Playable Video Generation: Supplementary Material

Willi Menapace
University of Trento
willi.menapace@unitn.it

Stéphane Lathuilière*
LTCI, Télécom Paris
Institut Polytechnique de Paris
stephane.lathuiliere@telecom-paris.fr

Sergey Tulyakov*
Snap Inc.
stulyakov@snap.com

Aliaksandr Siarohin
University of Trento
aliaksandr.siarohin@unitn.it

Elisa Ricci
University of Trento
Fondazione Bruno Kessler
e.ricci@unitn.it

In this Supplementary Material, we provide additional details regarding the neural network architecture used in our experiments (Sec. 1), our training procedure (Sec. 2) and the baseline selection process (Sec. 3). Then, we detail the results of our human evaluation in Sec. 4. Finally, we report more qualitative results in Sec. 6. Importantly, this document is completed by an accompanying website composed of two parts. The first (see play-bair.html, play.html and play-tennis.html) contains fully playable demos of CADDY running locally in the browser, the second (see main.html) shows qualitative results. Note that, to guarantee better compatibility across devices, our browser demo runs on CPU and may require up to ten seconds to load, while our complete model runs on GPU in real-time. Due to the limitations on the size of the submission files, it was possible to include only the model trained on the *Atari Breakout* dataset. Also, *html* format offers more possibilities in term of visualization (*e.g.* videos) and should be favored for qualitative evaluation.

1. Architecture details

In this section, we report further details regarding our network architecture. We show a detailed view in Fig. 1.

Block details. We mainly employ six types of blocks to build our architecture: convolution blocks, residual blocks, up-sampling blocks, convolutional LSTMs, fully connected layers and Gumbel-Softmax sampling blocks. All our blocks use Leaky-ReLu activations with the exception of convolutional LSTM blocks and of final convolutional blocks producing the outputs, which are terminated by a $\tanh$ function. Down-sampling is achieved using average-pooling. Our up-sampling blocks instead make use of a convolution and bilinear interpolation. We make use of 3×3 convolutions in all the layers, with the exception of the convolution outputting the predicted frame at the original resolution, which uses 7×7 filters.

Multiresolution. As explained in Sec. 3.3 of the main paper, the decoder network D outputs images at multiple resolutions. Practically, we produce two lower resolution versions of the output image, one with halved resolution and one with one-fourth resolution. These are produced by two auxiliary convolutional blocks which take as input the features produced by the up-sampling layers at the corresponding resolution.

*The second and third authors contributed equally to the work.

	T_f	T_{initial}	T_{final}	*batch size*	K
Atari Breakout	6	7	9	8	3
BAIR	6	7	12	8	7
Tennis	6	7	12	6	7

Table 1: Hyperparameters used on the different datasets.

2. Training details

Optimization. In all our experiments, we use an Adam optimizer with a fixed learning rate of $2e-4$.

Sequence length scheduling. During the initial phase of training, we notice greater stability with small values of T, while, on the other hand, training with large values of T increases the quality of long generated sequences. For this reason, we adopt a training scheme with a variable value of T. In particular, every 5000 iterations, we increase the current value of T by 1 until the target value. Tab. 1 shows the hyperparameter configurations for our datasets. In all the experiments, we set the initial value of T to 7 and use $T_f=6$ context frames.

Estimation of the number of actions K. Our method requires the number of actions K to be provided as a hyperparameter. For the *Atari Breakout* dataset, we pose $K=3$

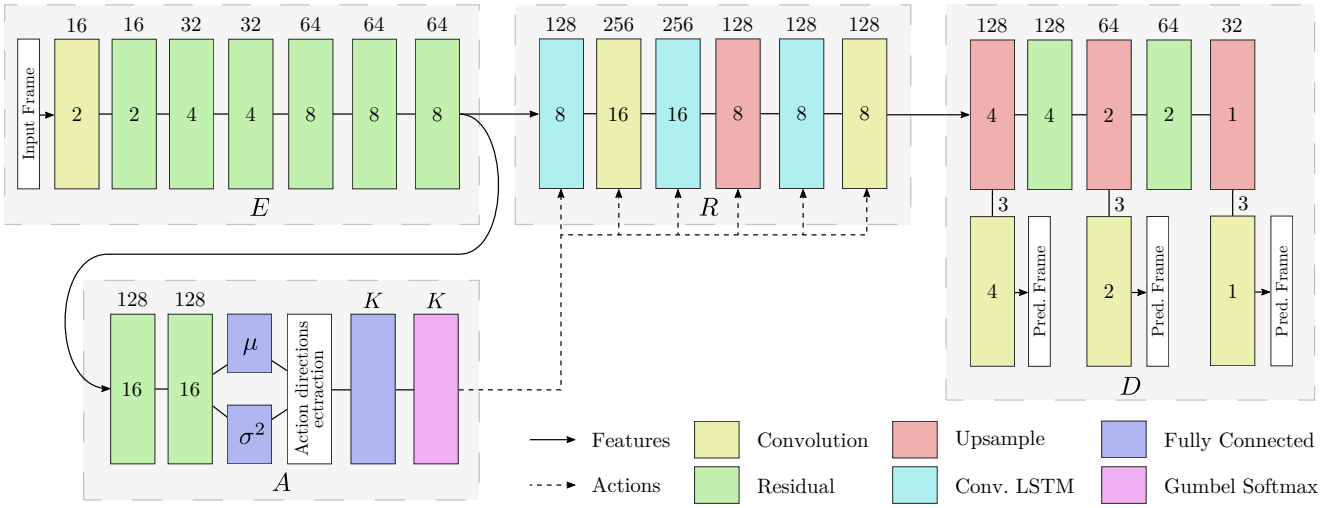


Figure 1: Overview of CADDY's architecture used for the *BAIR* and the *Tennis* datasets. On the *Atari Breakout* dataset, due to its simplicity, a version with lower capacity was employed. The model is composed of four main blocks: the encoder network E, the dynamics network R, the action network A and the decoder network D. We indicate the number of features at the top of each block. At the center, we show the factor such that the resolution of the output of the block is the input resolution divided by the indicated factor.

following the number of discrete actions in the real game. *BAIR* and *Tennis*, however, do not possess an intrinsic discrete action space, so the number of discrete actions required to capture the dynamics of the environment must be estimated. From initial experiments, we observe that this number can be over-estimated without negative consequences on the training process, with the model using extra actions to learn variations on the same action. On *BAIR*, we estimate $K=7$ which allows 2 movements on each of the 3 axes to be learned, plus a no movement action. On *Tennis*, we also choose $K=7$, expecting to learn 2 movements on the horizontal axis and 2 movements on the vertical axis, a *Stay* action, a *Hit the ball* action and an extra action that allows the model, if necessary, to learn an additional behavior of the player.

