Video^2 Game Generation: A Practical Study using Mario

Virtuals Protocol

Abstract

This work reports the study of video generative models for interactive video game generation and simulation. We discuss and explore the use of available pre-trained open-sourced video generation models to create playable interactive video games. While being able to generate short clips of broad range of described scenes, such models still lack controllability and continuity. Given these limitations, we focus on producing and demonstrating a reliable and controllable video game generator on a single game domain. We present MarioVGG, a text-to-video diffusion model for controllable video generation on the Super Mario Bros game. MarioVGG demonstrates the ability to continously generate consistent and meaningful scenes and levels, as well as simulate the physics and movements of a controllable player all through video.

1 Introduction

Recent progress and excitement in AI have been fueled by generative models that have been trained on large-scale internet data. These models have shown the ability to generate highly realistic text [24; 34; 2; 9], images [28; 26] and audio []. Despite being more complex modality, video generation models have also made tremendous strides with models showing capabilities of producing very detailed, high-resolution and complex scenes [5; 31; 27; 11]. The richness of the video modality and the ability to capture information about the world that is difficult or cannot be captured by text, mean that such models have enormous potential as a representation across a wide range of AI applications [39]. It has already demonstrated promise in the media and entertainment industries through examples productions of short films by creative directors and visual artists using such models [22].

Similarly, using video generation models for video games has tremendous potential. Recent work has seen the use of other generative AI models such as Large Language Models (LLMs) to help design the narrative of games [10], levels and worlds [32; 21; 20], and even the use of more coding focused models to help design within game engines. As these are language models operating over only text, they are limited to text-based narration games or still rely heavily on game engines, parsing from text to visual elements and fixed-elements and assets, making them constrained by the game engine's functionalities. However, one might draw the parallel to the potential effect of video models in the production of movies and could imagine or ask that given the final consumption format of a video games are in video, could the video game just directly be generated in video format? This would help to overcome constraints and limitations of the game engines and open up the space and accessibility for anyone to be able to easily create interactive video games, through a controllable text-interface. In other words, we ask the question can video generation models replace game engines? While there are still strides to be made in the capabilities of large video models in being able to accurately simulate physics and in their controllability, we explore some of these current challenges and what such a future model or method could look like and require.

Towards a new interface for designing, developing and playing video games, we highlight two key desiderata of video generation models in such a domain, (i) controllable generation of actions and (ii)

Preprint.

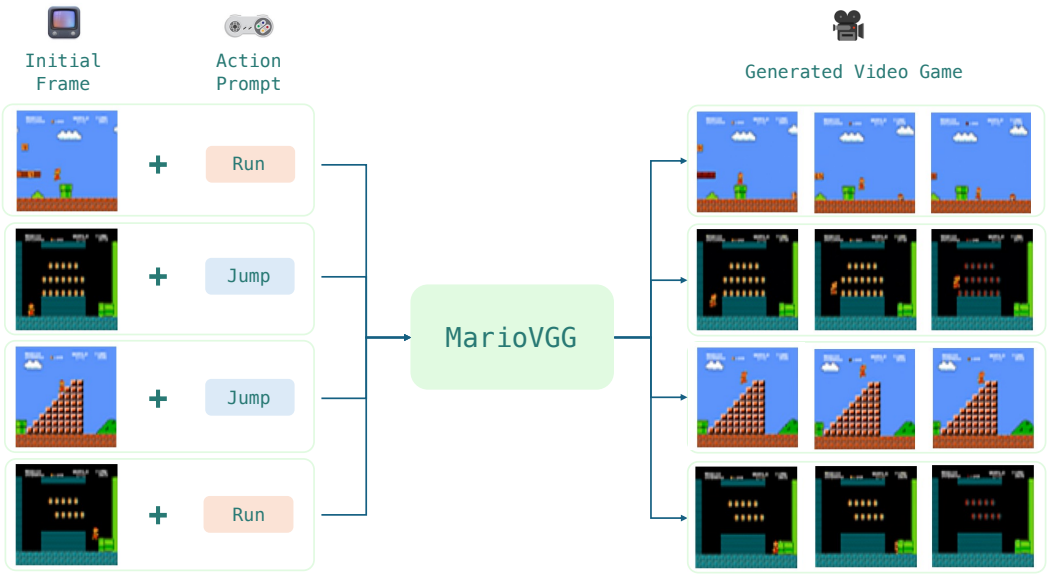

Figure 1: MarioVGG is a text-to-video model which takes in the initial game state as a single image frame, and an action in the form of a text prompt. It then outputs video (a sequence of frames). Examples of video produced by MarioVGG seen above across a diverse set of scenes. Video frames generated shown are skipped for illustration purposes to fit in the figure.

automatic or controllable generation of environments and worlds that are coherent and consistent. Controllable generation of actions enable novel, creative, imaginative actions that are difficult to design to be rendered and displayed easily. Similarly, worlds described by the designer or users could also create and lend itself to more immersive worlds and scenarios. Video generation models could be thought of as a new format of Procedural Content Generation (PCG) [30] which is a suite of methods and algorithms used to automatically design and generate elements and contents of games.

This report explores and discusses some of the key challenges of pre-trained video generation models for video games. These video generation models were designed for being very general generators that are able to generate scenes from arbitrary text descriptions well but struggle with continuity, controllable and coherence of many elements in the scene. Given the limitations of current large pre-trained video models, we focus on training a controllable video generation model on just a single 2-D domain and game, Super Mario Bros. We present, MarioVGG, a text-to-video diffusion model for controllable video generation, capable of producing a sequence of frames given a text prompt representing an action. MarioVGG demonstrates the ability to generate video that corresponds to the text action provided while also automatically generating the levels and environments in the game that is coherent. This coherence and continuity is also demonstrated by the ability to chain multiple sequences of video generated, closely mimicking how a live video game would be played.

2 Background

2.1 Diffusion Models

Diffusion models are a class of generative models that learn to approximate a distribution by iteratively transforming a random noise distribution into the target data distribution, referred to as the denoising process. Training such models involve a forward diffusion process, which adds Gaussian noise $\mathcal{N}(\mu, \sigma^2)$ to the data over several iterations, and a reverse diffusion process, which reconstructs the original data from a random noise distribution also over several iterations. The reverse process is modeled using a noise prediction network $\epsilon_\theta(x_t, t)$ which predicts the noise to remove at each iteration, optimized through learning.

Trained with just this denoising objective and this process, diffusion models have shown state-of-the-art performance in various tasks such as image generation, inpainting, and super-resolution.

Video diffusion models extend the principles of diffusion models to the temporal domain, allowing for the generation and refinement of entire video sequences. These models work by applying a similar noise addition and removal process as in image diffusion, but operate across both spatial and temporal dimensions. This involves learning consistency and smooth transitions between sequences of frames.

2.2 Pre-trained General Video Models for Video Game Generation

Such diffusion models have also powered much of the progress and development surrounding large pre-trained video models which have been trained on large scale internet data. These models are trained to be general-purpose video generators that demonstrate tremendous ability to generate high-resolution videos based on arbitrary text descriptions. Stable Video Diffusion (SVD) [3] is an open-source example of a large pre-trained video diffusion model that generates a short video given an input image and prompt. SVD was trained on a large-scale video dataset that was carefully curated and processed for video model training. Another example of such a model is Haiper [1], which also facilitates the creation of short video clips with text prompting. Naturally, one might ask whether these large pre-trained general video diffusion models can also generate realistic and reliable video game scenes.

We briefly explore this question and conduct a few tests with some of these pre-trained general video models. We found that while SVD and Haiper were able to generate realistic one-off clips of hypothetical game scenes, it had several shortcomings that render them unreliable for video game generation more generally. First, we found that these models lacked *fine-grained controllability* required for video game generation. The models had difficulty animating specific game-related actions reliably when prompted (e.g. "fire gun" or "jump towards the right"). Even if prompts were followed in some circumstances, the generated video was inconsistent across scenes. For example, a "jump" action in one generation would look different with that in another output generation. Second, these models struggled with game continuity and coherence. These models were trained to generate a fixed number of frames in each inference call, amounting to not more than 4 seconds of video. To generate coherent gameplay videos of arbitrary length, multiple inference calls need to be strung together. We found that the models were incapable of generating coherent videos across separate inference calls (e.g. character motion and art style changes abruptly). Some examples of these shortcomings are displayed in Figures 9, 10, and 11 in the Appendix. These findings are also observed elsewhere by users playing with such models when these models are called recursively to continue animating scenes. Hence, while such pre-trained video models are general enough to generate short clips of arbitrary descriptions, at this time of writing, they still currently lack the controllability and continuity needed to be able to generate consistent and coeherent videos needed for games.

