left	20.9%	0.738	0.813	0.946
Reverse	20.4%	0.542	0.653	0.858
Air-roll Right	18.7%	0.684	0.756	0.934
Aggregate	—	0.745	0.838	0.896

E Multiplayer Data Allocation at a Larger Budget

The single-player–multiplayer allocation sweep of Figure 14 uses the single-player model's own budget, where multiplayer training is starved of steps. At twice that budget (3.2M player-views), multiplayer is no longer undertrained (Figure 22): it trains adequately even from scratch, and multiplayer-heavy mixes fare best. A 75%-multiplayer blend (25% single-player) matches the lowest gFID and beats both pure multiplayer and pure single-player.

F Codec latent-space ablations

These experiments detail the codec latent-space design choices deferred from Section 6.3. The ablation protocol, metric suite, and baseline codec are all described there: each codec is judged by a 1B-parameter world model trained on it, generation columns are reported at the 4 s horizon, and in every table the best value in each column is bold and the second best underlined.

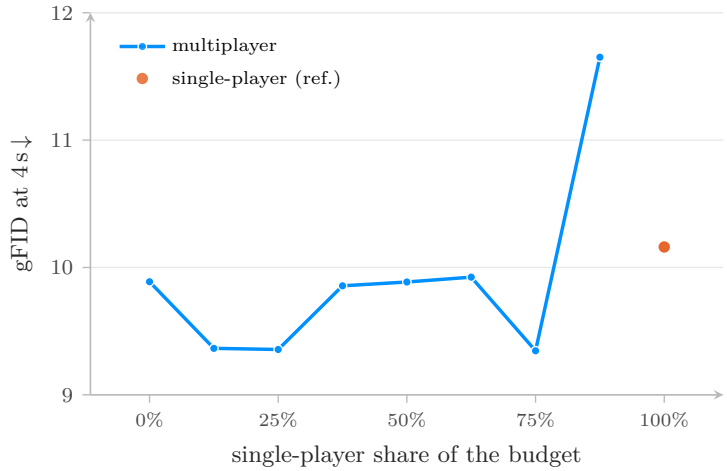


Figure 22 Multiplayer data allocation at the larger (3.2M-clip) budget. gFID of one player's view at the 4 s horizon across the single-player share. With enough multiplayer steps, multiplayer trains adequately even from scratch (0%), and multiplayer-heavy mixes reach the best gFID, beating both pure single-player (100%) and pure multiplayer (0%). Most mixes improve on pure single-player. Compare the single-player-budget sweep of Figure 14.

Latent-space ablations, in brief.

- **Any reasonable pretrained extractor works.** Smaller DINOv3 variants and EUPE-B trail DINOv3-L only slightly and all resist drift (Table 21).
- **Read several feature-extractor layers, not just the last.** Averaging a spread of the feature extractor's intermediate blocks keeps spatial detail that the deepest block discards (Table 22).
- **Compression can go in the codec or the world model.** The 2× reduction works in either, so we do all of it in the codec (Table 23).
- **Adaptive loss balancing helps.** Gradient-matched weights beat fixed weights on nearly all reconstruction and generation metrics (Table 24).
- **Upsample before decoding, not after.** Expanding the compact latent to the feature-extractor grid at the decoder's input, so the decoder runs on more tokens, far outperforms upsampling only at the output (Table 25).

Which pretrained feature extractor? We keep the feature extractor frozen and pretrained but swap DINOv3-L for the smaller DINOv3-B and DINOv3-S, and for EUPE-B, a compact feature extractor distilled from several vision foundation models into one (Meta AI, 2026) and distinct from the DINOv3 family. We aggregate each feature extractor's intermediate blocks the same way (block indices rescaled to its depth), with the bottleneck's input width set to its feature dimension. Because P-DINO and the Fréchet DINO Distances are computed with a DINOv3-B backbone, we gray out those three metrics for the DINOv3-B feature extractor, which shares the metric's backbone and therefore has an unfair advantage (Table 21).

