Morpheus: Generating 2D Games Without Game Engines on Edge Devices

Stanford CS229 Project

Shobhit Agarwal
Department of Computer Science
Stanford University
shobhit.agarwal@stanford.edu

Antonio Llano
Department of Computer Science
Stanford University
llano@stanford.edu

Chandra Suda
Department of Computer Science
Stanford University
csuda@stanford.edu

Abstract

World models have emerged as a promising paradigm for simulating game environments and training reinforcement learning agents without direct interaction with traditional game engines. While recent work has demonstrated impressive results in photorealistic 3D environments, these approaches typically require billions of parameters and substantial computational resources, limiting their accessibility for edge deployment. We address the problem of generating playable 2D games in real-time on consumer-grade hardware by comparing three distinct approaches: a classical action-conditioned convolutional network, selective state-space modeling with Mamba, and our novel Morpheus architecture—a lightweight latent diffusion model adapted from state-of-the-art world modeling approaches. Our experiments on Flappy Bird, Chrome Dino, Pokemon, and Super Mario demonstrate that Morpheus achieves stable, controllable gameplay at up to 20 FPS on a MacBook Pro CPU with 62.4M parameters, delivering superior visual fidelity (PSNR 36.68, LPIPS 0.192) while achieving a 3× speedup over diffusion baselines.

1 Introduction

World models have become increasingly popular as a means to simulate game environments and enable training of reinforcement learning (RL) agents without direct interaction with real environments. Recent work such as Google's Dreamer 4 (Hafner et al., 2025) has demonstrated that agents can learn complex behaviors in Minecraft solely from offline data collected through pre-trained world models, suggesting a general recipe for safety-critical domains: train an action-conditioned video model of the world, then train the policy inside it. In fields like robotics, where training in virtual environments transfers poorly to the real world (Zhao et al., 2020), world models that closely simulate ground-truth environments could unlock new frontiers for RL agents.

In parallel, generative video models have rapidly improved at simulating interactive environments. DeepMind's GENIE (Bruce et al., 2024) demonstrated action-controllable worlds learned from internet videos, while approaches like GameNGen (Valevski et al., 2024) and DIAMOND (Alonso et al., 2024) achieve state-of-the-art fidelity in DOOM and Atari. Yet these systems typically require billions of parameters, accelerator-class hardware, and are not optimized for edge deployment, limiting practical use in budget-constrained settings.

We tackle the problem of generating playable 2D games in real-time without traditional game engines, optimized for consumer-grade devices. The input to our system is a sequence of game frames (rendered as 320×240 pixel images) paired with one-hot encoded action vectors. We then use various architectures—including convolutional neural networks, selective state-space models (Gu et al., 2023), and our novel Morpheus model—to output predicted future frames conditioned on player actions, enabling autoregressive video generation that emulates gameplay. Unlike existing work emphasizing photorealistic 3D environments, 2D domains admit aggressive state abstraction while still requiring controllable physics and partial observability, enabling careful evaluation of dynamics fidelity versus computational cost under strict latency constraints. Ultimately, Morpheus enables real-time world modeling on consumer-grade laptops with CPU-only, making learned simulators practically accessible for RL training in resource-limited settings.

2 Related Work

Work on learned environments divides naturally into latent world models, diffusion-based neural game engines, structured dynamics models, and classical hand-coded simulators.

Latent world models and neural engines. Dreamer-style methods learn compact latent dynamics and train agents entirely inside those models Hafner et al. (2025), while GAIA-1 extends this idea to autonomous driving with discrete video tokens Hu et al. (2023). Diffusion-based approaches push this further toward generative engines: GENIE Bruce et al. (2024) uses diffusion transformers for action-conditioned video, and GameNGen Valevski et al. (2024) plus DIAMOND Alonso et al. (2024) achieve state-of-the-art fidelity and stable rollouts in DOOM and Atari. These works are clever in decoupling representation from control and in how they inject actions and tune sampling, but they assume accelerator-class hardware and prioritize visual quality over strict FPS/VRAM limits. Our diffusion pipeline is a deliberately compressed variant, designed to test how much of that robustness survives when forced onto consumer devices.