Gumbel-Softmax temperature annealing. Hard Gumbel-Softmax [8] discrete action sampling ensures that the action component a is truly discrete. Using a hard sampling strategy producing one hot vectors at the beginning of the training process, however, caused optimization difficulties. For this reason, we adopt a soft Gumbel-Softmax sampling approach. At the beginning of training, we perform sampling with a temperature of 1.0 which does not enforce a very low entropy on the sampled action vector a. As the training progresses, we linearly reduce the sampling temperature to 0.4 at step 20.000, enforcing that the sampled values of a are similar to one hot vectors.

Loss weights. We follow the hyperparameter selection procedure explained in Sec. 3.3 of the main paper to estimate the loss weights on the *Tennis* dataset. The mutual information maximization loss λ_{act} is used with weight 0.15, λ_{rep} is set to 0.2, and λ_{KL} is used with weight $1e{-}4$, while λ^{a}_{rec} is posed to $1e{-}5$. We found that these same values produce similar optimization behaviors on the *BAIR* and on the *Atari Breakout* datasets, so we use the same loss weights for all the experiments.

Pretraining. We notice that convergence speed is increased if the encoder network E and the decoder network D are initialized to perform frame reconstruction. For this reason, we integrate a short pretraining phase into our approach. In particular, instead of computing s_t using the dynamics network R, we employ a small auxiliary network to directly translate f_t to s_t so that E and D can be trained in isolation on the reconstruction task. In this phase, the dynamics network produces a reconstruction of s_t which we call $\hat{s}_t$, and a reconstruction loss between s_t and $\hat{s}_t$ is imposed. Gradients for this loss, however, are propagated through $\hat{s}_t$ only. The other loss terms remain unaltered.

Training times and GPU memory usage. We report in Tab. 2 the memory requirements and training times for our method on the different datasets and compare the results with the baselines. CADDY requires 16GB of GPU memory to train on the *Atary Breakout* and *Tennis* dataset, and 44GB on the *BAIR* dataset due to the increased resolution. In contrast, SAVP+ requires between 64GB and 128GB of memory. Moreover, training times for our method vary between 68 and 320 GPU hours, while SAVP+ requires significantly longer times between 730 and 1730 GPU hours.

	Atari Breakout	*BAIR*	*Tennis*
MoCoGAN [12]	16GB, 23h	16GB, 23h	16GB, 36h
MoCoGAN+	16GB, 72h	16GB, 96h	16GB, 39h
SAVP [10]	32GB, 144h	32GB, 144h	16GB, 144h
SAVP+	128GB, 1460h	128GB, 1730h	64GB, 730h
CADDY (Ours)	16GB, 107h	44GB, 320h	16GB, 68h

Table 2: GPU memory requirements in GB and time in GPU hours for training the different methods on the chosen datasets. CADDY trains significantly faster and with lower memory requirements than the SAVP+ baseline.

	FVD↓	Code Available
MoCoGAN [12]	503	✓
CDNA [5]	297	✓
SV2P [1]	263	✓
SVG-LP [4]	257	✓
SRVP [7]	181	✓
VideoFlow [9]	131	
SAVP [10]	116	✓
DVD-GAN-FP [3]	110	
TriVD-GAN-FP [11]	103	
Video Transformer [13]	94	

Table 3: FVD scores and code availability of a selection of video prediction methods on the *BAIR* dataset in 64x64 resolution. Note that CADDY is not listed since it is not a video prediction method.

3. Baselines selection

Since we present the first method for unsupervised PVG, we select a set of baselines from existing video prediction methods to adapt them to this new task. In Tab. 3, we compare existing video prediction methods in terms of FVD in the video prediction task on a 64x64 version of the BAIR dataset, which we use as a benchmark to guide selection. As the best and second-best performing methods with code publicly available, we choose SAVP [10] and SRVP [7] as baselines. Despite showing reduced performance, we choose MoCoGAN [12] as an additional baseline because its InfoGAN [2] loss for action learning makes it a good candidate for adaptation to the PVG problem. Note that our approach is not included in this comparison since CADDY is not designed for future frame prediction.

4. Human evaluation details

In Figs. 2, 3, 4, 5 and 6, we show user votes obtained during the AMT user study. Rows correspond to the different actions learned by the models that were used to generate the evaluated sequences, columns correspond to the action options that were presented to the users. The MoCoGAN, MoCoGAN+ and SAVP+ baselines do not learn a consistent

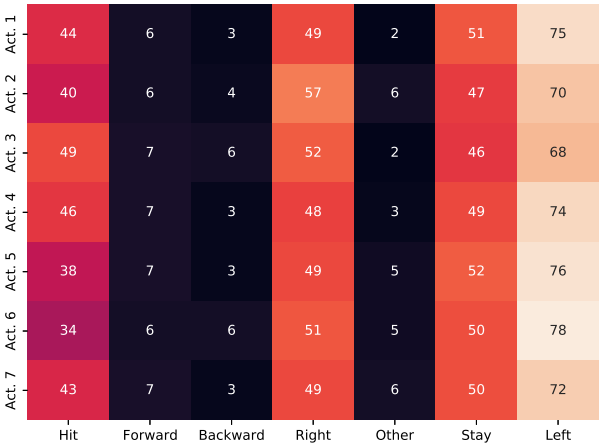


Figure 2: AMT votes for MoCoGAN [12] on the *Tennis* dataset. Rows correspond to the actions learned by the model, columns correspond to the action options presented to the users. A similar distribution of user votes is associated with each learned action.

action space. Indeed, their low Fleiss' kappa measures [6] (Sec. 4.2 of the main paper) show that users select different options for sequences generated with the same action, meaning that actions cause different effects based on the particular initial frame used to produce the sequence. On the other hand, the SAVP baseline (Fig. 4) learns actions that result in a different distribution of user votes for each row, indicating a partial capability of the model to condition its output based on the input action. Differently from the other methods, CADDY (Fig. 6) presents for each row a polarized response in a specific column, showing that our method associates the same meaning to an action independently from the starting frame. Some actions, including *Act. 1* and *Act. 5* present a lower user agreement. Despite the low number of votes given to the *Hit* action, a manual analysis of the corresponding sequences reveals that they correspond to the synthesis of ball hitting sequences, whose typical movement of the arm is difficult to spot and typically associated with movement of the player. This explains the portion of votes assigned to *Left*, *Right*, *Forward* and *Backward*.