Table 21 Pretrained feature extractor. Smaller DINOv3 variants and a different pretrained family (EUPE-B). P-DINO and the FDD distances use a DINOv3-B backbone, so the DINOv3-B row is biased on those (grayed).

	Codec (reconstruction)							World model (generation)		
Feature extractor	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
DINOv3-L (ours)	**29.7**	**0.891**	**0.051**	**0.021**	**5.4**	**38.6**	**0.17**	**10.7**	**163.1**	**0.55**
DINOv3-B	**29.7**	**0.891**	**0.051**	0.018	5.8	38.9	0.08	10.9	178.8	0.52
DINOv3-S	29.1	0.880	0.059	0.025	7.0	53.9	0.26	12.3	190.3	0.71
EUPE-B	29.0	0.876	0.058	0.023	6.2	52.2	0.18	11.4	180.4	0.56

Which pretrained feature extractor?

Any reasonable one. Smaller DINOv3 variants and EUPE-B trail DINOv3-L only slightly, which suggests richer representations may help downstream, and all of them stay equally low-drift over long rollouts (Figure 7): resistance to drift is a property of pretraining, shared across feature extractors.

How many feature-extractor layers should the latent use? The baseline averages features from seven DINOv3-L blocks ($\{11, 13, 15, 17, 19, 21, 23\}$) and adds the deepest as a residual, a choice adopted from RAEv2 (Singh et al., 2026) (Section 4.2). We compare against reading only the final block (Table 22).

Table 22 Layer aggregation. Aggregating seven DINOv3-L blocks (ours) against reading only the final block.

	Codec (reconstruction)							World model (generation)		
Features read	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
Multi-layer mean (ours)	**29.7**	**0.891**	**0.051**	**0.021**	**5.4**	**38.6**	**0.17**	**10.7**	**163.1**	**0.55**
Last block only	29.4	0.887	0.062	0.024	7.8	55.0	0.25	12.2	172.4	0.62

How many feature-extractor layers should the latent use?

Several. Aggregating a spread of DINOv3 blocks beats the final block alone on every metric, in line with RAEv2 (Singh et al., 2026): earlier blocks keep spatial detail the deepest layer discards.

Where should the spatio-temporal compression happen? Given that the latent is compressed, the 2× spatial and 2× temporal reduction can be done by the codec or by the world model, and we ask which is better. The baseline compresses in the codec, and the world model reads the latent with patch size 1. We instead keep the codec at the higher resolution and let the world model's patch embedding do the 2×, in time or in space and time, compressing the latent as it reads it (Table 23).

Table 23 Where the compression happens. Whether the 2× spatial and 2× temporal compression is done in the codec or in the world model's patch embedding.

Compression done in		Codec (reconstruction)							World model (generation)		
Spatial 2×	Temporal 2×	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
Codec	**Codec**	29.7	0.891	0.051	0.021	5.4	38.6	0.17	10.7	163.1	**0.55**
Codec	World model	30.6	0.909	0.039	0.016	4.3	23.0	0.13	**10.3**	**158.0**	0.56
World model	World model	**31.9**	**0.926**	**0.029**	**0.013**	**3.8**	**13.4**	**0.12**	11.8	214.3	0.63

Where should the spatio-temporal compression happen?

Either place works for the temporal reduction; the spatial reduction belongs in the codec. Moving the temporal 2× into the world model's patch embedding matches doing it in the codec on generation, so that choice is free, whereas moving the spatial 2× there too is worse on every generation metric. We therefore perform all of the compression in the codec.

Does adaptive loss balancing matter? The baseline weights each perceptual term by the adaptive gradient-norm balancing of the codec loss (Section 4.2). We compare against fixing every weight to 1, with no adaptive rescaling (Table 24).

Does adaptive loss balancing matter?

Yes. Fixed weights worsen nearly all reconstruction and generation metrics, so rebalancing the perceptual terms against reconstruction during training is worth its small cost.

