
Matrix-Game 4.0: Engine-Grounded Neural Rendering for Real-Time Interactive Worlds

Jihai Zhang^{1,2*} Runjia Qian^{1,*} Kaiqin Kong^{1,*} Biao Jiang^{1,*}
Puyi Wang² Siru Chen^{3,1} Tianle Li² Guanjie Chen^{4,5} Junyi Wu⁴
Chubin Chen¹ Yuanhang Li¹ Kaichen Huang¹ Haobo Zhang¹
Wei Yu^{1†} Guibin Chen¹ Yu Cheng^{6,1} Yahui Zhou¹

¹SkyworkAI ²The Chinese University of Hong Kong
³University of Science and Technology of China ⁴Shanghai Jiao Tong University
⁵Shanghai AI Lab ⁶Nanyang Technological University

Project page: [Matrix-Game-4.0](#) Code: [GitHub](#) Checkpoints: [Hugging Face](#)

Abstract

Moving from controllable videos to genuinely playable worlds requires persistent world state, executable game rules, and low-latency interaction. We outline a roadmap for the fourth generation of Matrix-Game that separates game execution from visual generation through an executable harness, and introduce Matrix-Game 4.0 as the first system built on this design. A lightweight game engine maintains world state and executes interaction rules, while a multimodal coding agent translates player ideas into game code and scene descriptions. An autoregressive video model transforms engine-rendered low-poly proxies into detailed visuals, conditioned on dynamic semantic captions and spatial-memory references. This architecture provides continuous external constraints on visual generation and enables natural-language creation and modification of scenes and gameplay. Built on Wan2.2 5B, our renderer is trained on pixel-aligned low-poly and high-fidelity video pairs and distilled into a causal generator that produces four-frame chunks with one or two denoising steps. Together with targeted inference optimizations, Matrix-Game 4.0 enables 480p, 60 FPS gameplay on a single NVIDIA RTX 5090 with 112 ms of key-press-to-display latency. Single-step generation achieves 83 FPS steady-state throughput with approximately 18.2 GiB of peak GPU memory usage. We demonstrate customizable shooter and driving games with engine-enforced mechanics, and will release code, model weights, and low-poly game assets to support research and creation of playable generative worlds.

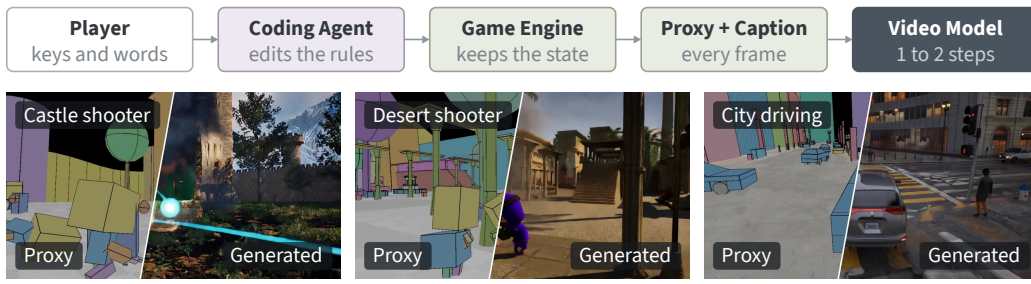
1 Introduction

Interactive world models such as the Genie series [1–3] move generative models from producing videos to watch toward producing worlds to take part in, and show that a model can build an interactive environment on the fly [4–7]. When the goal shifts from short explorations to games that run for a long time, offer rich gameplay, and follow controllable rules, systems that maintain the world mainly through video generation face four problems.

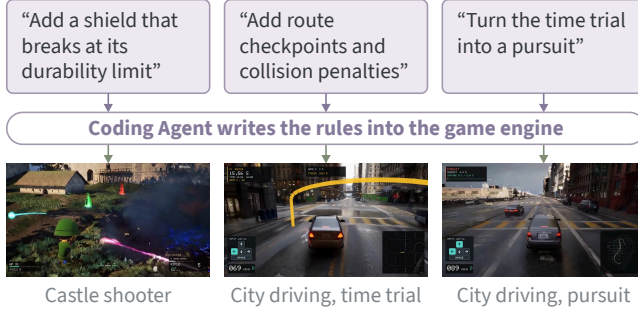
*Equal contribution.

†Project Lead and Corresponding Author.
Technical Report.

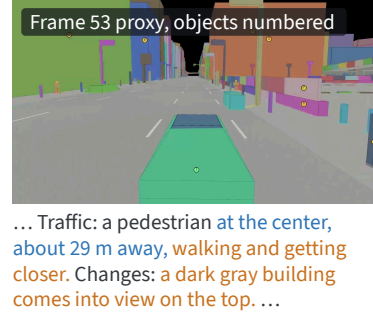
CODE KEEPS THE RULES, THE VIDEO MODEL DRAWS EVERY FRAME



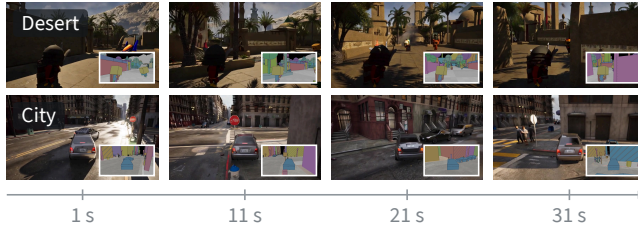
GAMEPLAY WRITTEN FROM ONE INSTRUCTION



DYNAMIC CAPTION, EVERY FRAME



40-SECOND ROLLOUTS · INSET: PROXY OF THE SAME FRAME



END-TO-END LATENCY (LOG SCALE)

Code World Model 1×H200	≈10.2 s
AgentGarten 1×H100	≈878 ms
StreamRender-H3 8×GB200	≈545 ms
Matrix-Game 4.0 1×RTX 5090	112 ms
83 FPS	57 FPS
1 step	2 steps
	18.2 GiB
	peak memory

Figure 1: **Matrix-Game 4.0**. Code keeps the game rules and state, and a video model draws every frame from the low-poly proxy and the dynamic caption that the engine produces. Top: the system loop. Second row: frames of held-out clips from three games, split diagonally into the engine proxy (left) and the frame generated from it (right). Middle left: gameplay that the coding agent built from a single instruction, rendered by the video model. Middle right: the proxy of one frame with its objects numbered, and an excerpt of the caption generated from the engine state; orange marks states and events, blue marks positions. Bottom left: two 40-second held-out rollouts at four time steps, with the proxy of the same frame inset. Bottom right: end-to-end latency of recent interactive systems (Table 3; for systems that generate in chunks, two chunk generation times), and the generation rate and peak memory of Matrix-Game 4.0 at 480p on one RTX 5090.

- **No continuous external correction.** Model rollouts proceed largely in isolation from external environments. World state exists implicitly in generated pixels and model context, without explicit state being continuously maintained for regions outside the field of view.
- **Accumulating errors.** Without external correction, small errors compound through autoregressive generation, progressively degrading consistency. Even with external 3D tools, errors in estimated camera poses can accumulate and eventually cause scene collapse.
- **Uncompressed frame-based memory.** Historical frames accumulate without their spatial information being consolidated into a compact representation. Memory costs therefore grow with rollout length, limiting long-horizon generation.

- **Limited gameplay mechanics.** Interaction centers on navigation and basic environmental responses, with limited support for executable rules, persistent action consequences, and explicit objectives that users can actively pursue.

More fundamentally, diffusion models primarily learn statistical correlations: visual realism does not necessarily imply causal or physical understanding. Following Roman Jakobson’s tripartite classification, the language-to-visual mapping in text- and image-to-video (TI2V) generation can be viewed as *intersemiotic translation*. We argue that current video generators are not yet *vision foundation models* for world modeling: much of their semantic understanding and apparent intelligence comes from upstream multimodal language models (MLLMs) or text encoders, while diffusion acts more like a renderer. Yet addressing these limitations alone is insufficient: genuinely playable worlds also require real-time generation with low interaction latency, meaningful game mechanics and rich interactions beyond navigation, and shared world state for multiplayer interaction. These considerations motivate an architecture where an external system maintains persistent world state and executes interaction rules, while the video model focuses on visual rendering.

