FLAM: Scaling Latent Action World Models with Factorization

Zizhao Wang[1*] **Chang Shi**[1*] **Jiaheng Hu**[1] **Roberto Martín-Martín**[1] **Peter Stone**[1,2]
[1]The University of Texas at Austin [2]Sony AI
{zizhao.wang,changshi,jiahengh}@utexas.edu,{robertomm,pstone}@cs.utexas.edu

Abstract

Learning latent actions from action-free video has emerged as a powerful paradigm for scaling up controllable world models learning. The latent actions offer an extra degree of freedom for users to generate videos iteratively. However, existing approaches often rely on monolithic inverse and forward dynamics models to learn one latent action that controls all, which struggle to scale in complex scenes where different entities act simultaneously. In this work, we propose FLAM, a factored dynamics framework that decomposes the latent state into independent factors, each with its own inverse and forward dynamics model. This structure enables more accurate modeling of complex, multi-entity dynamics and improves the video generation quality in action-free video settings. Evaluated on Multigrid, Procgen, nuPlan and Sports datasets, FLAM consistently outperforms the monolithic dynamics model, demonstrating the superiority of the factorized model.

1 Introduction

Recent advances in Latent Action Models (LAM) [29, 2] have unlocked the possibilities of learning world models from action-free videos that are abundant on the web. Specifically, these approaches use an inverse dynamics model to encode environmental changes into a *single* latent action. The latent action is then used to train a forward dynamics model, allowing controllable predictions of future frames purely from in-the-wild videos.

However, in-the-wild videos often contain complex scenes where many entities may be taking actions simultaneously: for instance, a robot video may include independent joint movements, object manipulation, and shifting camera perspectives; while a soccer game involves several players, the ball, and even background audience motion, each of which acts independently. Compressing all these motions into a *single* latent action is challenging, since the complexity of actions scales exponentially with the number of movable entities (Fig. 1). Consequently, existing methods struggle with latent action learning in such settings, which severely limits their scalability to in-the-wild scenarios.

In this work, we propose the Factored Latent Action Model (FLAM), where the latent state is decomposed into a set of factors, each independently predicting its latent action and its next state via shared factored inverse and forward dynamics models. By sharing a common latent action codebook across factors, FLAM reduces the challenge of learning a massive codebook covering all action combinations to the simpler task of learning a small codebook for each entity's action. Unlike most prior LAM approaches that use a monolithic scene representation entangling all entities, FLAM factorizes the scene into compositional entities with a shared forward dynamics model, inherently supporting permutation invariance and enabling stronger generalization. With next frame prediction as the training objective, FLAM learns structured state and action representations from action-free

[*]equal contribution

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Embodied World Models for Decision Making.

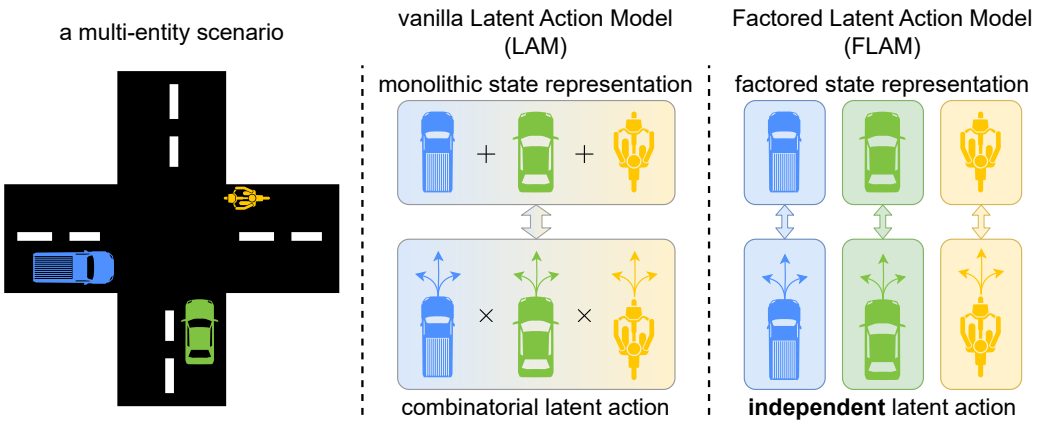

Figure 1: In multi-entity scenarios, **(left)** such as an intersection with three road users: **(middle)** a vanilla latent action model must compress all possible action combinations into a single latent action, which makes learning challenging due to the need for a huge codebook of size $|\mathcal{A}|^K$. **(right)** In contrast, FLAM decomposes the state and latent representation into K factors and adopts the same small action codebook of size $|\mathcal{A}|$ for each factor, enabling more efficient learning.

video data, leading to more accurate modeling of complex, multi-entity dynamics and improved prediction quality compared to previous work.

