Solaris: Building a Multiplayer Video World Model in Minecraft

Oscar Michel[†*] **Georgy Savva**[†*] **Daohan Lu**[*] **Suppakit Waiwitlikhit**
Timothy Meehan **Dhairya Mishra** **Egor Gikalo**
Srivats Poddar **Jack Lu** **Saining Xie**
New York University

Abstract

Existing action-conditioned video generation models (video world models) are limited to single-agent perspectives, failing to capture the multi-agent interactions of real-world environments. We introduce Solaris, a multiplayer video world model that simulates consistent multi-view observations. We develop a multiplayer data system designed for robust, continuous, and automated data collection on video games such as Minecraft. Unlike prior platforms built for single-player settings, our system supports coordinated multi-agent interaction and synchronized videos + actions capture. Using this system, we collect 12.64 million multiplayer frames and propose an evaluation framework for multiplayer movement, co-movement, grounding, building, and view consistency. We train Solaris using a staged pipeline that progressively transitions from single-player to multiplayer modeling, combining bidirectional, causal, and Self Forcing training. In the final stage, we introduce *Checkpointed Self Forcing*, a memory-efficient Self Forcing variant that enables a longer-horizon teacher. Results show our architecture and training design outperform existing baselines. We also demonstrate that scaling beyond two players is possible by training a 3-player proof-of-concept model and simulating up to 16 players on our hardware with efficient attention. Through open-sourcing our system and models, we hope to lay the groundwork for a new generation of multi-agent world models.

Project website: `https://solaris-wm.github.io/`

1 Introduction

Video world models generate future agent observations from past observations and actions, making them useful for synthetic training [2, 64], inference-time planning [71, 4], and policy learning and evaluation [17, 33, 52]. Yet current world models typically simulate only one agent's observations. In multi-agent settings, a useful model must generate the views of all agents sharing the environment.

Toward this end, we study the creation of multiplayer video world models trained on multiplayer data collected from Minecraft. Modeling the perspectives of multiple players at once is significantly more challenging than single-player modeling. Generated observations need to be consistent not only across time but also across agent perspectives. Actions taken by one agent, such as movement or block placement, must be simultaneously and accurately reflected in the views of all other agents.

Minecraft serves as an ideal testbed for multi-agent modeling. Its unbounded 3D worlds rigorously test perspective consistency, occlusion handling, and spatial memory. Its dynamic and malleable environment tests a model's ability to keep track of alterations that occur over time. Furthermore,

[†]Project lead; order determined by coin toss. [*]Equal technical contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

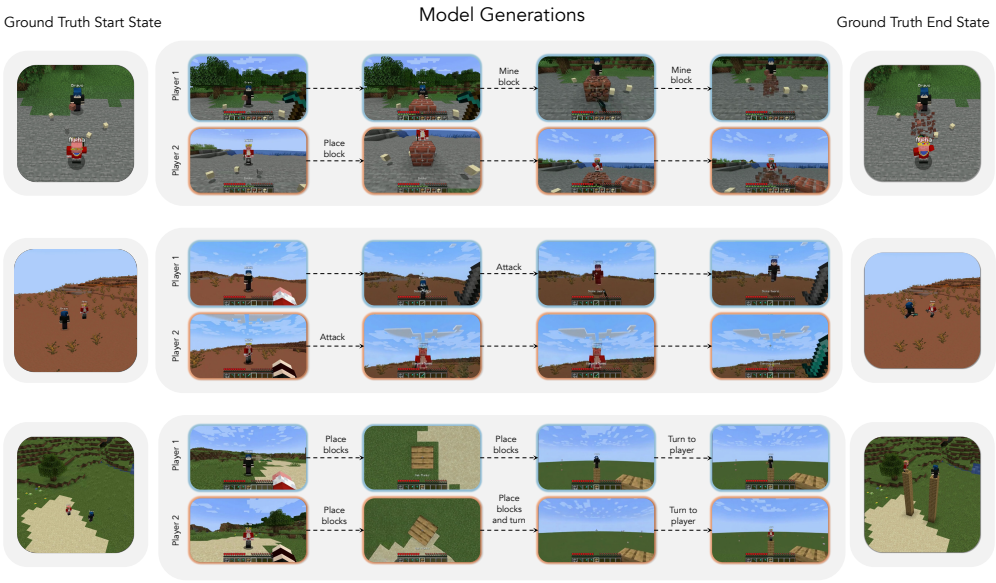

Figure 1: **Selected samples from our model.** Our model takes in starting frames from each player as input and generates action-conditioned videos. The action descriptions shown here are summaries of the fine-grained action sequences given to the model that span many frames. The third-person ground truth visualizations are not given to the model.

environmental stochasticity (e.g., mobs, weather) forces models to disentangle alterations caused by environmental randomness from those caused by controllable agents. And through its complex system of building, digging, and crafting, Minecraft allows for endless complexity. Lastly, its procedurally generated terrain naturally diversifies the environment.

Prior to this work, there were no existing publicly available systems for simulating multiplayer Minecraft gameplay, motivating us to build SolarisEngine: a system for collecting gameplay from pre-programmed bots. In SolarisEngine, our bots engage in non-trivial and realistic gameplay, including mining, attacking, building, and navigating terrains. Our system allows us to collect millions of frames in a matter of hours, and its modular design makes it easy to extend to new agent behaviors. We used this data engine to generate a multiplayer training dataset of 12.64 M frames and an evaluation set testing multiplayer movement, co-movement, grounding, building, and view consistency.

Our model, Solaris, adapts a single-agent pretrained video DiT to simulate multiple perspectives with minimal architectural changes. Given past player observations and actions, it autoregressively generates future observations; examples appear in Fig. 1. We also show that scaling beyond two players is possible: we train a 3-player proof-of-concept model and simulate up to 16 players on our hardware with an efficient attention implementation.

To improve long-horizon autoregressive generations, we adopt the Self Forcing [32] paradigm. We extend the setting of Self Forcing to allow the student to benefit from a long-context teacher. Naively applying Self Forcing with sliding-window generation leads to excessive memory usage, a problem we mitigate with a novel technique called Checkpointed Self Forcing analogous to gradient checkpointing. We also make simplifications to the initialization of the student model, finding that the initialization method of CausVid [69] can be replaced by simply finetuning with causal masking.

In summary, we present Solaris, a multi-agent video world model capable of simulating consistent multiplayer perspectives in Minecraft. We contribute a scalable data system, SolarisEngine, a large-scale training dataset of multiplayer gameplay, an evaluation system, and a new architecture that adapts a pre-trained video DiT for multiplayer videos. Finally, we introduce Checkpointed Self Forcing, a memory-efficient extension of Self Forcing, enabling effective autoregressive training, and demonstrate that scaling to more players is possible.

2 Related Work

World Models and Video World Models. A world model takes in an observation of a system along with a chosen action or intervention, and produces a prediction of the resulting state or outcome [14, 9, 36, 11, 56]. World models can be trained as generative models that learn latent dynamics from pixel observations and use them for policy optimization [23, 24, 25, 26, 27]. Some models learn more abstract predictive representations that can ignore irrelevant visual detail and better support understanding, prediction, and planning [71, 8, 4].

Recent video diffusion transformers [49] have led to video world models that simulate future observations as video streams [47, 65, 43, 7]. These models have been applied to robotics [46, 63, 40, 39, 66, 20], games [16, 48, 58, 3, 28, 62], self-driving [1, 31, 34], and physical simulation [38, 70]. Our focus is multiplayer simulation, where generated views must agree across agents sharing a world. Multiverse [18] is an early multiplayer video world model that trains a U-Net [54] on one race track in Gran Turismo 4, whereas we tackle the significantly more complex 3D open-world environment of Minecraft. Concurrent works MultiGen [50] and MultiWorld [61] introduce multiplayer world models, but MultiGen requires a hard-coded 2D map while MultiWorld is non-autoregressive and requires pre-recorded environment reference screenshots.

Autoregressive Video Generation. Large-scale Diffusion Transformer video models [49, 47, 37, 59] have made rapid progress. Diffusion Forcing [12] enables autoregressive generation by training with independent noise levels per frame. CausVid [69] converts bidirectional video models into efficient causal models, and Self Forcing [32, 15, 30] further reduces autoregressive train-test mismatch by training on a model's own generations.

Concurrent work, RELIC [30], also studies Self Forcing with a long-context teacher and uses recomputation during the backward pass. Our method performs this recomputation in one parallel masked forward pass, avoiding RELIC's multiple rolling passes.

AI Agents in Minecraft. Several Minecraft agent platforms have been developed, mainly for reinforcement learning [35, 19, 22]. Mineflayer [51] supports programmable bots and has been used in LoopNav [42] and Voyager [60]; VPT [6] provides large-scale human single-player data. As discussed in Section 3, these tools do not provide both multiplayer gameplay and rendered observations, motivating SolarisEngine.

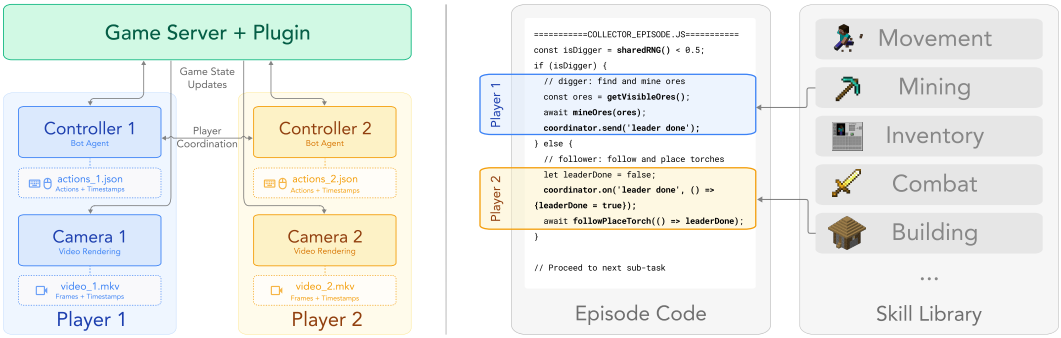

Figure 2: **SolarisEngine Overview.** *(Left)* Docker orchestration of server, camera, and controller bots where cameras mirror controllers via a server-side plugin. Although only two bots are shown, the system supports an arbitrary number of players. *(Right)* Episodes compose reusable skill primitives from a shared library. Simplified "collector" episode code is shown.