Structured dynamics and state-space models. Structured world belief models Singh et al. (2021) and modern state-space architectures such as Mamba Gu et al. (2023) learn dynamics over low-dimensional states rather than pixels. Their strengths are efficiency and stability when the state abstraction matches the environment; their weakness is sensitivity to the choice of state. Our SSM baseline follows this paradigm and exposes how hand-crafted coordinates struggle with camera motion and off-screen objects in scrolling 2D games.

Classical simulators. In most RL systems, the environment is still written by hand: PPO agents Schulman et al. (2017) and gameplay data pipelines Kozar (2023) rely on fixed simulators rather than learned engines. In contrast, we evaluate CNN, SSM, and diffusion models as drop-in simulators that must themselves generate playable rollouts under real-time, low-VRAM constraints.

3 Dataset and Features

In several of the de facto approaches for world model simulations, researchers model the environment as a Partially Observable Markov Decision Process (POMDP) Singh et al. (2021). A POMDP can be defined as: $(S, A, O, T, R, O, \gamma)$, where S is a set of states, A is the set of actions, and O is the set of observations. The transition function $T : S \times A \times S \to [0, 1]$ describes the environment dynamics (i.e. $p(s_{t+1}|s_t, a_t)$, and the reward function $R : S \times A \times S \to \mathbb{R}$ maps transitions to scalar rewards. In a POMDP, agents can not directly access states s_t and instead only tries to reason about the environment via noisy observations $x_t \in O$. Thus, for a baseline, we avoid modeling the environment as a POMDP, and instead construct an environment where an agent can directly access the states and try to learn environment dynamics (e.g. a state-space model), comparing its efficacy to a world model approach. Ultimately, we can learn any architecture that allows for autoregressive video frame prediction to emulate gameplay.

Our datasets are constructed under the assumptions of a POMDP. Specifically, following the collection paradigms of Valevski et al. (2024), we first train an agent to play a selected 2D game in Gymnasium Towers et al. (2025), an open-source library designed for gaming reinforcement learning, via Proximal Policy Optimization Schulman et al. (2017). For some games where training an agent is unreasonably difficult given the time constraint, we elect to have a human play a few games and record the data (via a custom script that tracks the action taken and frame of game). Then, we collect 20-100 "episodes" of the agent playing the game with varying starting states. Each episode consists of 800 states, with each state consisting of the environment observation (frame with pixel data) and a one-hot encoded action vector. It's important to note that for all games we maintain a frame size of 320×240. We adjust the total number of episodes collected depending on the complexity of the game (the complexity of the game is defined by action-space size). We set aside 3 episodes from each game dataset for testing. The table below shows the data collected for different games:

Game	# of Episodes	Human Generated
Flappy Bird	100	
Chrome Dino	45	
Pokemon	5	✓
Super Mario	20	✓

Table 1: Data collection quantities.

4 Methods

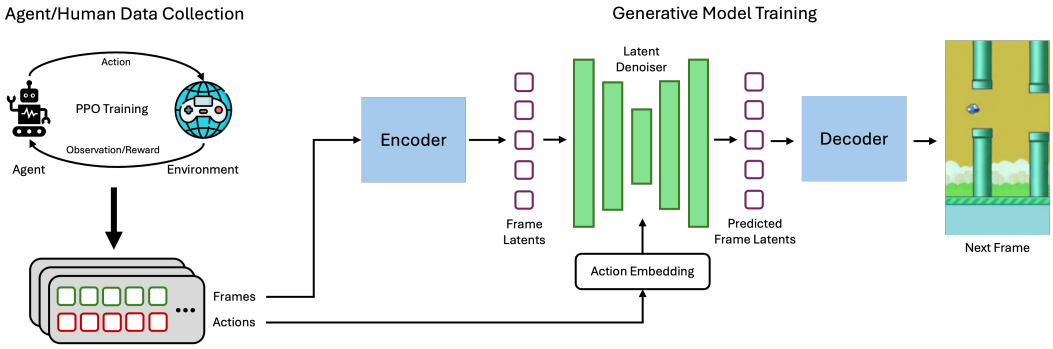

Figure 1: Training and model architecture for the best performing method we designed.