3 Methods

We identify two fundamental requirements for generating and simulating video games using video generation models. The first requirement is **game controllability**: in order for the generated video game to be deemed interactive, the player must be able to control the game precisely and reliably. The second requirement is **game state preservation**: the game state and parameters must be preserved and kept consistent throughout the duration of an episode of the game to maintain continuity.

To study the abilities of video models to satisfy these requirements, we design our model architecture and experiments based on the following assumptions. First, the game is controlled based on text instructions provided by a player. Text is a universal and generic interface for which game actions can be expressed. For very simple actions, these can be converted to simple text words. Second, the game state and parameters are entirely determinable from the game's visual output (a frame).

With the above assumptions, we learn a video diffusion model that is conditioned on text. To answer questions about the abilities of such video models for game controllability and continuity (state preservation) for video games, we limit the training of the model on gameplay data from just one specific video game. The scaling, generalization and creation of arbitrary video game environments requires much larger data requirements which we leave for future work. In our setting, a vidoe game generation model must be able to generate video that is consistent with playable actions provided in the form of text from this game. Specifically, our model takes an initial video frame of the game along with text of the desired action (e.g. "jump"), and learns to generate a sequence of frames that visually depicts the desired action.

Model Architecture. We adapt a video diffusion model proposed by [18] for this task. The model learns to approximate the distribution of subsequent frames $p(x_{1:T} \mid x_0, a_{text})$ given an input frame x_0 and a text description a_{text}. The model is based on a U-Net architecture that learns to predict the noise applied to the subsequent frames $\text{frame}_{1:T}$. Each downsampling and upsampling block of the U-Net [29] uses factorized spatial-temporal convolution: a convolution operation in the spatial dimension is first applied across all time steps, followed by a convolution in the time dimension applied across all spatial locations. Compared to a standard 3D convolution, the factorized spatial-temporal convolution significantly improves training and inference efficiency.

During training, the initial game frame is concatenated with T subsequent noisy frames. The action text is encoded into an embedding using CLIP [25]. The embedding is then fed to all blocks of the video diffusion U-Net model. The network is optimized using the mean squared error loss of the predicted noise on subsequent game frames.

During inference, an initial game frame along with the CLIP-embedded action text is passed to the model, and the model conditionally denoises the T subsequent frames to generate a video clip of the desired action. Because the model only generates a finite number of frames in each inference call, we can create arbitrarily long video clips by chaining together multiple model generations recursively. Specifically, the T-th frame of a generated sequence is used as the initial frame of the subsequent inference step. Crucially, chaining implicitly relies on our second assumption that the game state and parameters are entirely determinable from the game's visual output.

4 Experiments

4.1 Experimental Setup

For our study, we consider the Super Mario Bros game. While considering only 2-D game environments for simplicity, these games still have sufficient complexity such as some notions of simple physics (i.e. gravity, collisions etc.) and game interaction rules.

Super Mario Bros (SMB) is a 2D side-scrolling platform game. In SMB, players control the character Mario on-screen, with the goal of progressing through a sequence of levels to ultimately rescue a princess at the end of the final level. SMB contains a total of 32 levels, divided across 8 "worlds" (e.g. Level 4-1 refers to the first level in world 4). The relevant game mechanics are summarized in Table 1 and the full action space is provided in Appendix A.1.

Table 1: Summary of Super Mario Bros. Game Mechanics

Mechanic	**What It Does**
Movement	Move left/right and jump
Power-ups	Collect power-ups (Mushrooms, Fire Flowers, Starman) to gain abilities such as temporary invincibility or the ability to shoot fireballs
Enemies	Defeat by jumping or using fireballs
Obstacles	Navigate through gaps, blocks, spikes, and moving platforms
Death	Lose a life from colliding with enemies and dangerous obstacles, falling into holes, or exceeding the time limit
Lives	Begin with 3 lives. Lives can be gained from 1-Up Mushrooms or from collecting coins. The game ends when Mario runs out of lives

4.1.1 Dataset Processing and Curation

We use a publicly available dataset of frame-by-frame SMB gameplay data collected from a single player [23]. The dataset contains recorded gameplay from 280 distinct episodes, where each episode is an attempt to play through one of the levels of the game. Among the 280 episodes, 141 are winning episodes (i.e. the player successfully completes the level) and 139 are losing episodes, providing a balance between successful and failed demonstration data.

Each frame consists of the following: a 256x240 image frame of the game screen that the player observes, the frame number of the given episode, and the action made by the player.

To simplify the gameplay simulation, the action space was limited to only two actions, i.e. "run right" and "run right and jump" (hereafter referred to simply as "Run" and "Jump"). These two actions were chosen as they constitute a large fraction of recorded actions in our dataset, with "Run" and "Jump" representing 47% and 19% of all actions, respectively. Intuitively, these two actions alone equip Mario with sufficient movement capabilities to navigate through most of the game.

The dataset of the SMB gameplay was preprocessed to extract coherent sequences of frames depicting the targeted player actions to serve as demonstration data for model training. To capture a frame sequence of a particular action, the starting frame was identified, and then a fixed number of subsequent frames were captured together with the starting frame, forming the frame sequences of the desired player actions. We kept each sequence to 35 frames, as most of the desired player actions could be captured within this frame count. Depending on the desired temporal granularity, each frame sequence can be downsampled by by sampling a subset of frames in the sequence at uniform intervals. Further details on how we constructed these sequences are described in A.1.

We allocated frame sequences extracted from episodes of Level 1-1 for evaluation, and used sequences from episodes of all other levels for training. Table 2 shows the distribution of frame sequences extracted.

Action	No. of Training Sequences	No. of Evaluation Sequences
Run	8618	99
Jump	4634	49

Table 2: Distribution of frame sequences for training and evaluation.

4.1.2 Training and Inference

We train a video diffusion model with the architecture described in the Methods section. Action text is encoded using a pre-trained CLIP text encoder. The output tokens are aggregated by an attention-pooling network into an embedding vector which is then added to the time embedding of the video diffusion model. We downsample each 35-frame sequence by sampling 7 frames at uniform intervals. Each frame is downsized to 64x48 resolution.

The model is trained with Gaussian noise with zero mean and a variance that is controlled by a cosine beta schedule over 100 diffusion timesteps. The model is trained using MSE loss. Training was conducted over 300000 gradient steps with a batch size of 8 on a single NVIDIA RTX 4090 GPU. The training process took approximately 48 hours of compute. Further model training details are provided in the Appendix.

Using the same NVIDIA RTX 4090 GPU with 100 sampling steps, the MarioVGG model takes approximately 4 seconds to generate a 6-frame video sequence of the game at inference.

4.1.3 Evaluation Metric

To quantitatively assess the model's ability to generate realistic trajectories, we compute a normalized Learned Perceptual Image Patch Similarity (LPIPS) [41] score between actual and generated frame sequences. We opt for LPIPS as it is designed to measure perceptual similarity between images, which is in line with our goal of visually replicating gameplay footage generated by a game engine.