3 SolarisEngine: A Framework for Multiplayer Gameplay at Scale

We describe SolarisEngine, our system for capturing programmed multiplayer Minecraft gameplay at scale. Existing tools such as Malmo [35], MineRL [22], Minedojo [19], and Mineflayer [51] provide pieces of the required functionality, but not coherent cooperative gameplay with rendered observations

Method	Controllability	Multiplayer	Graphics
Malmo [35]	×	✓	✓
MineRL [22]	×	×	✓
MineDojo [19]	×	×	✓
Voyager [60]	✓	×	×
Mineflayer [51]	✓	✓	×
SolarisEngine	✓	✓	✓

Table 1: **Comparison of Minecraft AI frameworks.** Unlike prior systems, SolarisEngine uniquely combines controllable behavior, multiplayer support, and real Minecraft graphics.

(see Table 1). SolarisEngine combines a primitive skill library, multiplayer communication, and modular episode code to collect millions of aligned observation-action frames.

Enabling cooperative multiplayer gameplay. Existing frameworks like Malmo [35], MineRL [22], and Minedojo [19] use low-level action spaces, where, without RL training, agents can only take random actions. Data from these agents would be too simple and chaotic for world modeling, a problem that gets worse with multiple agents. Though agents could be trained using RL to learn meaningful action policies, the data collected from such agents would be geared toward collecting rewards and would not necessarily be diverse and realistic.

Video world models need realistic gameplay, so we use Mineflayer, a JavaScript Minecraft bot library also used by Voyager [60]. Its composable primitives, including *pathfinding*, *block placement*, and *combat*, let us program high-quality behavior directly.

However, Mineflayer was not designed for multiplayer coordination. To address this lack of multiplayer support, we built a communication layer enabling bot coordination to mimic collaborative human gameplay. We introduced high-level primitives such as building, tool use, and navigation. These primitives, shown in Fig. 2, when combined with the communication layer, form an episode where bots achieve a goal. We created a library of these episode types covering core Minecraft interaction aspects. For a full list, please refer to Section B.1. The episode code is high-level, but SolarisEngine logs the resulting low-level actions in a VPT-compatible format [6] (see Section B.3).

Extracting and aligning visuals with actions. Mineflayer also lacks rendering. We therefore pair each controller bot with a headless official Minecraft client, or camera bot, synchronized by a server-side plugin that mirrors state, actions, and animations. The pair forms one logical player (see Fig. 2). Frames are recorded within 100ms (2 MC ticks) after an action. Our main experiments use two players, but the system supports any number of players, e.g. 3 players in Fig. 8.

Robust and Scalable Docker System. We run controller bots, camera bots, and the Minecraft server as Docker Compose units managed by Python scripts, enabling many parallel workers. Bots repeatedly sample episodes and teleport to random locations to diversify terrain. To handle Minecraft errors and stuck episodes, a safety mechanism detects failures, notifies all components, aborts across bots, and starts a fresh episode without manual intervention. For more details, please see Section A.

Multiplayer Training Dataset. Using SolarisEngine, we collect a two-player training dataset covering building, combat, movement, and mining (see Section B). Episodes are evenly split between "Superflat" and "Normal" worlds. To our knowledge, this is the first action-annotated multiplayer Minecraft dataset. The data spans diverse times of day, biomes, tools, weather, and actions. In total, we collect 12.64M frames. Statistics appear in Fig. 3.

4 Solaris Model Design

4.1 Preliminaries

We consider a world model that predicts future observations for multiple agents from their past observations and actions. We extend the usual single-agent latent frame $\mathbf{x}_{\text{SP}} \in \mathbb{R}^{H \times W \times C}$ with a player dimension, performing diffusion over joint tensors of shape `(P, H, W, C)`. Let $\mathbf{x}^t =$

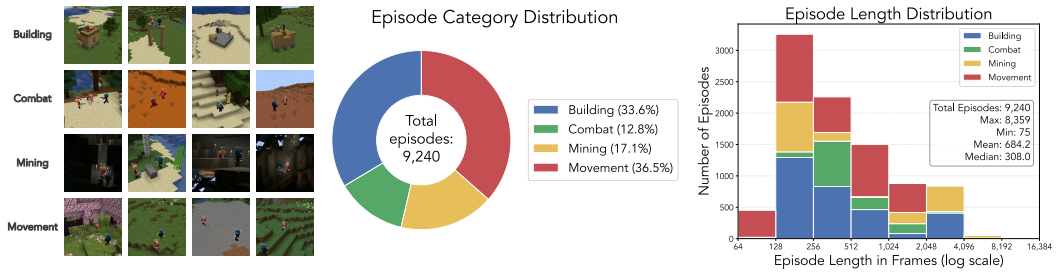

Figure 3: **Training dataset statistics.** *(Left)* Episodes cover building, combat, movement, and mining. *(Middle)* The dataset contains 9,240 episodes and 12.64 M frames, sampled with length-aware episode weights. *(Right)* Most episodes last 128–512 frames, or 6.4–25.6 seconds at 20 fps.

$\{x_1^t, \ldots, x_P^t\}$ and $\mathbf{a}^t = \{a_1^t, \ldots, a_P^t\}$ denote the joint observations and actions at time t, with sequence tensors $\mathbf{x} \in \mathbb{R}^{B\times P\times T\times H\times W\times C}$ and $\mathbf{a} \in \mathbb{R}^{B\times P\times T\times D}$. We model $p_\theta(\mathbf{x}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^t \mid \mathbf{x}^{<t}, \mathbf{a}^{<t})$ with diffusion. We train with conditional Flow Matching [45, 44], optimizing

$$\mathcal{L}_\theta = \mathbb{E}_{\mathbf{x},\mathbf{a},\sigma,\epsilon}\left[\left\|v_\theta(\mathbf{x}_{\boldsymbol{\sigma}}, \boldsymbol{\sigma}, \mathbf{a}) - (\epsilon - \mathbf{x})\right\|_2^2\right],$$

with forward process $\mathbf{x}_{\boldsymbol{\sigma}} = (1-\boldsymbol{\sigma})\mathbf{x} + \boldsymbol{\sigma}\epsilon$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. For the *bidirectional* model, all players and frames share one noise level: $\sigma \sim \mathcal{U}(0,1)$ and $\boldsymbol{\sigma} = \sigma \cdot \mathbf{1}_{P\times T}$. For the *causal* model, we use Diffusion Forcing [12], sampling independent noise levels $\sigma_{p,t} \sim \mathcal{U}(0,1)$ for each player and frame.

4.2 Network Architecture

Our architecture builds on Matrix-Game 2.0 [28], a single-player controllable video DiT [49] trained on multiple games, including Minecraft. We adapt it to multiplayer modeling with two changes.

Expanded action space. We extend Matrix-Game 2.0 to the MineRL [22] Minecraft action space by increasing the input dimension of its keyboard action module and reinitializing those weights. The action module runs independently per player by folding the player dimension into the batch.

Multiplayer attention. Players exchange information through shared self-attention in the DiT blocks. We apply 3D RoPE [55] independently to each player's tokens and add learned player-ID embeddings at each Multiplayer Self-Attention layer. The first-frame cross-attention block remains unchanged and is applied independently per player. Please see Fig. 4 for details.

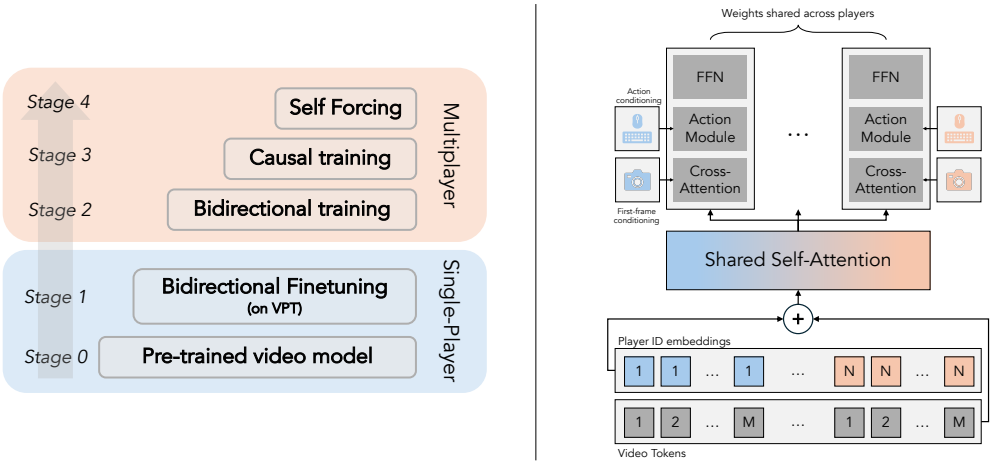

Figure 4: *(Left)* The stages of our training pipeline. *(Right)* DiT block. Player tokens are interleaved along the sequence dimension and exchange information through shared self-attention. Other modules are inherited from Matrix-Game 2.0 and applied independently per player.

5 Multiplayer Training Pipeline

We follow the established practice of first training a bidirectional base model and then adapting it to be causal, enabling auto-regressive generation. The stages of our training process are shown in Fig. 4. Training hyperparameters, TPU hardware used, and TPU chip-hours are listed in Section C. Our training and inference code is publicly released (Section H.2).

Stage 1: Bidirectional Single-Player. We initialize from Matrix-Game 2.0 [28][1], whose action space covers only camera and WASD movement. To support the full action space, we finetune on the VPT [6] dataset for 120K steps with bidirectional attention and a 33-frame context. The 3D VAE from Matrix-Game 2.0 remains frozen in all stages.

Stage 2: Bidirectional Multiplayer. We next adapt our single-player model to support multiplayer data. We employ the architectural modifications described in Section 4.2 and train the model with full-sequence diffusion with bidirectional attention on our dataset for 120K steps, where we find FID on a held-out test set converges at this point. This model is used as the teacher during Self Forcing.

Stage 3: Causal Multiplayer. We branch from the bidirectional model at 60K steps to initialize the causal model while the teacher continues to 120K. Following Matrix-Game 2.0, the causal model uses sliding-window attention with 6 latent frames (24 real frames), matching the maximum rolling KV cache. We train it for 60K steps with Diffusion Forcing before Self Forcing. Using Diffusion Forcing is simpler than the original CausVid [69] initialization assumed by Self Forcing, which uses ODE regression and DMD-based few-step distillation [68, 67], and we find it performs better. We compare these choices in Section 7.