Table 24 Loss balancing. Adaptive gradient-matched weights (ours) against fixed weights.

	Codec (reconstruction)							World model (generation)		
Balancing	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
Adaptive (ours)	**29.7**	**0.891**	**0.051**	**0.021**	**5.4**	38.6	**0.17**	**10.7**	163.1	**0.55**
Fixed weights	29.1	0.882	0.052	0.031	6.7	**37.7**	0.64	11.9	**160.3**	0.96

Where should the decoder upsample? The decoder first upsamples the latent spatially, from the /32 latent grid back to the feature extractor's /16 patch grid, with a strided transposed convolution. We test placing this upconv as the decoder's first layer, before the ViT blocks (the baseline, so attention runs at /16), or after them (so attention runs at the coarser /32 latent and upsampling comes last) (Table 25).

Table 25 Upsampling placement. The spatial upconv before the ViT blocks (ours) against after them.

	Codec (reconstruction)							World model (generation)		
Upconv	PSNR↑	SSIM↑	LPIPS↓	P-DINO↓	rFID↓	rFVD↓	rFDD↓	gFID↓	gFVD↓	gFDD↓
Before ViT (ours)	**29.7**	**0.891**	**0.051**	**0.021**	**5.4**	**38.6**	**0.17**	**10.7**	**163.1**	**0.55**
After ViT	27.6	0.842	0.105	0.038	11.6	125.2	0.53	15.7	232.2	0.89

Where should the decoder upsample?

At its input, not its output. Expanding the latent to the feature-extractor grid at the decoder's input, so the decoder runs on more tokens at the finer grid, far outperforms upsampling last, the worst codec in the study.

G Pixel-space rollouts

Without a codec, the pixel-space world model (Table 2) holds together for only a few frames before its rollout drifts, a failure the latent-space models do not share.

Figure 23 Pixel-space rollouts drift. A rollout from the plain pixel-space world model, shown left to right. The scene stays coherent for the first frames, then rapidly degrades into warped, unstructured texture, illustrating why pixel-space generation trails the latent baseline (Table 2).

H Context Noise Across Metrics

Figure 10 reports gFID against the past-noise standard deviation and rollout time for three representative codecs. Here we give the full set of six codecs (Figure 24) and the other two generation metrics, gFVD (Figure 25) and gFDD (Figure 26). All follow the same pattern: every codec is flat in noise except the distilled one, whose quality collapses over the rollout when the past is fed clean and is rescued by a small amount of noise.

I Human Evaluation

We complement the automatic metrics with the two human preference studies described in Section 6.2, run on a subset of the comparisons. Figure 27 reports the pairwise quality Elo and Figure 28 the action-adherence