We introduce **Matrix-Game 4.0**, a framework that places a video model within an external *harness* to separate game logic from visual generation. The system has three components (Figure 2). The *multimodal coding agent* translates player ideas into runnable game code and scene descriptions. The *game engine* maintains world state, executes interaction rules, and renders simplified geometry as a low-poly proxy. The *video generation model* transforms this proxy into detailed frames with materials, lighting, and atmosphere. At runtime, player actions update the engine state; the engine outputs proxy frames and structural, motion, and dynamic semantic conditions; and the video model generates the detailed frames displayed to the player. These continuous external constraints provide a basis for correction, so the game no longer relies on the video model’s memory to maintain state or execute rules. Our main contributions are as follows:

- **Real-time, playable video generation on a single consumer GPU.** Matrix-Game 4.0 enables 480p, 60 FPS gameplay on a single NVIDIA RTX 5090, with 112 ms of key-press-to-display latency. Single-step denoising achieves 83 FPS steady-state generation throughput with approximately 18.2 GiB of peak GPU memory usage, making video-model-driven games practical for local deployment and real-time interaction. (Table 3)
- **A roadmap toward playable worlds.** We outline Matrix-Game V4’s roadmap, separating game execution from visual generation through an executable harness. It combines a lightweight engine for world-state maintenance and rule execution, a multimodal agent for creating and modifying gameplay and scenes, and an autoregressive video model for real-time rendering, advancing from controllable videos to playable and extensible worlds.
- **A reusable pipeline spanning data, training, and runtime.** We unify aligned high- and low-fidelity rendering data collection, dynamic semantic annotation, controllable video training, few-step autoregressive distillation, and real-time inference.
- **An open-source foundation for generative games.** We plan to open-source Matrix-Game 4.0 at launch, providing a reusable technical foundation and adaptable game code. With the coding agent, researchers, developers, and creators can modify scenes, customize interactions, and design gameplay through natural language, lowering barriers to research and creation and turning ideas into playable games that can be continuously refined.

2 Roadmap

Matrix-Game V4 aims to build a truly playable world. Based on the analysis above, our approach is to wrap the video model in a harness that maintains world state and provides an **executable foundation for gameplay**. This architecture brings together three modules: a game engine, a multimodal coding agent, and a video generation model. The game engine serves as an external game manager, supplying spatial layout references, defining game rules, managing world state, and updating the semantic context. The multimodal coding agent bridges players’ ideas and game code, enabling flexible creation and modification of both gameplay and scenes. The video model then integrates the resulting context and control signals to autoregressively generate short sequences of realistic future frames. Figure 2 shows how the three parts work together.

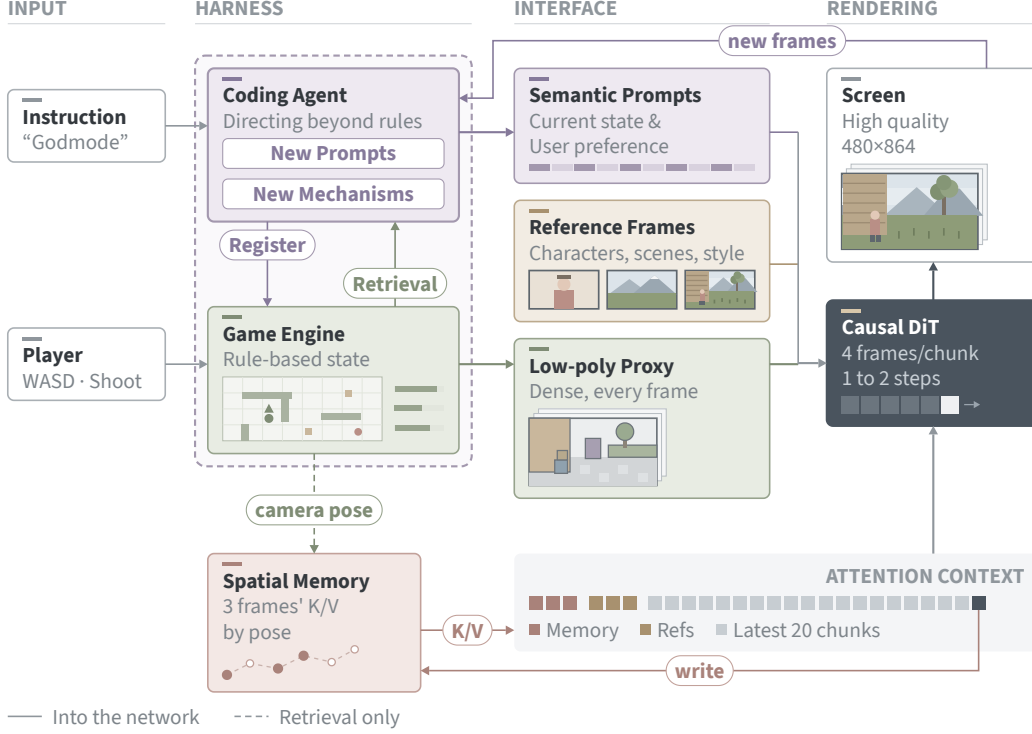


Figure 2: **Overview of Matrix-Game 4.0.** The coding agent and the game engine form the harness of the video model. The engine evolves the world state wherever rules apply and renders a dense low-poly proxy every frame. The coding agent turns user instructions into new prompts and new mechanisms that it registers into the engine, and retrieves the engine state to write semantic prompts that describe the current state and the user’s preference. The causal DiT renders one four-frame chunk at a time from the proxy, the prompt, and the reference frames, attending to a context of spatial-memory frames retrieved by camera pose, reference frames, and the latest 20 chunks. Solid lines enter the network; dashed lines are used for retrieval only.

2.1 Game Engine: The World’s “Referee”

The engine keeps track of what actually happens in the world and determines the consequences of player actions. Rather than producing fully detailed visuals, it represents roads, buildings, characters, and interactive objects through simplified geometry and scene structures. Game state and rules—including health, inventory, gravity, collisions, and quest progress—are explicitly maintained and enforced in code, rather than inferred from generated images.

As players act, the engine continuously updates the world and outputs structural renderings. These act as dynamic “scene sketches,” providing the video model with constraints on spatial layout, object motion, and camera viewpoint. Even when an object leaves the field of view, its state is still maintained by the engine according to the game rules.

2.2 Multimodal Coding Agent: The World’s “Scriptwriter” and “Translator”

The Multimodal Coding Agent provides the system’s intelligence and imagination, while also constructing its game rules. It communicates with the engine in both directions: translating established scenes and events into visual descriptions, and bringing players’ ideas or its own creative proposals into the game world.

Downstream: Translating the World. The agent converts engine-defined scene structures and interaction outcomes into text descriptions that guide video generation. With a reference image, a simple box can be described as “an old thatched cottage weathered by sea winds, with a dark red wooden front door.” When a player shoots an oil barrel, the engine determines whether the shot hits and triggers an explosion; the agent only describes the resulting visuals.

Upstream: Writing and Developing the World. The agent adds content and rules to the engine in two ways. When players have a specific idea, the agent turns it into a working game. From reshaping terrain and placing objects to modifying NPC behavior and designing new mechanics, it implements requests by editing scene configurations and game code. For example, “make monsters swarm in from all directions” requires more than adding monsters: it also means defining where they spawn, how they move and attack, and what determines victory or defeat. When players leave the details open, the agent improvises based on the existing scene and interaction context, filling in spatial layouts, storylines, or NPC dialogue. For example, a player might open a door with no predefined content behind it. The agent can imagine “a storage room with a locked iron chest in the corner” and submit it to the engine to be registered in the world. From that point on, the engine maintains the chest’s state and determines whether it can be opened.

2.3 Video Generation Model: The World’s “Painter”

The video model transforms structural renderings into rich, detailed visuals. While preserving scene structure and motion constraints, it generates materials, lighting, atmosphere, and fine visual details, reducing the need to create high-fidelity 3D assets individually.