5. Human evaluation for Video Quality

We perform an additional human evaluation on the *Tennis* dataset to assess the quality of the synthesized videos. Since our method produces results in 256×96, we compare only with baselines producing outputs in the same resolution to ensure fairness, namely MoCoGAN+ and SAVP+. We run our study on AMT and ask users to express preference between one of two videos based on video quality. One video is produced with CADDY and the other with

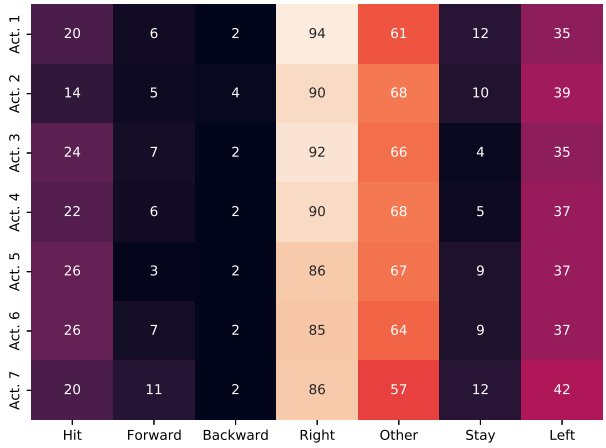


Figure 3: AMT votes for MoCoGAN+ on the *Tennis* dataset. Rows correspond to the actions learned by the model, columns correspond to the action options presented to the users. A similar distribution of user votes is associated with each learned action.

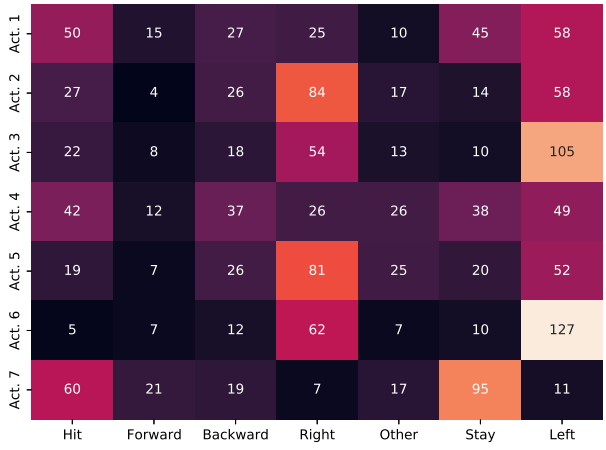


Figure 4: AMT votes for SAVP [10] on the *Tennis* dataset. Rows correspond to the actions learned by the model, columns correspond to the action options presented to the users. The distribution of user votes is weakly dependent on the learned action.

a baseline method. When compared to MoCoGAN+ and SAVP+, users express preference for our method in respectively 91.8% and 89.6% of cases.

6. Additional Qualitative Results

We provide the play-bair.html, play.html and play-tennis.html pages that contain fully playable CADDY models running in browser which allows the reader to directly evaluate the performance of our approach. In addition, we

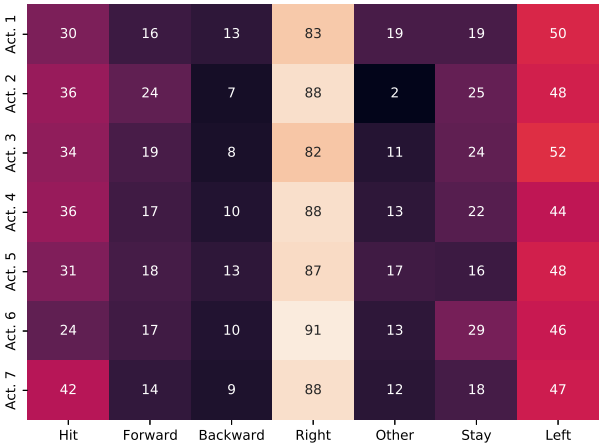


Figure 5: AMT votes for SAVP+ on the *Tennis* dataset. Rows correspond to the actions learned by the model, columns correspond to the action options presented to the users. A similar distribution of user votes is associated with each learned action.

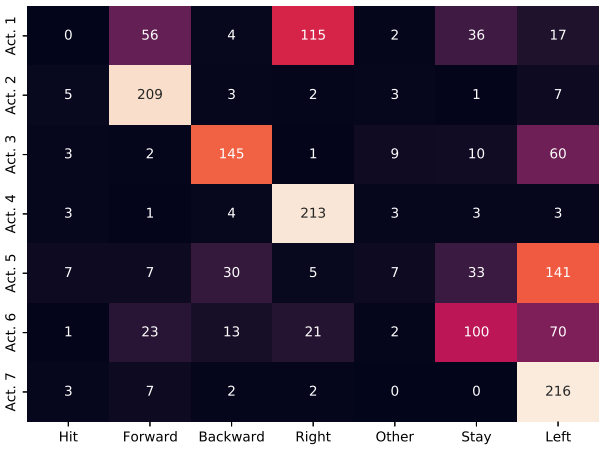


Figure 6: AMT votes for CADDY on the *Tennis* dataset. Rows correspond to the actions learned by the model, columns correspond to the action options presented to the users. The distribution of user votes is strongly dependent on the learned action and users express agreement on the effect produced by each learned action.

produce a set of qualitative results to show the capabilities of CADDY in the PVG task. In order to better visualize the results, we provide a main.html page showing qualitative results in the form of videos. In the accompanying website, we show demos of live user interaction with CADDY and examples of videos produced interactively by users. In addition, we present an action conditioning evaluation showing the effects of each of the actions learned by our model

and provide reconstruction results. The remaining of this section shows additional results fitted for visualization on paper.

6.1. Action Directions Space Visualization

In this section, we analyze the learned space of *action directions* d_t. Fig. 7a shows the action space learned on *BAIR*, where CADDY discovers two predominant categories of actions that correspond to common small movements of the robot hand. The remaining action categories are assigned to less common actions including lifting or lowering the robot arm or making fast horizontal movements. Fig. 7b illustrates the learned action space for *Atari Breakout*. The model clearly divides the *action directions* into three clusters corresponding to left movement, no movement and right movement. In *Tennis*, as shown in Fig. 7c, the model learns a rich action space whose structure is correlated with player movement. According to AMT user votes, Action 6 (orange) corresponds to *Stay* and its corresponding *action directions* occupy the center of the space. Action 4 (green) and 7 (red) correspond respectively to right and left movement and occupy opposing portions of the action space. Similarly, Action 2 (blue) and 3 (light blue) correspond to forward and backward movement and occupy opposing positions in the action space. Finally, according to human evaluation, Action 1 (dark blue) and Action 5 (yellow), which are positioned at boundary regions, have mixed correspondence to the movements of the neighboring regions and combine movement with ball hitting actions.

6.2. Interactively Generated Videos

We use CADDY to produce videos with direct user interaction. Starting from an initial frame, the user presses the button on its keyboard corresponding to the action to use at the current step, and the model generates the next frame. An extract of the generated results is shown in Fig. 8, while the complete videos are shown in the corresponding section of the main.html page. Videos can also be interactively generated by the reader through the play-bair.html, play.html and play-tennis.html pages.

6.3. Action Conditioning Evaluation

In order to visualize the effects produced by each action learned by CADDY, we consider an initial frame and, for each action, we produce a sequence by repeatedly using the current action as user input. We show the obtained results in Fig. 9 and Fig. 10. On all the datasets, our model learns actions that correspond to movement of the object of interest along each axis. In addition, on the *Tennis* dataset, CADDY learns actions related to ball hitting. In the corresponding section of the main.html page, we show additional video results both for CADDY and for the baselines.