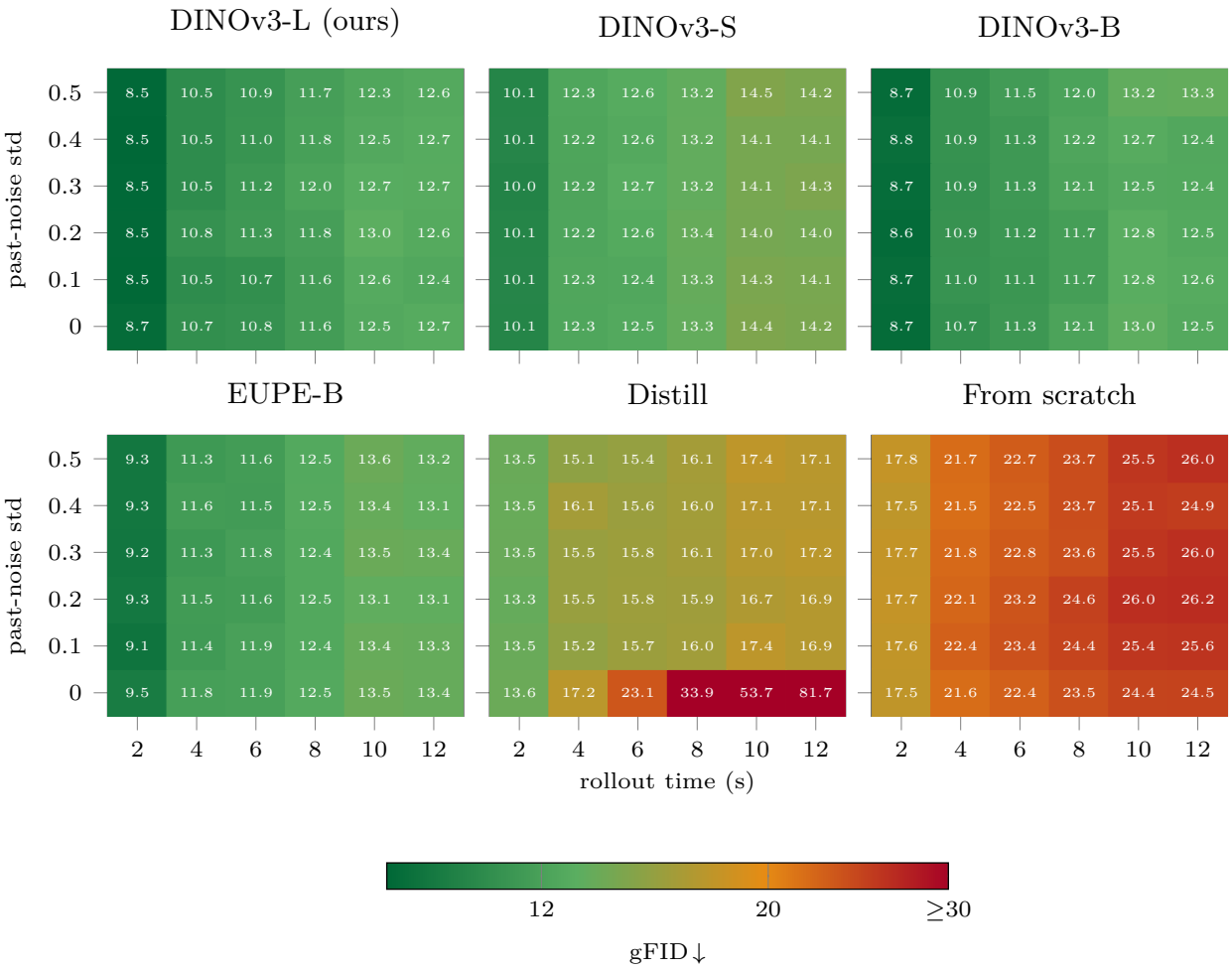


Figure 24 Context noise at inference, gFID (all codecs). gFID against the past-noise standard deviation (vertical) and rollout time (horizontal), for all six codecs; color is clipped at 30.

Elo delta. The action-adherence ranking correlates strongly with ARR, validating it as a controllability proxy (Section 6.5, Figure 12).

J Out-of-distribution recovery

Figure 29 shows a multiplayer rollout recovering on its own after being driven far out of distribution: the four views first diverge into noise, then resynchronize on a scripted goal replay and resume coherent play. We do not train for this behavior.