Conditioned on current control inputs and past frames, the model continuously generates video through causal autoregression. Few-step distillation and inference optimizations tailored to real-time re-rendering reduce the delay between player actions and visual feedback.

The real-time loop can be summarized as:

Player input → **Engine updates world state** → **Structural renderings and control signals** → **Real-time video rendering** → **Visual feedback to the player.**

From Controllable Videos to Executable Worlds. The key is to separate how the game runs from how it looks: code defines the rules, the engine maintains state, and the video model generates the visuals. The engine only needs to run a lightweight simulation, leaving video generation as the main real-time performance challenge—one that can be mitigated through few-step distillation and causal autoregressive inference. Multiple players can share the same world state, each seeing it from their own perspective. As AI coding agents become more capable, the barrier to building lightweight games continues to fall, making this approach increasingly practical. Through this division of responsibilities, Matrix-Game V4 aims to help more people turn their ideas into games that can be played, modified, and continually refined—moving game world models from “**controllable videos**” to “**executable worlds**.”

3 Data

3.1 Paired Low-Poly and High-Fidelity Capture

We build a cross-domain dataset of aligned low-poly and high-fidelity videos in Unreal Engine 5 (UE5) [8] for world modeling and video generation. It covers four representative scene types: the CitySample megacity [9], an open desert, a complex castle, and high-dynamic air combat. Table 1 lists the training data of each scene. Every capture is recorded at 1280×720 and contains three aligned videos (the UE5 RGB render, the low-poly structure render, and a depth render of the proxy geometry) plus a per-frame metadata record. Shooter and air-combat clips last about one minute. For the castle, desert, and city scenes we hold out three complete capture sessions, and for air combat we hold out whole capture batches; no held-out session contributes any training clip.

The capture pipeline abstracts each high-complexity scene into a lightweight geometric representation. Camera trajectories and timestamps are strictly synchronized between the two renders, so the low-poly structure frame and the high-fidelity frame of the same index are aligned pixel by pixel (Figure 3a). The structure representation keeps the spatial topology, the rigid poses, and the motion trajectories of the scene. For buildings, complex terrain, vegetation, and dynamic interactive objects, adaptive geometric fitting and view-frustum culling remove material reflections, high-frequency textures, and complex lighting, so the structure prior describes only the essential spatial geometry. In the resulting proxy the sky is black and the ground is a gray checkerboard; buildings and walls become flat-colored boxes and prisms, trees become a cylinder with a sphere, characters become figures made

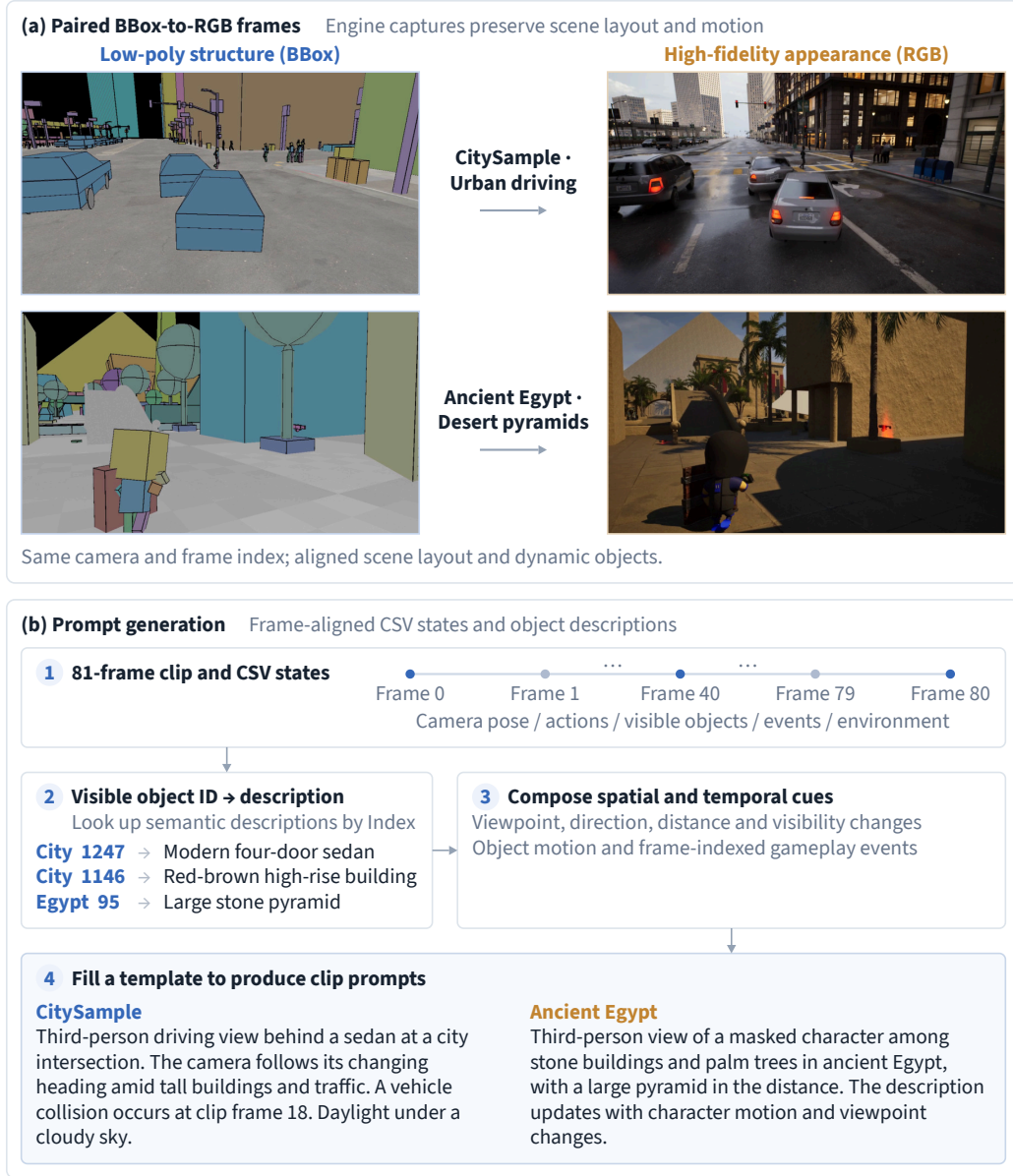
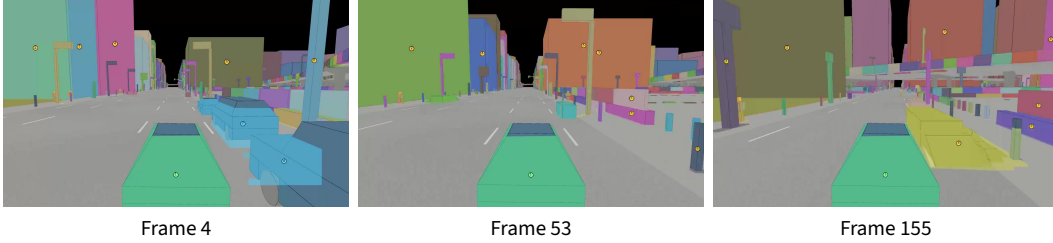


Figure 3: UE5 paired training data. (a) Each clip is captured twice in Unreal Engine 5 under the same camera and frame index: once as a low-poly structure render (BBox) that preserves scene layout and motion, and once as the high-fidelity RGB frame. (b) For every 81-frame clip, frame-aligned CSV records of camera pose, actions, visible objects, events, and environment are turned into a prompt: visible object IDs are looked up in a description table, spatial and temporal cues are composed, and a template produces the clip prompt.

of blocks, and every primitive is drawn with a black outline and no texture, reflection, or shadow (left part of each split frame in Figure 1). In parallel, the system records object instance IDs, visibility states, and causal interaction events, forming a semantic metadata stream for every clip. The data defines a conditional mapping from lightweight geometric topology to high-fidelity visuals, and gives deterministic supervision for spatiotemporal consistency and interaction stability in long rollouts.

Table 1: **UE5 paired training data.** Each clip is a pair of pixel-aligned low-poly and high-fidelity videos at 1280×720 . Held-out clips come from capture sessions (for air combat, whole multi-machine capture batches) that contribute no training clip.

