TRAINING EXPRESSIVE VIDEO DIFFUSION WORLD MODELS FROM SCRATCH IN UNDER 50 GPU HOURS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive video diffusion has emerged as a promising approach for training expressive world models which can simulate complex interactions in dynamic environments. However, existing video diffusion models are typically highly expensive to train, which can often limit open-source research to fine-tuning of existing models or inference-time approaches with limited flexibility. In this work, we introduce WDDQ, a computationally efficient training recipe for a world model for Minecraft. In particular, we develop an autoencoder with a highly compressed latent space, a high-fidelity causal diffusion decoder, and a world model architecture combining an efficient masked predictor with a diffusion model that generates latent frames conditional on predictor outputs. Using this recipe, we train a world model that outperforms existing open-source models while training from scratch using less than 50 GPU hours. Unlike prior work, we accomplish this without distilling pretrained visual encoders, overfitting on specific scenes, or leveraging privileged information about 3D positions or geometry.

1 INTRODUCTION

Understanding the dynamics of the world is a crucial capability for intelligent agents to learn, plan, and achieve goals in complex environments. Recently, video diffusion has emerged as a promising paradigm for training high-fidelity, expressive world models (Alonso et al., 2024; Valevski et al., 2024; Decart et al., 2024; Guo et al., 2025). However, video diffusion models can be highly expensive to train. For example, Open-Sora 2.0, a recent open-source video diffusion model, was trained using over 99,800 H200 hours (Zheng et al., 2025). High training costs can increase the environmental footprint of model development (Strubell et al., 2019; Patterson et al., 2021; Schwartz et al., 2020) and increase the barrier to entry for research outside of a few select organizations with large compute budgets (Bommasani et al., 2021; Whittaker, 2021).

In this work, we introduce **WDDQ** (World Diffusion Done Quick), a significantly improved recipe for training video diffusion world models. Using this recipe, we produce an expressive world model for Minecraft that outperforms existing open-source models while training from scratch using less than 50 hours on a single GH200 GPU. Our recipe includes leveraging a highly diffusable latent space by training a generative autoencoder with improved trade-offs between reconstruction fidelity and latent compression. We also introduce several improvements to a separable causal diffusion world model architecture, involving efficient learning of temporal dynamics via a masked next-latent prediction objective. Combined with tweaks and modernizations to various low-level components of the architecture and optimizer, we show experimentally that this recipe can produce models with high visual fidelity and strong controllability via action conditioning compared to existing baselines. Notably, we achieve this without leveraging representations from pretrained visual encoders (aside from LPIPS loss used for autoencoder training), overfitting on specific scenes, or leveraging privileged information about the underlying environment state or 3D geometry. We hope that this recipe will help reinvigorate open-source research in world modeling as well as provide broadly useful ideas and components for improving the training efficiency of video diffusion models in general.

Our contributions can be summarized as follows:

- We introduce **WDDQ**, a computationally efficient recipe for training expressive video diffusion world models from scratch in under 50 hours on a single GH200 GPU.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- We develop a highly compressed video autoencoder with $4\times$ temporal downsampling, $32\times$ spatial downsampling, and 32 latent channels, paired with a generative diffusion decoder that preserves visual reconstruction fidelity.
- We design a separable world model architecture that decouples action-conditioned temporal prediction from next-frame diffusion, enabling masked predictor training which simultaneously improves training throughput and generation quality.
- We demonstrate that our world model outperforms existing open-weight Minecraft world models in terms of generation quality and action controllability, without distilling pretrained visual encoders, privileged simulator state, or scene-specific overfitting.

2 Related Work

Video world models. Recently, world models have become a topic of interest in model-based reinforcement learning and for building interactive environments (Hafner et al., 2023; 2025). Recent, high-fidelity world models forgo predicting explicit states, instead focusing on action-conditioned video generation, producing rollouts directly in pixel space (Decart et al., 2024; Guo et al., 2025; Alonso et al., 2024; Valevski et al., 2024). Minecraft, a video game with open-world, procedurally generated environments and long-horizon dynamics, has become an important testbed for world model research. Open-weight Minecraft models such as Lucid-v1 (Hojel & Seid, 2024), Oasis (Decart et al., 2024), and MineWorld (Guo et al., 2025) demonstrate impressive interactive generation, but face trade-offs in terms of coherence, controllability, training cost, and inference throughput. Meanwhile, some works demonstrate efficient training, but study only the low-resolution (128×128) and low-diversity TECO Minecraft dataset which contains only swamp scenes (Po et al., 2025; Yan et al., 2023). In this work, we target efficient from-scratch training for high-resolution Minecraft world models, operating at 640×360 resolution across diverse scenes without distilling pretrained visual encoders, overfitting to specific scenes, or leveraging privileged simulator state.

Latent video diffusion. Latent diffusion models reduce the cost of generative modeling by operating in the compressed latent space of a pretrained autoencoder (Rombach et al., 2022; Podell et al., 2023; Yang et al., 2024b). Large-scale video diffusion models such as HunyuanVideo (Kong et al., 2024) and Wan (Wan et al., 2025), often leverage highly spatially and temporally compressed latents to support high-resolution video synthesis. However, low-dimensional latents are often lossy representations of the underlying pixels, resulting in imperfect reconstructions. Recently, autoencoders with pixel diffusion decoders have been shown to be capable of producing reconstructions with high visual fidelity (Chen et al., 2025; OpenAI, 2023), generating convincing low-level details absent in the latent space. In this work, we develop an autoencoder designed specifically to improve the efficiency of world model training, with higher spatial compression and dimensionality reduction than existing works, while maintaining reasonable fidelity via a pixel diffusion decoder trained to autoregressively decode video frames conditional on latents.

Efficient video architectures. Existing image diffusion works, such as Zheng et al. (2023) and Krause et al. (2025), leverage sparsity in order to improve training efficiency. Meanwhile, in video diffusion, architectures that separately model causality and diffusion have demonstrated efficiency improvements during training and inference (Bai et al., 2026). For WDDQ, we develop a video diffusion architecture that leverages the strengths of both approaches, enabling memory-efficient masked training of a large predictor backbone while maintaining strong generation fidelity with low-latency inference via a smaller spatial-only diffuser.

3 Preliminaries

Our world model parameterizes a generative model over frames via rectified flow (Liu et al., 2022). Given a sample $x^{(0)}$ from a dataset $\mathcal{D}$, a noise sample ε from isotropic Gaussian noise, and a noise level (denoted with superscripts, reserving subscripts for video timesteps), we denote

$$x \sim \mathcal{D}, \qquad \varepsilon \sim \mathcal{N}(0, I), \qquad x^{(\sigma)} = (1-\sigma)x + \sigma\varepsilon. \tag{1}$$

f_θ is trained to predict $\mathbb{E}[v^{(\sigma)} \mid x^{(\sigma)}]$, where $v^{(\sigma)} = \varepsilon - x$, via a mean-squared-error (MSE) loss:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left\| f_\theta(x^{(\sigma)}, \sigma) - (\varepsilon - x)\right\|^2\right]. \tag{2}$$

At inference time, a random noise vector $x^{(1)} = \varepsilon$ is sampled from isotropic Gaussian noise and iteratively mapped to a clean sample $x^{(0)}$ over the course of K sampling steps, starting from a set of $K+1$ distinct boundary noise levels $\sigma_1, \ldots, \sigma_{K+1}$ with $\sigma_1 = 1, \sigma_{K+1} = 0$. With the standard deterministic Euler solver, the update step from noise level σ_i to σ_{i+1} is given by

$$x^{(\sigma_{i+1})} = x^{(\sigma_i)} + (\sigma_{i+1} - \sigma_i) f_\theta(x^{(\sigma_i)}, \sigma_i). \tag{3}$$

3.1 AUTOENCODER DESIGN

Temporal compression. Compressing multiple input frames to a single latent frame is a common approach used in autoencoders for video diffusion (Kong et al., 2024; Wan et al., 2025). Temporal compression is desirable for several reasons. It reduces the number of tokens used to encode any given video, increasing the viable training and inference context lengths. For autoregressive world models, temporal compression may have the potential to reduce inference-time error accumulation by reducing the time horizon of autoregressive rollouts.