4.1 Classical Neural Network

We implement an action-conditioned convolutional model (primarily as a baseline to compare our other architectures) which learns a deterministic next frame function: $f_\theta = (x_t, u_t) \to \hat{x}_{t+1}$, where x_t is the previous rendered frame and u_t is a one-hot encoded control input of length n number of actions. Video frames serve as targets, controls are approximated, and no explicit latent state is provided—memory is instead approximated by feeding the previous frame (and, with some probability, the model's own prior prediction) back as input. We minimize a reconstruction loss $L(\hat{x}_{t+1}, x_{t+1})$ with teacher forcing, scheduled sampling, and frame dropout. At test time, we can run the network in a tight autoregressive loop, producing a playable world without hand-coded logic—simple to deploy and stable over short horizons.

4.2 State-Space Modeling

We apply Mamba Gu et al. (2023), a selective state-space model (SSM), to approximate the underlying transition dynamics of the Flappy Bird environment. Given the observable state vector $s_t \in \mathbb{R}^{10}$—comprising the coordinates of the three nearest pipe sets $\{p_{i,j}\}$ and the agent's kinematics $[x, y, v]$ (fully expanded, $s_t = [p_{0,0}, p_{1,0}, p_{0,1}, p_{1,1}, p_{0,2}, p_{1,2}, x, y, v, a]$)—we formulate the task as learning a world model $f(s_t, a_t) \to s_{t+1}$. Unlike Linear Time-Invariant (LTI) systems, Mamba employs a selection mechanism where the discretization step Δ and projection matrices $\mathbf{B}$ and $\mathbf{C}$ are functions of the input $u_t = [s_t; a_t]$. This allows the model to dynamically modulate the recurrent rule $h_t = \bar{\mathbf{A}}_t h_{t-1} + \bar{\mathbf{B}}_t u_t$, where $\bar{\mathbf{A}}_t = \exp(\Delta_t \mathbf{A})$ represents the discretized state transition via a Zero-Order Hold (ZOH). The final prediction is projected via $s_{t+1} = \mathbf{C}_t h_t + \mathbf{D} u_t$, allowing the network to selectively compress historical context or reset its latent state h_t in response to high-impulse events, such as the instantaneous velocity change caused by a "flap" action.

4.3 World Modeling

We propose adapting GameNGen Valevski et al. (2024) and DIAMOND Alonso et al. (2024), two state-of-the-art approaches for autoregressive video game generation, to emphasize performance on smaller, lightweight 2D games (we choose these particular methods since they are currently state-of-the-art at generating video games without engines).

Both architectures rely on a formulation of diffusion, which attempts to model the data distribution by learning to reverse a gradual noise corruption process. DIAMOND specifically adopts the Elucidated Diffusion Model (EDM) framework Karras et al. (2022), which formulates the diffusion process as a probability flow Ordinary Differential Equation (ODE). Unlike standard approaches that predict noise residuals, the network D_θ is trained to directly estimate the clean data x from a noisy input $x + \sigma\epsilon$ by minimizing the weighted L_2 loss $\mathbb{E}_{x,\sigma}[\lambda(\sigma)\|D_\theta(x + \sigma\epsilon; \sigma) - x\|^2]$, where σ represents the amount of noise added to the image and ϵ the quantification of noise. We choose DIAMOND's formulation specifically because EDM models diffusion as a continuous rather than discretized process, which

is critical for our optimization, as it decouples the noise schedule from the network architecture, maintaining stability even when the inference trajectory is collapsed to a single step.

Taking inspiration from GameNGen, each frame x_t is compressed into a highly compact latent space of size $64 \times 48 \times 4$ encoded into a continuous vector $z_t \in \mathbb{R}^{C \times H' \times W'}$ using a convolutional autoencoder and model $p_\theta(z_{t+1}|z_{t-K:t}, a_{t-K:t})$. The diffusion model D_θ operates directly in this latent space (thus reducing generation cost significantly), taking as input the noisy latent $z_t + \sigma\epsilon$ concatenated channel-wise with the recent history of previous frames $\{z_{t-k}, \ldots, z_{t-1}\}$. We implement D_θ as a U-Net architecture Ronneberger et al. (2015) which is conditioned on the noise level σ, conditioning noise σ_{cond}, and the sequence of actions $\{a_{t-k}, \ldots, a_{t-1}\}$ via Adaptive Group Normalization (AdaGN) layers injected throughout the network. To curb long-horizon drift observed in DIAMOND, we train with conditioning noise: Gaussian perturbations, random frame-drop/masking, and periodic self-conditioning (replace a conditioning frame with a model prediction). The loss is standard diffusion denoising on z_{t+1} with optional action-consistency auxiliary (predict a_t from $\hat{z}_{t+1}$) to discourage visually plausible but control-inconsistent futures. At test time, we roll out autoregressively with a a 1-step sampler, yielding stable, controllable 2D generations under real-time budgets.