Scene	Game	Train clips	Hours	FPS	Held-out
Medieval castle	Third-person shooter	17,848	290	24	3 sessions, 683 clips
Ancient Egypt desert	Third-person shooter	40,296	643	24	3 sessions, 617 clips
Desert canyon air combat	Flight combat	28,720	467	24	whole capture batches
CitySample megacity	Driving	27,470	312	17	3 sessions
Total		114,334	1,712		



Frame 4. Third-person driving view behind the ego car on a city street. The ego car, a modern four-door sedan, is driving forward. Traffic: a passenger minivan on the bottom-right, about 4 m away, parked; a passenger minivan on the right, about 10 m away, parked; a modern four-door sedan at the center, about 17 m away, parked; a dark-skinned woman pedestrian on the left, about 32 m away, standing still; a dark-skinned woman pedestrian on the left, about 35 m away, standing still; a dark-skinned woman pedestrian at the center, about 54 m away, standing still. Around: a dark gray high-rise building on the top-left, about 60 m away; a reddish-brown high-rise building on the top-left, about 88 m away; a silver-gray high-rise building on the top-left, about 96 m away; a gray building on the top, about 129 m away; a gray street lamp on the top-right, about 7 m away.

Frame 53. Third-person driving view behind the ego car on a city street. The ego car, a modern four-door sedan, is driving forward. Traffic: a dark-skinned woman pedestrian at the center, about 29 m away, walking and getting closer; a dark-skinned woman pedestrian on the left, about 30 m away, walking and getting closer. Changes: a dark gray building comes into view on the top. Around: a silver-gray high-rise building on the top-left, about 79 m away; a gray building on the top-right, about 101 m away; a dark gray building on the top, about 97 m away; a gray street lamp on the top, about 15 m away; an off-white barrier on the right, about 15 m away.

Frame 155. Third-person driving view behind the ego car on a city street. The ego car, a modern four-door sedan, is driving forward. Traffic: a dark-skinned woman pedestrian on the left, about 15 m away, standing still and getting closer; a dark-skinned woman pedestrian on the left, about 16 m away, standing still and getting closer; a dark-skinned woman pedestrian at the center, about 41 m away, walking and getting closer; a four-door pickup truck at the center, about 194 m away, driving. Changes: a gray trash bag comes into view on the bottom-right; a street lamp comes into view on the top-left. Around: a gray building on the top-left, about 36 m away; a silver-gray high-rise building on the top-right, about 180 m away; a gray trash bag on the bottom-right, about 9 m away; a street lamp on the top-left, about 14 m away; an off-white barrier on the right, about 11 m away.

Figure 4: **Dynamic captions follow the engine state.** Top: low-poly proxy frames of one CitySample driving clip; numbered markers are the visible objects that the engine reports for this frame, each linked by its instance ID to a semantic description. Below: the caption that the rule-based template produces for the same frame, without any vision-language model. Orange text comes from object states and events, and blue text from object positions and distances. As the car moves, parked vehicles leave the description, pedestrians change from standing to approaching, and newly visible objects are announced under “Changes”.

3.2 Dynamic Captions

In an interactive game the picture changes with every player action. Turning the camera changes which objects are visible, and moving or triggering events changes the state of objects. A fixed text description cannot follow these changes, so Matrix-Game 4.0 uses dynamic captions: the description given to the video model is updated with the current state of the game world.

Each object in the scene has an ID that is linked to a semantic description. The system reads the currently visible objects and their real-time states from the game engine and fills them into a prompt template, producing a description that matches the current frame (Figure 3b). When the player changes the view, enters a new area, or triggers an event, the description is reorganized, so the video

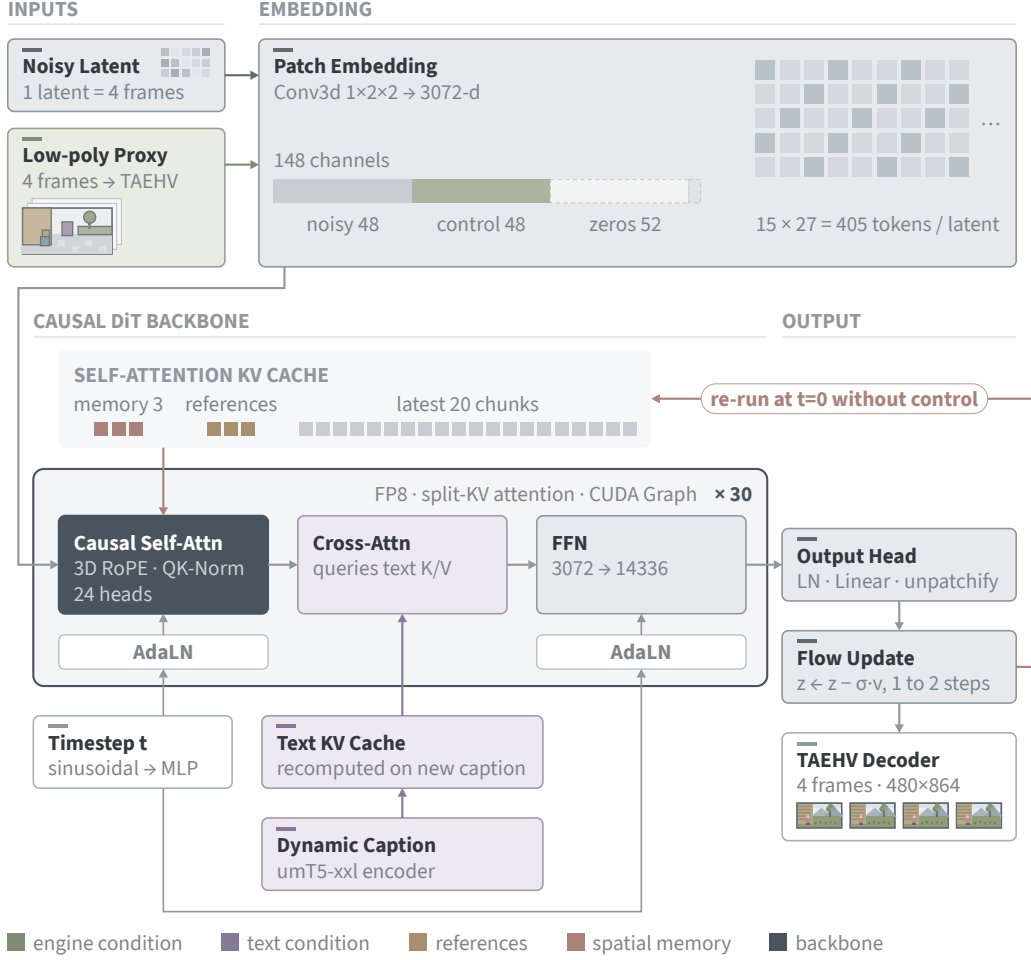


Figure 5: **Architecture of the rendering model.** The noisy latent of one chunk (one latent frame, four video frames) and the low-poly proxy encoded by TAEHV are concatenated into 148 channels and patchified into 405 tokens per latent frame at 480×864 . Thirty causal DiT blocks apply self-attention over a KV cache that holds spatial-memory frames, reference frames, and the latest 20 chunks, cross-attention to the cached text keys and values of the dynamic caption, and a feed-forward network, each modulated by the timestep through AdaLN. The output head predicts a velocity, one or two flow updates produce the clean latent, and TAEHV decodes four frames. The finished chunk is then re-run at $t = 0$ without the control input to write its keys and values into the cache.

model always receives the latest semantic condition (Figure 4). The dynamic caption and the low-poly proxy complement each other. The proxy supplies structure: position, layout, and motion. The caption supplies semantics: which objects are in the frame and what state each is in. Together they constrain generation, so the frames follow the scene structure produced by the engine and also follow changes in events and object states.

4 Rendering Model

4.1 Architecture

The rendering model starts from Wan2.2 5B [10, 11] in its Fun Control variant [12] and is turned into a video-to-video (V2V) model. Four inputs condition each chunk, and each has its own job:

- The **low-poly proxy** rendered by the engine is concatenated with the latent video being generated along the channel dimension. It tells the model what to draw and where.