One may expect that autoencoders with temporal compression are incompatible with low-latency interaction in world models, assuming that a full chunk of actions must be available before the next chunk can be generated. Supposing a temporal compression factor of $4\times$, if the next latent, z_t, corresponds to frames x_t through x_{t+3}, then action a_{t+2} needs to be available in order to generate the full chunk of frames. Without temporal compression, observation x_t would typically be generated after receiving action a_{t-1}, so there are 3 additional timesteps of latency before the agent receives observation x_t during interactive inference.

One way to prevent additional latency is by resampling the latent each time a new action is received. However, when implemented naively—by resampling the latent independently each time—frames within the same chunk may be inconsistent with each other, particularly in stochastic or partially observable environments. We find that a simple and effective mitigation is to initialize the diffusion process from the same noise sample for each new observation within a single chunk, i.e. starting the sampling process from the same $z_t^{(1)} = \varepsilon$ all four times that the latent z_t is generated while producing observations $x_t, \ldots, x_{t+3}$.

Only the final resampled latent within each chunk is passed as context for following chunks, and distinct initial noise samples are still used across chunks, so this resampling strategy does not affect the distribution of the sequence of latents observed by the world model itself. In other words, the intermediate generated latents within each chunk only affect the intermediate frames passed to the agent or downstream user, providing immediate observations, while only the final latents within each chunk are committed to the world model's context. Thus, the distribution of latents committed to context under this resampling strategy is identical to the distribution of latents observed under the naive strategy of only generating each latent after all actions within a chunk are received.

Generative decoding. For the sake of improving diffusability and reducing the memory requirements of world model training, we use a latent space with $4\times$ temporal compression and $32\times$ spatial compression. To enable high-fidelity reconstructions, we parameterize the decoder itself as a diffusion model, an approach previously shown to be effective for high-fidelity image autoencoders (Chen et al., 2025; OpenAI, 2023). Although each temporally compressed latent corresponds to multiple frames, since we already regenerate the latent for each new action during interactive inference, only a single frame decoded from the current latent is actually returned to the agent/user at any given time. So, our diffusion decoder is trained to only decode a single frame within the chunk rather than the entire frame chunk simultaneously, reducing requirements on decoder capacity. An additional input embedding, corresponding to the index of the current frame within the latent chunk (ranging from 0 to 3), is used to modulate the decoder's output via AdaRMSNorm (together with the embedding of the noise level, σ).

To avoid visible differences between generated low-level details from adjacent frames, we additionally condition our diffusion decoder on the pixels of the *previous* frame by concatenating them to the noised image input. This previous frame is provided as input both within chunks and across chunk boundaries. The combined previous frame and current noised frame inputs to the decoder are processed via a hierarchical encoder, architecturally similar to, but separate from, the encoder used to produce latents. Skip connections within the decoder pass representations at each resolution to the corresponding blocks in the hierarchical upsampling layers used to decode the latent. The overall architecture of our diffusion decoder is broadly similar to HDiT (Crowson et al., 2024), albeit with fewer layers, with convolutions instead of local neighborhood attention, and with small architectural

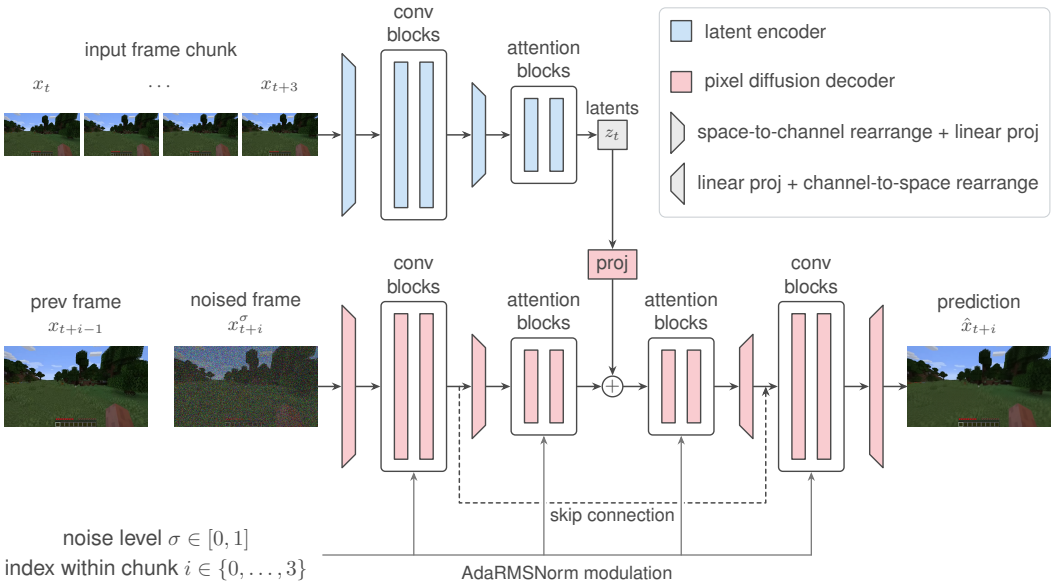


Figure 1: **Autoencoder architecture.** During training, the encoder deterministically produces a single latent frame z_t given four input frames $x_t, \ldots, x_{t+3}$. The decoder generates reconstructions via pixel diffusion, denoising frame x_{t+i} conditional on the latent z_t as well as the previous frame x_{t+i-1}. The architecture of our pixel diffusion decoder is similar to HDiT (Crowson et al., 2024), and includes its own encoder (distinct from the latent encoder) for taking as input the clean previous frame and noised current frame, concatenated channelwise. (For clarity, only two levels of the hierarchy are shown, but our autoencoder contains four.)

tweaks such as GGRoPE (Xiong, 2025) and 2D depthwise convolutions within the attention blocks (similar to those used in Cai et al. (2022)). In the decoder, latent embeddings are injected at the lowest level of the hierarchy via a linear projection. An overview of our autoencoder architecture is provided in Figure 1. Similar to Chen et al. (2025), we limit latent capacity by normalizing the latents to unit root-mean-square (RMS) magnitude along the channel dimension, then interpolating them with a random amount of Gaussian noise. In particular, we sample a latent noise level τ from the shifted uniform distribution with shift $s = 0.1$ and pass the noised latent, z_t^τ, as input to the decoder:

$$z_t^\tau = (1 - \tau) z_t + \tau \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I), \quad \tau = su/(1 + (s - 1)u), \quad u \sim \mathcal{U}(0, 1). \tag{4}$$

Assuming a compression factor of $4\times$, our encoder E produces a latent z_t as a function of frames x_t through x_{t+3}, and our diffusion decoder D produces a prediction for frame $t + i$ for $i \in \{0, \ldots, 3\}$ given the (corrupted) latent, the noised frame, the clean previous frame, and the noise level σ:

$$z_t = E(x_t, \ldots, x_{t+3}), \quad \hat{x}^0_{t+i} = D(z_t^\tau, x^\sigma_{t+i}, x_{t+i-1}, \sigma, i).$$

Similar to DiTo (Chen et al., 2025), the encoder and decoder are trained end-to-end, minimizing a combined MSE loss and LPIPS loss weighted by λ_{LPIPS}. Rather than computing latent per-channel normalization statistics post hoc, we simply add a regularization term to encourage the latents to have a per-channel mean of zero, denoted $\mathcal{L}_{\text{reg}}$ (details provided in Appendix A). The overall autoencoder objective can be written:

$$\mathcal{L}_{\text{autoencoder}} = \|\hat{x}^0_{t+i} - x_{t+i}\|_2^2 + \lambda_{\text{LPIPS}} \, \text{LPIPS}(\hat{x}^0_{t+i}, x_{t+i}) + \lambda_{\text{reg}} \, \mathcal{L}_{\text{reg}}. \tag{5}$$