Concretely, for a sequence of n actual frames $S_{\text{a}} = \{s_{\text{a},1}, s_{\text{a},2}, \ldots, s_{\text{a},n}\}$, and corresponding sequence of model-generated frames $S_{\text{g}} = \{s_{\text{g},1}, s_{\text{g},2}, \ldots, s_{\text{g},n}\}$, the LPIPS is the mean LPIPS between frames, i.e.

$$\text{LPIPS}(S_{\text{a}}, S_{\text{g}}) = \frac{1}{n} \sum_{i=1}^{n} \text{LPIPS}\left(s_{\text{a},i}, s_{\text{g},i}\right)$$

We further normalize the LPIPS scores by computing the LPIPS score attained by a null model that simply regenerates the initial conditioning frame across all time steps, $S_{null} = \{s_{\text{a},0}\}^n$. Intuitively, the normalization enables us to quantify the improvement in perceptual similarity between the actual and generated game video, relative to a motionless video. It is worth nothing, however, that LPIPS

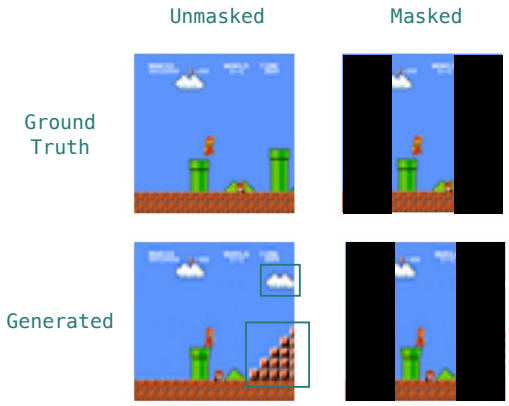

Figure 2: Illustration of the masking done to process frames when computing the LPIPS score to remove generated objects and artifacts and focus on similarity of actions.

uses extracted features from pre-trained ImageNet models to represent human vision, which may be overly sensitive towards certain image characteristics compared to a human.

To compute the normalized LPIPS score for an evaluation set E with M pairs of sequences $\{(S_{\text{a}}^{(j)}, S_{\text{g}}^{(j)})\}_{j=1}^{M}$

$$\text{Normalized LPIPS}(E) = \frac{\sum_{j=1}^{M} \text{LPIPS}(S_{\text{a}}^{(j)}, S_{\text{g}}^{(j)})}{\sum_{k=1}^{M} \text{LPIPS}(S_{\text{a}}^{(k)}, S_{\text{null}}^{(k)})}$$

Finally, the LPIPS score is computed over masked frames. Masking is important because in a given sequence of frames, MarioVGG will procedurally synthesize and generate novel tiles and environments that differ from the ground truth as the game scrolls forward, illustrated in Figure 2. Masking allows us to evaluate the distance on the generated movement of Mario, measuring the controllability, feasibility and closeness of the video actions generated w.r.t the ground truth, without unnecessarily penalizing the creative synthesis by MarioVGG which is a clear benefit.

4.2 Results

4.2.1 Quantitative Results

We evaluate the performance of MarioVGG on a held-out validation dataset consisting of 148 sequences. Figure 3 shows the evolution of the normalized LPIPS score as training progresses. The best normalized validation LPIPS score of 0.636 is observed after 210k training steps. Despite using just embedding distance of the frames as a score approximation, there is still a clear decreasing LPIPS score trend demonstrating learning of video sequences that get closer to what a ground truth Mario actions and game movement would look like. This score loosely suggests that MarioVGG generates sequences that have 37% better perceived similarity to actual game footage, relative to a motionless video.

4.2.2 Qualitative Results

Figure 4 shows side-by-side comparisons of video frames generated by MarioVGG (bottom row) and ground truth video frames from the validation dataset (top row), given the same initial frame and action. We can observe visually that the model very successfully animates and simulates the action and movement of Mario's similar to the actual game video. This demonstrates that MarioVGG has a good level of controllable video generation using the text actions provided. Additionally, we can also observe new tiles in the environment (ramp) that were generated procedurally by the model which is coherent with the surrounding environment, naturally preserving many aspects of the game state and flow. Please see Figure 8 in Appendix A.3 for a greater diversity of these generations and comparisons.

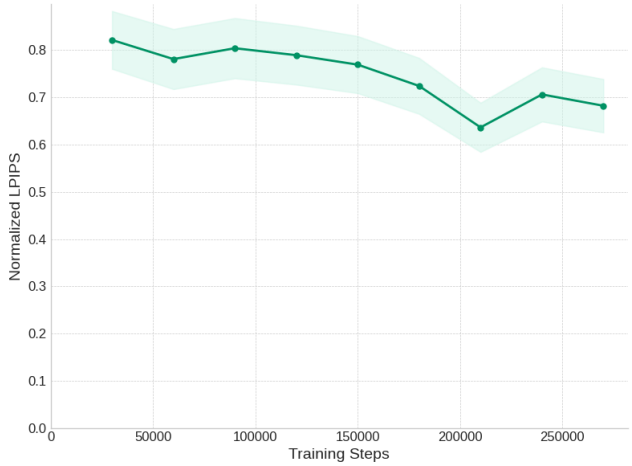

Figure 3: Normalized LPIPS scores on our validation dataset across training steps.

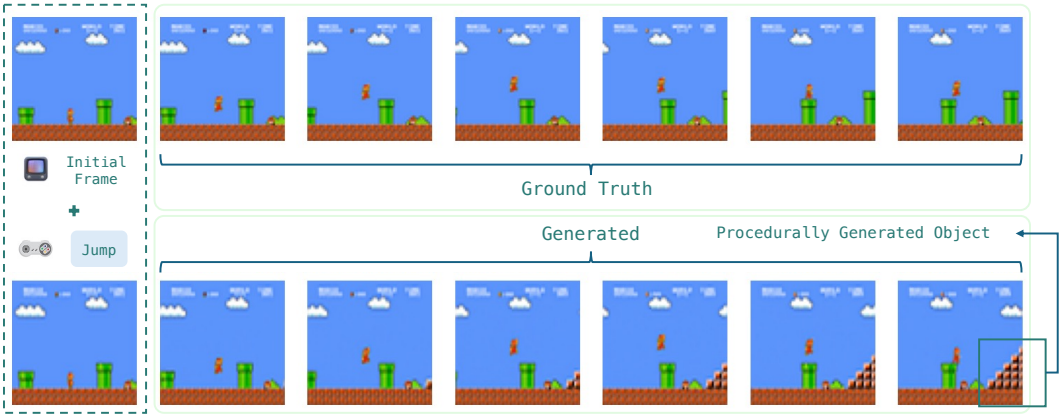

Figure 4: Comparison of video generated by the MarioVGG model and the ground truth sequence of frames, given the same initial frame and action. There is no visible difference in the action of the character in the game between. As it is a generative model, the frames generated also consist of procedural generated objects and levels.

Game Physics Simulation. We also highlight elements of the game mechanics and interactions rules that the model has learnt. In the top panel of Figure 5, we observe that MarioVGG succesfully simulates and generates the gravity mechanics of the game. The generated video demonstrates Mario falling off a platform as the "Run" action is used, all this while maintaining consistency of the surrounding environment and generating coherent new tiles. We also observe that the model also successfully learns interactions and collisions with objects. By conditioning the model generation on an initial frame in which Mario is positioned directly behind an obstacle, and providing a "Run" action text prompt the model generate gameplay video as shown in the bottom panel of Figure 5. We observe that the model correctly generates a video in which Mario does not move (while animation of the background continues) showing that the model has learnt that Mario cannot physically pass through obstacles. These observations are seen consistently across many of the examples shown demonstrating that the MarioVGG is able to learn the physics of the games purely from video frames in the training data without any explicit hard-coded rules (game rules are in the weights of the model). While there are still occasions in which failures occur in which the character disappears, a vast majority of the video generations are consistent.