- The **dynamic caption** (Section 3.2) describes the scene and its current events and supplies the details.
- **Spatial-memory reference frames** keep the scene consistent when the player returns to a place.
- The **first frame** connects the generated video to what is already on screen.

Figure 5 shows the network, whose DiT has 5.0B parameters. The input of the patch embedding has 148 channels: 48 for the noisy latent, 48 for the control latent of the proxy, and 52 zero channels. A $1 \times 2 \times 2$ 3D convolution maps it to a width of 3072, which gives $15 \times 27 = 405$ tokens per latent frame at 480×864 . The backbone has 30 DiT blocks. Each block applies causal self-attention, cross-attention to the text keys and values, and a feed-forward network with a hidden width of 14336, and the timestep modulates the block through AdaLN. Self-attention has 24 heads of width 128 with QK-Norm, and 3D RoPE assigns 44, 42, and 42 of the 128 dimensions to time, height, and width. The text encoder is umT5-xxl [13]. The key and value tensors of the caption are cached and recomputed only when the caption changes. The output head (LayerNorm, linear projection, unpatchify) predicts a velocity, and the latent is updated by $z \leftarrow z - \sigma v$ for one or two steps. TAEHV [14] decodes the four frames of the chunk.

The self-attention KV cache has three parts: the keys and values of three spatial-memory frames, those of the reference frames, and those of the latest 20 chunks. After a chunk is generated, the model runs once more on the finished latent at $t = 0$ without the control input and writes the resulting keys and values into the cache, so later chunks attend to the clean history.

4.2 Supervised Fine-Tuning

We train one model per game. Starting from the Wan2.2-Fun-5B Control weights, the model is fine-tuned as a bidirectional video-to-video model with the flow-matching objective of Wan, on 81-frame windows of the paired data at 704×1280 . Each window is conditioned on its low-poly proxy, its caption, the first frame, and up to three context frames that serve as spatial memory. The context frames are taken from at least 40 frames before the window: candidates whose view overlaps the window are kept, and the closest ones by camera pose are selected. They enter the sequence as clean latents with timestep 0 and are excluded from the loss. To support classifier-free guidance and generation without a given first frame, the first frame is dropped with probability 0.3, the prompt with probability 0.2, and the proxy control with probability 0.2. Training uses 64 GPUs with a global batch of 64, AdamW with a learning rate of 10^{-5} , and 4,000 steps.

4.3 Causal Few-Step Distillation

Following Causal Forcing++ [15], we convert the bidirectional model into a causal autoregressive generator that denoises one chunk (one latent frame) at a time with one or two steps. The three stages below run at 480×864 on 81-frame windows with 64 GPUs and a global batch of 64.

Stage 1: teacher-forced autoregressive diffusion. Starting from the fine-tuned model, each latent frame receives its own noise level and attends only to the clean ground-truth frames before it, the first frame, and the context frames. The model is trained with the same flow-matching loss for 20,000 steps at a learning rate of 2×10^{-6} . The result is a causal teacher that samples each chunk with a multi-step ODE solver.

Stage 2: causal consistency distillation. A frozen copy of the causal teacher, an online student, and an EMA student start from the same weights. On a grid of 48 noise levels, the teacher takes one Euler step with classifier-free guidance of 3 from a noisy chunk to the next lower level; the online student predicts the clean chunk from the higher level and is trained to match the EMA student’s prediction from the lower level. After 5,000 steps with an EMA decay of 0.99, the EMA student generates a chunk in four steps.

Stage 3: distribution matching. The student then learns from its own autoregressive rollouts with asymmetric DMD [16–18]. The real score is the frozen bidirectional model with classifier-free guidance, $v_c + 3(v_c - v_u)$; the fake score is initialized from the same weights and trained online. The fake score is updated at every step and the student every five steps, with learning rates of 4×10^{-7} and 2×10^{-6} . The student denoises each chunk in two steps and the first generated latent frame in

Table 2: **Inference speed and latency, applied cumulatively.** Each row adds one optimization to all rows above it. FPS is the steady-state generation rate of the model; latency is the time from a key press to the first generated frame that reflects it, with the game running at 60 FPS. Speedup is the FPS ratio to the previous row. Setting: 480×864 , chunk size of one latent frame (four frames), KV window of 21 latent frames, one NVIDIA RTX 5090.

Configuration	FPS $\uparrow$	Latency (ms) $\downarrow$	Speedup
Causal Wan2.2 DiT, 4 steps, full Wan VAE	4.5	1110	–
+ tiny VAE (TAEHV)	6.5	867	1.4 $\times$
+ RoPE kernel, cached text KV	15.4	551	2.4 $\times$
+ 8-bit weights + CUDA Graph	24.2	466	1.6 $\times$
+ 1-step distillation	51.2	376	2.1 $\times$
+ asynchronous I/O pipeline	57.7	367	1.1 $\times$
+ fused Triton kernels	65.8	121	1.1 $\times$
+ ring KV cache	73.7	117	1.1 $\times$
+ split-KV FlashAttention	83.4	112	1.1 $\times$
Overall	19 $\times$ faster	9.9 $\times$ lower	

four. Our latest recipe adds three changes that match the deployed inference setting. The student rolls out 21, 41, 61, or 121 latent frames with a rolling cache of the first frame and the latest 20 latent frames, and the two scores evaluate the last 21. At the start of every 20-frame segment, up to three spatial-memory frames are retrieved by camera pose and their keys and values join the cache. Because the deployed system encodes the proxy with TAEHV, the student receives TAEHV-encoded control on 80% of the steps and Wan-VAE-encoded control on the rest, while both scores always see the Wan-VAE encoding. This stage runs for 1,000 steps from the end of Stage 2.

5 Real-Time Inference

The rendering model generates one chunk at a time, and a chunk is one latent frame, *i.e.*, four video frames. Short chunks keep the response to an action short, and they also make model calls frequent: with a single denoising step, each call does little arithmetic, and the GPU spends a large share of its time waiting for the CPU to launch kernels. We remove these costs in the order listed in Table 2.

Launch overhead. The computation of the DiT backbone is recorded once as a CUDA Graph [19] and replayed in a single launch at run time. Linear layers use 8-bit (FP8) weights [20].

Fused kernels. We write fused GPU kernels in Triton [21] for RoPE, normalization, and FP8 quantization. The query, key, and value projections of a layer share one quantization of their input. The key and value tensors of the text condition are cached and recomputed only when the caption changes.

KV cache and attention. The key/value cache of the video history is a ring buffer: when the window slides, the newest latent frame overwrites the oldest slot in place and no history is moved. Self-attention splits the KV window into parts that are computed in parallel and merged [22].

Decoding and I/O. The full Wan VAE decoder is replaced by the lightweight TAEHV decoder [14], and model input and output run in an asynchronous pipeline.

Result. Together these changes raise the steady-state rate from 4.5 to 83.4 FPS (19 $\times$) and cut the key-to-frame latency from 1110 ms to 112 ms (9.9 $\times$). Measuring model inference alone on one RTX 5090 at 480p, the steady-state generation rate is about 83 FPS with one denoising step and about 57 FPS with two, and the peak memory is about 18.2 GiB in both cases. With one step, the model denoises one four-frame chunk in 48 ms. The text encoder, the DiT, and the VAE all reside on the same card.