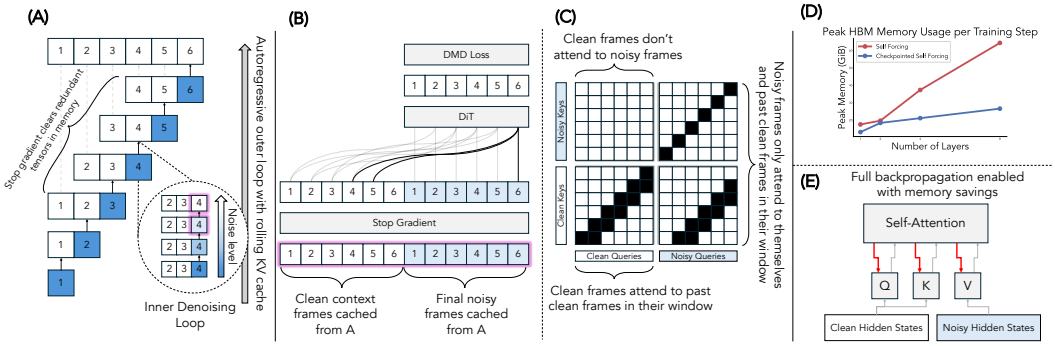

Figure 5: Overview of Checkpointed Self Forcing. **(A)** Sliding-window generation with a KV cache creates redundant tensors, making naive backpropagation memory prohibitive. **(B)** We use the cached clean context and final noisy frames from part A to recompute the final denoising step for each frame in parallel. **(C)** The mask restricts each noisy frame's attention to clean frames in its sliding-window context, mirroring part A. **(D)** The savings enable KV backpropagation, improving visual generation in Section 7. **(E)** Peak memory comparison between naive and Checkpointed Self Forcing.

Stage 4: Self Forcing. We apply Self Forcing [32] to improve long generations. Different from the original implementation (see Section E.1 for an overview), we extend it to a stronger teacher with a longer context than the student. When the student generates with a sliding window, naive backpropagation through the rollout must retain overlapping context windows, making memory scale as $O(L_t \cdot L_s)$ for generation length L_t and student context length L_s.

Checkpointed Self Forcing removes this redundant memory problem by separating rollout from backpropagation (Fig. 5). We first generate the sequence and cache clean estimates $\hat{\mathbf{x}}_0^{1:N}$ and noisy transition states $\mathbf{x}_\sigma^{1:N}$ without gradients. We then recompute the generator outputs in one parallel forward pass, using a custom attention mask so each noisy frame attends only to the preceding clean frames in its sliding context. This reproduces inference-time conditioning while reducing memory to $O(L_t)$.

[1]We use the `base_distilled_model` checkpoint.

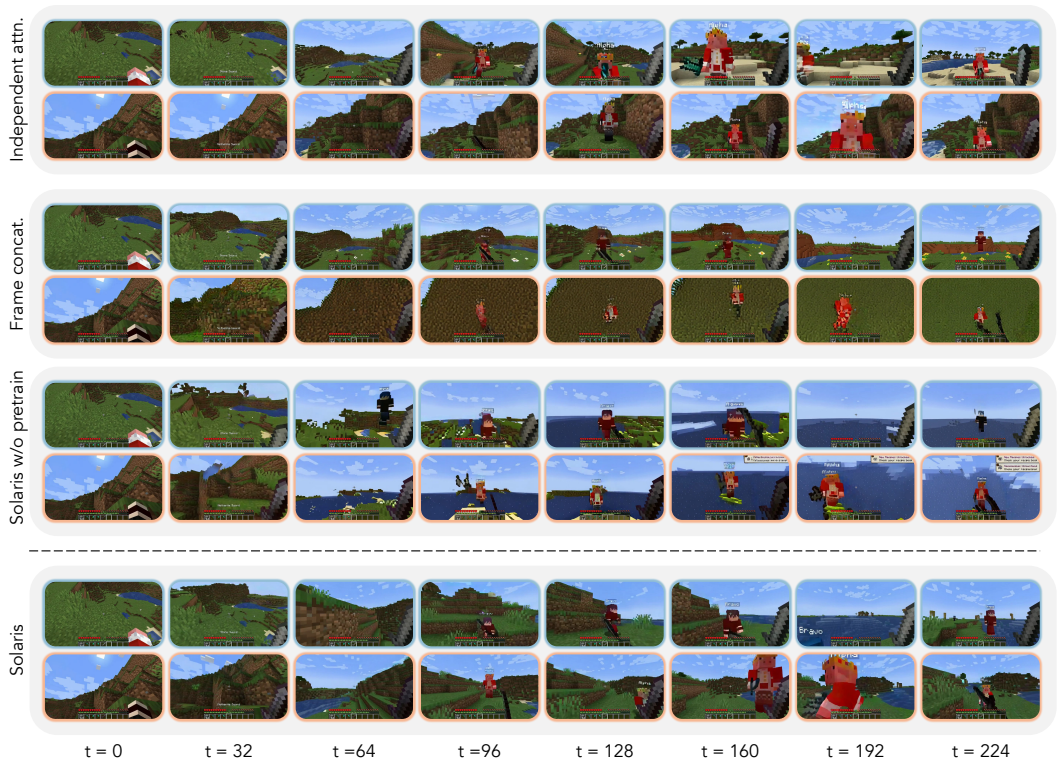

Figure 6: **Qualitative comparison.** Solaris maintains coherent fighting behavior and textured terrain. Independent attention is unable to disentangle player viewpoints, showing Player 1 in both. Frame concatenation degrades for the second player and flattens terrain. "Solaris w/o pretrain" shows artifacts such as duplicated player bodies, incorrect pop-ups, and unrealistic scene changes.

The procedure is analogous to gradient checkpointing [13, 21], with the rollout stage storing tensors needed in the recomputation step. The savings also make it feasible to backpropagate through recomputed KV representations, which improves visual quality in our ablations. A pseudocode comparison between Checkpointed Self Forcing and the original implementation appears in Section E.2.

6 Evaluation Benchmark

We evaluate on held-out episode types covering Movement, Co-Movement, Grounding, Building, and Consistency. We use FID [29] to measure visual quality, while we use a VLM judge (here `Gemini-3-Flash`) to measure binary task success by answering a verifiable multiple-choice question about each generation. The generation is counted as correct if the VLM answers match known ground-truths. On ground-truth episodes, the judge is correct on 82–100% of episodes, depending on the task (Table 7). A blind human evaluation of Consistency ranks the models in the same order as the judge (Section D.1). We generate 32 different random episodes for each type, except for "Consistency", which has 64. We show ground-truth and generated evaluation episodes in the supplementary material. We also include FVD [57] computed on 2,048 random 16-frame test-split segments in Section D, where FVD is stable on this large sample size. We describe our custom evaluation episode types below:

Movement. In a "Superflat" world, one bot moves while the other observes. The VLM judges from each agent's view how the other player moved on screen.

Co-Movement. In a "Superflat" world, two agents facing each other take the same movement action at the same time (both forward, back, left, or right). The VLM judges from each agent's view how the other player moved on screen.

Method	Movement		Co-Movement	Grounding		Building		Consistency	
	VLM ↑	FID ↓	VLM ↑	VLM ↑	FID ↓	VLM ↑	FID ↓	VLM ↑	FID ↓
Independent	12.5 ± 0.0	58.9	19.8 ± 2.9	26.0 ± 1.5	67.1	0.0 ± 0.0	77.8	38.0 ± 1.5	104.8
Frame concat	**84.4 ± 4.4**	69.5	90.6 ± 0.0	50.0 ± 2.6	66.6	0.0 ± 0.0	103.2	25.5 ± 3.2	129.4
Solaris w/o pretrain	69.8 ± 1.5	37.11	65.6 ± 4.4	25.0 ± 0.0	49.9	0.0 ± 0.0	86.6	32.8 ± 2.2	121.4
Solaris	67.7 ± 5.3	**36.84**	**100.0 ± 0.0**	**53.1 ± 0.0**	**38.0**	**9.4 ± 0.0**	**83.6**	**56.8 ± 2.9**	**99.4**

Table 2: Quantitative comparison against frame concatenation [18] and training without single-player pretraining. "VLM" reports accuracy ± one standard deviation over three VLM eval runs. Co-Movement reports VLM accuracy only.

Grounding. Two nearby agents start facing each other in a "Superflat" world. One turns away, pauses, and turns back while the other remains stationary. The VLM checks whether the turning agent loses sight of the other player while turned away and sees them again after returning.

Building. In a "Superflat" world, one bot builds a simple shape while the other observes. The VLM evaluates whether the observer sees the completed structure.

Consistency. Two nearby agents in a "Normal" world turn 90 degrees to the same side or opposite sides. The VLM judges whether their views are *similar* (same-side) and *different* (opposite-side).

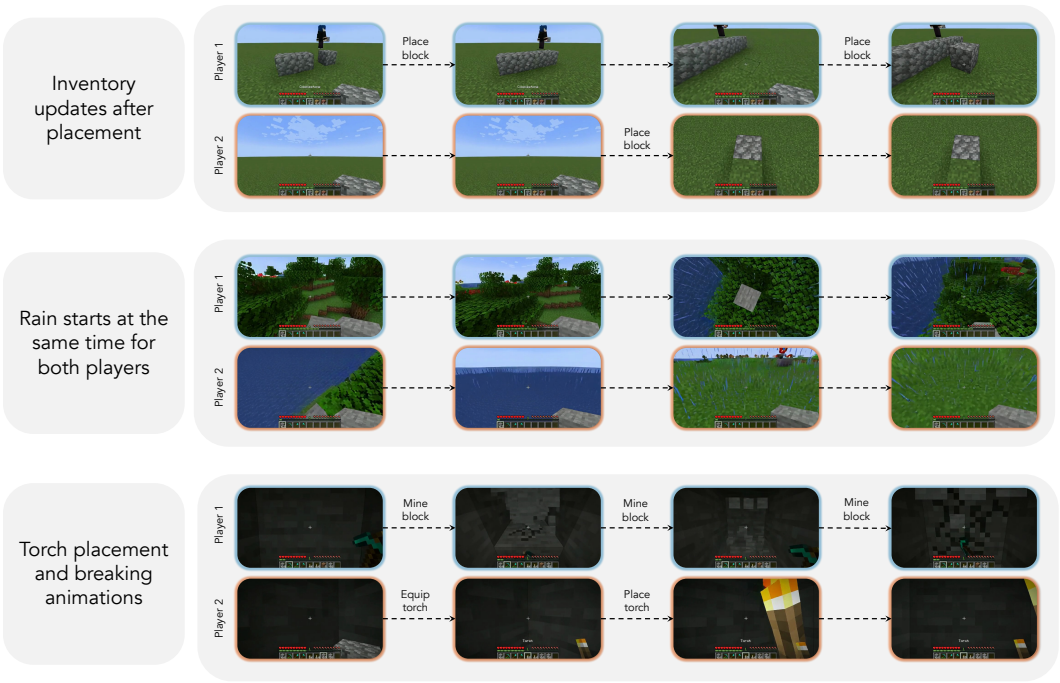

Figure 7: Qualitative examples of learned capabilities: inventory tracking after block placement, synchronized weather, and active item tracking with torch placement and mining animations.