3.2 World model design

Our world model is parameterized as an autoregressive latent video diffusion model with two components: a temporally causal, masked predictor and a spatial-only next-frame diffuser. We start from the idea that the majority of the information from past observations useful for generating the next frame is likely to be encoded in the conditional expectation of the next frame given those past observations. For example, in static scenes, generally each part of the scene either has or has not been observed in the past, thus making every pixel of the next observation frame either known with certainty or essentially unknown. Based on this intuition, we aim to structure our world model

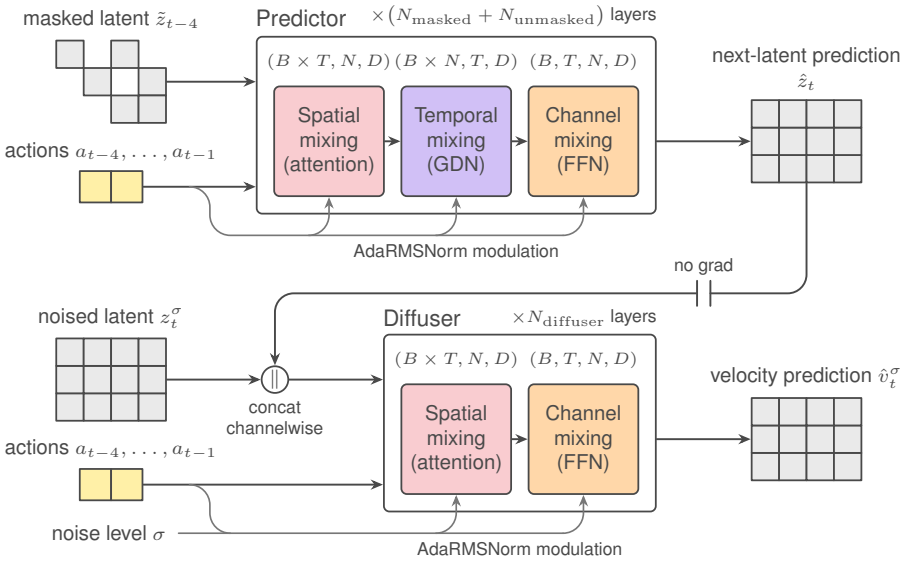


Figure 2: **World model architecture.** The predictor takes masked latents as input and autoregressively predicts the latent corresponding to the next chunk of frames. The diffuser conditions on this prediction (without gradients) and parameterizes a rectified flow over the latent. Action embeddings are appended as input tokens to both the predictor and diffuser, and also modulate intermediate layers via AdaRMSNorm (along with an embedding of the noise level for the diffuser).

architecture such that it serves to inpaint unknown portions of the next frame, conditional on known predicted portions of the next frame.

In practice, we implement this idea by using a separable architecture with some similarities to Bai et al. (2026). During training, the predictor takes as input a history of past latents and produces an action-conditional prediction of the next latent frame, trained via a typical MSE loss. The next-frame diffuser parameterizes a rectified flow, taking as input the detached prediction of the next latent, the noised next latent, and the current noise level and outputting a velocity prediction for a single latent frame. Unlike Bai et al. (2026), in our architecture, potentially noisy gradients from the diffusion model are not backpropagated to the causal predictor; instead, the predictor is trained with only the next-frame prediction objective. In principle, this can enable training the predictor and diffuser separately, but for simplicity we train the entire model in a single phase.

We apply VideoMAE-style (Tong et al., 2022) tube masking of the predictor's input, which serves to reduce FLOPs and memory usage during training by reducing the number of tokens processed by the predictor. Each latent contains 12×20 observation tokens from the autoencoder, which are flattened and concatenated with 16 tokens embedding the preceding action chunk, resulting in 256 tokens per latent frame. We randomly drop a subset of 128 observation tokens, consistent across all frames of a training video, before passing the remaining 128 tokens as input to the predictor. We re-insert copies of a learnable mask token at the missing positions before the last four layers of the predictor, in order to produce a next-latent prediction at every spatial location, trained via MSE:

$$\hat{z}_t = \text{Predictor}(z_{<t}, A_{<t}), \qquad \mathcal{L}_{\text{predictor}} = \|\hat{z}_t - z_t\|^2. \tag{6}$$

where $z_{<t}$ and $A_{<t}$ denote the history of latents and actions preceding z_t, respectively. We note that unlike Bai et al. (2026), we do not pass the entire predictor's hidden state to the diffusion model, since it can induce a train-test gap as a result of representations downstream of the mask tokens being present in the decoder's input during training, while no mask token embeddings are inserted at inference time. Providing only next-frame predictions to the diffuser mitigates this train-test gap, enabling efficient masked training without degrading downstream generation quality.

The architecture of our predictor broadly follows TRecViT (Patraucean et al., 2026), an efficient causal video transformer architecture which forgoes costly 3D full-sequence attention over all video tokens, instead alternating between spatial-only attention blocks operating independently over each frame, and temporal-only linear recurrent blocks operating independently over each spatial location. We use Gated DeltaNet (GDN) blocks (Yang et al., 2024a) for our linear recurrent layers, due to open-source availability of efficient training and inference kernels.

The diffuser takes as input this prediction (with gradients detached), concatenates it channelwise to a noised latent $z_t^\sigma = (1-\sigma)z_t + \sigma\varepsilon$, appends its own action token embeddings, and outputs a prediction for the velocity, $\hat{v}_t^\sigma$, minimizing an MSE loss:

$$\hat{v}_t^\sigma = \text{Diffuser}(z_t^\sigma, \hat{z}_t, A_t, \sigma), \qquad \mathcal{L}_{\text{diffuser}} = \|\hat{v}_t^\sigma - (\varepsilon - z_t)\|^2 \tag{7}$$

where A_t denotes the chunk of four actions preceding the last frame of z_t. Our diffuser contains no temporal mixing, alternating between only spatial attention layers and FFNs and parameterizing an independent flow for each latent frame. For both the predictor and diffuser, we include various minor architectural components inspired by the modded NanoGPT speedrun (Jordan et al., 2024), such as RMSNorm, ReLU^2 activations for FFNs, and learnable per-head scales for attention queries.

For action conditioning, we embed the discrete keyboard and click actions as binary features and use a Fourier-style feature embedding for continuous mouse actions. The embeddings for the four actions within a chunk are multiplied by learned channelwise weights (one for each of the four actions), passed through an FFN block, and used to modulate the intermediate representations of the model via AdaRMSNorm (Crowson et al., 2024). These embeddings are also concatenated to the input to both the predictor and the diffuser as additional action tokens, processed identically to the latent observation tokens and discarded before the final output projection. Further details on action tokenization and embedding are provided in Appendix B.

4 Experiments

We evaluate WDDQ on Minecraft, a complex video game with low-level actions and pixel observations. Minecraft features diverse, procedurally generated open worlds with a wide variety of items, structures, and monsters that the player can use and/or interact with, and is commonly used as a benchmark for training world models (Decart et al., 2024; Guo et al., 2025; Hafner et al., 2023; 2025). We use OpenAI's VPT dataset (Baker et al., 2022) featuring around 2500 hours of 20 FPS gameplay at a resolution of 360×640. Due to storage limitations, as well as the fact that it is infeasible to run even a single training epoch on the entire dataset within 50 GPU hours, we limit our experiments to only the higher-quality 8.x and 10.x versions of the dataset, comprising approximately 500 hours of gameplay.