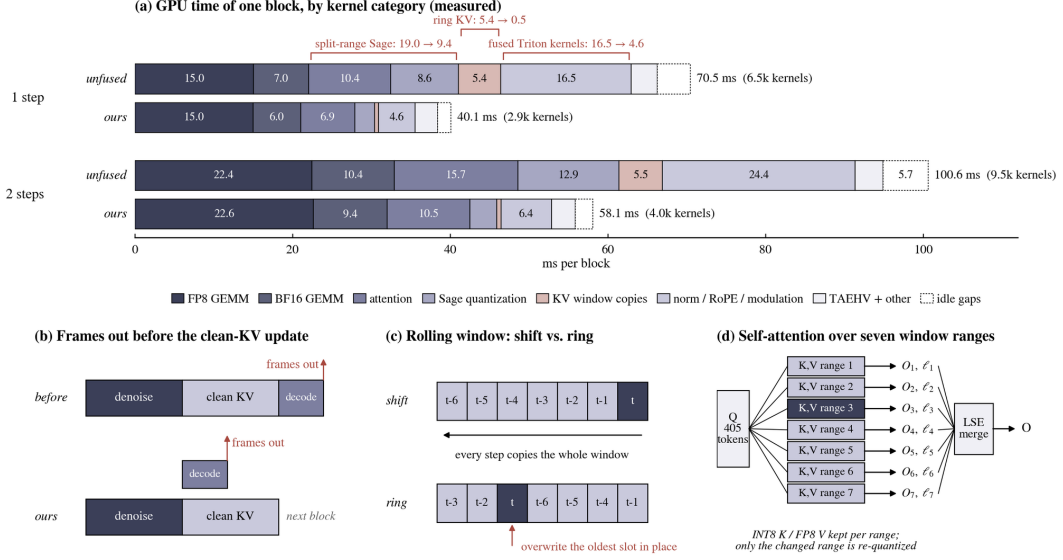


Figure 6: Overview of the inference optimizations. (a) Per-block GPU time on an RTX 5090, broken down by kernel category. (b) Decoding is moved ahead of the clean-KV update, so frames are output earlier. (c) Instead of shifting the whole KV window at every step, the ring cache writes the new frame into the oldest slot. (d) Self-attention is split into seven window ranges whose quantized keys and values are cached; the partial outputs are combined by log-sum-exp.

Table 3: **Latency, hardware, and openness of recent real-time interactive generation systems.** For the systems that generate in chunks, end-to-end latency is listed as two chunk generation times. Matrix-Game 4.0 latency is measured from a key press to the first frame that reflects it, with the game running at 480p and 60 FPS.

System	End-to-end latency	Compute	Open release
Code World Model [23]	≈ 10.2 s (2×5.1 s per chunk, 17 frames per chunk)	$1 \times$ H200 141 GB; peak memory 108–123 GB, does not fit on 80 GB cards	Code, weights, and low-poly assets
StreamRender-H3 [24]	≈ 545 ms (2×273 ms per chunk, about 8.5 frames per chunk)	$8 \times$ GB200 with context parallelism	Code, weights, and low-poly assets
AgentGarten [25]	≈ 878 ms (2×439 ms per chunk, 16 frames per chunk)	$1 \times$ H100 80 GB (data-center GPU)	Code and weights; low-poly assets not released
Matrix-Game 4.0	112 ms from key press to frame; playable at 480p, 60 FPS	$1 \times$ RTX 5090 , a single consumer GPU with 32 GB memory	Code, weights, and low-poly assets, released at launch

6 Experiments

6.1 Setup

We train one rendering model per game on the data in Table 1. The results in this section use the distilled causal models of the castle shooter, the desert shooter, and city driving, each generating two denoising steps per chunk. A held-out rollout starts from the ground-truth first frame and runs autoregressively for 961 frames (40 seconds at 24 FPS) at 480×864 , conditioned on the low-poly proxy and the caption of a held-out clip. Speed and latency are measured on a single NVIDIA RTX 5090 (Section 5).



Figure 7: **Re-rendering over 40-second held-out rollouts.** For each game, the upper row is the low-poly proxy from the engine and the lower row is the frame generated by the distilled causal model at the same time step. The generated frames follow the proxy layout, camera, and object motion throughout the rollout, while the model fills in materials, vegetation, lighting, smoke, and muzzle effects.

6.2 Comparison with Recent Interactive Systems

Table 3 compares Matrix-Game 4.0 with three recent systems that generate interactive video from structured or code-defined worlds. Matrix-Game 4.0 has the lowest end-to-end latency, about one fifth of the next system, and it is the only one that runs on a single consumer GPU.

6.3 Qualitative Results

Figure 7 shows three held-out rollouts at four time steps. The generated frames keep the layout of the proxy: buildings, walls, trees, and vehicles appear where the proxy places them, and the camera follows the same path. The model adds what the proxy omits. Flat-colored blocks become stone walls, thatched houses, palm trees, sandstone buildings, and city facades; the box-shaped player becomes a textured character; and combat produces smoke, sparks, and projectile trails. The look of each scene stays stable over the 40 seconds, and the character keeps its appearance from the first chunk to the last.

6.4 Gameplay Customization with the Coding Agent

With the coding agent, a player can customize a game in natural language on top of the existing game code: reshape the terrain, place scene objects, change NPC behavior and interaction logic, or design an entirely new mode. A shooter where everyone fights alone can become a team battle, and a classic time trial can become a cops-and-robbers chase. Figure 8 shows three modes built this way. The shooter gains health and ammunition control, a shield with limited durability, a shotgun with a higher



Figure 8: **Game modes written with the coding agent, rendered in real time.** Top: in the third-person shooter, the agent adds health and ammunition control, shield durability, a faster shotgun, a game-over condition, and simple sound effects. Middle: in the driving game, the agent implements a time trial with route checkpoints and collision penalties. Bottom: the same driving game turned into a pursuit mode, in which the player must intercept a target vehicle and stay close to it for five seconds. The engine evaluates every rule; the video model renders each frame from the engine output.

fire rate, a game-over condition at zero health, and simple sound effects. The driving game gains a time trial with checkpoints, collision penalties, and drifting, and a pursuit mode in which the player intercepts a target vehicle. All of these rules run in the engine, and the video model renders whatever the engine outputs. We encourage creators to adapt and extend the released game code to explore further gameplay.

7 Related Work

7.1 Interactive Video World Models

Interactive world models generate the next observations of an environment from user actions. Genie learns a latent action space and an autoregressive dynamics model from unlabeled Internet videos [1]; Genie 2 and Genie 3 generate playable 3D environments, and Genie 3 does so in real time at 720p and 24 FPS [2, 3]. GameNGen runs DOOM as a diffusion model at 20 FPS on a single TPU [4], DIAMOND trains agents inside a diffusion world model [26], and Oasis generates a Minecraft-like game in real time [5]. Later systems add richer controls and longer horizons: Hunyuan-GameCraft maps keyboard and mouse input to a shared camera representation [27, 28], Yan targets real-time 1080p simulation and text-driven editing [29], WorldPlay and LingBot-World stream minute-scale interactive video [30, 31], and Matrix-Game 2.0 and 3.0 combine few-step autoregressive diffusion with long-horizon memory for real-time 720p generation [6, 7]. In these models the network carries both the world dynamics and the visuals. Matrix-Game 4.0 keeps the dynamics in game code and uses the network for the visuals.

7.2 Code-Defined Worlds Rendered by Video Models

A recent line of work assigns state and rules to executable code and rendering to a video model. In Code World Model, a coding agent writes code that maintains the world state, and a proxy video compiled from that state conditions a fine-tuned MiniMax-H3 renderer [23]. StreamRender-H3 drives a two-step streaming renderer built on MiniMax-H3 with the semantic maps of a code game engine [24], on top of the Sol inference engine [32]. AgentGarten couples simulators and game engines with a shared neural renderer and uses the resulting worlds to train agents [25]. CoDeR builds executable worlds in code through coordinated roles and renders them with video generation

models [33], and Magpie separates a game engine that resolves player actions from a render server that generates visuals from engine frames [34]. Matrix-Game 4.0 follows the same division of labor and is designed for interactive latency on one consumer GPU: it reaches 112 ms from key press to frame on a single RTX 5090 (Table 3), and it adds a coding agent that edits the game rules and writes the prompts during play.

7.3 Neural Rendering and Structure-Conditioned Video Generation

Neural rendering combines learned generative models with the structure of the graphics pipeline [35, 36]. Enhancing Photorealism Enhancement translates rendered game frames toward photographs using G-buffers from the renderer [37], GameGAN learns a game renderer with a memory module from frames and actions [38], and DLSS 5 adds photoreal lighting and materials to game frames in real time [39]. vid2vid and World-Consistent vid2vid convert semantic maps into temporally coherent video, the latter keeping revisited regions consistent through projected 3D guidance [40, 41]. With diffusion models, ControlNet and VideoComposer add spatial and temporal controls to pretrained generators [42, 43]; Cosmos-Transfer1 turns segmentation, depth, and edge maps into realistic video [44]; and DiffusionRenderer and FrameDiffuser synthesize video from G-buffers [45, 46]. Our rendering model starts from the Wan2.2 Fun Control weights [12, 47], conditions on the low-poly proxy video rendered by the game engine, and is trained on pixel-aligned UE5 pairs.