Moreover, we analyze the action-conditional distribution of the displacement Δ associated with the object of interest. We show the results in Fig. 11 for the *BAIR* dataset, in Fig. 12 for the *Atari Breakout* dataset and in Fig. 13 for the *Tennis* dataset. We observe that each action corresponds to a distinct distribution of the displacement Δ which captures a specific movement direction. The other methods instead present distributions that are more uniform across actions, indicating a limited capability of learning actions that correspond to the movement of the object of interest.

6.4. Action Variability Embeddings Evaluation

In this section, we perform an evaluation of the capacity of *action variability embeddings* to capture variations of the relative action. In particular, we consider a pair of actions a_i and a_j. At inference time, since we pose *action variability embeddings* $v_i = v_j = 0$, for Eq. (8) we have $d_i = c_i$ and $d_j = c_j$ i.e. the associated *action directions* are centered on the *action direction centroids*. We argue that it is possible to produce actions with intermediate effects between a_i and a_j by sampling *action variability embeddings* corresponding to *action directions* in intermediate locations between the two *action direction centroids*.

Let $l \in [0, 1]$ be an *interpolation factor*. We pose

$$a, c = \begin{cases} a_i, c_i & \text{if } l <= 0.5 \\ a_j, c_j & \text{if } l > 0.5 \end{cases}$$

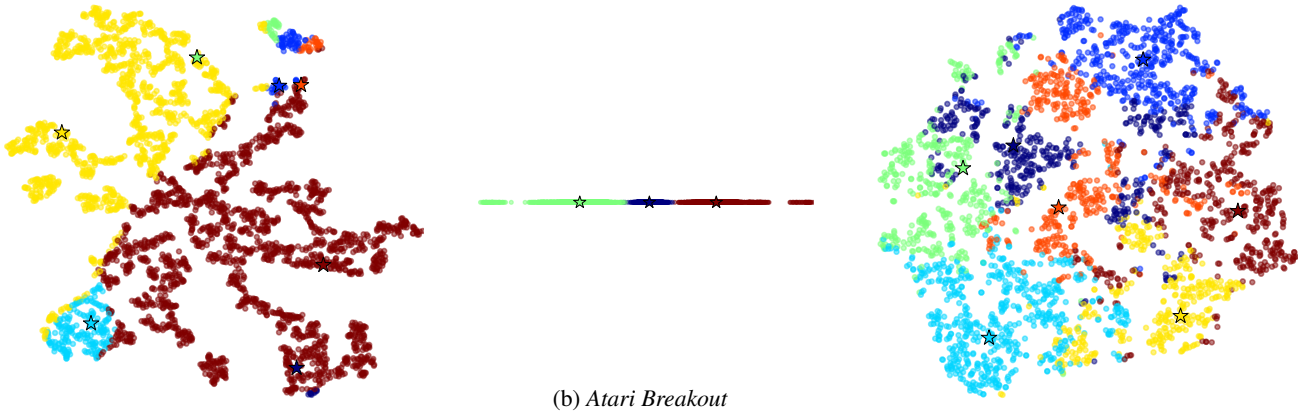
$$v = l(c_j - c_i) + c_i - c$$

The resulting *action* a and *action variability embedding* v represent an intermediate action between a_i and a_j. In Fig. 14 we show qualitative results representing the effects obtained using intermediate actions that interpolate between a pair of discrete actions.

6.5. Reconstruction Results

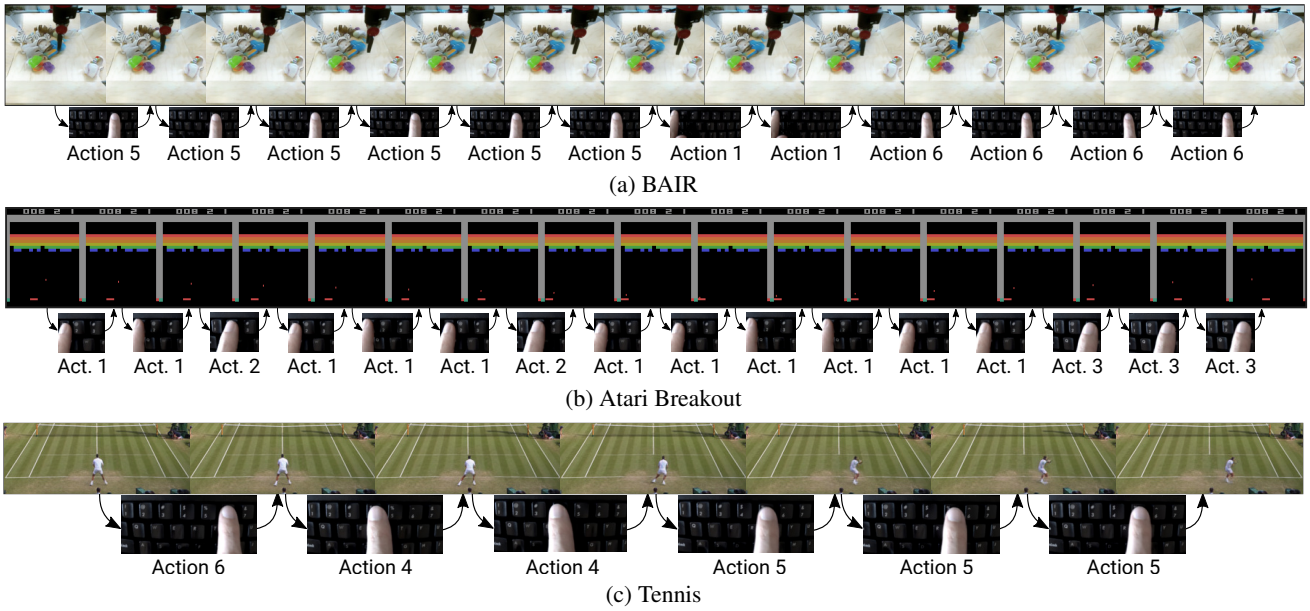
In this section, we show reconstructed sequences produced by our method. Note that additional results are present in the corresponding section of the main.html page. On the *BAIR* dataset (Fig. 15), our model correctly generates a robot arm that follows the relative movements of the original sequence. We observe that the appearance of the arm remains consistent in the whole sequence. On the other hand, in the SAVP+ and MoCoGAN+ baselines the arm disappears or shows artifacts towards the end of the sequence.

In Fig. 16, we show reconstruction results on the *Atari Breakout* dataset. Our method correctly learns the action space and the physics of the environment. The generated player-controlled platform correctly matches the position of that in the ground truth sequence, blocks that are hit by the ball correctly disappear and the trajectory of the ball



(a) *BAIR* (b) *Atari Breakout* (c) *Tennis*

Figure 7: Visualizations of the learned space of *action directions* d_t with corresponding *action direction centroids* $\{c_k\}_{k=1}^{K}$, grouped by action.