5 Experiments / Results / Discussion

Overall, over long horizons, our method achieves simulation quality comparable to the original game while maintaining a small architecture capable of being deployed onto a device as small as the iPhone 17. We evaluate performance on how well the generated game compares to the ground truth (e.g. does it look similar?, is it playable?, etc.) and how fast it can run in real-time. As such, we choose the following metrics:

Image Quality: We measure LPIPS Zhang et al. (2018) and PSNR using the setup described in Figure 1, where we sample an initial state and predict a single frame based on ground-truth contextual frames. PSNR measures the raw signal integrity of the reconstruction, whereas LPIPS serves as a proxy for human visual assessment, computing the visual distance between two images via a pre-trained neural network. Figure 2 shows the rollout of the model on various games.

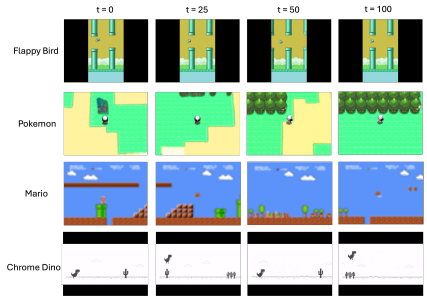

Figure 2: Rollouts of our world model architecture on different video games. Videos of flappy bird gameplay for all models tested can be seen here: `https://drive.google.com/drive/folders/1zEiN1368QEJPyW7eJ_mquYfDm6wS_itR?usp=sharing`.

FPS Throughput We measure the maximum number of frames the model can render per second on an M3 Max Macbook Pro, relying only on the CPU as a sufficient representative sample of consumer hardware. Our flagship model achieves 18.13 FPS, inference time of 53.7ms, and a total of 62.4M parameters on Flappy Bird.

Model	Flappy Bird	Chrome Dino	Pokemon	Super Mario
Classical NN	18.15 / 0.710 / **145.00**	22.80 / 0.445 / **138.50**	18.50 / 0.680 / **125.00**	7.20 / 0.885 / **112.00**
MAMBA	31.40 / 0.285 / 62.50	-	-	-
GameNGen (Modified)	33.90 / 0.205 / 5.20	33.75 / 0.228 / 5.80	29.80 / 0.275 / 2.17	26.50 / 0.315 / 1.3
Morpheus (Ours)	**36.68** / **0.192** / 18.13	**34.85** / **0.210** / 20.41	**31.20** / **0.254** / 13.90	**28.15** / **0.290** / 11.50

Table 2: Comparisons of PSNR/LPIPS/FPS for every model. Note: MAMBA was only tested on Flappy Bird due to insufficient environment representations for other games.

As shown in Table 2, the trade-off between simulation quality and computational throughput is evident across the baselines. The *Classical NN* achieves the highest throughput (up to 145 FPS) but fails to capture granular details, resulting in the poorest image quality (LPIPS > 0.4 across all environments), likely due to a lack of explicit memory within the convolutional layers, making it effectively impractical to scale beyond a few frames. Conversely, the modified *GameNGen* produces competitive visual fidelity (attributed to its diffusion approach) but suffers from prohibitive inference costs (approx. 5 FPS even after scaling the architecture down to allow for inference on a Macbook Pro), rendering it unplayable in real-time scenarios.

Our modified architecture of DIAMOND achieves the highest visual fidelity among all tested methods (PSNR 36.68, LPIPS 0.192) while maintaining a playable throughput of 18.13 FPS on an M3 Max CPU. This represents a $3\times$ speedup over the diffusion-based baseline while also outperforming in reconstruction quality and having 100x fewer parameters (62.4M). To achieve this efficiency beyond architectural modifications, hyperparameters were rigorously tuned to maximize the ratio of perceptual quality to parameter count.