We evaluated our method in challenging domains including real-world autonomous driving, sports, and egocentric activity videos, where FLAM outperforms previous state-of-the-art methods both in dynamics prediction quality and video generation controllability.

2 Related Work

In this section, we first discuss previous works in learning world models from action-free videos. Next, we discuss prior works in factorized decision making and object-centric representations, whose strength and weakness inspire FLAM.

Dynamics Modeling without Action Labels Given the abundant source of videos and the scarcity of action labels, several methods have been developed to learn from pure observations. PlaNet [14] learns a latent dynamics model for planning. [23] learns both object structure and dynamics from videos using keypoint representation. ILPO [10] learns a latent policy with forward dynamics model only, and later maps the latent policy output to real action through an action remapping network. LAPO [29] and Genie [2] jointly learn an inverse dynamics model jointly with a forward dynamics model. [24] then introduce a small portion of ground-truth actions as dynamics modeling supervision. [34] expand learning from observation from vision only to vision and language modalities. Past work [36, 35, 1, 34] has proven the value of dynamics modeling without action labels in applications such as learning to drive, play games, and manipulate with robot arms from videos. Our work is mainly based on the LAPO framework, since it is the SOTA method that incorporates all four components: inverse dynamics modeling, forward dynamics modeling, real action remapping and policy learning. However, noisy real-world videos usually include more than one entity, and LAPO falls short in modeling those complex scenes, which motivates our method focusing on the dynamics modeling for multi-entity videos through factorization.

Factorization in Decision Making Factorization has long been used to exploit structured state and action spaces in complex environments, often through the factored MDP formulation [26, 13]. Recent works have applied this principle to derive factored forward dynamics [28, 33], factored policies [15, 16], and factored value functions [31]. FLAM is motivated by the same principle that factorization simplifies complexity into manageable components. However, it applies this idea to learning world models from action-free video, where the underlying factorization is not given. [19] is a similar work that also deals with world models from action-free video. However, FLAM is the first

factored latent world model that does not use pre-trained or co-trained object-centric representation, and specifically focus on multi-entity scenarios.

Object-Centric Representation Learning Object-centric learning aims to represent complex scenes by isolating individual objects from the background and from each other, leading to improved generalization and modeling capabilities. Key challenges include the need for supervision, leading to the development of many unsupervised methods. MONet[3] and IODINE[12] achieved unsupervised multi-object segmentation through attention and iterative amortized inference. Slot Attention [20] uses iteratively updated slots to learn disentangled, object-based representations. DINO [5, 25] uses a transformer-based architecture instead. Recent work such as [6, 18] has focused on improving object-centric learning fidelity in noisy real-world data. In our work, we use Slot Attention as the factorizer for the state representation and use the forward prediction task as the learning signal. One thing that distinguishes our representation from the object-centric representation is that we do not use the reconstruction task for representation learning; thus the representation tends to focus on gathering groups of entities with similar dynamics instead of those with similar superficial visual properties.

3 Preliminaries

Latent Action Model Given a video dataset, we seek to model the dynamics with only observations; without any action labels. The Latent Action Model (LAM) is a model that uses an inverse dynamics model (IDM) and a forward dynamics model (FDM) to infer latent actions. Though not fully aligned with real actions with physical meanings, the inferred latent actions capture the most essential change during the transition, thus can be used for subsequent policy learning.

As shown in Fig. 2, both the IDM and the FDM observe o_t, but only the IDM observes o_{t+1}. Therefore, in order to accurately predict o_{t+1}, the IDM must extract useful transition information through the latent action a_t for the FDM. However, if no bottleneck constraints are added in a_t, the model is likely to collapse, as a naive solution would be simply to copy o_{t+1}.

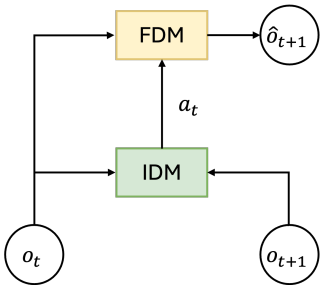

Figure 2: Latent action model architecture.