(a) BAIR

(b) Atari Breakout

(c) Tennis

Figure 8: Video sequences generated by CADDY with direct user interaction on the *BAIR* (a), *Atari Breakout* (b) and *Tennis* (c) datasets. Additional videos are shown in the corresponding section of the main.html or can be directly generated using the play-bair.html, play.html and play-tennis.html pages.

closely follows that of the original video, even after multiple bounces. In the baselines, instead, the generated platform does not match the behavior of that in the original sequence. Artifacts are present, especially in the MoCoGAN+ and SAVP+ baselines, where multiple platforms are generated of the platform disappears. Moreover, when the ball is generated, its trajectory follows the original one less accurately.

On the *Tennis* dataset (Fig. 17 and Fig. 18) CADDY correctly reconstructs the pose and the movements of the player and generates its shadow. The other baselines instead generate sequences with artifacts such as disappearing or faded player, and disappearing or detached shadow.

References

[1] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations (ICLR)*, 2018.

[2] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya

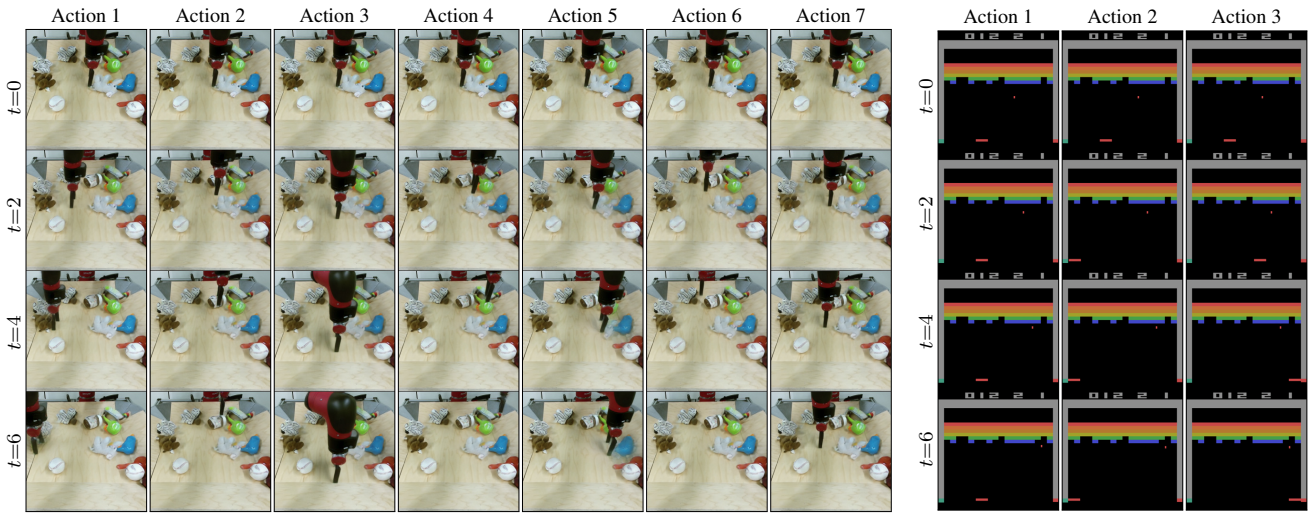


Figure 9: Videos generated by CADDY on the *BAIR* (left) and on the *Atari Breakout* (right) datasets. We generate a sequence for each learned action by repeatedly inputting the current action starting from the same initial frame. In all datasets, the model learns actions that correspond to movement on each axis. Additional videos are shown in the corresponding section of the main.html page.

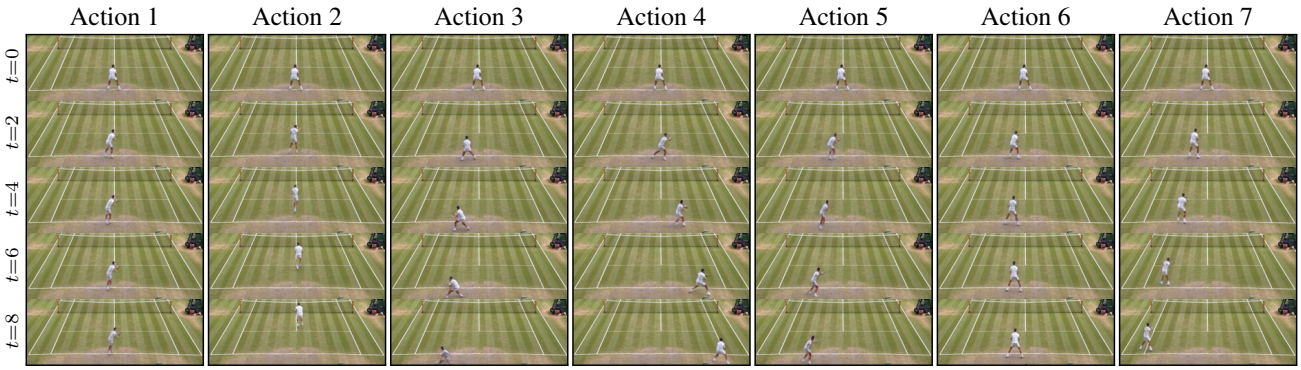


Figure 10: Videos generated by CADDY on the *Tennis* dataset. We generate a sequence for each learned action by repeatedly inputting the current action starting from the same initial frame. The model learns actions that correspond horizontal (Act. 4 and Act. 7) and vertical player movement (Act. 2 and Act. 3), no movement (Act. 6), and ball hitting (Act. 1, Act. 5). Additional videos are shown in the corresponding section of the main.html page.

Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 2172–2180, 2016.

[3] Aidan Clark, Jeff Donahue, and Karen Simonyan. Efficient video generation on complex datasets. *CoRR*, abs/1907.06571, 2019.

[4] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. volume 80 of *Proceedings of Machine Learning Research*, pages 1174–1183, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[5] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 64–72, 2016.

[6] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[7] Jean-Yves Franceschi, Edouard Delasalles, Mickael Chen, Sylvain Lamprier, and P. Gallinari. Stochastic latent residual video prediction. *ArXiv*, abs/2002.09219, 2020.

[8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017.

[9] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A conditional flow-based model for stochastic video generation. In *International Conference on Learning Representations (ICLR)*, 2020.