7 Experiments

Qualitative Results. As shown in Fig. 1, Solaris simulates building, mining, fighting, and multiplayer viewpoint changes from held-out bot action sequences. The generated final frames align with third-person ground-truth game states that are not given to the model. Additional capabilities, including inventory tracking, weather, torch placement, and animations, appear in Fig. 7. In the architecture comparison in Fig. 6, baselines degrade over time while Solaris maintains visual fidelity. We include uncurated model generations on the evaluation benchmark in the supplementary material, and more videos, including generations on held-out data, on the project website.

Architecture Experiments. We compare against two architectural baselines: an Independent model, where self-attention is applied separately to each player with no cross-player information exchange, and the frame-concatenation design used by Multiverse [18], where we concatenate player observations in the channel dimension of our DiT instead of the sequence dimension. Solaris performs best overall, with stronger qualitative results in Fig. 6 and higher scores on Co-Movement and the harder Building, Consistency, and Grounding tasks in Table 2. In Co-Movement, where both players move at once, the Independent model reaches only 19.8%, since each player's view is generated without the other player's actions. We also train Solaris without single-player pretraining. Its weaker results suggest that single-player pretraining teaches useful Minecraft dynamics, allowing a relatively smaller multiplayer dataset to adapt the model to multi-agent generation. We also study how Solaris scales with model size by training the bidirectional model from scratch at 25%, 50%, and 100% of its original size. Larger models achieve lower held-out loss and FID. Please see Section G for the details.

Self Forcing Ablations. We ablate each component of our Self Forcing pipeline. We study the two main stages of CausVid, ODE regression initialization and few-step distillation, and find that straightforward causal finetuning is sufficient instead. Although the original Self Forcing paper assumes the generator to be a few-step model at the start of training, we find that the few-step ability can be learned simultaneously with stable autoregressive generation in Self Forcing, removing the need for CausVid beforehand. KV backpropagation, enabled by our memory savings, achieves the best FID in Table 3. Although it reduces action following in some categories, it remains competitive overall and performs best on the challenging Building and Consistency tasks.

Components			Movement		Co-Movement	Grounding		Building		Consistency	
Init.	Pre-DMD	KV-BP	VLM ↑	FID ↓	VLM ↑	VLM ↑	FID ↓	VLM ↑	FID ↓	VLM ↑	FID ↓
ODE Reg	×	✓	19.8 ± 1.5	77.42	57.3 ± 1.5	4.2 ± 1.5	56.6	3.1 ± 0.0	95.7	34.9 ± 1.5	142.3
Causal FT	✓	✓	35.4 ± 3.9	54.36	62.5 ± 0.0	3.1 ± 0.0	40.4	2.1 ± 1.5	90.5	45.8 ± 2.7	160.1
Causal FT	×	×	**88.5 ± 1.5**	53.76	**100.0 ± 0.0**	**77.1 ± 1.5**	55.2	5.2 ± 2.9	87.4	52.6 ± 1.9	105.1
Causal FT	×	✓	67.7 ± 5.3	**36.84**	**100.0 ± 0.0**	53.1 ± 0.0	**38.0**	**9.4 ± 0.0**	**83.6**	**56.8 ± 2.9**	**99.4**

Table 3: Ablations on Self Forcing training variants. We study the initialization of the causal model, finding simple causal finetuning with Diffusion Forcing [12] to suffice. We also find that doing few-step distillation before Self Forcing (pre-DMD) is ineffective. Finally, we find that enabling KV cache backpropagation improves visual quality, at a cost of accuracy on Movement and Grounding. "VLM" reports accuracy ± one standard deviation over three VLM eval runs. Co-Movement reports VLM accuracy only.

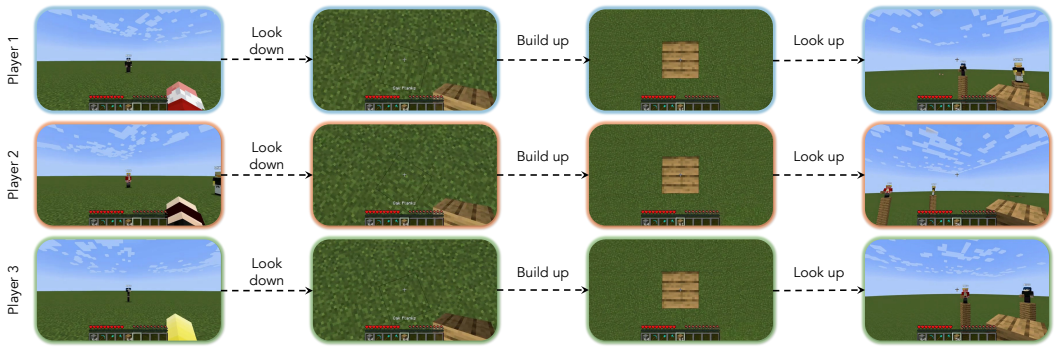

Figure 8: Our 3-player model generation on a test-split tower-building episode. Item counters update correctly when building, and players observed at the end of episode are shown accurately.

N-Player Scalability. We showcase the potential of scaling our method and data engine beyond two players by training a 3-player variant of Solaris. SolarisEngine can be scaled to any number of players, provided the episode code is written. We create a limited 3-player dataset as a proof of concept. See Section F for details on our dataset construction and training. We show a qualitative example of the 3-player model successfully generating the tower building task in Fig. 8. We also study the memory and time complexity of scaling our model beyond three players on simulated inputs. With an efficient attention implementation, up to 16 players can be run on our hardware (Fig. 9). This

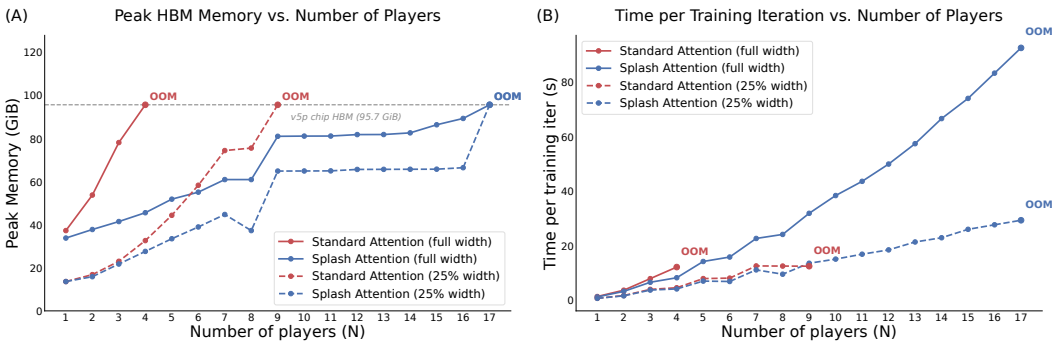

Figure 9: Memory and time scaling vs. number of players. Splash Attention compared to Standard Attention at the default width and a reduced 25% width. Markers labeled "OOM" indicate the smallest N at which the configuration exceeds the TPU's HBM. **(A)** Peak HBM scales quadratically with Standard Attention and linearly with Splash Attention. **(B)** Time per training iteration is asymptotically quadratic for both, with Splash Attention consistently faster.

shows that training with many players is feasible, although we have not collected datasets with more than three players. Please see Section F for the details.

8 Limitations

Our model faces several limitations. The use of scripted bot-generated training data limits generalization to out-of-distribution actions, and performance on human gameplay remains unexplored. Interactions with the environment remain challenging, particularly building. Qualitative results can exhibit failures in cross-view geometric consistency, duplicated or missing players, and occasionally ignored or hallucinated camera actions. We show examples of these failures, including on the Building task, on the project website. Our evaluation relies on VLM judgments, which may not fully capture visual artifacts or geometric inconsistencies. Finally, we focus on short-horizon multiplayer modeling and do not investigate long-term memory or persistent world representations. Addressing these limitations remains an important direction for future work.

9 Acknowledgments

Srivats Poddar completed this work while studying at NYU. Suppakit Waiwitlikhit and Timothy Meehan contributed to the project during their time as visiting students at NYU. We thank Nanye Ma for his help with our Jax codebase. We are grateful for Jihan Yang, Sihyun Yu, Shusheng Yang, and Yucen Lily Li for their advice on the draft and Charles Herrmann for helpful discussions. This work was primarily supported by the Google TPU Research Cloud (TRC) program and the Google Cloud Research Credits program (GCP19980904). S.X. acknowledges support from the MSIT IITP grant (RS-2024-00457882) and the NSF award IIS-2443404. O.M. is supported by the NSF Graduate Research Fellowship Program.

References

[1] Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.

[2] Arslan Ali, Junjie Bai, Maciej Bala, Yogesh Balaji, Aaron Blakeman, Tiffany Cai, Jiaxin Cao, Tianshi Cao, Elizabeth Cha, Yu-Wei Chao, et al. World simulation with video foundation models for physical ai. *arXiv preprint arXiv:2511.00062*, 2025.

[3] Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos J Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. In *NeurIPS*, 2024.

[4] Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

[5] Xingjian Bai, Guande He, Zhengqi Li, Eli Shechtman, Xun Huang, and Zongze Wu. Causality in video diffusers is separable from denoising, 2026.

[6] Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In *NeurIPS*, 2022.

[7] Amir Bar, Gaoyue Zhou, Danny Tran, Trevor Darrell, and Yann LeCun. Navigation world models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 15791–15801, 2025.

[8] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *arXiv preprint arXiv:2404.08471*, 2024.

[9] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[10] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[11] Arthur E. Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control*. Taylor & Francis, London, 1975.

[12] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. In *NeurIPS*, 2025.

[13] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[14] Kenneth J. W. Craik. *The Nature of Explanation*. Cambridge University Press, Cambridge, UK, 1943.