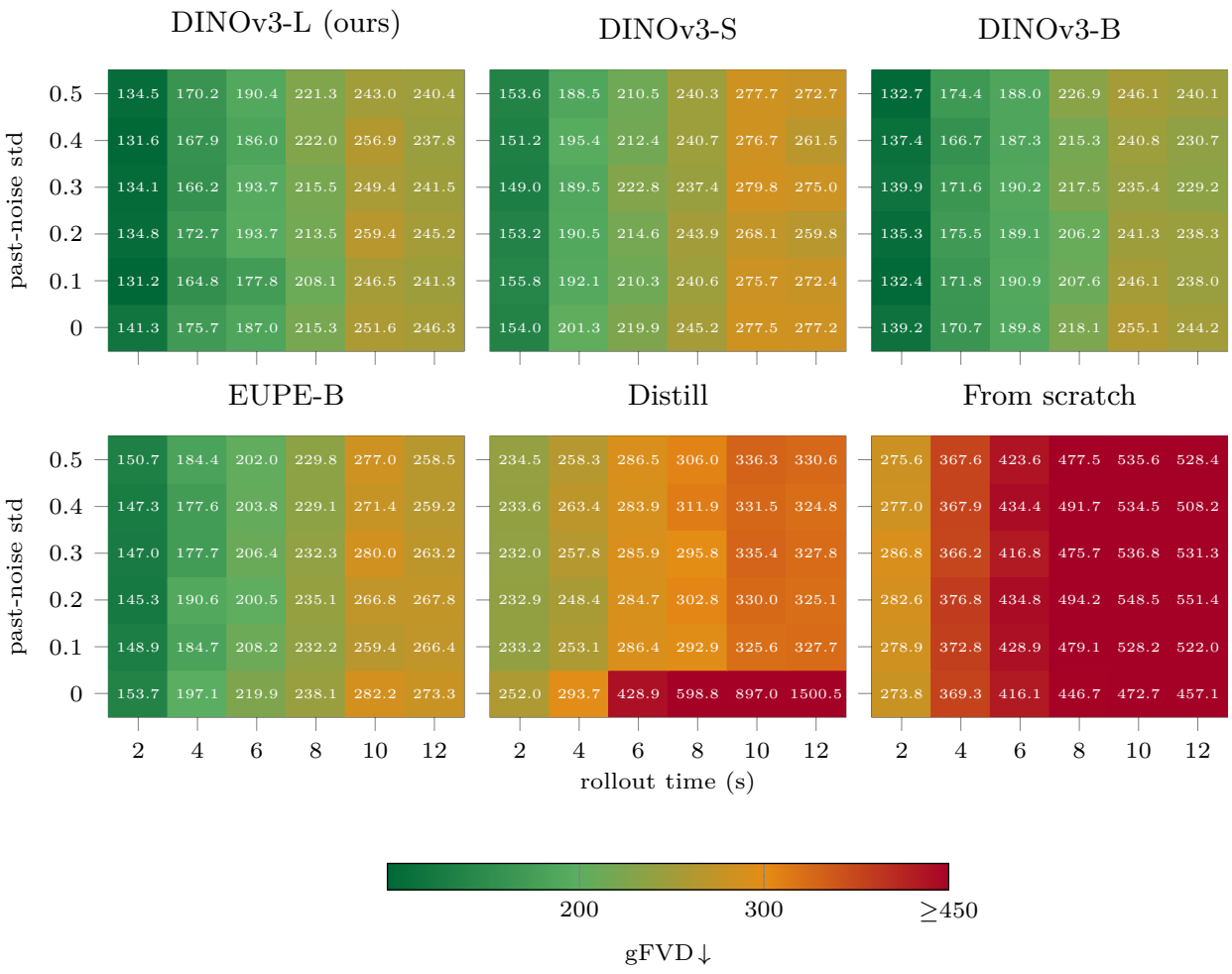


Figure 25 Context noise at inference, gFVD. gFVD against the past-noise standard deviation (vertical) and rollout time (horizontal), for all six codecs; color is clipped at 450.