[10] Alex X. Lee, Richard Zhang, Frederik Ebert, P. Abbeel,

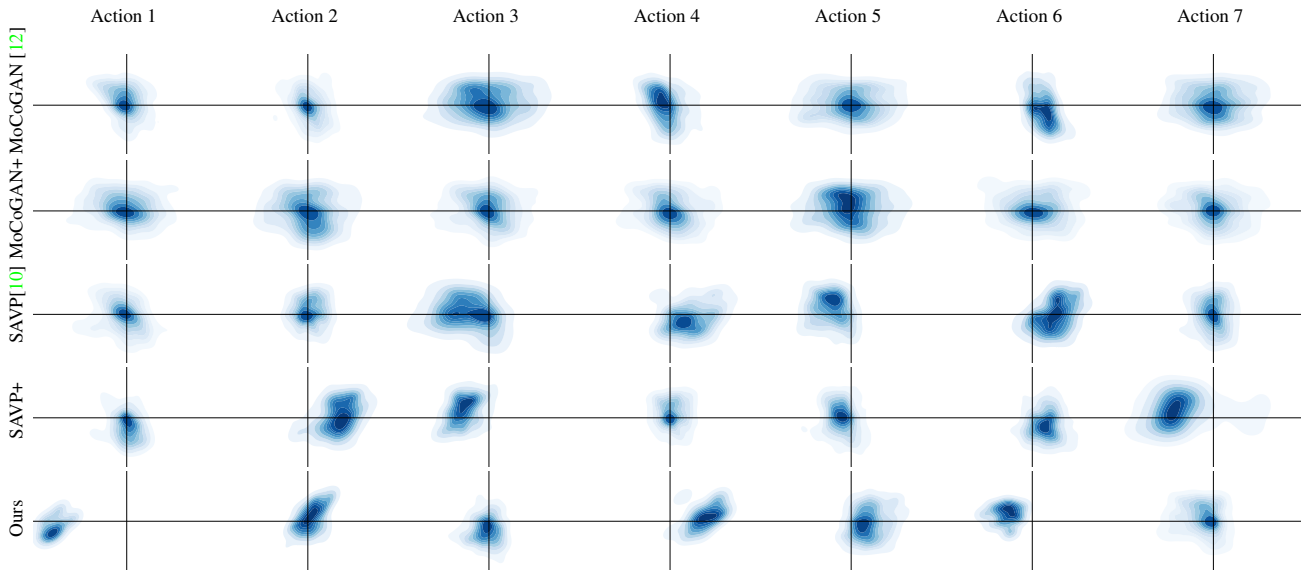


Figure 11: Distribution of the displacement Δ associated with robot arm on the *BAIR* dataset. Ideal distributions are different for each action and have low variance, meaning that they capture specific movements. The displacement component associated with movement on the vertical axis z is not shown. While the distributions of the displacement Δ on the x and y axes associated with the SAVP and SAVP+ baselines are influenced by the input action, they do not successfully capture movement of the robot arm on the z axis. This is reflected in the Δ-*MSE* score which shows that the actions learned by our model correspond to more specific movements than the ones learned by the baselines.

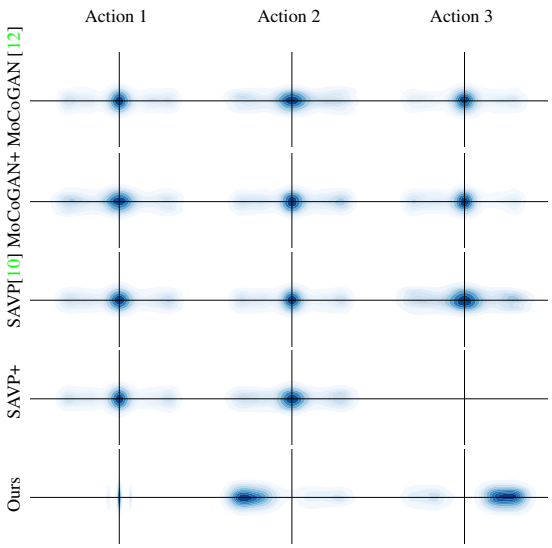


Figure 12: Distribution of the displacement Δ associated with the player-controlled platform on the *Atari Breakout* dataset. Ideal distributions are different for each action and have low variance, meaning that they capture specific movements. Differently from the baselines, our model learns actions that correspond to specific movements of the platform.

Chelsea Finn, and S. Levine. Stochastic adversarial video prediction. *ArXiv*, abs/1804.01523, 2018.

[11] Pauline Luc, A. Clark, S. Dieleman, D. Casas, Yotam Doron, Albin Cassirer, and K. Simonyan. Transformation-based adversarial video prediction on large-scale data. *ArXiv*, abs/2003.04035, 2020.

[12] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[13] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit.

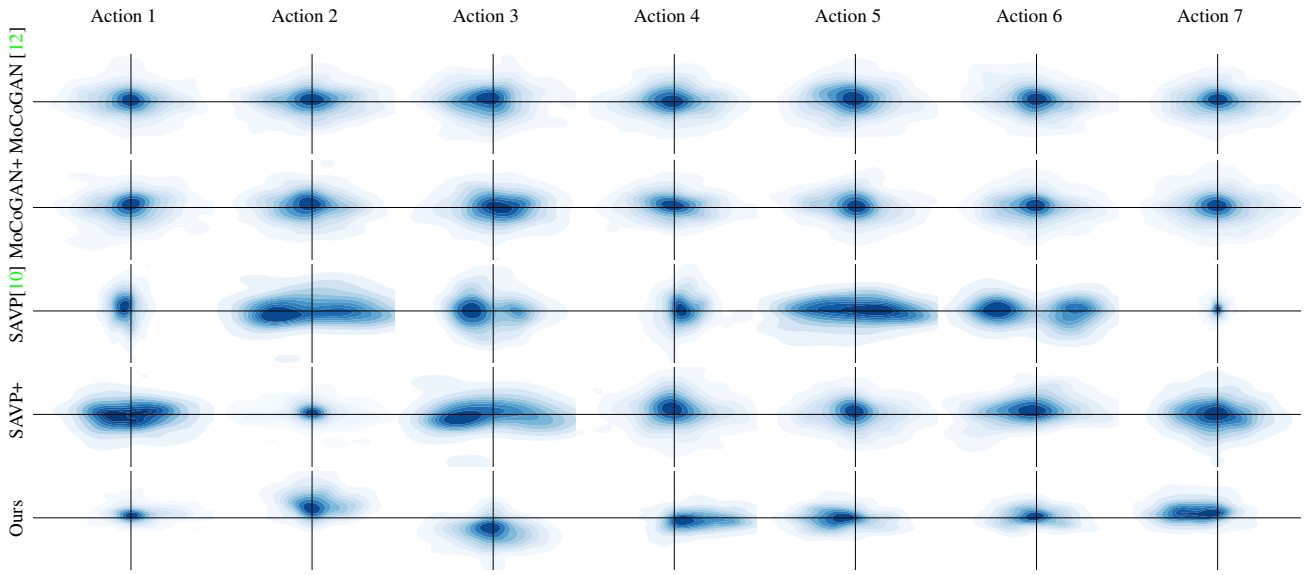


Figure 13: Distribution of the displacement Δ associated with the player on the *Tennis* dataset. Ideal distributions are different for each action and have low variance, meaning that they capture specific movements. Our model successfully conditions the movement of the player on the action, while the other baselines show limited conditioning capabilities.