4.1 Autoencoder results

For a fair comparison with prior work, we train a 128-channel autoencoder with an overall latent dimensionality similar to that of baseline VAEs. The autoencoder has two blocks per resolution and initially rearranges the image into 4×4 patches, then performs $2\times$ down/up-sampling between each of the four encoder/decoder levels, resulting in $32\times$ total spatial downsampling. From highest to lowest resolution, we use channel counts of 384, 512, 768, and 1024.

Our autoencoder is trained for less than 16 hours on a single GH200, using batches of 4 videos with 16 frames each. Many of our optimizer choices are inspired by the modded-NanoGPT speedrun (Jordan et al., 2024). We use the NorMuon optimizer (Li et al., 2025) with a learning rate of 0.001 for most matrix parameters, AdamW (Loshchilov & Hutter, 2017) with a learning rate of 0.001 for non-matrix and embedding parameters and a learning rate of 0.0001 for input/output projections. We use a (coupled) weight decay of 1e-5, corresponding to a decoupled, AdamW-style weight decay of 0.1 for matrix and input/output parameters and 0.01 otherwise, and train for a total of 100K steps. Full hyperparameters are listed in Appendix C.

Considering that our autoencoder is generative in nature, we focus on feature-based distributional metrics such as rFID and rKDD, the latter defined as the squared Maximum Mean Discrepancy (MMD) computed using features from DINOv2-ViT-B/14 with a polynomial kernel of degree 3, following recommendations from Stein et al. (2023). We evaluate rFID with 5K samples and rKDD with 1K samples, where each sample is a video clip of 5 frames treated as 5 independent images, resulting in 25K and 5K total images, respectively. For baselines, we compare to the open source ViT VAE for Minecraft released by Decart et al. (2024) as well as the HunyuanVideo-1.5 VAE (Kong et al., 2024), a modern video autoencoder trained on significantly more diverse internet video data.

As shown in Table 1, our 128-channel autoencoder (WDDQ-AE c128) has better reconstruction fidelity than the baselines at the same or higher overall compression ratio, validating our recipe for autoencoder training. However, we also find that there is a trade-off between fidelity and diffusability,

Table 1: **Autoencoder results.** We train high-fidelity (c128) and high-compression (c32) variants of our autoencoder. WDDQ-AE c128 outperforms prior video autoencoders in terms of reconstruction fidelity at a similar overall compression ratio, while WDDQ-AE c32 provides a more diffusable latent space for efficient world-model training with a minor fidelity degradation.

Autoencoder	Latent design			Reconstruction	
	Downsample	Channels	Compression ↑	rFID@5K ↓	rKDD@1K ↓
Open Oasis VAE	1×20×20	16	75×	2.198	0.103
HunyuanVideo1.5 VAE	4×16×16	32	96×	3.952	0.256
WDDQ-AE c128	4×32×32	128	96×	**1.197**	**0.046**
WDDQ-AE c32	4×32×32	32	**384**×	2.485	0.114

in which latents with higher channel counts are more difficult to diffuse from the perspective of the world model. In other words, the approximately $96\times$ compression ratio used by existing works may actually be suboptimal for world models of this scale. Indeed, we show that a 32-channel variant of our autoencoder results in higher downstream generation quality for our world model in Section 4.3.

4.2 World model results

We train our world model for 200K steps with a batch size of 1 and a sequence length of 128 latent frames, corresponding to around 25 seconds of context. This requires less than 24 hours on a single GH200. We train the world model using the 32-channel autoencoder (WDDQ-AE c32) due to its improved modelability, and use VideoMAE-style tube masking (Tong et al., 2022) with a masking ratio of 50%. We use a predictor with a depth of 28 and a diffuser with a depth of 16. We use the same optimizer hyperparameters as for autoencoder training.

To evaluate generation quality, we randomly sample 500 pairs of starting frames and corresponding action sequences from a held-out evaluation split of the VPT dataset, and generate action-conditional rollouts 128 timesteps long. We compare our model to Open Oasis (Decart et al., 2024), as well as MineWorld (Guo et al., 2025), using the default sampling hyperparameters provided in the open-source implementations for both baselines. For Oasis, we perform 10 denoising steps with a stabilization noise level of 15, and for fairness, we optimize Oasis's sampling implementation to cache KVs after the first denoising step, precompute RoPE rotation matrices, and integrate with `torch.compile` without graph breaks, significantly improving throughput (without affecting model outputs) before measuring FPS. For MineWorld, we report generation quality metrics with diagonal decoding disabled (maximizing fidelity) and report FPS with diagonal decoding enabled (maximizing throughput). For WDDQ, we sample with 10 Euler steps for the diffusion model and 8 steps for the autoencoder's diffusion decoder. We measure generation quality using Fréchet Video Distance (FVD) (Unterthiner et al., 2018) by dividing each sampled rollout into eight chunks of 16 frames each, resulting in 4000 video clips. All reference and generated clips are resized to 224×224 before computing FVD, following standard practice. We measure FPS based on end-to-end throughput, including autoencoder decoding latency.

Generation quality vs. throughput. As shown in Table 2, WDDQ achieves an FVD of 63.4, compared with 104.9 for MineWorld and 365.1 for Open Oasis, corresponding to reductions of 39.6% and 82.6%, respectively. WDDQ generates at the dataset's native 640×360 resolution with an end-to-end throughput of 8.81 FPS, compared with 9.60 FPS for Oasis and 3.59 FPS for MineWorld. MineWorld operates at a lower resolution of 384×224. Thus, WDDQ substantially improves the reported generation quality while retaining throughput close to Oasis and exceeding MineWorld's throughput.

Action controllability. To evaluate controllability, i.e., the faithfulness of the generations to provided action inputs, we follow the protocol in Guo et al. (2025). Generated rollouts are labeled with actions using the inverse dynamics model (IDM) from VPT (Baker et al., 2022), which are then compared with the ground-truth input actions to compute macro-averaged F1 scores for each discrete action group and L1 errors for mouse actions after VPT-style binning. Table 3 shows that WDDQ achieves the best F1 scores in all discrete categories, aside from drop actions, where it ties with the baselines. WDDQ also reduces mouse L1 error from 0.534 to 0.261, a 51.1% reduction relative to MineWorld.

Qualitative comparisons. Figure 3 illustrates qualitative model differences in forest and cave rollouts conditioned on identical starting frames and action sequences. Oasis exhibits error accumulation,

Table 2: **World model results.** We evaluate generation quality via FVD (Unterthiner et al., 2018), sampling 500 autoregressive rollouts of 128 frames conditioned on starting frames and action sequences from the VPT (Baker et al., 2022) dataset. WDDQ achieves the best generation quality while operating at the native 640×360 resolution.

Model	Params (AE)	Params (WM)	Resolution ↑	FPS ↑	FVD ↓
Open Oasis	229M	608M	640×360	**9.60**	365.1
MineWorld	146M	1.229B	384×224	3.59	104.9
WDDQ	138M	696M	640×360	8.81	**63.4**