Specifically, we conducted an ablation study on the number of base channels for the autoencoder. Starting from a higher baseline, we iteratively reduced the channel width by factors of two. We observed that reducing the base channels to 64 provided the optimal inflection point; further reductions led to a sharp increase in reconstruction artifacts (blurriness), while higher channel counts yielded diminishing returns in PSNR at the cost of significantly higher inference latency. A similar heuristic was applied to the selection of residual blocks and conditioning channels. The noise parameter σ for the EDM SDE was selected via a random search within the interval $[0.5, 1.0)$. We found that values near the lower bound resulted in insufficient diversity, while values approaching 0.9 were optimal to stabilize trajectories.

Despite the strong performance on Flappy Bird and Chrome Dino, our method exhibits degradation in environments with higher visual complexity or limited training data. In data-constrained regimes like Super Mario and Pokemon, the model struggles to maintain object permanence over long horizons. As observed in Figure 2, the generated Mario sprite vanishes at $t = 50$. Similarly, the Pokemon generations appear significantly blurrier than the 2D scrolling games. This indicates that the current architecture requires a density of training data that scales with the complexity of the game's state space to prevent manifold collapse during autoregressive generation. Earlier collapse in these games also suggest the model overfits closely to the starting sequence and is unable to generalize a robust internal "world" representation of the environment. Furthermore, our use of 1-step denoising, while essential for real-time performance, introduces artifacts during discontinuous state changes. Specifically, "crash" sequences in Flappy Bird often result in a blurry bird (visible in the provided video link). We hypothesize that the multimodal distribution of possible next states upon collision is averaged out by the single-step inference, resulting in a mean-pixel prediction rather than a sharp, distinct outcome.

6 Conclusion / Future Work

In this work, we demonstrated that high-fidelity, playable world models can be effectively deployed on consumer-grade hardware through aggressive architectural optimization. By adapting latent diffusion models with efficient 1-step sampling, our Morpheus architecture achieves a critical balance between visual coherence and inference latency, outperforming classical baselines in quality and unoptimized diffusion models in speed (reaching up to $\sim$20 FPS on a CPU). While we successfully simulated simple 2D dynamics, the model's struggle with object permanence in complex environments like Super Mario highlights the dependency on dense training data for maintaining robust internal representations and substantiate lower performance of SSM-style and classical neural network architectures. Future work will focus on mitigating manifold collapse in multi-sprite scenarios by exploring consistency distillation to sharpen stochastic transitions (e.g., collisions), scaling to 3D games, real-time style control (e.g. changing the backgrounds/characters in real-time with text prompts), and evaluating whether RL agents trained inside these lightweight simulators can transfer policies back to real engines.

7 Contributions

- **Shobhit Agarwal**: Wrote code to modify DIAMOND architecture to support autoencoder instead of UNET. Trained models for neural network & MAMBA, and GameNGen modifications, also synthesized video results/analysis.

- **Antonio Llano**: Implemented DIAMOND improvements to GameNGen via EDM sampling and AdaNorm action injection. Synthesized video results/analysis across architectures and games.
- **Chandra Suda**: Conducted ablations with latent space sizes and denoising steps and tested with Chrome Dino game for generalization. Helped with results and benchmakring.

References

E. Alonso et al. 2024. Diffusion for world modeling: Visual details matter in atari (diamond). In *NeurIPS 2024 (Spotlight)*.

J. Bruce et al. 2024. Genie: Generative interactive environments. *arXiv preprint arXiv:2402.15391*.

A. Gu et al. 2023. Linear-time sequence modeling with selective state spaces (mamba). *arXiv preprint arXiv:2312.00752*.

D. Hafner et al. 2025. Training agents inside of scalable world models (dreamer 4). *arXiv preprint arXiv:2509.24527*.

A. Hu et al. 2023. Gaia-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. 2022. Elucidating the design space of diffusion-based generative models.

A. Kozar. 2023. Data collection using deep reinforcement learning for game play data. Technical report, University of Manitoba.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

G. Singh, S. Peri, J. Kim, H. Kim, and S. Ahn. 2021. Structured world belief for reinforcement learning in pomdp. In *Proceedings of the 38th International Conference on Machine Learning*.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2025. Gymnasium: A standard interface for reinforcement learning environments.

D. Valevski, Y. Leviathan, M. Arar, and S. Fruchter. 2024. Diffusion models are real-time game engines.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric.

W. Zhao, J. P. Queralta, and T. Westerlund. 2020. Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744.