Chaining. We also evaluate the ability of MarioVGG to continuously generate and maintain long-horizon video sequences by chaining sequences through the model. This provides a better illustration of end-to-end simulated gameplay. In Figure 6, we show six chained generations from MarioVGG

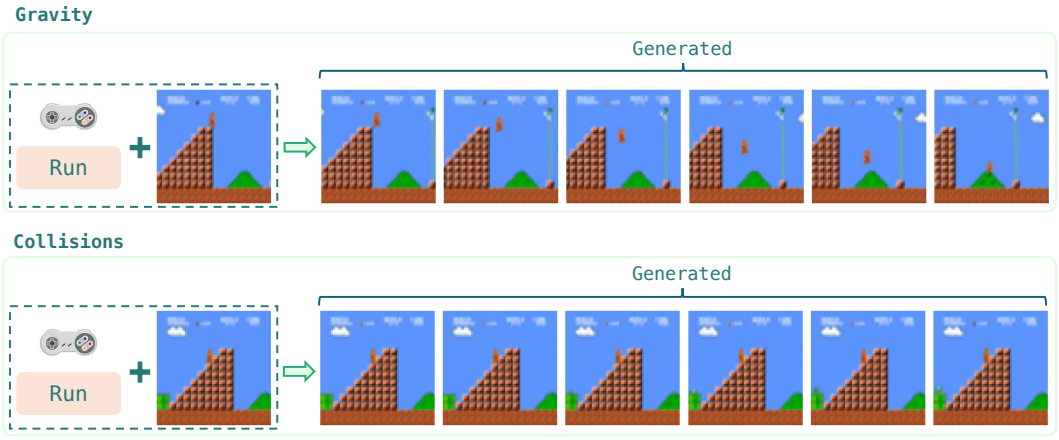

Figure 5: Examples of how MarioVGG has learned to simulate key elements of the game physics from actions taken such as gravity (top) and collisions with objects/items (bottom).

along with the corresponding action for each generation and where the final frame of each generated video sequence is used as the input frame for the next generation in the chain. We observe that MarioVGG is able to generate and carry coherent and consistent gameplay videos between successive generation steps, which means that we can generate gameplay videos that are arbitrarily long and are feasible. We also note that as the chain progresses, the model synthesizes the levels and environments completely independently that are coherent with the graphical language of the game. Only the first input frame is truly grounded from a true game state.

4.2.3 Scaling Resolution

In this subsection, we conduct experiments to scale the (i) frame rate and (ii) resolution of the videos generated by MarioVGG. First, we trained a model to generate 13 frames instead of 6 frames to evaluate this effect and the ability of the model to potentially either generate longer sequences or higher frame rate videos. Next, we trained another model with input frame dimensions of resolution 128x96 instead of 64x48 (as trained in our models presented) to evaluate the ability of the model and method to generate higher resolution videos. In both experiments, we kept the model architecture the same as in the previous sections with similar training parameters including training steps as well as diffusion steps.

Table 3 shows the normalized LPIPS scores obtained on our evaluation dataset. While the normalized LPIPS scores are higher than the scores obtained with our original model, they still demonstrate an improvement over a null model. Given that all these models were trained for the same number of steps, there could be potential room for improvement if provided with longer training runs when especially when scaling resolution. These results show that our methods can be extended to generate video games with higher video quality. In Figure 7, we compare the resolution quality between the two models trained on frames with different resolutions. We can clearly observe a higher resolution output generated which is to be expected, demonstrating that resolution generation generally dependent on the data the model is trained on and the compute.

Experiment	**Normalized LPIPS**
Baseline	0.636
Scaling Output Length	0.719
Scaling Image Resolution	0.798

Table 3: LPIPS scores from our experiments on scaling generation of videos to higher resolution.

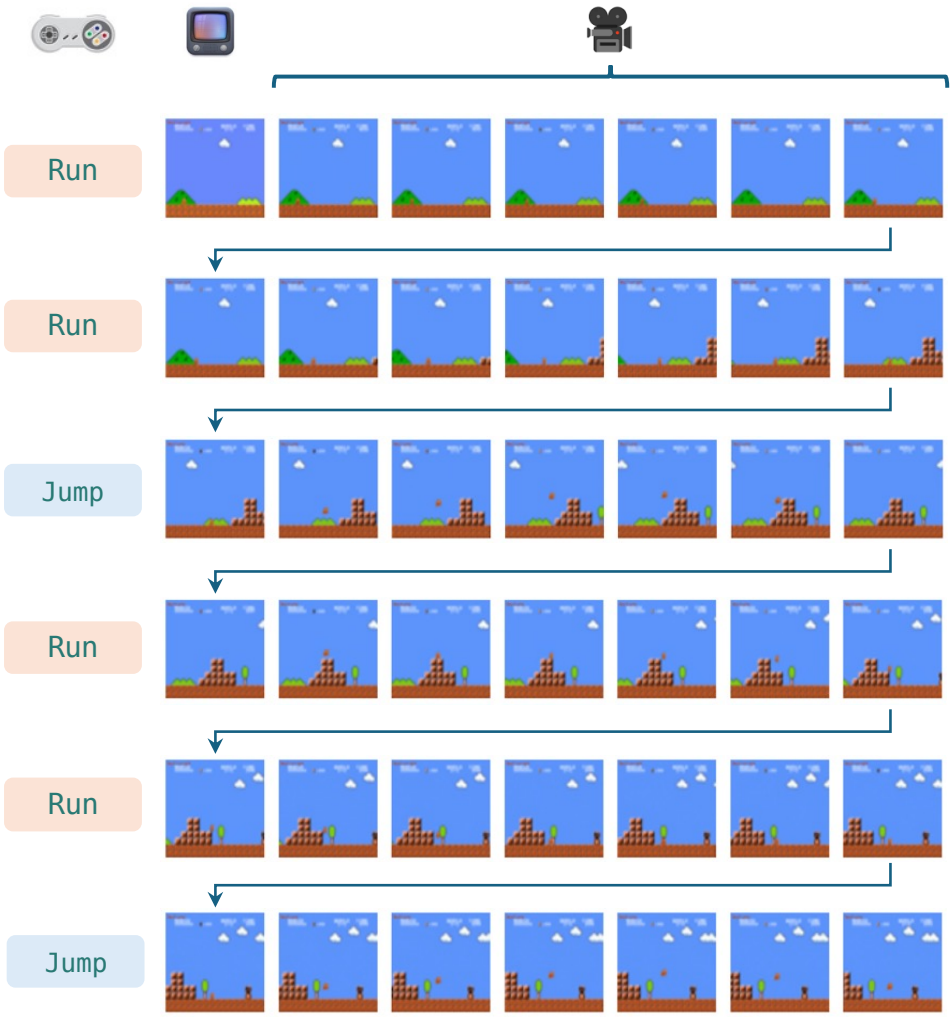

Figure 6: Video generated when chained across multiple actions where the last frame of one generation is used as the initial frame for the next generation. Video shows controllability and coherence over a long time horizon.

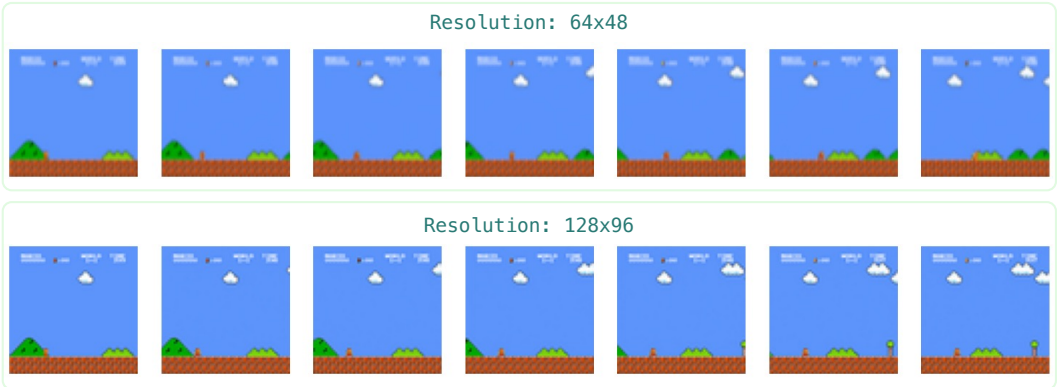

Figure 7: Comparison between the baseline model (top) and a one that is trained to generate higher resolution model (bottom).

5 Related Work

Our work is related to the wider body of work on video generation models. These models conventionally predict a sequence of frames when conditioned or provided on an initial image [4; 3; 16; 12]. In many cases, these models serve to animate the provided image or scene and are not controllable as the outcomes of the sequence of frames generated cannot be controlled, only predicting the most likely next frames.

Another class of models are controllable video generation models, which are commonly provided signals for conditioning the generation process. These conditioning signals can take multiple forms like specific vectors [17] or modalities like text. Most common recent controllable models are text-to-video models, due to the general interface and vast representation capabilities of text [36; 5; 38]. Such models have been shown to work on small scale datasets and constrained text-space [18] and also scale to heavily captioned large-scale datasets. An alternative to captioning and text-labels that commonly are required for text-to-video models, are controllable latent vectors or latent actions [19; 6] that are also learned from the data. This enables video models to be more easily trained on larger-scale video data as it does not have to be captioned or labelled but such a representation comes at a cost of no semantics that come with text. Similar to our work, Genie [6] also learns a video model capable of simulating 2-D platformer games.

