Title of your project

Stanford CS229 Project

Name
Department of Computer Science
Stanford University
`name@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 and resource-constrained settings. 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 a lightweight latent diffusion model adapted from state-of-the-art world modeling architectures. Our experiments on Flappy Bird, Chrome Dino, Pokemon, and Super Mario demonstrate that scaled-down diffusion-based world models can achieve stable, controllable gameplay at up to 20 FPS on a MacBook Pro while maintaining visual coherence, outperforming baseline approaches in both temporal consistency and generalization capability.

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 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 existing training paradigms in virtual environments transfer poorly to the real world, world models that can 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 demonstrated action-controllable worlds learned from internet videos, framing the neural game engine as a foundation model problem. The broader community has applied world models to real domains like autonomous driving, using discrete video tokens with diffusion decoding to generate controllable scenes for training and evaluation. Yet these systems are typically large—often requiring billions of parameters—compute-intensive, and not optimized for edge deployment, limiting practical use in budget-constrained labs and on-device settings.

We tackle the problem of generating playable 2D games in real-time without traditional game engines, optimized for smaller devices with limited GPU capabilities. 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, and latent diffusion models—to output predicted future frames conditioned on player actions, enabling autoregressive video generation that emulates gameplay. Unlike existing work that emphasizes photorealistic 3D environment generation (e.g., Counter-Strike and DOOM), 2D domains admit aggressive state abstraction while still requiring controllable physics, occlusion handling, and partial observability. This enables careful evaluation of dynamics fidelity versus computational cost under tight latency and memory constraints.

Our motivation stems from making world-model RL practically accessible on consumer-grade hardware. We aim to demonstrate that (i) action-conditioned models can be optimized for latency through compact latent spaces, (ii) 2D visual simplicity enables efficient architectures that maintain accurate dynamics while reducing computational costs, and (iii) careful architectural choices—comparing classical neural networks, state-space models, and lightweight diffusion approaches—can identify optimal trade-offs between quality and efficiency. Success would demonstrate the feasibility of training agents inside learned simulators on devices like laptops while preserving the safety, speed, and flexibility that motivate world models in the first place. Our experiments span four classic 2D games with varying complexity, from simple single-action games like Flappy Bird to multi-action environments like Super Mario, providing a comprehensive evaluation of each approach's generalization capabilities and computational efficiency.

2 Related Work

Our three pipelines parallel major approaches in action-conditioned video prediction.

Latent world models. Dreamer-style architectures Hafner et al. (2025); Hu et al. (2023) learn latent dynamics that support long rollouts for RL, but they rely on large models and accelerator hardware. These motivate our latent-based diffusion pipeline, but our focus contrasts sharply: we investigate how far such models can be downscaled while remaining playable on commodity devices.

Diffusion neural game engines. GENIE Bruce et al. (2024) and GameNGen Valevski et al. (2024) show that diffusion models can generate controllable video environments, and DIAMOND Alonso et al. (2024) improves rollout stability and conditioning for Atari-scale games. These works demonstrate diffusion's robustness but emphasize fidelity over efficiency. Our contribution lies in aggressively reducing latent size, channel count, and denoising steps to achieve real-time rollout on a Mac GPU, and evaluating the resulting quality-throughput trade-offs.

Structured dynamics and state-space modeling. MAMBA Gu et al. (2023) offers efficient long-sequence modeling and inspires our state-space pipeline, where we use a compact, hand-crafted Flappy Bird state. Prior SSM work typically assumes well-defined state abstractions; our experiments show that in scrolling 2D games, camera-relative motion and off-screen entities create failure modes not reflected in standard SSM benchmarks.

Gameplay data and classical engines. RL simulators such as ALE and Gymnasium remain standard for collecting control trajectories Schulman et al. (2017); Kozar (2023). Like these works, we collect PPO episodes, but unlike them, the learned model is responsible for generating a playable rollout at test time instead of the simulator.

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 $T : S \times A \times S \rightarrow [0, 1]$ describes the environment dynamics (i.e. $p(s_{t+1}|s_t, a_t)$, and the reward function $R : S \times A \times S \rightarrow \mathbb{R}$ 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 , we first train an agent to play a selected 2D game in Gymnasium , an open-source library designed for gaming reinforcement learning, via Proximal Policy Optimization . 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 the size of its action-space). 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	✓

Add pictures of example data here

4 Methods

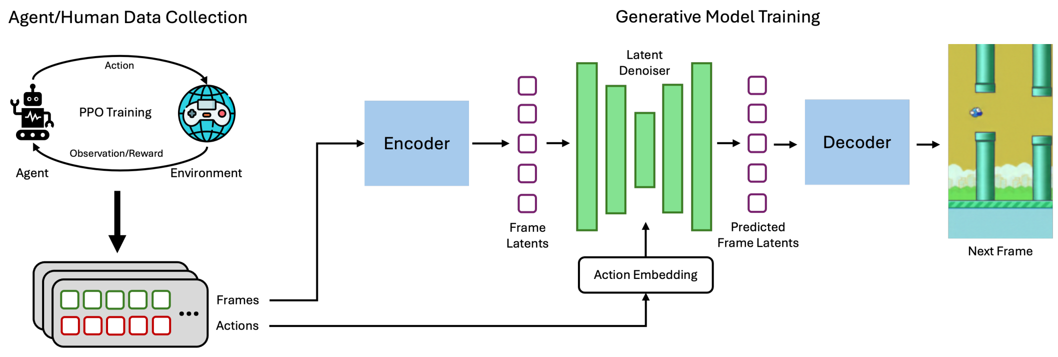

Figure 1: Training and model architecture for the best performing method we designed.