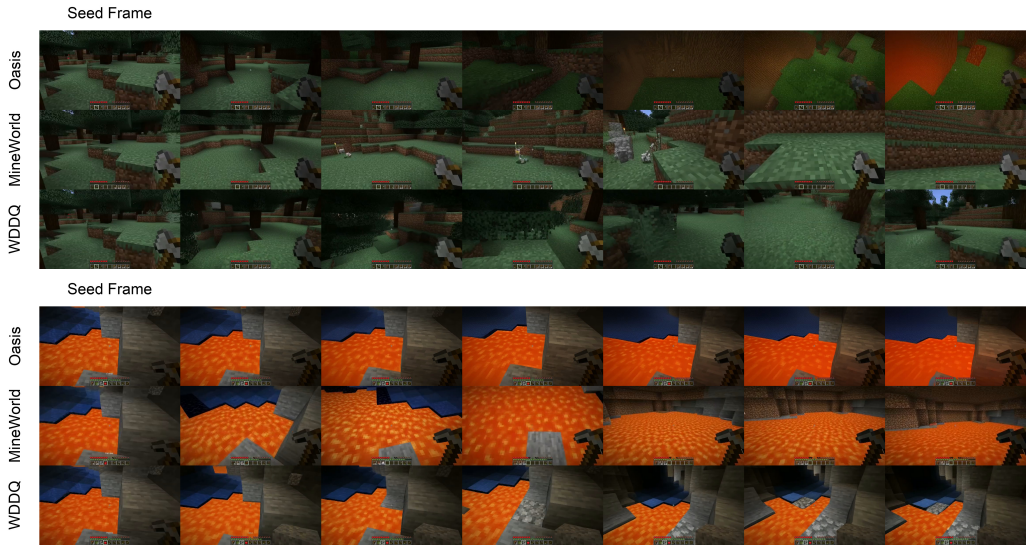


Figure 3: **Example rollouts.** Each row shows an autoregressive rollout conditioned on identical starting frames and corresponding sequences of input actions taken from the held-out validation split of our dataset, corresponding to a forest and cave scene, respectively. **Oasis** demonstrates significant *error accumulation*, while **MineWorld** appears to *hallucinate random changes* to the contents of the player's hotbar. In the cave example, MineWorld generates a camera movement that fails to respect the input mouse action, while **WDDQ** is the only model which correctly models the player placing a block.

while MineWorld introduces changes to the player's hotbar. In the cave example, MineWorld's camera movement does not follow the input mouse action, and WDDQ is the only evaluated model that correctly depicts the player placing a block. These examples illustrate both visual consistency and action following beyond the aggregate metrics. Additional rollout examples are provided in Appendix E.

Training cost. The two WDDQ training stages require less than 16 GH200 hours for the autoencoder and less than 24 GH200 hours for the world model. Their combined training time is therefore less than 40 GH200 hours, within the 50-GPU-hour budget. Both stages are trained from scratch, with a pretrained network used only to compute the autoencoder's LPIPS loss.

Although MineWorld (Guo et al., 2025) does not report a wall-clock training time, we can estimate its overall training FLOPs based on the reported model architecture and training configuration. MineWorld trains a 1.2B-parameter autoregressive transformer for 200K steps with sequences of 5,376 tokens on 32 A100 GPUs. Because its global batch size is not reported, we conservatively assume that MineWorld was trained on only one sequence per batch per GPU, which corresponds to around 34.4B training tokens. This gives a lower bound of 248 EFLOPs for training the MineWorld world model, based on the standard $6ND$ estimate. Unfortunately, MineWorld does not provide training details for its VQGAN tokenizer.

For WDDQ, the same $6ND$ estimate gives around 18.4 EFLOPs, taking predictor masking into account. (We use separate $6ND$ estimates for the unmasked and masked portions of the model, with D halved for the masked portion.) Thus, even under assumptions favorable to MineWorld, WDDQ's world model training requires less than 8% as many FLOPs as that of MineWorld.

Table 3: **Action controllability results.** We report action controllability metrics following the evaluation protocol from MineWorld (Guo et al., 2025), based on labels produced by the VPT IDM (Baker et al., 2022). WDDQ achieves the best controllability for mouse actions and all discrete action groups, aside from drop actions, which rarely appear in the dataset.

Model	Discrete Actions (F1 Score) ↑							Mouse (L1) ↓
	Fwd/Back	Left/Right	Sprint/Sneak	Use	Attack	Jump	Drop	
Open Oasis	0.615	0.584	0.598	0.610	0.607	0.711	0.500	0.597
MineWorld	0.845	0.825	0.816	0.684	0.811	0.869	0.500	0.534
WDDQ	**0.895**	**0.852**	**0.820**	**0.785**	**0.933**	**0.896**	0.500	**0.261**

Training details for Oasis (Decart et al., 2024) are not publicly available. DreamerV4 (Hafner et al., 2025) did not release model checkpoints, and there is no standardized evaluation protocol for the metrics they report. We therefore cannot reliably compare training costs with Oasis or directly evaluate DreamerV4 under our protocol. However, DreamerV4 does report training on 256 to 1024 TPUv5p, indicating that its training recipe also required significant compute.

4.3 Ablations

We ablate various components of our world model architecture, showing results in Table 4. All models are trained with the same number of steps, with the dataset and optimizer held constant. First, we train a variant of our model without tube masking in the predictor. We reduce the number of predictor layers from 28 to 24 in order to prevent out-of-memory errors during training. This variant performs a full forward pass with all tokens during training, significantly increasing overall training FLOPs. We find that this variant achieves similar downstream performance despite higher training compute, validating predictor tube masking as an effective method for improving training efficiency. Second, we train a variant that uses a single, unified transformer with 24 layers instead of separate predictor and diffuser components. This variant is significantly more expensive than the separable architecture during inference, since each denoising step requires a forward pass through the entire model rather than only the diffuser beyond the first step. We find that this variant also has poor generation quality. Third, we compare our model to a variant trained using the 128-channel autoencoder. We find that this variant results in worse overall generation quality than the 32-channel version, likely due to higher-dimensional latents being more difficult to model accurately.

Table 4: **World model architecture ablations.** We compare generation quality and macro-averaged action controllability for WDDQ and variants without predictor masking, with a single unified transformer, and with a 128-channel rather than 32-channel autoencoder. We also report estimated total training FLOPs and inference FLOPs per generated frame using 10 Euler steps.

Variant	Train FLOPs ↓	Infer FLOPs ↓	FVD ↓	Action F1 ↑
WDDQ	1.85×10^{19}	1.27×10^{12}	63.4	0.812
No predictor tube masking	2.37×10^{19}	1.24×10^{12}	64.9	0.819
Non-separable architecture	1.58×10^{19}	2.06×10^{12}	87.0	0.795
c128 autoencoder instead of c32	1.85×10^{19}	1.27×10^{12}	85.6	0.800

5 Conclusion

In this work, we introduce a computationally efficient recipe for training world models from scratch. By leveraging a highly compressed latent space with a high-fidelity diffusion decoder, as well as an efficient world model architecture based on masked next-frame prediction and prediction-conditioned diffusion, we train our world model, WDDQ, in under 50 GPU hours while outperforming open-weight Minecraft world models in terms of generation quality and action controllability. Of course, generations for world models of this scale remain imperfect. At larger scales, training on higher-dimensional latent spaces could lead to improved visual fidelity. Future work could also explore improved methods for maintaining consistency across long temporal horizons. Regardless, this work demonstrates that expressive diffusion world models can be trained at a smaller scale than demonstrated in prior work, and we hope that insights from our training recipe can be broadly useful for improving the efficiency of world model training in general.

AI USE STATEMENT

In this work, we used generative AI tools for reviewing and editing code. All AI-written code was checked manually for correctness. We also used AI to identify spelling and grammatical errors and provide feedback on the paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

In addition to the method description in the main paper, we provide hyperparameter tables in Appendix C and additional implementation details for the autoencoder architecture in Appendix A, action embeddings in Appendix B, and dataset details in Appendix D. Training code and model checkpoints will be released on GitHub at a future date.

ETHICS STATEMENT

WDDQ aims to make video world-model research more accessible by reducing the computation required for training from scratch. Our experiments are confined to Minecraft and use existing gameplay videos publicly released as part of the VPT contractor dataset (Baker et al., 2022).

Our experiments do not establish suitability for real-world or safety-critical deployment. Our evaluation is limited to virtual environments; extending the training recipe to real-world video would require additional consideration of privacy, consent, and potential misuse, including the creation of misleading synthetic content.

REFERENCES

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in Atari. *Advances in Neural Information Processing Systems*, 37:58757–58791, 2024.