7.4 Real-Time Autoregressive Video Generation

Streaming generation produces video chunk by chunk. Diffusion Forcing and Rolling Diffusion assign different noise levels across frames [48, 49], and StreamDiT trains on a moving frame buffer for real-time text-to-video [50]. Distribution matching distillation (DMD) turns a multi-step diffusion model into a one-step or few-step generator [16, 17]. CausVid applies DMD to convert a bidirectional video model into a causal one [18]; Self Forcing and Self-Forcing++ train on the student’s own KV-cached rollouts [51, 52]; Causal Forcing initializes the student from an autoregressive teacher [53]; and Causal Forcing++ uses causal consistency distillation to reach frame-wise generation with one or two steps [15]. We follow Causal Forcing++ to distill our bidirectional rendering model. At the systems level, CUDA Graphs [19], FlashAttention and split-KV decoding [22, 54, 55], FP8 arithmetic [20], Triton kernels [21], and the TAEHV decoder [14] supply the components of our inference stack.

8 Conclusion and Future Work

Matrix-Game 4.0 handles how a game runs and how it looks in separate components: code carries the rules, the engine maintains the state, and a video model generates the visuals in real time. A coding agent links the player’s ideas to the game code, so scenes, interactions, and gameplay can be built and changed with natural language and played immediately. The rendering model re-renders low-poly proxy frames at 480p on a single RTX 5090, generates 83 frames per second with one denoising step, and shows the response to a key press after 112 ms. The pipeline from paired UE5 data and dynamic captions to few-step causal distillation and real-time inference is shared across games, so a new scene, interaction, or gameplay mode reuses the same conditions and training recipe.

In the near term, after further efficiency work that lowers the hardware requirement, we plan to release a technical preview on Steam. Next, we will keep improving the model, deploy a refiner to raise the output resolution, explore multiplayer sessions in which the views of different players stay consistent, provide finer control over the generated frames, and support more game genres and scenes.

References

- [1] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *International Conference on Machine Learning*, 2024.
- [2] Jack Parker-Holder, Philip Ball, Jake Bruce, Vibhavari Dasagi, Kristian Holsheimer, Christos Kaplanis, Alexandre Moufarek, Guy Scully, Jeremy Shar, Jimmy Shi, Stephen Spencer, Jessica Yung, Michael Dennis, Sultan Kenjeyev, Shangbang Long, Vlad Mnih, Harris Chan, Maxime Gazeau, Bonnie Li, Fabio Pardo, Luyu Wang, Lei Zhang, Frederic Besse, Tim Harley, Anna Mitenkova, Jane Wang, Jeff Clune, Demis Hassabis, Raia Hadsell, Adrian Bolton, Satinder Singh, and Tim Rocktäschel. Genie 2: A large-scale foundation world model. 2024.
- [3] Philip J. Ball, Jakob Bauer, Frank Belletti, Bethanie Brownfield, Ariel Ephrat, Shlomi Fruchter, Agrim Gupta, Kristian Holsheimer, Aleksander Holynski, Jiri Hron, Christos Kaplanis, Marjorie Limont, Matt McGill, Yanko Oliveira, Jack Parker-Holder, Frank Perbet, Guy Scully, Jeremy Shar, Stephen Spencer, Omer Tov, Ruben Villegas, Emma Wang, Jessica Yung, Cip Baetu, Jordi Berbel, David Bridson, Jake Bruce, Gavin Buttimore, Sarah Chakera, Bilva Chandra, Paul Collins, Alex Cullum, Bogdan Damoc, Vibha Dasagi, Maxime Gazeau, Charles Gbadamosi, Woohyun Han, Ed Hirst, Ashyana Kachra, Lucie Kerley, Kristian Kjems, Eva Knoepfel, Vika Koriakin, Jessica Lo, Cong Lu, Zeb Mehring, Alex Moufarek, Henna Nandwani, Valeria Oliveira, Fabio Pardo, Jane Park, Andrew Pierson, Ben Poole, Helen Ran, Tim Salimans, Manuel Sanchez, Igor Saprykin, Amy Shen, Sailesh Sidhwani, Duncan Smith, Joe Stanton, Hamish Tomlinson, Dimple Vijaykumar, Luyu Wang, Piers Wingfield, Nat Wong, Keyang Xu, Christopher Yew, Nick Young, Vadim Zubov, Douglas Eck, Dumitru Erhan, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Raia Hadsell, Aäron van den Oord, Inbar Mosseri, Adrian Bolton, Satinder Singh, and Tim Rocktäschel. Genie 3: A new frontier for world models. 2025.
- [4] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. *arXiv preprint arXiv:2408.14837*, 2024.
- [5] Decart. Oasis: A universe in a transformer. 2024.
- [6] Xianglong He, Chunli Peng, Zexiang Liu, Boyang Wang, Yifan Zhang, Qi Cui, Fei Kang, Biao Jiang, Mengyin An, Yangyang Ren, et al. Matrix-game 2.0: An open-source real-time and streaming interactive world model. *arXiv preprint arXiv:2508.13009*, 2025.
- [7] Zile Wang, Zexiang Liu, Jiaying Li, Kaichen Huang, Baixin Xu, Fei Kang, Mengyin An, Peiyu Wang, Biao Jiang, Yichen Wei, et al. Matrix-game 3.0: Real-time and streaming interactive world model with long-horizon memory. *arXiv preprint arXiv:2604.08995*, 2026.
- [8] Epic Games. Unreal Engine: The most powerful real-time 3d creation tool. <https://www.unrealengine.com>, 2024.
- [9] Epic Games. City sample project Unreal Engine demonstration. <https://dev.epicgames.com/documentation/en-us/unreal-engine/city-sample-project-unreal-engine-demonstration>, 2022. Unreal Engine documentation, accessed 2026-10-15.
- [10] Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Fei Wu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.
- [11] Wan-Video. Wan2.2. <https://github.com/Wan-Video/Wan2.2>, 2025. GitHub repository; T12V-5B weights at <https://huggingface.co/Wan-AI/Wan2.2-T12V-5B>.
- [12] alibaba-pai. Wan2.2-Fun-5B-Control. <https://huggingface.co/alibaba-pai/Wan2.2-Fun-5B-Control>, 2025. Hugging Face model weights.
- [13] Hyung Won Chung, Noah Constant, Xavier Garcia, Adam Roberts, Yi Tay, Sharan Narang, and Orhan Firat. UniMax: Fairer and more effective language sampling for large-scale multilingual pretraining. In *International Conference on Learning Representations*, 2023.
- [14] Ollin Boer Bohan. TAEHV: Tiny autoencoder for hunyuan video. <https://github.com/madebyollin/taehv>, 2025. Commit 011dfc2, accessed 2026-10-15.
- [15] Min Zhao, Hongzhou Zhu, Kaiwen Zheng, Zihan Zhou, Bokai Yan, Xinyuan Li, Xiao Yang, Chongxuan Li, and Jun Zhu. Causal forcing++: Scalable few-step autoregressive diffusion distillation for real-time interactive video generation. *arXiv preprint arXiv:2605.15141*, 2026.
- [16] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6613–6623, 2024.
- [17] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and Bill Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024.