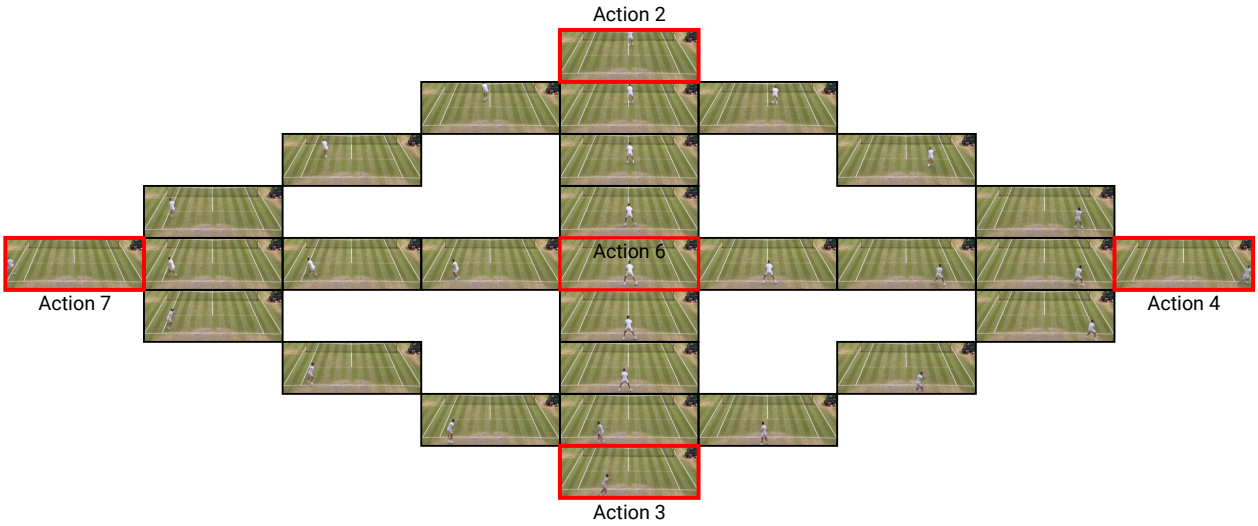


Figure 14: Visualization of the last frame of sequences produced from the same initial frame with differing *actions* and *action variability embeddings*. The sequences in red represent pure discrete actions where, *action variability embeddings* are posed to 0. For the sequences in black instead, *actions* and *action variability embeddings* are derived as described in Sec. 6.4 to represent intermediate actions between the two closest discrete actions using values of the *interpolation factor* of 0.3, 0.5 and 0.7. Note how the player positions depicted in the black sequences smoothly interpolate between the positions obtained with pure discrete actions, showing the capability of *action variability embeddings* to capture variations on discrete actions.

Scaling autoregressive video models. In *International Conference on Learning Representations (ICLR)*, 2020.

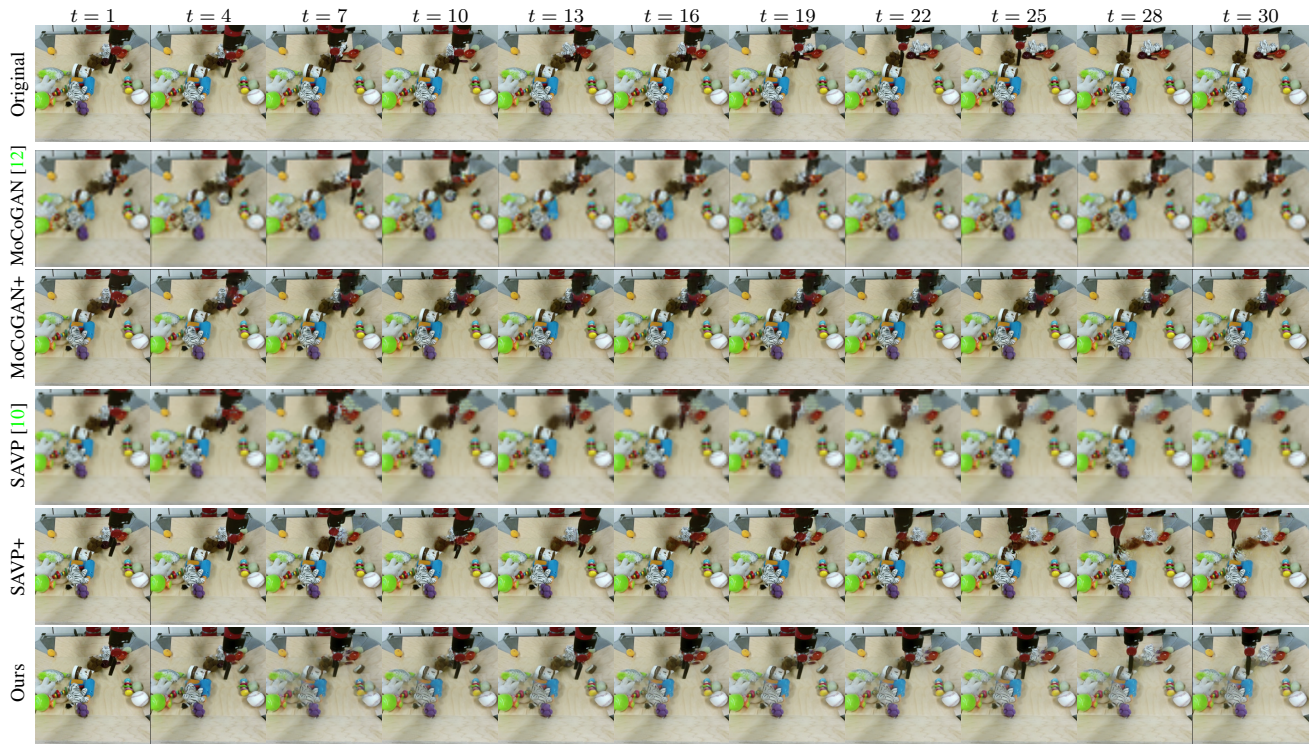


Figure 15: Reconstructed sequences on the *BAIR* dataset using the learned, discrete actions extracted from the original sequence as inputs.