[15] Justin Cui, Jie Wu, Ming Li, Tao Yang, Xiaojie Li, Rui Wang, Andrew Bai, Yuanhao Ban, and Cho-Jui Hsieh. Self-forcing++: Towards minute-scale high-quality video generation. In *ICLR*, 2026.

[16] Decart, Julian Quevedo, Quinn McIntyre, Spruce Campbell, Xinlei Chen, and Robert Wachen. Oasis: A universe in a transformer. Project Website, 2024.

[17] Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *NeurIPS*, 2023.

[18] Enigma-team. Introducing multiverse: The first ai multiplayer world model. Engima Blog, 2025.

[19] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *NeurIPS*, 2022.

[20] Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, et al. Dreamdojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026.

[21] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, March 2000.

[22] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. In *IJCAI*, 2019.

[23] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

[24] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

[25] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *ICLR*, 2021.

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

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

[28] Xianglong He, Chunli Peng, Zexiang Liu, Boyang Wang, Yifan Zhang, Qi Cui, Fei Kang, Biao Jiang, Mengyin An, Yangyang Ren, et al. Matrix-game 2.0: An open-source real-time and streaming interactive world model. *arXiv preprint arXiv:2508.13009*, 2025.

[29] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017.

[30] Yicong Hong, Yiqun Mei, Chongjian Ge, Yiran Xu, Yang Zhou, Sai Bi, Yannick Hold-Geoffroy, Mike Roberts, Matthew Fisher, Eli Shechtman, et al. Relic: Interactive video world model with long-horizon memory. *arXiv preprint arXiv:2512.04040*, 2025.

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

[32] Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv preprint arXiv:2506.08009*, 2025.

[33] Joel Jang, Seonghyeon Ye, Zongyu Lin, Jiannan Xiang, Johan Bjorck, Yu Fang, Fengyuan Hu, Spencer Huang, Kaushil Kundalia, Yen-Chen Lin, Loïc Magne, Ajay Mandlekar, Avnish Narayan, You Liang Tan, Guanzhi Wang, Jing Wang, Qi Wang, Yinzhen Xu, Xiaohui Zeng, et al. Dreamgen: Unlocking generalization in robot learning through video world models. In *CoRL*, 2025.

[34] Chiyu Max Jiang, Xander Masotto, and Bo Sun. The waymo world model: A new frontier for autonomous driving simulation. Waymo Blog, 2026.

[35] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 2016.

[36] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960.

[37] 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.

[38] Chenyu Li, Oscar Michel, Xichen Pan, Sainan Liu, Mike Roberts, and Saining Xie. PISA experiments: Exploring physics post-training for video diffusion models by watching stuff drop. In *ICML*, 2025.

[39] Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, et al. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.

[40] Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. *arXiv preprint arXiv:2503.00200*, 2025.

[41] Sizhe Lester Li, Evan Kim, Xingjian Bai, Tong Zhao, Tao Pang, Max Simchowitz, and Vincent Sitzmann. Turning video models into generalist robot policies, 2026.

[42] Kewei Lian, Shaofei Cai, Yilun Du, and Yitao Liang. Toward memory-aided world models: Benchmarking via spatial consistency. *arXiv preprint arXiv:2505.22976*, 2025.

[43] Junbang Liang, Ruoshi Liu, Ege Ozguroglu, Sruthi Sudhakar, Achal Dave, Pavel Tokmakov, Shuran Song, and Carl Vondrick. Dreamitate: Real-world visuomotor policy learning via video generation. In *CoRL*, 2024.

[44] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023.

[45] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.

[46] Riccardo Mereu, Aidan Scannell, Yuxin Hou, Yi Zhao, Aditya Jitta, Antonio Dominguez, Luigi Acerbi, Amos Storkey, and Paul Chang. Generative world modelling for humanoids: 1x world model challenge technical report. *arXiv preprint arXiv:2510.07092*, 2025.

[47] OpenAI. Sora: A text-to-video model by openai. OpenAI Blog, 2024.

[48] Jack Parker-Holder and Shlomi Fruchter. Genie 3: A new frontier for world models. Google DeepMind Blog, 2025.

[49] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[50] Ryan Po, David Junhao Zhang, Amir Hertz, Gordon Wetzstein, Neal Wadhwa, and Nataniel Ruiz. Multigen: Level-design for editable multiplayer worlds in diffusion game engines. *arXiv preprint arXiv:2603.06679*, 2026.

[51] PrismarineJS. Mineflayer: Create minecraft bots with javascript. `https://github.com/PrismarineJS/mineflayer`, 2025. Version 4.35.0.

[52] Julian Hector Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. Worldgym: World model as an environment for policy evaluation. In *ICLR*, 2026.

[53] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[54] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241. Springer, 2015.

[55] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[56] Richard S. Sutton. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bulletin*, 2(4):160–163, July 1991.

[57] 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.

[58] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. In *ICLR*, 2025.

[59] 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.

[60] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *TMLR*, 2024.

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

[62] Zeqi Xiao, Yushi LAN, Yifan Zhou, Wenqi Ouyang, Shuai Yang, Yanhong Zeng, and Xingang Pan. Worldmem: Long-term consistent world simulation with memory. In *NeurIPS*, 2025.

[63] Sherry Yang, Yilun Du, Seyed Kamyar Seyed Ghasemipour, Jonathan Tompson, Leslie Pack Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *ICLR*, 2024.

[64] Sherry Yang, Jacob C Walker, Jack Parker-Holder, Yilun Du, Jake Bruce, Andre Barreto, Pieter Abbeel, and Dale Schuurmans. Position: Video as the new language for real-world decision making. In *ICML*, 2024.

[65] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1389–1399, 2023.

[66] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[67] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and Bill Freeman. Improved distribution matching distillation for fast image synthesis. *NeurIPS*, 2024.

[68] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *CVPR*, 2024.

[69] Tianwei Yin, Qiang Zhang, Richard Zhang, William T Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *CVPR*, 2025.

[70] Jianhao Yuan, Xiaofeng Zhang, Felix Friedrich, Nicolas Beltran-Velez, Melissa Hall, Reyhane Askari-Hemmat, Xiaochuang Han, Nicolas Ballas, Michal Drozdzal, and Adriana Romero-Soriano. Inference-time physics alignment of video generative models with latent world models. *arXiv preprint arXiv:2601.10553*, 2026.

[71] Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *ICML*, 2025.

A Solaris Multiplayer Framework

A.1 Mineflayer Modifications

To facilitate a wide range of Minecraft action recording, we modify the Mineflayer API to expose access to the most recently applied camera action in its physics module and extend its event system to send events on one-off semantic actions such as attacking, using, placing, and hotbar changes.

In addition, we introduce the following Mineflayer code to improve the quality of the collected data:

- Make the bot correctly look at the face of the block when placing a new block.
- Add camera smoothing to all non-Pathfinder look commands.

A.2 GPU Data Collection

We initially used `libOSMesa` for CPU-based headless rendering of the Minecraft clients (camera bots). In the open *Normal* world, however, the videos were often laggy, with up to four repeated frames at 20 fps in graphics-intensive scenes such as forests. GPU-based rendering produced consistently smooth videos without repeated frames. The GPU also let `ffmpeg` encode the screen captures with NVENC, Nvidia's hardware video encoder, which further reduced the CPU load.

B Multiplayer Training Dataset

B.1 Episode Types

Our dataset contains 14 distinct episode types that cover the 4 broader Minecraft game mechanics. We describe the episode types in Table 4. We show qualitative examples of selected episodes in Fig. 10.



Figure 10: **Episode Demonstrations from our training dataset.** We show the recorded frames from 3 different training episodes at various points in time. Note that the third-person "start state" and "end state" screenshots are for visualization only and are not part of the dataset.

B.2 Episode filtering

Due to the teleportation logic we use before every episode to diversify the terrain, the bots might end up underwater in some of the episodes, as shown in Fig. 11. The underwater mechanics break the logic of most of our episodes, so we choose not to include those. Out of 6000 episodes collected in

Table 4: Episode types in our training dataset.

Episode Name	Category	Description
buildStructure	Building	Each bot builds a wall or tower in front of the other, or bots build a platform together at midpoint
buildTower	Building	Both bots build a tall 1-block tower by jumping and placing blocks underneath themselves.
buildHouse	Building	Bots build a house together.
towerBridge	Building	Bots build a 1-block tower by jumping, then build a bridge connecting the two towers, and meet.
pve	Combat	Bots defend two random positions on the ground, facing each other against spawning mobs.
pvp	Combat	Bots equip swords and fight each other.
collector	Mining	One bot digs underground to search and mine ores within line-of-sight, while the other bot follows and places torches.
mine	Mining	Bots dig 1 block underground and mine their way towards each other.
placeAndMine	Mining	Bots stand facing each other. One bot places blocks, and the other one destroys them.
chase	Movement	One bot runs away in a zig-zag pattern and the other bot pursues it.
orbit	Movement	Bots move in a circular trajectory around a shared center, periodically looking at each other.
straightLineWalk	Movement	One bot runs towards and past the other bot, then spins to look at it.
walkLook	Movement	One or both bots move in a random direction with just WASD actions in front of each other.
walkLookAway	Movement	One bot moves in a random direction, looks away, looks back at the other bot. The other observes.

the "Normal" world, 340 are underwater. Leveraging the fact that Minecraft shows oxygen bubbles in the GUI, see Fig. 12, when the character is underwater, we filter out the water episodes using a linear classifier that we run on the episode frames, detecting the bubble icons. The classifier achieves a 100% accuracy.



Figure 11: An episode where a bot has been teleported underwater.

B.3 Action Space

The gold standard of actions for a Minecraft dataset for world modeling is the VPT [6] dataset, which contains the full range of Minecraft game actions. SolarisEngine and hence our multiplayer

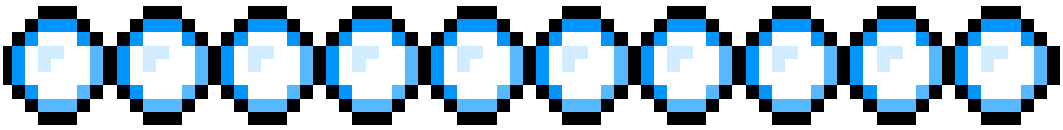

Figure 12: The "bubbles" HUD template PNG used to filter underwater episodes.

dataset comes close to it and supports everything except for inventory opening/closing and raw mouse movement recording (only relative pitch/yaw character rotations are recorded). These two limitations don't allow SolarisEngine to capture GUI activities such as inventory manipulation and crafting. Another difference between VPT and our multiplayer dataset is that we record and store actions in the form of semantic actions (either continuous or discrete) instead of raw keyboard and mouse events, as the VPT dataset does. The full list of actions recorded by SolarisEngine and available in our multiplayer training dataset is presented in Table 5.