Xingjian Bai, Guande He, Zhengqi Li, Eli Shechtman, Xun Huang, and Zongze Wu. Causality in video diffusers is separable from denoising. *arXiv preprint arXiv:2602.10095*, 2026.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (VPT): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Han Cai, Junyan Li, Muyan Hu, Chuang Gan, and Song Han. Efficientvit: Multi-scale linear attention for high-resolution dense prediction. *arXiv preprint arXiv:2205.14756*, 2022.

Yinbo Chen, Rohit Girdhar, Xiaolong Wang, Sai Saketh Rambhatla, and Ishan Misra. Diffusion autoencoders are scalable image tokenizers. *arXiv preprint arXiv:2501.18593*, 2025.

Katherine Crowson, Stefan Andreas Baumann, Alex Birch, Tanishq Mathew Abraham, Daniel Z Kaplan, and Enrico Shippole. Scalable high-resolution pixel-space image synthesis with hourglass diffusion transformers. In *Forty-first International Conference on Machine Learning*, 2024.

Decart, Julian Quevedo, Quinn McIntyre, Spruce Campbell, Xinlei Chen, and Robert Wachen. Oasis: A universe in a transformer, 2024. URL `https://oasis-model.github.io/`.

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

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

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

A Hojel and R Seid. Lucid v1: Real-time latent world models. *Int J Cur Res Sci Eng Tech*, 7(4): 116–119, 2024.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024. URL https://github.com/KellerJordan/modded-nanogpt.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Felix Krause, Timy Phan, Ming Gui, Stefan Andreas Baumann, Vincent Tao Hu, and Björn Ommer. TREAD: Token routing for efficient architecture-agnostic diffusion training. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15703–15713. IEEE, 2025.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. NorMuon: Making muon more efficient and scalable. *arXiv preprint arXiv:2510.05491*, 2025.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

OpenAI. DALL·E 3 System Card. https://openai.com/index/dall-e-3-system-card/, 2023.

Viorica Patraucean, Xu Owen He, Joseph Heyward, Chuhan Zhang, Mehdi S. M. Sajjadi, George-Cristian Muraru, Artem Zholus, Mahdi Karami, Ross Goroshin, Yutian Chen, Simon Osindero, Joao Carreira, and Razvan Pascanu. TRecViT: A recurrent video transformer. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

Ryan Po, Yotam Nitzan, Richard Zhang, Berlin Chen, Tri Dao, Eli Shechtman, Gordon Wetzstein, and Xun Huang. Long-context state-space video world models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8733–8744, 2025.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

George Stein, Jesse Cresswell, Rasa Hosseinzadeh, Yi Sui, Brendan Ross, Valentin Villecroze, Zhaoyan Liu, Anthony L Caterini, Eric Taylor, and Gabriel Loaiza-Ganem. Exposing flaws of generative model evaluation metrics and their unfair treatment of diffusion models. *Advances in Neural Information Processing Systems*, 36:3732–3784, 2023.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 3645–3650, 2019.

Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022.

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

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. *arXiv preprint arXiv:2408.14837*, 2024.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Meredith Whittaker. The steep cost of capture. *Interactions*, 28(6):50–55, 2021.

Jerry Xiong. On n-dimensional rotary positional embeddings, 2025. URL `https://jerryxio.ng/posts/nd-rope/`.

Wilson Yan, Danijar Hafner, Stephen James, and Pieter Abbeel. Temporally consistent transformers for video generation. In *International Conference on Machine Learning*, pp. 39062–39098. PMLR, 2023.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024a.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. CogVideoX: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024b.

Hongkai Zheng, Weili Nie, Arash Vahdat, and Anima Anandkumar. Fast training of diffusion models with masked transformers. *arXiv preprint arXiv:2306.09305*, 2023.

Zangwei Zheng, Xiangyu Peng, Yuxuan Lou, Chenhui Shen, Tom Young, Xinying Guo, Binluo Wang, Hang Xu, Hongxin Liu, Mingyan Jiang, et al. Open-Sora 2.0: Training a commercial-level video generation model in $200 k. *arXiv preprint arXiv:2503.09642*, 2025.

A AUTOENCODER DETAILS

As described in Section 3.1, our autoencoder involves a chunkwise, hierarchical encoder which takes as input a chunk of four frames of shape $(4, 360, 640, 3)$ and outputs a latent of shape $(12, 20, C)$ where C is either 32 or 128, depending on the desired trade-off between latent diffusability and reconstruction quality. Initially, we pad the input (using the "symmetric" strategy, which pads with a reflection of the image including a repeat of the value on the edge) to $(384, 640)$ and perform a $4 \times 4 \times 4$ patchify (spatially and temporally). Between each level of the hierarchy, we downsample by applying a 2×2 space-to-channel rearrange and linearly projecting the result (equivalently, performing a 1×1 convolution) to the dimensionality of the next level of the hierarchy. (We note that this can be equivalently implemented via a convolution with a 2×2 kernel and 2×2 stride, without a bias.) From highest to lowest resolution, the levels of our hierarchy have residual stream dimensionalities of 384, 512, 768, and 1024.

Our decoder, a pixel diffusion model, takes as input the $(360, 640, 3)$ current noised frame (padded to 640×384) and conditions on the $(12, 20, C)$ latent, the $(360, 640, 3)$ previous frame, the current noise level $\sigma \in [0, 1]$, as well as the index $i \in \{0, 1, 2, 3\}$ of the current frame within the chunk, to produce a $(360, 640, 3)$ prediction. Our decoder itself has an encoder-decoder architecture which is broadly similar to HDiT (Crowson et al., 2024). To avoid confusion, we refer to the encoder that produces latents as the 'latent encoder', and the encoder/decoder halves of the pixel diffusion decoder itself as the 'decoder's downsampling branch' and the 'decoder's upsampling branch', respectively. The decoder's downsampling branch is nearly architecturally identical to the latent encoder. There are only two differences: the first is that the initial linear projection following the latent encoder's patchify has an input dimensionality of 192 (corresponding to the 3 RGB channels of the $4 \times 4 \times 4$ patch of the input chunk) while the patchify of the decoder's downsampling branch has half the input dimensionality, at 96 (corresponding to the concatenation of the 3 RGB channels of the 4×4 patch of the current noised frame and the 3 RGB channels of the 4×4 patch of the previous frame). The second difference is that the RMSNorm layers at the beginning of each block use standard learnable affine weights in the latent encoder, but are replaced with AdaRMSNorm blocks in the decoder's downsampling branch, where they are adaptively modulated depending on σ and i. The decoder's upsampling branch is essentially the exact inverse of the downsampling branch. HDiT-style (Crowson et al., 2024) skip connections with learned linear interpolation weights pass the hidden states at the end of each level of the decoder's downsampling branch to the corresponding level of the decoder's upsampling branch.

A common approach to normalize the latent space of an autoencoder for diffusion model training is to compute the per-channel mean and standard deviation of the latents over the dataset, and normalize/denormalize the inputs/outputs of the diffusion model, respectively. For simplicity, we add a regularization term, $\mathcal{L}_{\text{reg}}$, which directly encourages the latents to be mean 0, and omit separate normalization for the diffusion model entirely. This has some similarities to the typical gaussian-KL loss applied during VAE training, though unlike typical KL regularization, we would like to enforce zero per-channel mean over the dataset, rather than encourage the latents of each individual sample to be near zero. To achieve this, we simply track a per-channel EMA of the latents, $\mu_z \in \mathbb{R}^C$, and we add a loss term penalizing the dot product of μ_z with the latents of the current video batch, clamped to be nonnegative: $\mathcal{L}_{\text{reg}} = [\langle \mu_z, z \rangle]_+$ where the inner product is an average over all (B, T, H, W) latents. After computing the loss, the EMA μ_z is updated, $\mu_z = (1 - \gamma_{\text{reg}})\mu_z + \gamma_{\text{reg}} \text{ mean}(z)$, where once again, the mean is taken over all (B, T, H, W) elements of a batch.