Another closely related family of work are world models, which are models that have a representation and understanding of how the world or an environment operates, capturing essential features and relationships. Similar to video models, world models should encode both spatial and temporal information to act as real-world simulators [38] which can be used for prediction, planning and decision making [8]. This is also commonly known as dynamics models which has a history in robotics, optimal control, model-based reinforcement learning [33; 37]. Most of the work in this area involve learning forward dynamics models which predicts the next state given the current state and action or control input. States can be low-dimensional representations of a system [7] or higher-dimensional ones such as pixel or image-based representations [13; 14; 15] The video game models we are learn are forward dynamics models, but predicts a sequence of future frames rather than just a single frame.

Closest to our work is concurrently released work GameNGEN [35] which also explores using diffusion models to be used directly as game-engines. Both work similarly operates over just a single game domain and focuses on controllability over player and consistency and coherence over game-play sequences. Where we differ is in the modelling choices and the game domain. MarioVGG generates a sequence of frames while conditioning on a single frame and a text-encoded action, while Valevski et al. [35] conditioned the model on a sequence of previous states and actions (i.e. vectors) to predict a single frame. One major focus and result in GamNGEN was the interactive generation times which MarioVGG is not capable of at the moment.

6 Discussion and Conclusion

Limitations and Future Work. We discuss several limitations of MarioVGG relating to the two fundamental requirements for video game generation (game controllability and game state preservation) highlighted in this work. For game controllability, unlike game engines which enable a deterministic relationship between the player's input and the game response, MarioVGG is highly dependent on the quality and scale of data available, which opens up the possibility for generated trajectories to be inconsistent with the player's input given scenes and actions that are out of distrbution or diverge. Video models are also probabilistic in nature and weobserve that the input action text is not obeyed all the time. Some examples of failure cases can be observed in Figure 12 in the Appendix. However, we believe this due to the limited gameplay data. Potential improvements could be done with some data augmentation methods, and scaling and collecting more diverse gameplay data through reinforcement learning agents. Architecturally, there is also more to be explored in conditioning the model on previous frames and actions to enhance consistency.

For game state preservation, we note that MarioVGG currently does not have an explicit terminal state. Thus, even if MarioVGG generates a trajectory resulting in a terminal state, it will continue generating subsequent trajectories when prompted, instead of reverting the initial game state to the latest checkpoint. While this can be solved using a network head dedicated for tracking game states

outside of information that can be captured from video frames, an open research question is how terminal states are decided in a fully generative game that has no "ground truth" equivalent and if the terminal state can also be included in a prompt-like description.

From a game development perspective, there are also some practical limitations. The non-negligible inference time of the model means that a player would need to wait several seconds for MarioVGG to generate the trajectory of their desired action which is not practical and friendly for interactive video games. However, these models have yet to be optimized for inference and could be potentially sped up to interactive speeds through the use of weight quantization, and rewriting code in lower-level languages to remove runtime overhead. We would also like to not that this all experiments and inference was done on a single consumer grade GPU only.

Other future directions also include the option for controllable environment generation. Levels and objects in MarioVGG are not controllable and are generated procedurally. While this procedural generation of environments is a positive and desirable features, having controllability over the generation of environment features and elements is also potentially important. For example, where prior work which has used LLMs to output tiles from text descriptions which are then parsed in to the game engine [32].

Lastly, while we only focus on a single game and domain of Mario, the long-term goal of such a research direction is to be able to use these models as game engines and democratize interactive game creation development the same way LLMs have democratized programming. Hence, a step in this direction could be to improve and extend our model's ability to generate gameplay videos beyond Super Mario Bros. to generate novel variations in Super Mario Bros. art styles on the fly, or even accommodating general 2-D platformers altogether. Training data augmentation is a potential approach to generalize our model beyond SMB specifically, such as the use of text-guided diffusion models and inpainting techniques on existing game play data to generate synthetic scenes to be used to train the model [40].

Conclusion In summary, this work presents a technical report and investigation into the use of video generation models for interactive and controllable video games, by passing game engines. We highlight some desiderata and key features that are required of video generation models if they are to be used as potential game engines namely (i) controllable videos and actions, and (ii) game state preservation and continuity through consistent and coherent frames over long sequences. We then briefly explore some current limitations and shortcomings of pre-trained general video generation models for this application. To demonstrate and answer if video generation models could potentially have this capability in the future, we focus on developing a model that perform reliable and controllable video game generation on just a single 2-D game domain of Mario, which consists of game mechanics such as interactions and game rules as well. Trained on just a single consumer-grade GPU, our text-to-video diffusion model, MarioVGG can generate controllable game frames that are consistent and follow game mechanics.

While replacing game development and game engines completely using video generation models might still not be practical and plausible at the moment due to the lack of generality and control in many aspects of video generation, we show that it is possible and an option with just a limited set of data on a single game domain. With an increase in scale and diversity of data, using video generation models for such an application should not be discounted.

References

[1] Haiper - AI video generator. `https://haiper.ai/home`, 2024. Accessed: August 19, 2024.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.

[4] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent

diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22563–22575, 2023.

[5] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024. URL `https://openai.com/research/video-generation-models-as-world-simulators`.

[6] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

[8] Yilun Du, Sherry Yang, Pete Florence, Fei Xia, Ayzaan Wahid, Pierre Sermanet, Tianhe Yu, Pieter Abbeel, Joshua B Tenenbaum, Leslie Pack Kaelbling, et al. Video language planning. In *The Twelfth International Conference on Learning Representations*, 2023.

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[10] Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Large language models and games: A survey and roadmap. *arXiv preprint arXiv:2402.18659*, 2024.

[11] Rohit Girdhar, Mannat Singh, Andrew Brown, Quentin Duval, Samaneh Azadi, Sai Saketh Rambhatla, Akbar Shah, Xi Yin, Devi Parikh, and Ishan Misra. Emu video: Factorizing text-to-video generation by explicit image conditioning. *arXiv preprint arXiv:2311.10709*, 2023.

[12] Yuwei Guo, Ceyuan Yang, Anyi Rao, Zhengyang Liang, Yaohui Wang, Yu Qiao, Maneesh Agrawala, Dahua Lin, and Bo Dai. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning. *arXiv preprint arXiv:2307.04725*, 2023.

[13] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. URL `https://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution`. `https://worldmodels.github.io`.

[14] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

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

[16] Yingqing He, Tianyu Yang, Yong Zhang, Ying Shan, and Qifeng Chen. Latent video diffusion models for high-fidelity long video generation. *arXiv preprint arXiv:2211.13221*, 2022.

[17] Jiahui Huang, Yuhe Jin, Kwang Moo Yi, and Leonid Sigal. Layered controllable video generation. In *European Conference on Computer Vision*, pages 546–564. Springer, 2022.

[18] Po-Chen Ko, Jiayuan Mao, Yilun Du, Shao-Hua Sun, and Joshua B Tenenbaum. Learning to Act from Actionless Videos through Dense Correspondences. *arXiv:2310.08576*, 2023.

[19] Willi Menapace, Stephane Lathuiliere, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. Playable video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10061–10070, 2021.

[20] Timothy Merino, Roman Negri, Dipika Rajesh, Megan Charity, and Julian Togelius. The five-dollar model: generating game maps and sprites from sentence embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 107–115, 2023.

[21] Muhammad U Nasir, Steven James, and Julian Togelius. Word2world: Generating stories and worlds through large language models. *arXiv preprint arXiv:2405.06686*, 2024.

[22] OpenAI. Sora: First impressions. `https://openai.com/index/sora-first-impressions/`, 2023.

[23] R.C. Pinto. Super mario bros. gameplay dataset. `https://github.com/rafaelcp/smbdataset`, 2021.

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[25] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL `https://arxiv.org/abs/2103.00020`.

[26] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[27] Runway Research. Gen2: Text driven video generation. 2023. URL `https://research.runwayml.com/gen2`.