Table 5: Action space of our multiplayer dataset. "Sustained" denotes action is recorded for as long as it lasts. "Once" denotes action is recorded for one tick only, usually applicable to instantaneous events.

Action key	Type	Description
forward	bool/sustained	Player moving forward (W).
back	bool/sustained	Player moving backward (S).
left	bool/sustained	Player strafing left (A).
right	bool/sustained	Player strafing right (D).
jump	bool/sustained	Player jumping.
sprint	bool/sustained	Player sprinting.
sneak	bool/sustained	Player sneaking.
camera	vec2f/sustained	Change in player camera orientation (yaw, pitch).
attack	bool/once	Player attacks.
use	bool/once	Player uses / interacts with the environment.
mount	bool/once	Player mounts an entity/vehicle.
dismount	bool/once	Player dismounts.
place_block	bool/once	Player places a block using the currently selected item.
place_entity	bool/once	Player places an entity item.
mine	bool/sustained	Player mining a block.
hotbar.1	bool/once	Player selects hotbar slot 1.
hotbar.2	bool/once	Player selects hotbar slot 2.
hotbar.3	bool/once	Player selects hotbar slot 3.
hotbar.4	bool/once	Player selects hotbar slot 4.
hotbar.5	bool/once	Player selects hotbar slot 5.
hotbar.6	bool/once	Player selects hotbar slot 6.
hotbar.7	bool/once	Player selects hotbar slot 7.
hotbar.8	bool/once	Player selects hotbar slot 8.
hotbar.9	bool/once	Player selects hotbar slot 9.

B.4 Mouse Action Distribution

Most of our episode types use the Pathfinder Mineflayer plugin for intelligent navigation of our bots. Pathfinder turns the camera at a fixed maximum speed of 0.15 rad per tick, or about 172 degrees per second at 20 ticks per second. Our dataset also contains manual camera movements (not controlled by Pathfinder) at random, lower speeds, but the camera actions are dominated by Pathfinder's fast turns, which form the spike in Fig. 13.

C Model Training

We provide our training hyperparameters and TPU Chip-hours in Table 6. We train our Bidirectional Single-player model on v5p-128 and all other models on v5p-64 TPUs. We pick hyperparameters typically found in the literature for the respective training type, and pick a global batch size equivalent to 1 sample per device. We train until FID convergence on the test split.

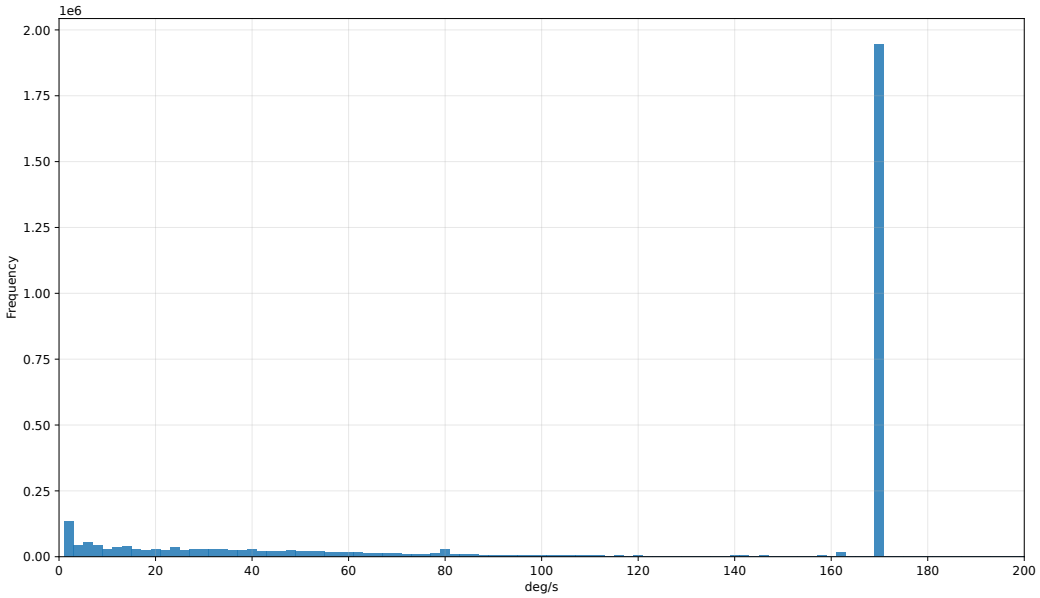

Figure 13: A histogram showing the mouse action magnitude distribution in our training dataset.

Table 6: Training hyperparameters across all training stages.

Setting	Learning Rate	Batch Size	Num Steps	Adam β_1	Adam β_2	Weight Decay	TPU Chip-hours
Bidirectional Single-player	1e−4	64	120K	0.9	0.95	0	2.7K
Bidirectional Multiplayer	1e−4	32	120K	0.9	0.95	0	3.6K
Causal Multiplayer	1e−4	32	60K	0.9	0.95	0	1.6K
Self Forcing (generator)	3e−6	32	240	0.9	0.95	0	89
Self Forcing (critic)	3e−7	32	1200	0.9	0.95	0	89

D Evaluation Details

In this section, we present the implementation details of our VLM-as-a-judge evaluation method, providing the exact VLM prompts and its accuracy on ground truth videos. The VLM accuracy is calculated based on whether the VLM gives the expected answer for all queries across all query perspectives for the same episode. E.g., for "Grounding", where one player turns away, an episode is judged as accurate if and only if the VLM's answer matches the expected answer at all three query points (3 total queries). By contrast, for "Consistency", there are two queries per episode, but it includes both players' perspectives. Please see Table 7 for the exact prompt and accuracy on ground-truth episodes ("Episode GT Acc."), and Fig. 14 for a visualization. For all tasks except Movement and Co-Movement, we query the VLM an additional time at the end of the episode. The VLM judge is `gemini-3-flash-preview`, with thinking disabled for Movement, Co-Movement, and Grounding and enabled for Building and Consistency, on both generated and ground-truth episodes.

We additionally report the test-split FVD metric in Table 8 to complement the eval-episodes FID and VLM accuracy reported in Table 2 and Table 3. The test-split FVD is computed by comparing test-split generations to the ground truth on 2,048 random 16-frame segments, pooling the clips of both players. Note that the FVD is to be interpreted as a measure of overall video quality. It is agnostic of actions, number of players, and episode types; plus, FVD requires thousands of diverse videos to be stable. As a result, on our evaluation episodes (Section 6), we use the image-based FID instead, which needs fewer video samples.

Task	Ep. Type	Query Point, Perspective	Prompt	Exp. Answer(s)	GT Ep. Acc. $\pm$ 1 SD (%)
Movement	Translation	Movement Ends, Each Player	Here are Minecraft screenshots showing another player on the screen. Between the first frame and the second frame, did the player being shown move closer, farther, to the left, or to the right on-screen? Answer with a single word from "closer", "farther", "left", "right", or "no motion".	closer, farther, left, right	100.00 ± 0
Co-Movement	Both Players Move	Movement Ends, Each Player	Here are Minecraft screenshots showing another player on the screen. Between the first frame and the second frame, did the player being shown move closer, farther, to the left, or to the right on-screen? Answer with a single word from "closer", "farther", "left", "right", or "no motion".	closer, farther, left, right	98.96 ± 1.47
Grounding	One Player Turns Away and Back	Turned Away, Acting Player	Here is a Minecraft screenshot. Is there another player visible on-screen? Answer with a single word: "yes", "no".	no	98.96 ± 1.47
		Turned Back, Acting Player		yes	
		Episode Ends, Acting Player		yes	
Building	Building	Building Ends, Observing Player	Here is a Minecraft screenshot. Can you tell me whether there is a visible structure built about 6 blocks away from the player? Answer with a single word from "yes", "no".	yes	82.29 ± 5.31
		Episode Ends, Observing Player		yes	
Consistency	Turn 90° (Same Side)	Turning Ends, Both Players	You will be shown two Minecraft screenshots. Do these two screenshots show the same scenery? Be careful and answer based on the content of the screenshots, not just the camera angles. Answer with a single word: "yes", "no".	yes	95.83 ± 1.47
		Episode Ends, Both Players		yes	
	Turn 90° (Opposite Sides)	Turning Ends, Both Players		no	93.75 ± 5.10
		Episode Ends, Both Players		no	

Table 7: Detailed description for our evaluation episodes. The "GT Ep. Acc." column shows the VLM accuracy for ground-truth episodes (which is expected to be 100%). In all tasks except for "Consistency" (where both players turn) and "Co-Movement" (where both players move), one "acting" player performs actions while the other "observing" player stands completely still. In the Query Perspective column, "Each Player" means we ask the VLM the same text prompt twice, each time about a different player's perspective (2 queries). "Both Players" means we ask the VLM judge one text prompt, but about both players' perspectives (1 query). For all tasks except Movement and Co-Movement, we additionally query the VLM at the end of the episode with the same expected answer as the prior query point. Standard deviation calculated across 3 trials.

Ablation	Variant	FVD ↓
Architecture	Independent	88.7
	Frame concat	113.2
	Solaris w/o pretrain	102.0
	Solaris	**81.3**
Self Forcing	ODE Reg, Pre-DMD ×, KV-BP ✓	151.9
	Causal FT, Pre-DMD ✓, KV-BP ✓	131.8
	Causal FT, Pre-DMD ×, KV-BP ×	**76.9**
	Causal FT, Pre-DMD ×, KV-BP ✓	81.3

Table 8: Test-split generation FVD comparison across ablation variants. FVD is computed on the 2,048 test-split segments (not split by task type). Lower is better. The shaded row is our final Solaris configuration, which appears in both ablations. Note FVD is agnostic to actions, episode types, and the number of players.