B ACTION EMBEDDING DETAILS

We tokenize each Minecraft action into an 18-dimensional binary 'keyboard' action, a 2-dimensional binary 'click' action, and a 2-dimensional continuous 'mouse' action. The 18 keyboard dimensions correspond to the 'forward', 'back', 'left', 'right', 'sprint', 'sneak', 'jump', 'inventory', and 'ESC' actions, following the terminology used in VPT, along with 9 hotbar actions. We exclude remaining keyboard actions, such as 'drop', 'pickItem', and 'swapHands', since they appear extremely rarely in the training dataset. The 'click' actions correspond to 'attack' and 'use'. The keyboard and click actions are concatenated and encoded as a binary vector of length 20, which we pass through a simple linear projection. We embed the continuous mouse action (also known as the 'camera' action in VPT) using Fourier features. First, we apply symlog scaling (Hafner et al., 2023) to the magnitude of the 2D mouse action (rather than applying symlog to each dimension independently, which would not be

a rotationally symmetric operation). Next, we take the dot product of the scaled mouse action with a fixed set of 256 frequency vectors in $\mathbb{R}^2$. We use a deterministic set of frequency vectors, where the ith vector, $i \in \{0, 1, \ldots, 255\}$, has magnitude $\omega_i = \omega_{\text{min}}(\omega_{\text{max}}/\omega_{\text{min}})^{i/255}$ and direction $i\,\Delta\theta$, where $\Delta\theta$ is a fixed angular spacing. We use $\omega_{\text{min}} = 0.20, \omega_{\text{max}} = 20.0$, and take $\Delta\theta$ to be π/ϕ where ϕ is the golden ratio, following recommendations in Xiong (2025). We take the sine and cosine of each of the dot products and concatenate them, resulting in a 512-dimensional Fourier embedding, which we then pass through a linear projection.

Since the latent space of our autoencoder is temporally compressed, each latent corresponds to four timesteps of actions. For AdaRMSNorm modulation, we multiply each of the keyboard and mouse action embeddings by a learned per-channel weight, distinct for each of the four timesteps, before averaging to obtain a single vector for the entire latent frame. We also perform action conditioning by appending action tokens to the spatial input tokens. These action tokens are also constructed by multiplying each of the four keyboard and mouse action embeddings with separate per-channel weights, distinct for each timestep. Our autoencoder outputs 240 spatial tokens, so we produce 16 action tokens for the sake of passing an even 256 tokens as input to the world model. Thus, we use 2 learnable weights for each of the four button actions and four mouse actions, mapping the 8 embeddings to 16 tokens.

C Hyperparameters

C.1 Autoencoder hyperparameters

token counts	[(96, 160), (48, 80), (24, 40), (12, 20)]
# blocks	[2, 2, 2, 2]
channels	[384, 512, 768, 1024]
block types	[conv, conv, attn, attn]
MLP expand	2
attn head dim	64
GGRoPE $\omega_{\text{min}}, \omega_{\text{max}}$	0.20, 20.0
GGRoPE prop. zero freqs	0.125
latent noise shift s	0.1
LPIPS weight λ_{LPIPS}	0.05
Regularizer weight λ_{reg}	0.001
Regularizer EMA γ_{reg}	0.01
training steps	100,000
batch size	4
sequence length	(1 + 12) RGB / 4 latent frames
NorMuon lr	0.001
AdamW lr (embeddings)	0.001
AdamW lr (scalars)	0.001
AdamW lr (in/out projs)	0.0001
NorMuon β_1, β_2	0.95, 0.99
AdamW β_1, β_2	0.95, 0.99
lr schedule	constant + linear cooldown
lr warmup	none
lr cooldown start	50,000
weight decay (coupled)	1e-5

C.2 WORLD MODEL HYPERPARAMETERS

predictor masked # blocks N_{masked}	24
predictor unmasked # blocks N_{unmasked}	4
diffuser # blocks N_{diffuser}	16
predictor masking ratio	50%
MLP expand	4
attn head dim	64
GDN head dim	128
GGRoPE $\omega_{\text{min}}, \omega_{\text{max}}$	0.20, 20.0
GGRoPE prop. zero freqs	0.25
training steps	200,000
batch size	1
sequence length	(1 + 508) RGB / 128 latent frames
tokens per frame N	256 (240 latent + 16 action)
NorMuon lr	0.001
AdamW lr (embeddings)	0.001
AdamW lr (scalars)	0.001
AdamW lr (in/out projs)	0.0001
NorMuon β_1, β_2	0.95, 0.99
AdamW β_1, β_2	0.95, 0.99
lr schedule	constant + linear cooldown
lr warmup	none
lr cooldown start	100,000
weight decay (coupled)	1e-5

D DATASET DETAILS

Our dataset is based on the open-source implementation in VPT's (Baker et al., 2022) GitHub repository. In our implementation, we include several minor bug fixes and modifications related to dataset preprocessing: (1) The VPT implementation rounded actions to the nearest integer as a result of initializing null actions during tokenization with an integer-dtype NumPy array, resulting in a small amount of precision loss. We store mouse actions as floating-point values, removing this rounding. (2) We include runs of null actions up to three timesteps long rather than filtering out all null actions. This ensures that null action inputs during inference are in-distribution, while still filtering out most completely static data. (3) The VPT implementation's detection of hotbar actions with scroll wheel inputs resulted in actions delayed by one timestep. These types of hotbar actions were also not properly detected as non-null actions, which may occasionally result in them being dropped from the dataset due to null action filtering. We add both types of hotbar actions to the non-null detection and fix the timestep offset for scroll actions. (4) Many episodes in which the contractor starts a new world are almost completely black for a few frames at the beginning of the episode. To reduce their occurrence in our training data, we simply remove the first 10 frames from any episode in which the 'serverTick' field indicates that the episode is a new world. (5) We concatenate adjacent videos from the same 'contractor session' (based on the session ID information in the episode paths) into contiguous trajectories, and drop all trajectories shorter than 512 RGB frames.

E ADDITIONAL QUALITATIVE EXAMPLES

Figures 4 and 5 compare additional 128-frame rollouts from Oasis, MineWorld, and WDDQ. For each example, all three models receive the same starting frame and action sequence sampled randomly from the held-out validation split of our dataset.

Figure 4 illustrates differences in the evolution of scene appearance across the generated rollouts. In the first and fourth examples, Oasis fails to accurately model the dynamics of the player breaking the log, whereas MineWorld and WDDQ accurately model the effects of the player's actions on the surrounding environment. In the second example, Oasis fails to preserve much of the cave's visible structure, producing largely featureless gray or orange regions. MineWorld and WDDQ both

preserve the visual coherence of the blocks and items in the scene, though WDDQ demonstrates higher diversity in the generated rollouts, generating a view of a lava lake as the player turns around.

Figure 5 provides further comparisons across different environments and interaction types. The outputs of Oasis tend to collapse quickly, replacing much of the visible scene with nearly uniform red or orange regions. WDDQ and MineWorld retain recognizable walls, doorways, and terrain throughout the displayed frames, though MineWorld lacks the capability of generating GUI elements such as inventories, while hallucinating seemingly random changes to the contents of the player's hotbar in many samples.