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

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[30] Noor Shaker, Julian Togelius, and Mark J Nelson. Procedural content generation in games. *Computationl Synthesis and Creative Systems*, 2016.

[31] Abhishek Sharma, Adams Yu, Ali Razavi, Andeep Toor, Andrew Pierson, Ankush Gupta, Austin Waters, Aäron van den Oord, Daniel Tanis, Dumitru Erhan, Eric Lau, Eleni Shaw, Gabe Barth-Maron, Greg Shaw, Han Zhang, Henna Nandwani, Hernan Moraldo, Hyunjik Kim, Irina Blok, Jakob Bauer, Jeff Donahue, Junyoung Chung, Kory Mathewson, Kurtis David, Lasse Espeholt, Marc van Zee, Matt McGill, Medhini Narasimhan, Miaosen Wang, Mikołaj Bińkowski, Mohammad Babaeizadeh, Mohammad Taghi Saffar, Nando de Freitas, Nick Pezzotti, Pieter-Jan Kindermans, Poorva Rane, Rachel Hornung, Robert Riachi, Ruben Villegas, Sander Dieleman Rui Qian, Serena Zhang, Serkan Cabi, Shixin Luo, Shlomi Fruchter, Signe Nørly, Srivatsan Srinivasan, Tobias Pfaff, Tom Hume, Vikas Verma, Weizhe Hua, William Zhu, Xinchen Yan, Xinyu Wang, Yelin Kim, Yuqing Du, and Yutian Chen. Veo. 2024. URL `https://deepmind.google/technologies/veo/`.

[32] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[34] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[35] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines, 2024. URL https://arxiv.org/abs/2408.14837.

[36] Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual descriptions. In *International Conference on Learning Representations*, 2022.

[37] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning, 2020. URL https://openreview.net/forum?id=H1lefTEKDS.

[38] Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 2023.

[39] Sherry Yang, Jacob Walker, Jack Parker-Holder, Yilun Du, Jake Bruce, Andre Barreto, Pieter Abbeel, and Dale Schuurmans. Video as the new language for real-world decision making. *arXiv preprint arXiv:2402.17139*, 2024.

[40] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Jodilyn Peralta, Brian Ichter, et al. Scaling robot learning with semantically imagined experience. *arXiv preprint arXiv:2302.11550*, 2023.

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

A Appendix

A.1 SMB Dataset Action Space

The player actions are represented by an unsigned 8-bit integer from 00000000_2 (0_{10}) to 11111111_2 (255_{10}) where each bit corresponds to a button used during the gameplay in the captured frames, in the following order from MSB to LSB: Up, Down, Left, Right, A, B, Start and Select. Table 4 shows some examples of actions and their corresponding decimal integer representations.

Action	Up	Down	Left	Right	A	B	Start	Select	Integer (Decimal)
Walk right	0	0	0	1	0	0	0	0	16
Run right	0	0	0	1	0	1	0	0	20
Walk left	0	0	1	0	0	0	0	0	32
Jump	1	0	0	0	0	0	0	0	128
Run right and jump	1	0	0	1	0	1	0	0	148
Run left and jump	1	0	1	0	0	1	0	0	164

Table 4: Examples of action representation in the action space

The frame sequences of the "run right" action were relatively simple to capture, as Mario only ran to the right on a flat ground or an object. It was easier to determine the starting frame of the "run right" action for a complete frame sequence to be captured, as the frame would have the action integer 20 and provided its previous frame would depict a different action. Then, a certain number of the subsequent frames were captured. The frame sequences captured might include the character running into an obstacle or falling off the ground or the object it was running on. This was acceptable as it could introduce the game mechanics in the model training.

However, the frame sequences of a "run right and jump" action were more complicated. The starting frame was harder to determine because the character was already leaving the ground or the object from which it jumped when action integer 148 was captured. Thus, the algorithm needed to be designed so as to look back on previous frames to find the most recent frame in which the character was just about to leave the ground or the object from which it jumped. Usually, Mario was walking or running right before the jump, thus the most recent frame with action integer 16 or 20 was considered to be the starting frame of the jumping action. Also, depending on Mario's surrounding environment, the number of frames needed for Mario to complete the jumping action could vary. For example, the character took the highest number of frames to complete the action of jumping from a floating object and landing on an open ground; whereas the character took the lowest number of frames to complete the action of jumping from the group and onto an obstacle like pipes, cannons or rock hills. From our observations of the dataset inspection, the total length of the frame sequence of jumping right needed to be at least 32. The subsequent frames captured must not include any action making the character moving to the left either amid the air or on the ground, as this would introduce noise to the training dataset.

A.2 Model Training Details

Additional model training details are provided in Table 5.

A.3 Supplementary Results

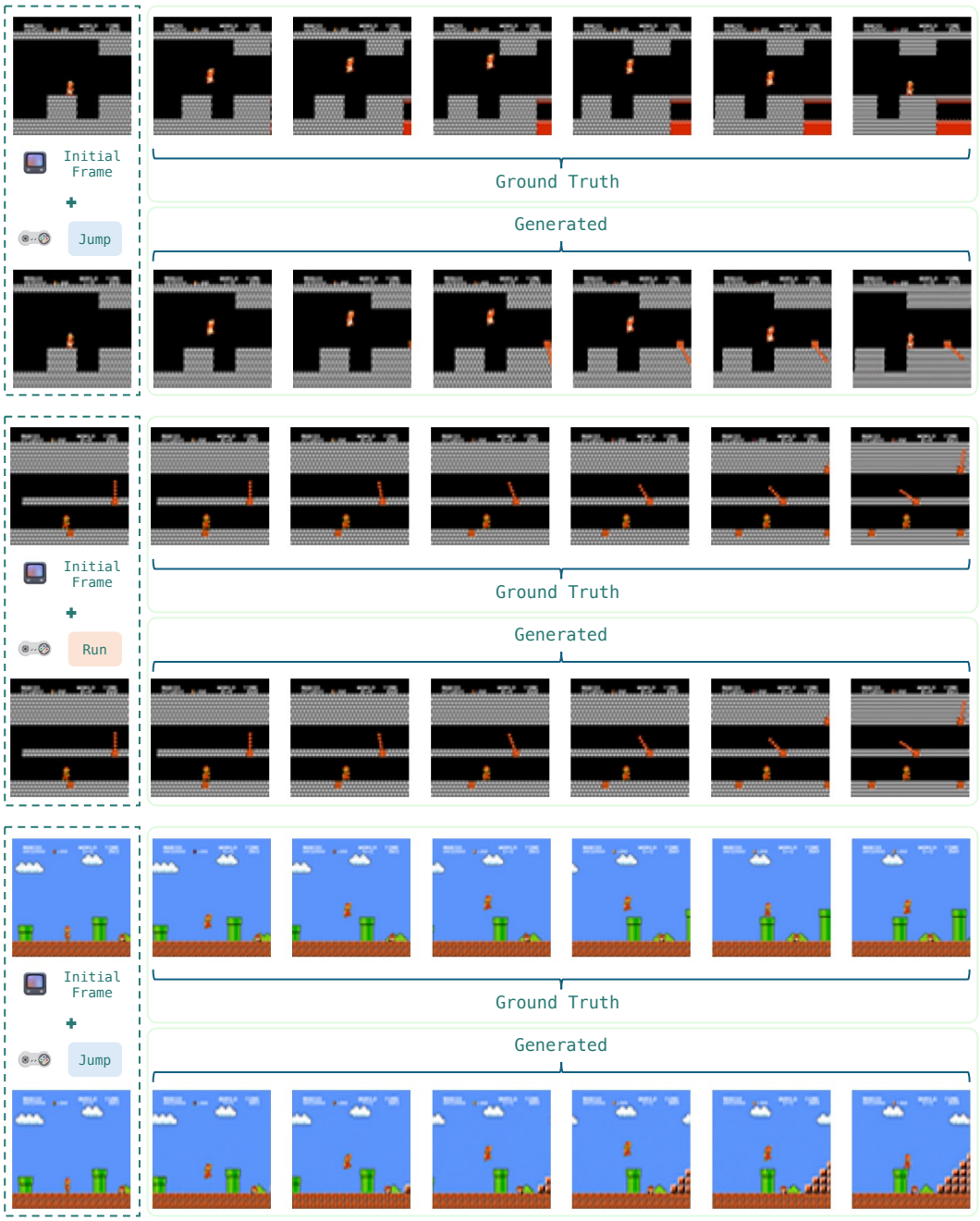

Figure 8: More diverse examples and demonstrations of MarioVGG across different levels that have different styles.