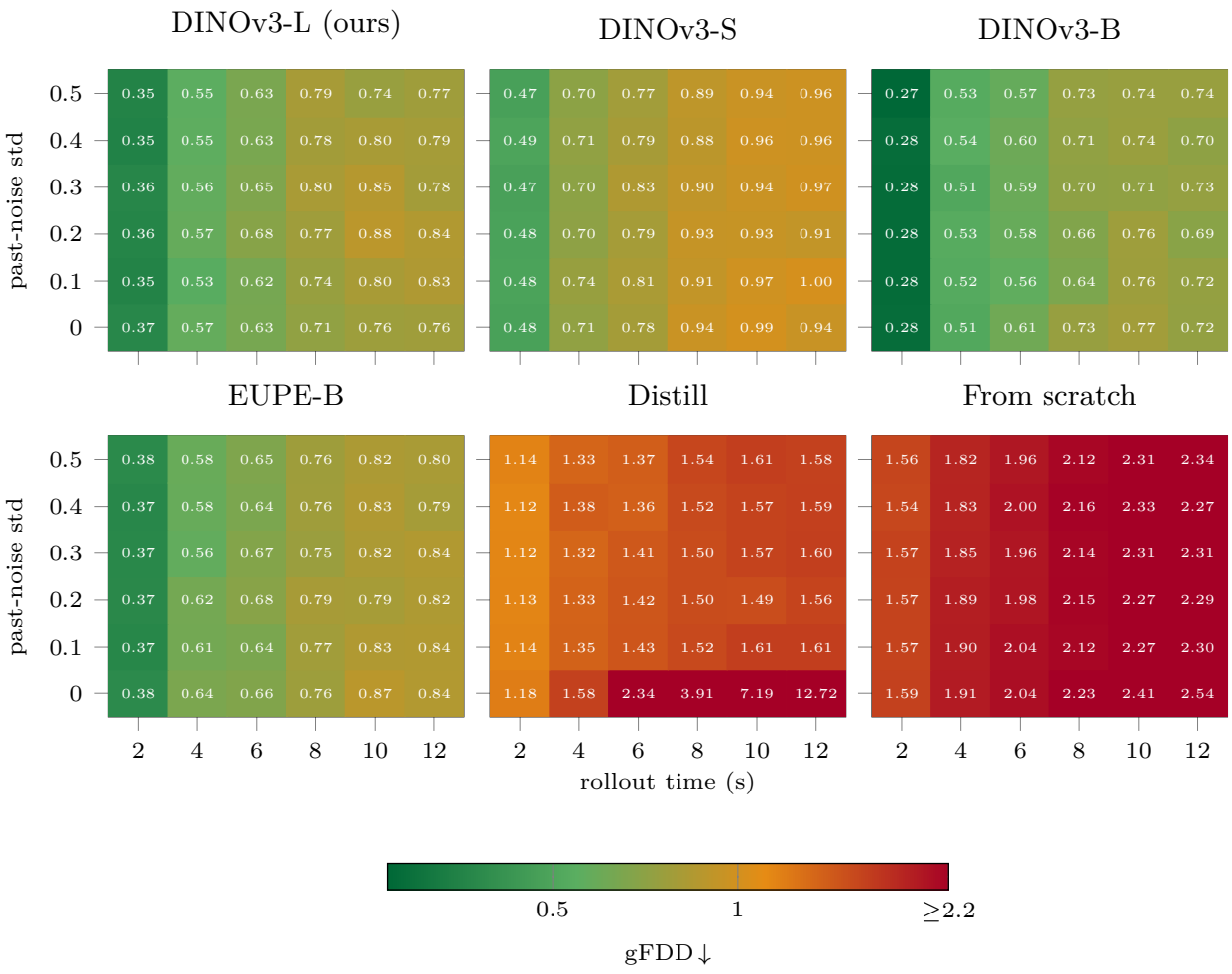


Figure 26 Context noise at inference, gFDD. gFDD against the past-noise standard deviation (vertical) and rollout time (horizontal), for each codec; color is clipped at 2.2.