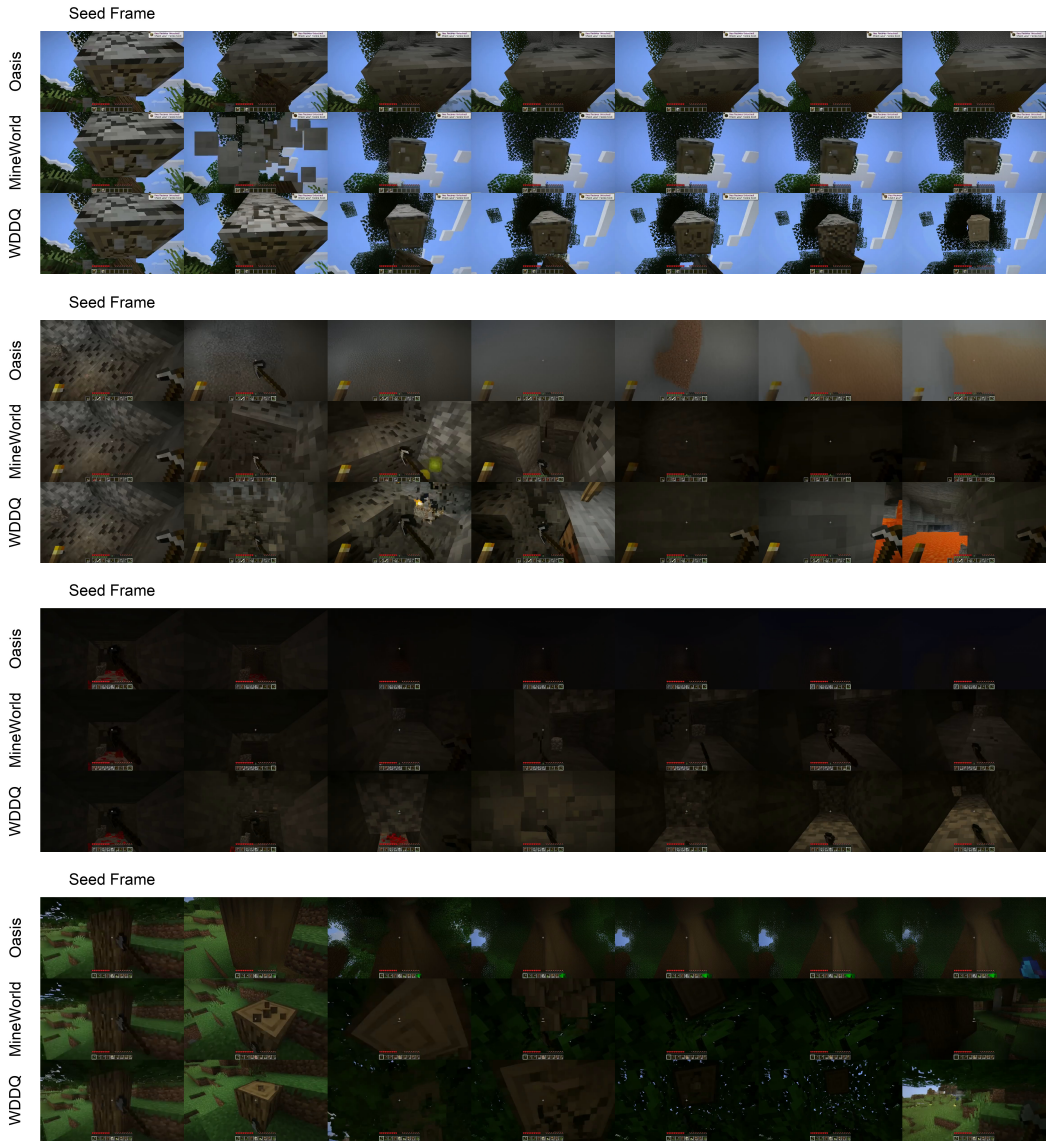


Figure 4: **Additional rollouts in forest and cave scenes.** From top to bottom, the four examples show a scene with birch trees, a torch-lit cave, a dark underground passage, and an scene with oak trees. From top to bottom, the rows of each sample correspond to Oasis, MineWorld, and WDDQ. The first column shows the shared seed frame, and subsequent columns show uniformly spaced frames generated via autoregressive rollout. Within each example, all generations are conditional on the same input action sequence.

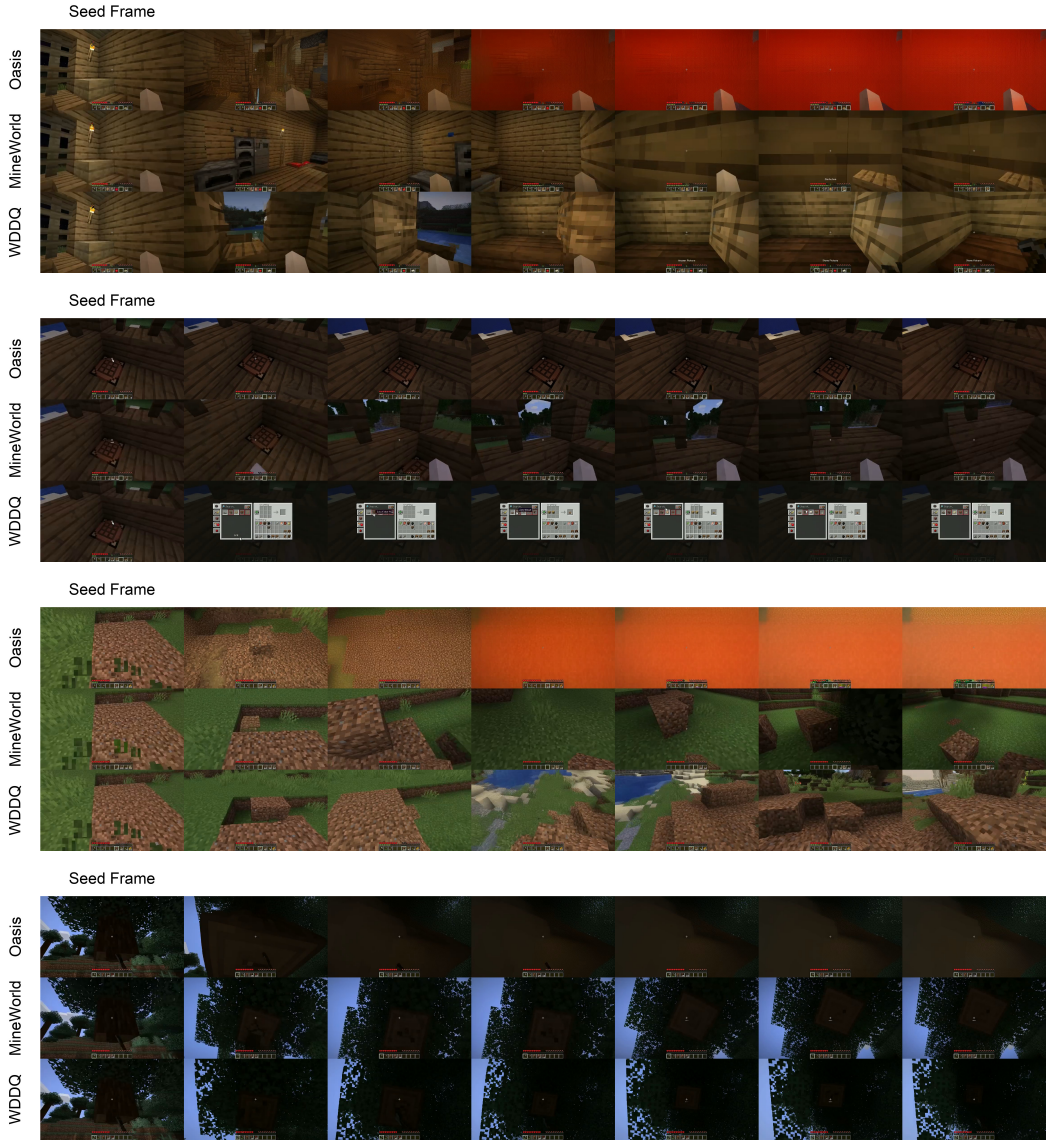


Figure 5: **Additional rollouts across indoor and outdoor environments.** From top to bottom, the four examples show a wooden interior, a scene with a crafting table, outdoor terrain, and a tree canopy. From top to bottom, the rows of each sample correspond to Oasis, MineWorld, and WDDQ. The first column shows the shared seed frame, and subsequent columns show uniformly spaced frames generated via autoregressive rollout. Within each example, all generations are conditional on the same input action sequence.