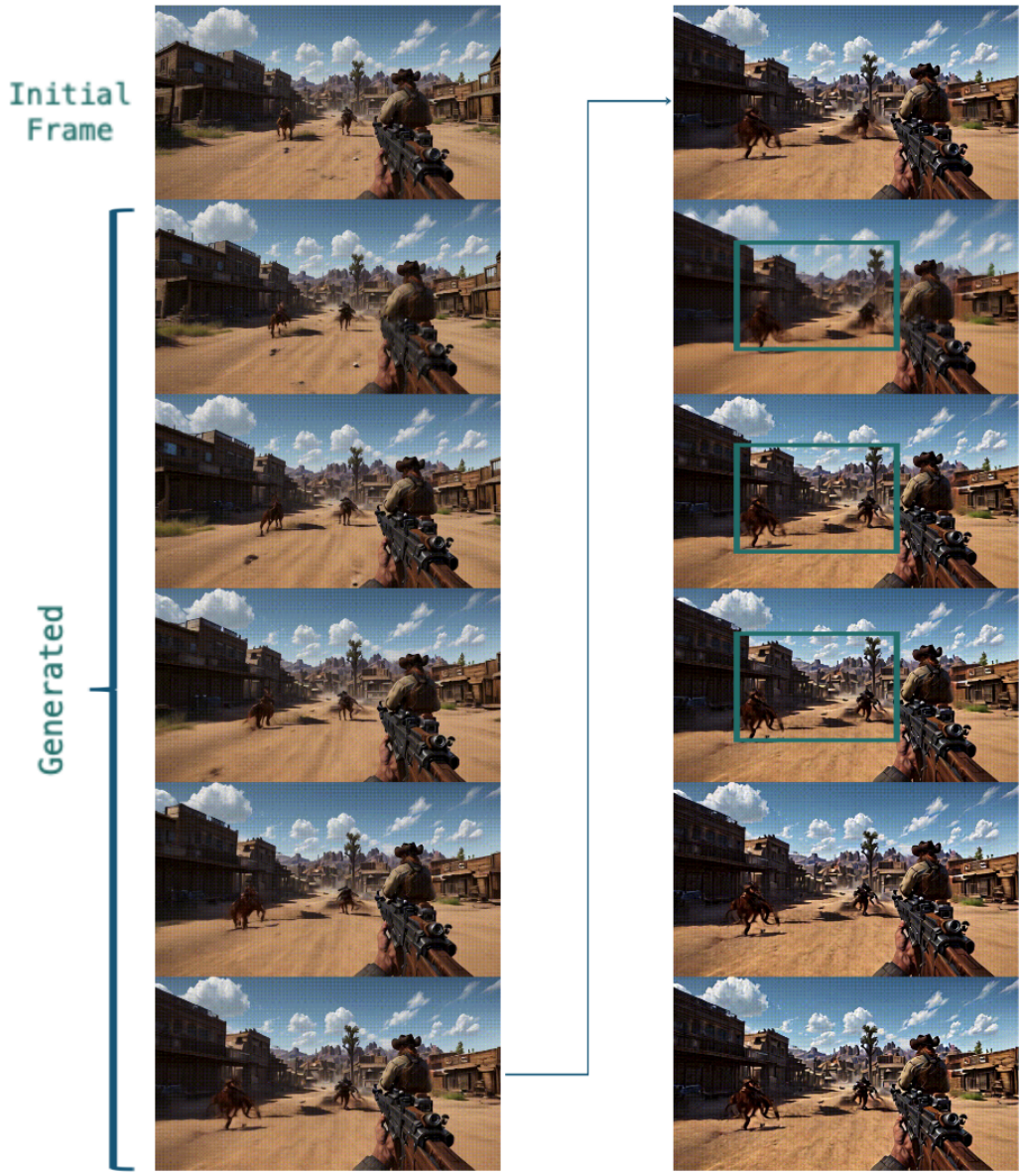

Figure 9: An example of SVD's shortcomings in generating coherent videos across multiple inference calls. In the first column on the left, we prompt SVD to generate game footage given an initial screenshot of a hypothetical first-person shooter game. We observe that SVD initially generates coherent video of the character progressing forward on the screen. However, after attempting to generate subsequent footage from the final frame of the first video (right column), the generated video is no longer coherent: the character's motion abruptly stops, and on-screen movement becomes blurry (see boxed region).

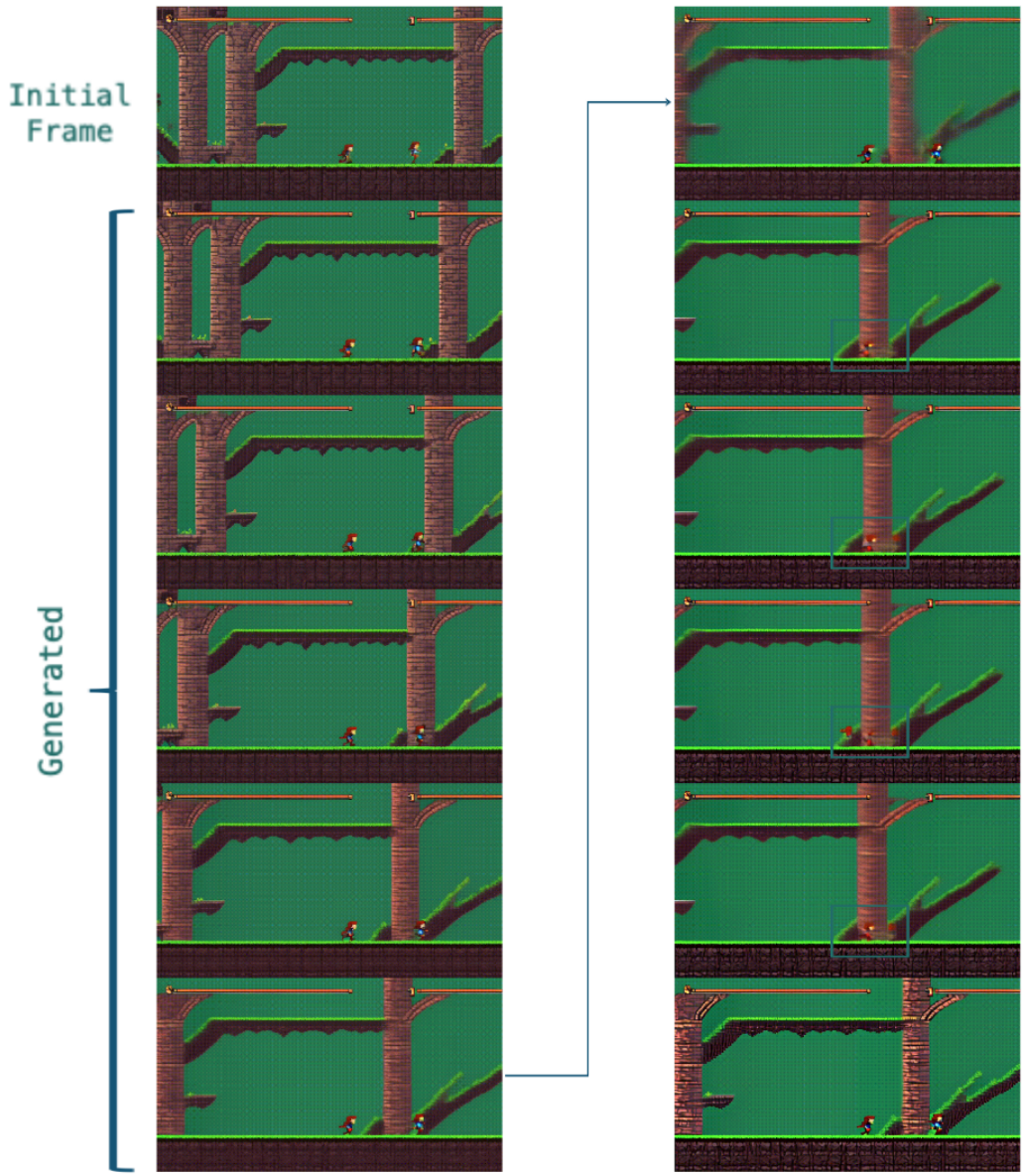

Figure 10: Another example of SVD's shortcomings in generating coherent videos across multiple inference calls. In the first column on the left, we prompt SVD to generate game footage given an initial screenshot of a hypothetical 2-D platformer game. We observe that SVD generates a coherent video of the game characters moving to the right of the environment. However, after attempting to generate subsequent footage from the final frame of the first video (right column), the generated video is no longer coherent: the characters appear blurry and jittery around the tree (see boxed region), instead of continuing their movement to the right.