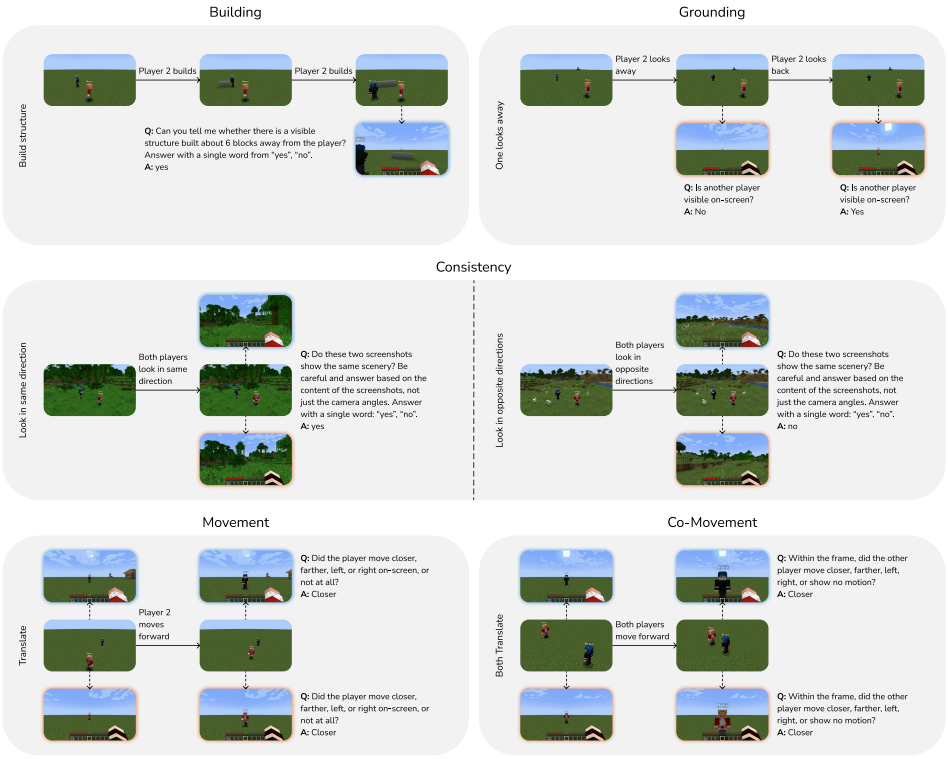

Figure 14: An illustration of the frames and prompts provided to the VLM for evaluation. Each scenario is a ground-truth video for illustration. The third-person views show what happens in the episode, and the frames sent to the VLM are outlined in blue (Player 1's view) or orange (Player 2's view), next to the corresponding question and correct answer. Prompts are abbreviated; please see Table 7 for the full text.

D.1 Human Evaluation

To check that the VLM judge's Consistency scores reflect what people see, five annotators who are authors of this paper answered the judge's question on the same screenshot pairs that the judge received. The annotators were blind to which model produced each pair: items carry opaque IDs, and each annotator saw them in a different shuffled order. Before starting, each annotator saw two calibration examples taken from ground-truth episodes, one showing the same scenery and one showing different scenery. The annotators labeled the full Consistency set (32 same-side and 32 opposite-side episodes, each queried at two points) for three models of Table 2: Independent, Frame concat, and Solaris, which is 384 screenshot pairs per annotator. Independent was labeled as a separate task with the same interface. We score the human answers exactly as the VLM judge's: an episode counts as correct only if the answers at both of its query points are correct.

Judge	**Independent**	**Frame concat**	**Solaris**
Human annotators (5)	40.0 ± 3.9	32.5 ± 7.3	$\mathbf{67.5 \pm 6.2}$
VLM judge (3 trials)	38.0 ± 1.5	25.5 ± 3.2	$\mathbf{56.8 \pm 2.9}$

Table 9: Consistency accuracy (%) from human annotators and from the VLM judge on the same screenshot pairs. $\pm$ is one standard deviation over the five annotators or over the three VLM eval runs. The VLM row is the Consistency column of Table 2.

Table 9 shows that humans and the VLM judge rank the models the same way, with Solaris first by a wide margin. Every annotator and every VLM run ranks Solaris above both baselines. On average, the annotators score all three models higher than the judge does, so the judge is stricter overall rather than favoring Solaris. Every annotator also ranks Independent above Frame concat, but for two of the five the gap is under two points, so we do not consider that part of the ranking firm.

We also compare answers pair by pair. An annotator agrees with each other annotator on $68.2 \pm 5.3\%$ of the pairs on average (Cohen's $\kappa = 0.36 \pm 0.06$), and the VLM judge agrees with each annotator on $65.7 \pm 0.3\%$ ($\kappa = 0.26 \pm 0.01$). The judge therefore disagrees with a human about as often as two humans disagree with each other. Agreement on individual pairs is only fair for both, because the same-scenery question is hard to answer for a single pair; the ranking of the models is stable.

E Self Forcing

E.1 Overview

Here, we give a brief overview of the original implementation of the Self Forcing algorithm. Let N be the number of frames in the video sequence. We employ a few-step denoising strategy with a discretized noise schedule $\{\sigma_0 = 0, \ldots, \sigma_K = 1\}$. A core component of Self Forcing is the ability to obtain a clean prediction from any intermediate noisy state. In Flow Matching, given a noisy latent $\mathbf{x}_\sigma$ and the vector field prediction $v_\theta(\mathbf{x}_\sigma, \sigma)$, the estimated clean data $\hat{\mathbf{x}}_0$ can be derived analytically: $\hat{\mathbf{x}}_0 = \mathbf{x}_\sigma - \sigma v_\theta(\mathbf{x}_\sigma, \sigma)$.

During training, fully unrolling the denoising process for every frame is memory-prohibitive. To address this, Self Forcing employs a *stochastic gradient truncation* strategy. For each training iteration, a random target noise level $\sigma_{stop} \sim \{\sigma_0, \ldots, \sigma_K\}$ is sampled. The model generates the video autoregressively, but for each frame, the denoising process is truncated once it reaches σ_{stop}. The input to the next frame i is conditioned on the clean estimates of the previous frames, derived from this truncated state:

$$\hat{\mathbf{x}}_0^{1:N} \sim p_\theta = \prod_{i=1}^{N} p_\theta(\hat{\mathbf{x}}_0^i \mid \hat{\mathbf{x}}_0^{<i}).$$

The DMD loss is applied to the sequence of estimates $\hat{\mathbf{x}}_0^{1:N}$. Gradients are backpropagated only from this final truncation step, and in the original implementation, a stop-gradient operation is added to the KV cache. This random sampling of σ_{stop} ensures that the model receives supervision across all noise levels while maintaining constant memory complexity with respect to the number of diffusion steps.

Algorithm 1: Checkpointed Self Forcing

Input: Denoise timesteps $\{t_1, \ldots, t_T\}$, teacher context length L_t, student context length L_s
Input: AR diffusion model G_θ (returns KV embeddings via G_θ^{KV})

Initialize model output $\mathbf{X}_0 \leftarrow []$
+ Initialize noisy inputs $\mathbf{X}_s \leftarrow []$
Initialize KV cache $\mathbf{KV} \leftarrow []$
Sample $s \sim \text{Uniform}(1, 2, \ldots, T)$
for $i \leftarrow 1$ **to** L_t **do**
  Initialize $x^i_{t_T} \sim \mathcal{N}(0, I)$
  for $j \leftarrow T$ **to** s **do**
    if $j = s$ **then**
      + $\mathbf{X}_s$.append($x^i_{t_j}$)
      Set $\hat{x}^i_0 \leftarrow G_\theta(x^i_{t_j}; t_j, \mathbf{KV})$
      $\mathbf{X}_0$.append($\hat{x}^i_0$)
      - Cache $\mathbf{kv}^i \leftarrow G_\theta^{\text{KV}}(\texttt{stop_grad}(\hat{x}^i_0); 0, \mathbf{KV})$
      + Cache $\mathbf{kv}^i \leftarrow G_\theta^{\text{KV}}(\hat{x}^i_0; 0, \mathbf{KV})$
      $\mathbf{KV}$.append($\mathbf{kv}^i$)
      if $|\mathbf{KV}| > L_s$ **then** $\mathbf{KV}$.pop(0)
    end
    else
      - Set $\hat{x}^i_0 \leftarrow \texttt{stop_grad}(G_\theta(x^i_{t_j}; t_j, \mathbf{KV}))$
      + Set $\hat{x}^i_0 \leftarrow G_\theta(x^i_{t_j}; t_j, \mathbf{KV})$
      Sample $\epsilon \sim \mathcal{N}(0, I)$
      Set $x^i_{t_{j-1}} \leftarrow \texttt{euler_step}(\hat{x}^i_0, \epsilon, t_{j-1})$
    end
  end
end
+ $\mathbf{X}_s \leftarrow \texttt{stop_grad}(\mathbf{X}_s)$
+ $\mathbf{X}_0 \leftarrow \texttt{stop_grad}(\mathbf{X}_0)$
+ $M \leftarrow \texttt{TeacherForcingMask}(L_s, L_t)$
+ $\mathbf{X}_{\text{in}} \leftarrow [\mathbf{X}_0, \mathbf{X}_s]$
+ $\hat{\mathbf{X}}_0 \leftarrow G_\theta(\mathbf{X}_{\text{in}}; [0, t_s], \text{mask} = M)$
Update θ via distribution matching loss

Figure 15: Jax-style diff pseudocode comparing a single generator train step of naively applying sliding-window self forcing to our efficient implementation, Checkpointed Self Forcing. Green "+" lines denote lines added from our algorithm while red "-" denote lines that have been removed. Please also see Section E.3 for mask pseudocode and Fig. 5 for a visualization.

E.2 Algorithmic Comparison

We show the pseudo-code of Checkpointed Self Forcing in a "diff" style comparing it to the original Self Forcing implementation in Algorithm 1.

E.3 Teacher Forcing Mask Implementation

In Checkpointed Self Forcing, a special teacher forcing mask is used to re-simulate the final step of denoising across all frames in parallel. This mask enforces sliding window causality and that noisy frames can only attend to past clean frames. We present pseudocode in Algorithm 2.