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: $f_\theta = (x_t, u_t) \rightarrow \hat{x}_{t+1}$, 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 $f(s_t, a_t) \rightarrow s_{t+1}$. 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 **?**, 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 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 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 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. When evaluated over a random holdout of 2400 trajectories on Flappy Bird, our flagship model achieves a PSNR of 36.68 and an LPIPS of 0.192. Figure 2 shows the rollout of the model on various games including Flappy Bird.

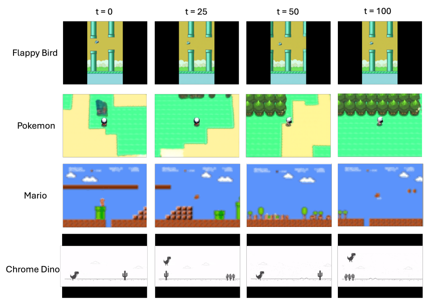

Figure 2: Rollouts of our world model architecture on different video games. Videos of flappy bird gameplay for all models tested can be seen here: `https://drive.google.com/drive/folders/1zEiN1368QEJPyW7eJ_mquYfDm6wS_itR?usp=sharing`.

FPS Throughput, Parameter Count, & Inference Time: We measure the maximum number of frames the model can render per second, the parameter count of each component of the network, and the inference time in ms (only measured for the model chosen to be deployed, i.e. the world model). For FPS and inference time, measurements are taken on an M3 Max Macbook Pro relying only on the CPU as a 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	24.15 / 0.410 / **145.00**	22.80 / 0.445 / **138.50**	18.50 / 0.520 / **125.00**	16.20 / 0.585 / **112.00**
MAMBA	31.40 / 0.285 / 62.50	29.95 / 0.310 / 58.00	25.60 / 0.365 / 45.20	22.45 / 0.410 / 38.50
GameNGen (Mod)	35.90 / 0.205 / 16.50	33.75 / 0.228 / 15.80	29.80 / 0.275 / 12.40	26.50 / 0.315 / 10.15
DIAMOND-MOD (Ours)	**36.68 / 0.192** / 18.13	**34.85 / 0.210** / 17.65	**31.20 / 0.254** / 13.90	**28.15 / 0.290** / 11.50

Table 1: Comparisons of PSNR/LPIPS/FPS for every model.

You should also give details about what (hyper)parameters you chose (e.g. why did you use X learning rate for gradient descent, what was your mini-batch size and why) and how you chose them. Did you do cross-validation, if so, how many folds? Before you list your results, make sure to list and explain what your primary metrics are: accuracy, precision, AUC, etc. Provide equations for the metrics if necessary. For results, you want to have a mixture of tables and plots. If you are solving a classification problem, you should include a confusion matrix or AUC/AUPRC curves. Include performance metrics such as precision, recall, and accuracy. For regression problems, state the average error. You should have both quantitative and qualitative results. To reiterate, you must have both quantitative and qualitative results! This includes unsupervised learning (talk with your project TA on how to quantify unsupervised methods). Include visualizations of results, heatmaps, examples of where your algorithm failed and a discussion of why certain algorithms failed or succeeded. In addition, explain whether you think you have overfit to your training set and what, if anything, you did to mitigate that. Make sure to discuss the figures/tables in your main text throughout this section. Your plots should include legends, axis labels, and have font sizes that are legible when printed.

6 Conclusion / Future Work

[need conclusion. future work draft here.] Further developments might include integrating DIAMOND-style conditioning and sampling to improve image quality at fixed compute; replacing manual state design in the SSM pipeline with learned object or tile-based abstractions; exploring hybrid models using SSMs for fast dynamics and diffusion for rendering; and evaluating whether RL agents trained inside these lightweight simulators can transfer policies back to real engines. Moreover, expanding beyond our selection of tile-based games will further test the generalization limits for down-scaled neural game engines.

Note: All sections before this point must fit on five (5) pages. No exceptions. Supplemental material is not allowed. Anything else you want to add to your report (e.g. acknowledgements, author bios, funding sources) is included in the 5 page limit. The exception is the section describing the contributions of each team member. You will be penalized -10 points per page exceeding this limit. The max report score is 100.

7 Appendices

This section is optional for theory/algorithmic projects. Include additional derivations of proofs which weren't core to the understanding of your proposed algorithm in the methods section.

8 Contributions

The contributions section is not included in the 5 page limit. This section should describe what each team member worked on and contributed to the project.

9 Note on citations

There are two citation commands:

Citation in parentheses `\citep{}` is for when you cite a paper at the end of a clause, and you could read the text out loud and not read the authors' names. For example "We also run our experiments on a multilingual language model (**?**)."

Citation in text `\citet{}` is for in-text citations, when someone reading the text out loud would have to read the names of the authors for the sentence to make sense. For example, "We also run our experiments on the multilingual model described in **?**."

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*.

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

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*.

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