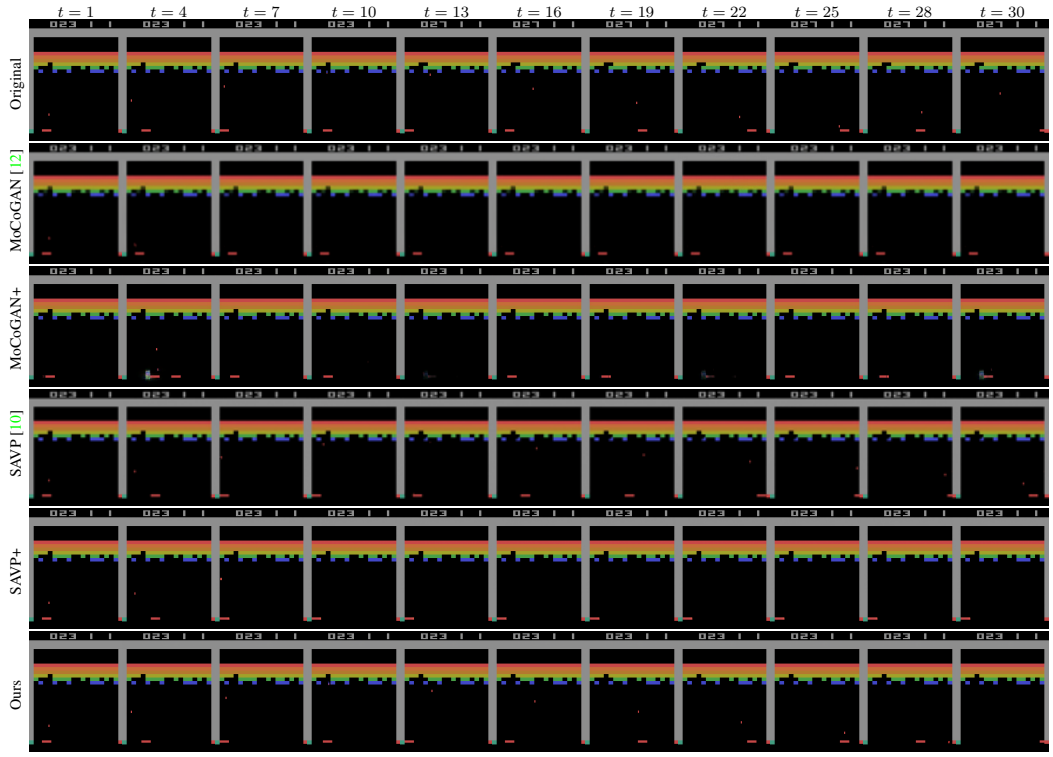


Figure 16: Reconstructed sequences on the *Atari Breakout* dataset using the learned, discrete actions extracted from the original sequence as inputs.

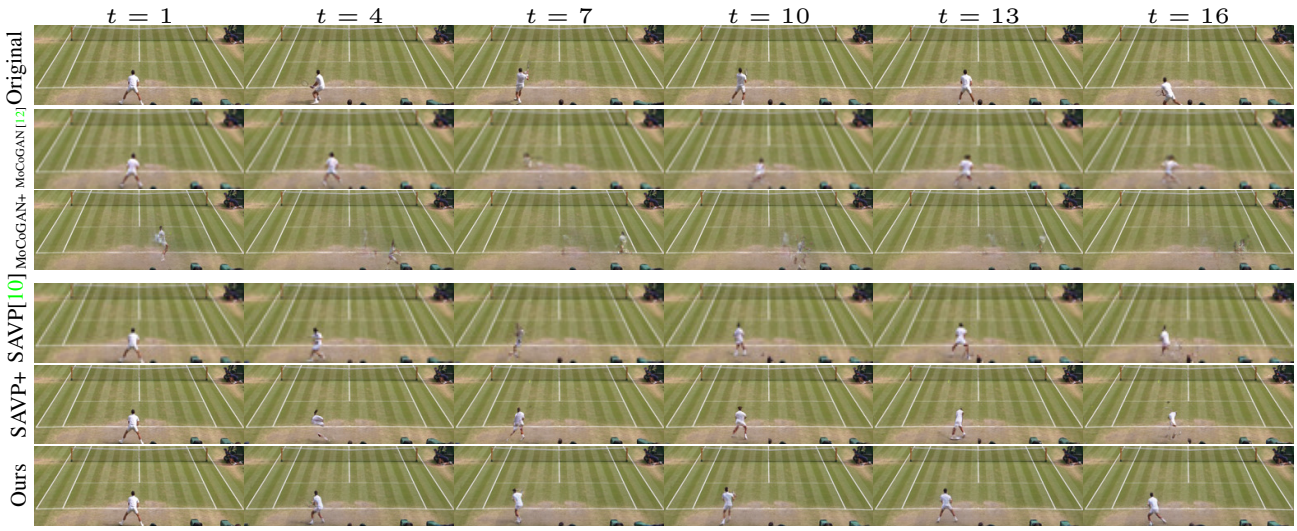


Figure 17: Reconstructed sequences on the *Tennis* dataset using the learned, discrete actions extracted from the original sequence as inputs.

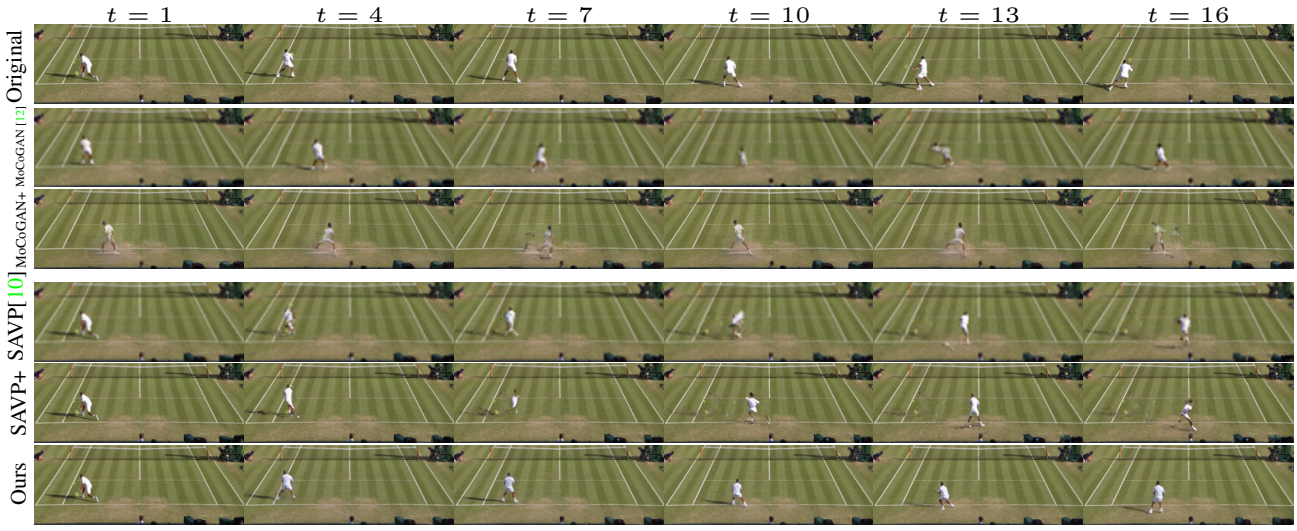


Figure 18: Reconstructed sequences on the *Tennis* dataset using the learned, discrete actions extracted from the original sequence as inputs.