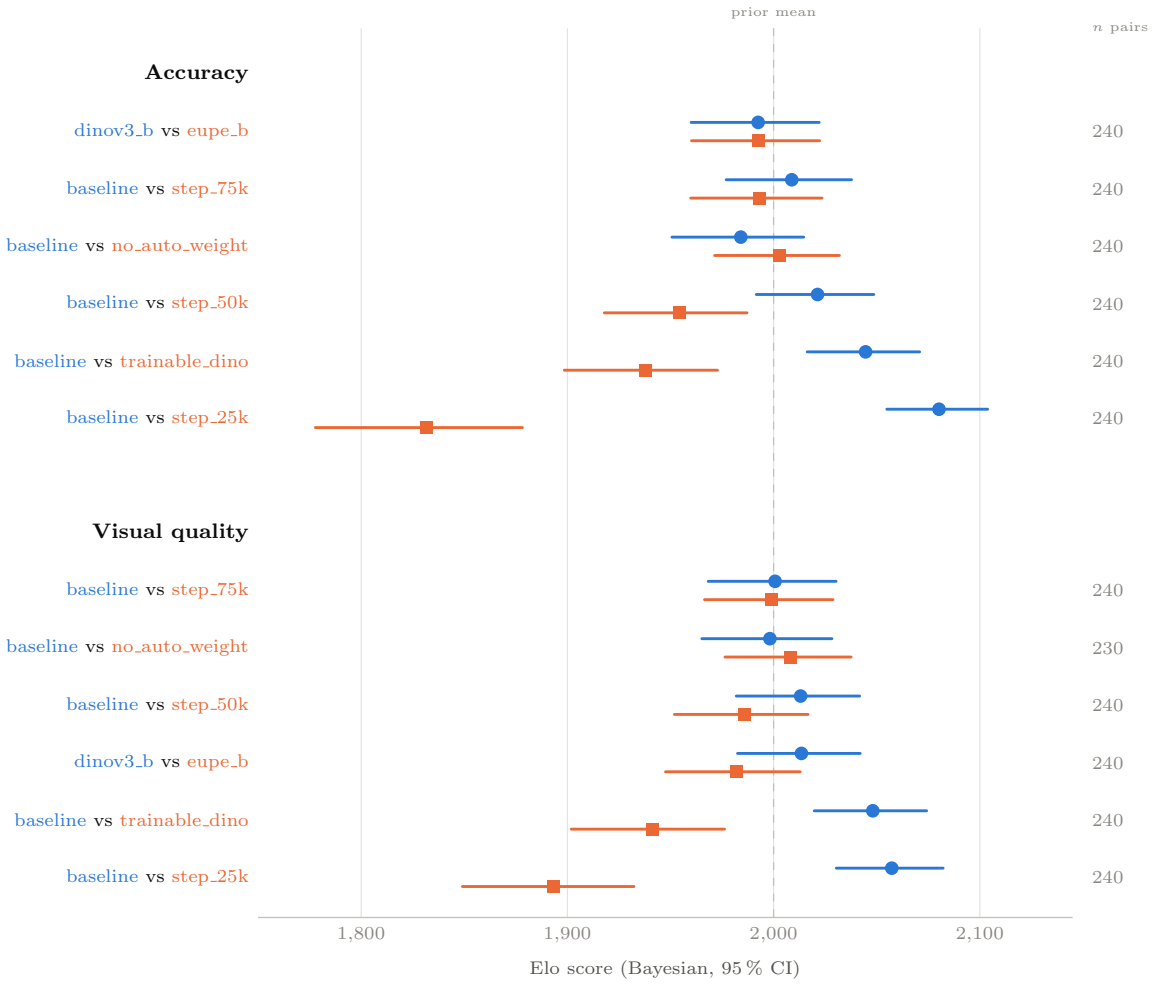


Figure 27 Human evaluation: quality. Pairwise Elo (Bayesian, 95% CI) from the quality study of Section 6.2, on accuracy and on visual quality, across a subset of comparisons.