Parameter	Value
Resolution	48x64
Base Channels	160
Num. Res Blocks	3
Num. Network Parameters	166M
Diffusion Timesteps	100
Diffusion Beta Schedule	cosine
Batch Size	8
Training Steps	300k
Learning Rate	1e-4
Loss	L2
Optimizer	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.99$)
EMA Update Steps	10
EMA Decay Factor	0.999

Table 5: Training configuration for MarioVGG.

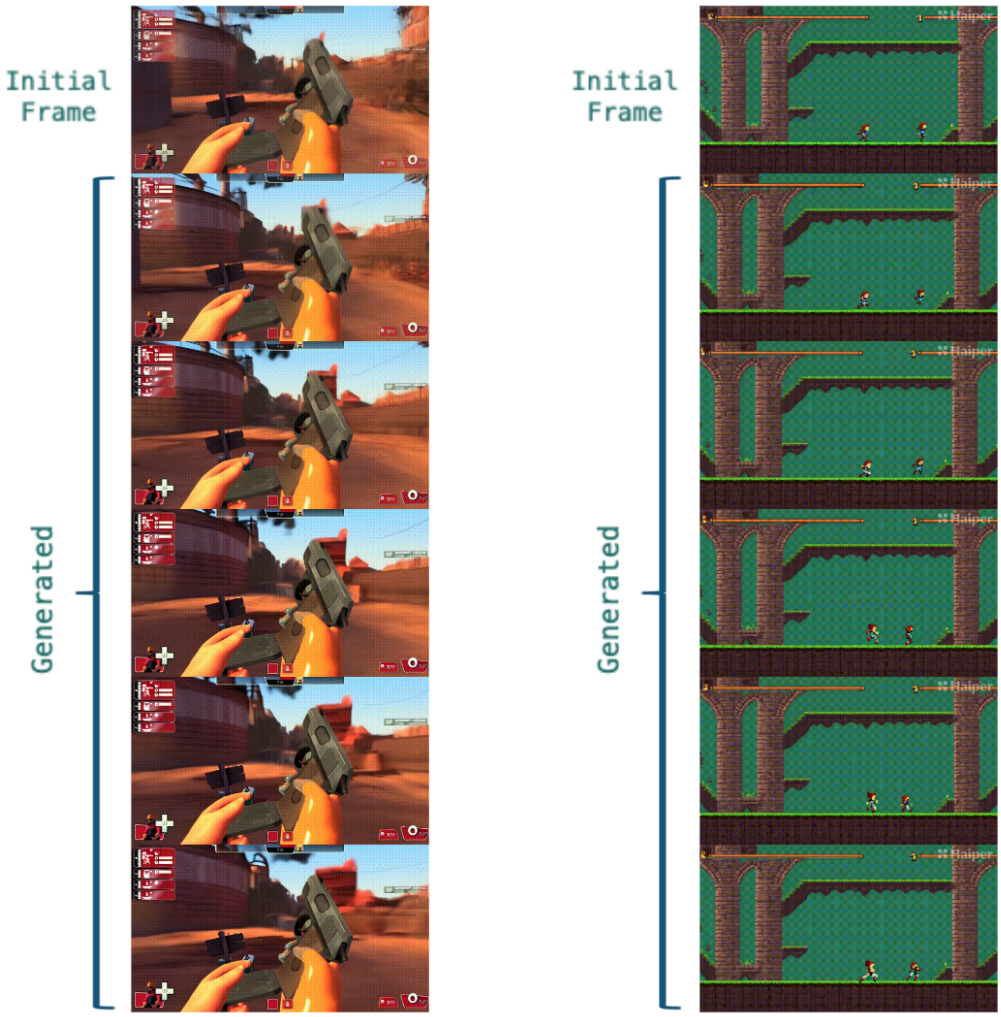

Figure 11: Examples of SVD and Haiper's inability to generate fine-grained player actions. In the column on the left, the model was prompted to generate a video corresponding to the action "reload gun", but instead generated motion in the background and kept the reloading animation unchanged across frames. In the column on the right, the model was prompted to generate a video corresponding to the action "jump", but generated little to no vertical movement of the game characters.

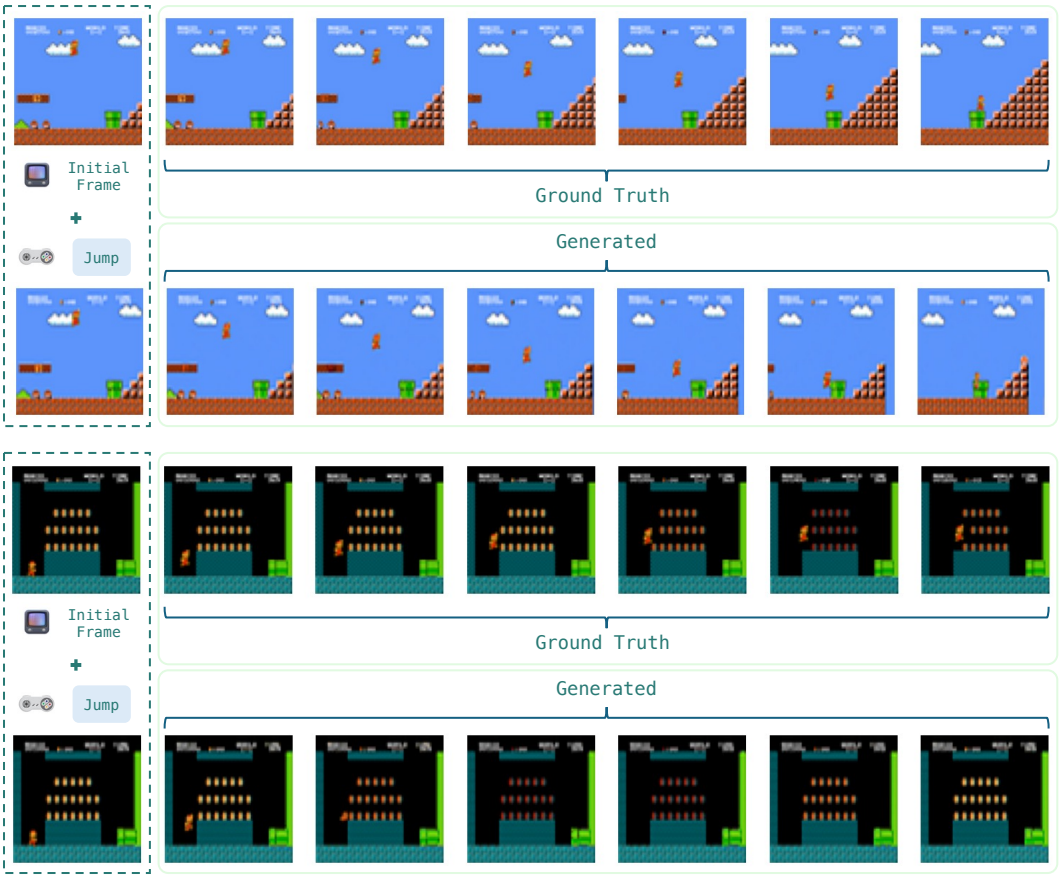

Figure 12: Illustration of some failure cases of the MarioVGG model. (Top) when completing the "jump" action, Mario lands inside the obstacle not obeying collisions. (Bottom) Mario suddenly disappears while jumping on to a platform with many coins that have similar appearances to Mario