- [18] Tianwei Yin, Qiang Zhang, Richard Zhang, William T Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 22963–22974, 2025.
- [19] NVIDIA. CUDA C++ programming guide: CUDA graphs. <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>, 2024.
- [20] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.
- [21] 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*, pages 10–19, 2019.
- [22] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. <https://crfm.stanford.edu/2023/10/12/flashdecoding.html>, 2023.
- [23] Yiwen Chen, Guosheng Lin, and Chi Zhang. Code world model: Coding agent as world brain. *arXiv preprint arXiv:2608.25927*, 2026.
- [24] Yanzuo Lu, Tian Ye, Yitong Li, Shuchen Xue, Haozhe Liu, Song Han, and Enze Xie. StreamRender-H3: Interactive coding worlds with real-time streaming diffusion render. <https://nvlabs.github.io/Sana/Sol-Engine/StreamRender-H3/>, 2026. NVIDIA Research project page.
- [25] Jiawei Chi, Shangchen Miao, Zhiyuan Shi, Kailu Wu, Hanyang Wang, Weiliang Chen, Qiyu Dai, Chengyang Li, Jinshan Ren, Jun Gao, Mingsheng Long, Yueqi Duan, Jiangran Lyu, Jialong Wu, and Fangfu Liu. AgentGarten: Code worlds for evolving agents. *arXiv preprint arXiv:2610.12374*, 2026.
- [26] Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos J Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. In *Advances in Neural Information Processing Systems*, volume 37, pages 58757–58791, 2024.
- [27] Jiaqi Li, Junshu Tang, Zhiyong Xu, Longhuang Wu, Yuan Zhou, Shuai Shao, Tianbao Yu, Zhiguo Cao, and Qinglin Lu. Hunyuan-gamecraft: High-dynamic interactive game video generation with hybrid history condition. *arXiv preprint arXiv:2506.17201*, 2025.
- [28] Junshu Tang, Jiacheng Liu, Jiaqi Li, Longhuang Wu, Haoyu Yang, Penghao Zhao, Siruis Gong, Xiang Yuan, Shuai Shao, Linfeng Zhang, et al. Hunyuan-gamecraft-2: Instruction-following interactive game world model. *arXiv preprint arXiv:2511.23429*, 2025.
- [29] Deheng Ye, Fangyun Zhou, Jiacheng Lv, Jianqi Ma, Jun Zhang, Junyan Lv, Junyou Li, Minwen Deng, Mingyu Yang, Qiang Fu, Wei Yang, Wenkai Lv, Yangbin Yu, Yewen Wang, Yonghang Guan, Zhihao Hu, Zhongbin Fang, and Zhongqian Sun. Yan: Foundational interactive video generation. *arXiv preprint arXiv:2508.08601*, 2025.
- [30] Wenqiang Sun, Haiyu Zhang, Haoyuan Wang, Junta Wu, Zehan Wang, Zhenwei Wang, Yunhong Wang, Jun Zhang, Tengfei Wang, and Chunchao Guo. Worldplay: Towards long-term geometric consistency for real-time interactive world modeling. *arXiv preprint arXiv:2512.14614*, 2025.
- [31] Robbyant Team, Zelin Gao, Qiuyu Wang, Yanhong Zeng, Jiapeng Zhu, Ka Leong Cheng, Yixuan Li, Hanlin Wang, Yinghao Xu, Shuailei Ma, et al. Advancing open-source world models. *arXiv preprint arXiv:2601.20540*, 2026.
- [32] Yitong Li, Junsong Chen, Haopeng Li, Haozhe Liu, Jincheng Yu, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Sol video inference engine: Agent-native full-stack acceleration framework for efficient video generation. *arXiv preprint arXiv:2606.23743*, 2026.
- [33] Zixun Fang, Yawen Shao, Kai Zhu, Jie Xiao, Shihan Chen, Yu Liu, Xueyang Fu, Yang Cao, Wei Zhai, and Zheng-Jun Zha. Code plans, diffusion renders: Open-ended generative world modeling. *arXiv preprint arXiv:2609.26458*, 2026.
- [34] Xiaoyu Zhan, Xinyu Wang, Xiaohong Zhang, Huanjie Zhu, Tengjiao Sun, Pengcheng Fang, Jiaxing Yu, Yanwen Guo, and Dongjie Fu. Magpie: Real-time world renderer for interactive games. *arXiv preprint arXiv:2608.27168*, 2026.
- [35] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. State of the art on neural rendering. *Computer Graphics Forum*, 39(2):701–727, 2020.
- [36] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Nießner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhöfer, and Vladislav Golyanik. Advances in neural rendering. *Computer Graphics Forum*, 41(2):703–735, 2022.

- [37] Stephan R. Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1700–1715, 2023.
- [38] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with GameGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1228–1237, 2020.
- [39] NVIDIA. NVIDIA DLSS 5 delivers AI-powered breakthrough in visual fidelity for games. <https://nvidianews.nvidia.com/news/nvidia-dlss-5-delivers-ai-powered-breakthrough-in-visual-fidelity-for-games>, 2026. NVIDIA Newsroom, March 16, 2026.
- [40] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *Advances in Neural Information Processing Systems*, volume 31, 2018.
- [41] Arun Mallya, Ting-Chun Wang, Karan Sapra, and Ming-Yu Liu. World-consistent video-to-video synthesis. In *European Conference on Computer Vision*, pages 359–378, 2020.
- [42] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.
- [43] Xiang Wang, Hangjie Yuan, Shiwei Zhang, Dayou Chen, Jiuniu Wang, Yingya Zhang, Yujun Shen, Deli Zhao, and Jingren Zhou. VideoComposer: Compositional video synthesis with motion controllability. In *Advances in Neural Information Processing Systems*, volume 36, pages 7594–7611, 2023.
- [44] NVIDIA, Hassan Abu Alhaija, Jose Alvarez, Maciej Bala, Tiffany Cai, Tianshi Cao, Liz Cha, Joshua Chen, Mike Chen, Francesco Ferroni, Sanja Fidler, Dieter Fox, Yunhao Ge, Jinwei Gu, Ali Hassani, Michael Isaev, Pooya Jannaty, Shiyi Lan, Tobias Lasser, Huan Ling, Ming-Yu Liu, Xian Liu, Yifan Lu, Alice Luo, Qianli Ma, Hanzhi Mao, Fabio Ramos, Xuanchi Ren, Tianchang Shen, Xinglong Sun, Shitao Tang, Ting-Chun Wang, Jay Wu, Jiashu Xu, Stella Xu, Kevin Xie, Yuchong Ye, Xiaodong Yang, Xiaohui Zeng, and Yu Zeng. Cosmos-Transfer1: Conditional world generation with adaptive multimodal control. *arXiv preprint arXiv:2503.14492*, 2025.
- [45] Ruofan Liang, Zan Gojcic, Huan Ling, Jacob Munkberg, Jon Hasselgren, Zhi-Hao Lin, Jun Gao, Alexander Keller, Nandita Vijaykumar, Sanja Fidler, and Zian Wang. DiffusionRenderer: Neural inverse and forward rendering with video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26069–26080, 2025.
- [46] Ole Beiswenger, Jan-Niklas Dihlmann, and Hendrik P. A. Lensch. FrameDiffuser: G-Buffer-conditioned diffusion for neural forward frame rendering. *arXiv preprint arXiv:2512.16670*, 2025.
- [47] aigc-apps. VideoX-Fun: A video generation pipeline for diffusion transformer. <https://github.com/aigc-apps/VideoX-Fun>, 2026. GitHub repository.
- [48] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.
- [49] David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogetboom. Rolling diffusion models. In *International Conference on Machine Learning*, volume 235 of *PMLR*, pages 42818–42835, 2024.
- [50] Akio Kodaira, Tingbo Hou, Ji Hou, Markos Georgopoulos, Felix Juefei-Xu, Masayoshi Tomizuka, and Yue Zhao. StreamDiT: Real-time streaming text-to-video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 29200–29210, 2026.
- [51] Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv preprint arXiv:2506.08009*, 2025.
- [52] Justin Cui, Jie Wu, Ming Li, Tao Yang, Xiaojie Li, Rui Wang, Andrew Bai, Yuanhao Ban, and Cho-Jui Hsieh. Self-forcing++: Towards minute-scale high-quality video generation. *arXiv preprint arXiv:2510.02283*, 2025.
- [53] Hongzhou Zhu, Min Zhao, Guande He, Hang Su, Chongxuan Li, and Jun Zhu. Causal forcing: Autoregressive diffusion distillation done right for high-quality real-time interactive video generation. *arXiv preprint arXiv:2602.02214*, 2026.
- [54] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.
- [55] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.