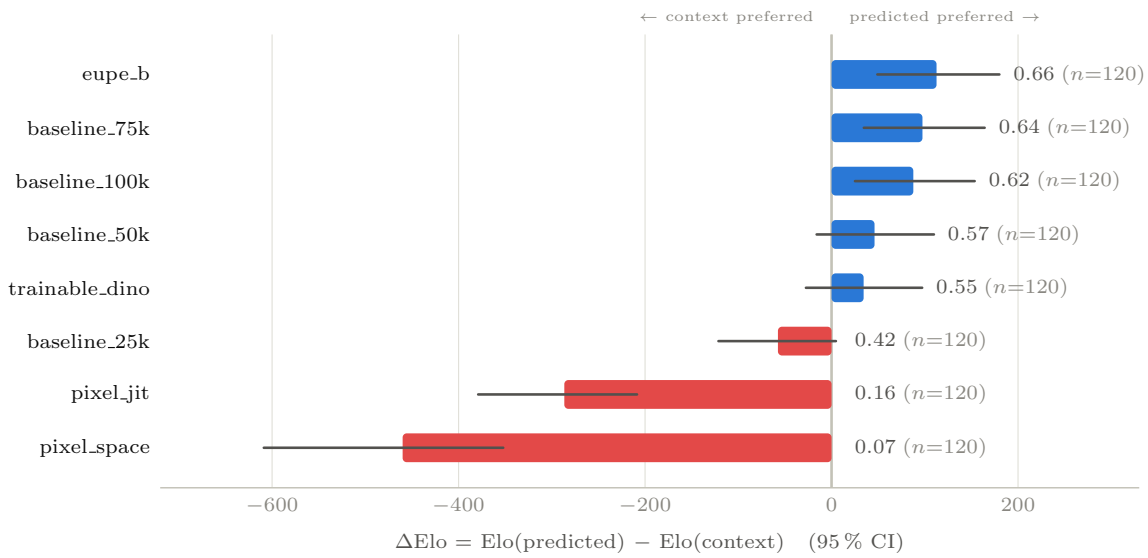
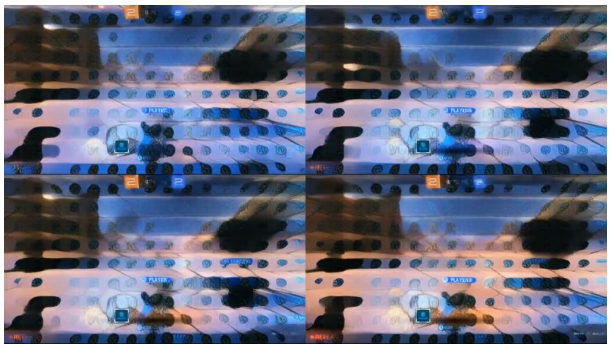


Figure 28 Human evaluation: action adherence. Elo delta between the action-conditioned prediction and the context from the adherence study of Section 6.2, across a subset of comparisons; higher means raters more often prefer the predicted video. This ranking correlates strongly with ARR (Section 6.5, Figure 12).

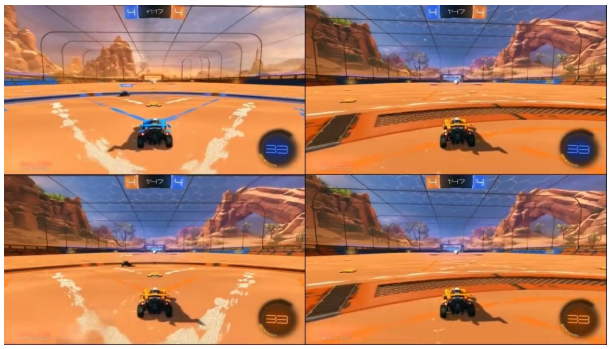
1. *purposely driven out of distribution*



2. *divergence grows*

3. *recovery begins*

4. *views resync on a goal replay*



5. *players respawn in each view*

6. *car color snaps back*

Figure 29 Recovering from an out-of-distribution excursion. A multiplayer rollout with the four tiled views, read left to right and top to bottom. We drive the model out of distribution on purpose **(1)** and its four views diverge into noise **(2)**; recovery then begins **(3)** as the model resynchronizes the views on a scripted goal replay **(4)**, respawns the players in their own views at the post-goal kickoff **(5)**, and snaps the last mis-colored car back to its correct color **(6)**. We do not train for this recovery.