Algorithm 2: Pseudocode of `TeacherForcingMask` in a NumPy-like style.

```
# L_s: student context length (sliding window size)
# L_t: teacher context length (number of frames)
# tokens_per_frame: number of tokens per frame

ctx_len = L_t * tokens_per_frame

# create index grids for queries and keys
q_idx = np.arange(2 * ctx_len)[:, None]
kv_idx = np.arange(2 * ctx_len)[None, :]

# compute frame indices
q_frame = (q_idx // tokens_per_frame) % L_t
kv_frame = (kv_idx // tokens_per_frame) % L_t

# determine if query/key is from noisy or clean frames
q_is_noisy = q_idx >= ctx_len
kv_is_noisy = kv_idx >= ctx_len

# teacher forcing mask logic
teacher_forcing = (
    # noisy queries attend to same-frame noisy keys
    (q_is_noisy & kv_is_noisy & (q_frame == kv_frame))
    # noisy queries attend to earlier clean keys
    | (q_is_noisy & ~kv_is_noisy & (q_frame > kv_frame))
    # clean queries attend causally to clean keys
    | (~q_is_noisy & ~kv_is_noisy & (q_frame >= kv_frame))
)

# sliding window constraint
sliding_window = kv_frame > (q_frame - L_s)

mask = teacher_forcing & sliding_window
```

F N-Player Scalability

F.1 3-Player Dataset

To study the scalability of our method to more than two players, we design three new episode types in SolarisEngine that work with any number of players. The episodes feature building and navigation; please refer to Table 10 for the detailed episode type descriptions. Using the new episode types, we collect a dataset with 3 players. The total number of frames is 1.1 M, with 946 episodes equally split between normal and flat worlds.

Table 10: N player Episode types.

Episode Name	Category	Description
buildTowerNp	Building	bots build a tall 1-block tower by jumping and placing blocks underneath themselves.
chaseNp	Movement	One bot runs away in a zig-zag pattern and other bots pursue it.
walkLookNp	Movement	bots move in a random direction with WASD actions in front of each other.

F.2 3-Player Model Training

Using the three-player dataset, we train our model with the model architecture unchanged, except for increasing the player ID embedding table size to 3. We repeat our training pipeline, starting from stage 2, and use our Solaris 2-player bidirectional checkpoint as initialization. The bidirectional training takes 30K steps, causal 45K steps, and self-forcing 1.2K steps. We keep all hyperparameters unchanged.

F.3 Scaling Beyond Three Players

Additionally, we analyze the computational time and memory complexity as a function of the number of players. Using random inputs, which cost the same memory and time as real gameplay data, we incrementally increase the number of players up to 16. To enable a more thorough investigation of the scaling behavior, we also conduct experiments at a decreased model width (25% of the default), which

lets us collect a wider range of N before exhausting TPU's memory. We compare Splash Attention implementation against Standard Attention. We find that the time complexity scales quadratically and the memory usage scales linearly when using Splash Attention, and quadratically when using the Standard Attention implementation. Please see Fig. 9 for a visualization.

G Model Size Scaling

We trained the bidirectional model from scratch on the multiplayer data for 120K steps, using 25%, 50%, and 100% of the original model size (Table 11). We did not use single-player initialization in these runs because the single-player checkpoint is only available at the original model size, so using it would make the 25% and 50% settings incomparable. We report held-out loss because VLM evaluation would require training the full pipeline at each model size. Prior work on causal video world models has similarly used held-out diffusion loss to compare architectural and scaling choices [5, 41]. Held-out loss decreases monotonically with model size, and the gap persists throughout training; FID follows the same ordering. Please see Fig. 16 for a visualization.

Table 11: Model size ablation. The bidirectional model is trained from scratch on the two-player data for 120K steps. Loss is the flow-matching loss on the held-out test split. Both metrics are measured at the final checkpoint.

Size	Layers	Hidden	Heads	FFN	Loss	FID
25%	17	896	7	5,248	0.1412	66.1
50%	24	1,152	9	6,720	0.1303	53.5
100%	30	1,536	12	8,960	**0.1248**	**46.9**

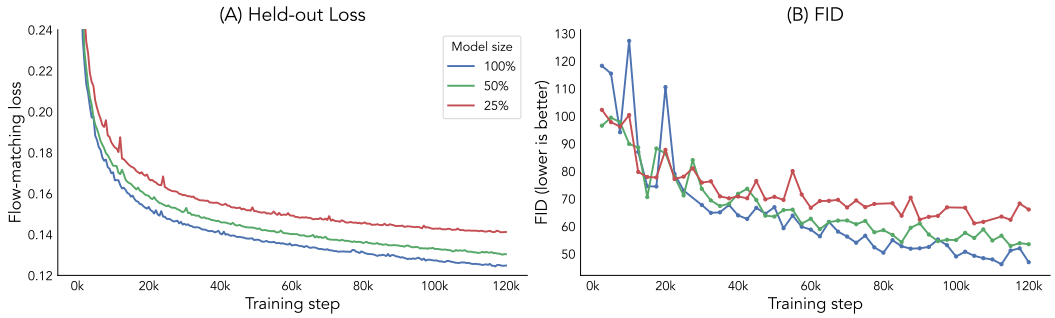

Figure 16: Held-out loss and FID vs. training step for 25%, 50%, and 100% of the original model size. **(A)** Held-out flow-matching loss decreases monotonically with model size, and the gap persists throughout training. **(B)** FID follows the same ordering.

H Asset Documentation

H.1 Existing Assets

We use the following third-party assets. Each is cited at first use; versions and license details are summarized below.

Pre-trained models.

- **Matrix-Game 2.0** [28]. Used as the initialization for our model architecture and weights, adapted from single-player to multiplayer. License: MIT. URL: `https://huggingface.co/Skywork/Matrix-Game-2.0`.
- **Wan2.1 3D Causal VAE (Wan-VAE)** [59]. The frozen 3D VAE inherited from Matrix-Game 2.0; not retrained at any stage of our pipeline. License: Apache 2.0. URL: `https://github.com/Wan-Video/Wan2.1`.
- **OpenAI CLIP** [53]. Used as the image conditioner required by the Matrix-Game 2.0 pipeline. We use the released ViT checkpoints unchanged. License: MIT. URL: `https://github.com/openai/CLIP`.

Datasets.

- **OpenAI VPT contractor dataset** [6]. Used for single-player finetuning; approximately 2,000 hours of human Minecraft gameplay with paired video and action labels. License: MIT (released by OpenAI). Note: Minecraft itself is the intellectual property of Mojang/Microsoft and is not relicensed by OpenAI; downstream use of the gameplay frames must comply with Mojang's EULA. URL: `https://github.com/openai/Video-Pre-Training`.

Game environment.

- **Minecraft: Java Edition** (Mojang / Microsoft). Used as the simulation environment for all data collection. Used under the Minecraft EULA and Commercial Usage Guidelines. We hold valid Minecraft Java Edition licenses for every account and instance involved in collection, do not redistribute any Minecraft game assets (textures, sounds, or world files), and release only model outputs and gameplay recordings derived from our self-hosted servers. URL: `https://www.minecraft.net/en-us/eula`.

Libraries and tooling.

- **Mineflayer v4.35.0** (PrismarineJS). JavaScript Minecraft client used for bot control during data collection. License: MIT. URL: `https://github.com/PrismarineJS/mineflayer`.
- **JAX** [10]. Machine learning framework used for training and inference. License: Apache 2.0. URL: `https://github.com/jax-ml/jax`.
- **Docker / Docker Compose**. Service orchestration for game servers and bots. License: Apache 2.0. URL: `https://www.docker.com/`.
- **Standard scientific Python stack.** We use unmodified, publicly released versions of PyTorch (BSD-3-Clause; `https://pytorch.org`), HuggingFace `transformers` and `diffusers` (Apache 2.0; `https://github.com/huggingface`), NumPy (BSD-3-Clause), SciPy (BSD-3-Clause), Pillow (HPND/MIT-CMU), and OpenCV (Apache 2.0). No modifications were made; each is used in accordance with its respective license.

Evaluation models and metrics.

- **Gemini** (Google DeepMind). Used as the VLM judge for semantic task-adherence evaluation (Section 6). Accessed via the Gemini API on the paid tier, under the Gemini API Additional Terms of Service so that prompts and outputs are not used to train Google's models. Specific model snapshot: `gemini-3-flash-preview`. Terms: `https://ai.google.dev/gemini-api/terms`.

- **FID via** `matthias-wright/jax-fid`. Used for image-quality FID computation in JAX. License: Apache 2.0. URL: `https://github.com/matthias-wright/jax-fid`.
- **FVD via** `flateon/FVD-I3D-torchscript`. Used for the TorchScript I3D weights in our FVD computation. License: MIT (per its Hugging Face model card); the underlying I3D weights were originally released by DeepMind in `google-deepmind/kinetics-i3d` under the Apache 2.0 license, and the FVD metric convention follows Unterthiner et al. [57]. URLs: `https://huggingface.co/flateon/FVD-I3D-torchscript`, `https://github.com/google-deepmind/kinetics-i3d`.

H.2 New Assets

We publicly release the following new assets alongside the paper:

- **SolarisEngine** (data-collection system). Released under the Apache-2.0 license. URL: `https://github.com/solaris-wm/solaris-engine`.
- **Solaris training and inference code.** Released under the Apache-2.0 license. URL: `https://github.com/solaris-wm/solaris`.
- **Solaris pre-trained models.** Released under the Apache-2.0 license. URL: `https://huggingface.co/nyu-visionx/solaris`.
- **Solaris Multiplayer Training and Eval Dataset**. Released under the Apache-2.0 license. URLs: `https://huggingface.co/datasets/nyu-visionx/solaris-training-dataset` (train), `https://huggingface.co/datasets/nyu-visionx/solaris-eval-datasets` (eval). Downstream users must additionally comply with the Mojang/Microsoft Minecraft EULA for any rendered game content. The dataset card documents collection methodology, the episode-type taxonomy, splits, known limitations, and intended research use.

Consent and human subjects. No human-subject data is included. All gameplay was performed by automated bots in self-hosted servers operated by the authors; no third-party players' data is recorded.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We contribute: a multiplayer world model, a multiplayer data collection system, a multiplayer dataset, a multiplayer evaluation framework, and a modification to the Self Forcing algorithm. This is exactly what we claim in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our paper in Section 8.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not contain theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides a detailed description of how our model architecture works and our training pipeline. We publicly release all code, data, and model weights; see Section H.2 for the URLs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We publicly release all code, data, and model weights; see Section H.2 for the URLs.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We provide the training details in Section C of the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We include standard deviation estimates for our main results, which have randomness in Table 2 and Table 3.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide details in Table 6 (Section C).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work uses a simulated Minecraft environment rather than human-subject or personally identifiable data, and we discuss potential negative social impact in Section 8.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss it in Section 8.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our model and data are trained to generate video-game-like videos, which are unlikely to be misused. The dataset itself is recorded with bots, with no humans or any personally identifiable information involved.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We address this in Section H.1.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We detail the licenses for new assets in the paper in Section H.2. Details about training and limitation can be found in the paper at Section C and Section 8.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification:

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.