Thus, prior methods use vector quantization to discretize the latent action into a small set of codes, ensuring that it is compact [32]. While action quantization is a necessary step to ensure that it only encodes the most important changes rather than copying the future frame, it also limits the expressiveness of the latent action. One example is scenarios where many state variables have independent actions, such as a crowded intersection, and it is suboptimal and even infeasible to compress all entities' actions into a single latent action.

As a solution, FLAM factorizes both the state representation and the latent action into independent factors, enabling us to share a small codebook across all factors, as discussed in detail in Section 4.

4 Factored Latent Action Model (FLAM)

From a high-level perspective, FLAM scales latent action models to multi-entity scenarios by inferring a set of latent actions rather than a single latent action between each pair of frames in a factored manner, enabled by two learning phases shown in Fig. 3:

- **Encoder learning** (Section 4.1): FLAM pre-trains a VQ-VAE to extract high-level features from pixels, allowing the latent action model to learn in the feature space rather than the pixel space for the purpose of efficient learning.
- **Latent action model (LAM) learning** (Section 4.2): Using the extracted features, FLAM decomposes the scene into several independent factors, also referred to as *slots*. For each slot, an inverse dynamics model infers a separate latent action from its current and next-frame values. Then, based on its current value and the corresponding latent action, a forward dynamics model independently predicts the next-frame value for each slot. Finally, all predicted slots are mapped back to the feature space and decoded into the next video frame.

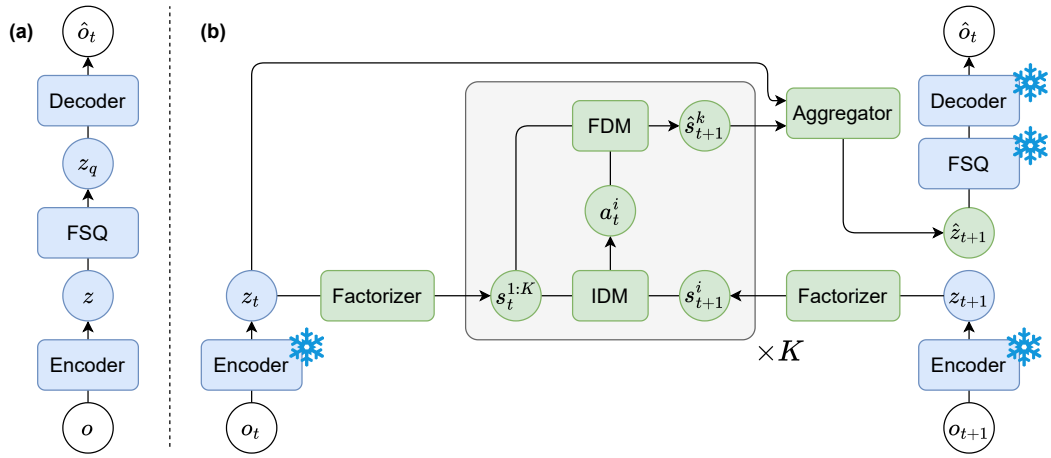

Figure 3: Two training stages of FLAM. **(a)** A VQ-VAE is pretrained to extract features for latent action model learning. **(b)** FLAM infers latent actions and makes predictions for each slot independently, with all modules trained to minimize prediction error.

After FLAM is well trained to model the world from action-free videos, its latent action outputs can be used for either controllable video generation or policy learning to achieve tasks. We refer to this as the third phase, and discuss how to leverage FLAM for both settings in Section 4.3.

4.1 Pretrained Encoder

As shown in Fig. 3 (a), FLAM learns a VQ-VAE [32] to extract features from raw pixels, enabling fast LAM learning. For each frame $o \in \mathbb{R}^{H \times W \times 3}$, a CNN encoder first extracts N patch-level features $z \in \mathbb{R}^{N \times d_z}$. The features are then quantized from continuous encoder outputs to the nearest entry in a discrete codebook, denoted as $z_q \in \mathbb{R}^{N \times d_z}$, using finite scalar quantization `FSQ` [22], which forces features into a finite set of learned embeddings. At the same time, we also record their codebook indices as $c \in \{1, \ldots, C\}^N$. By applying vector quantization, FLAM reformulates the next-step feature prediction from a regression task (predicting features z) to a classification task (predicting indices c), which has been shown to improve prediction quality [30] (see details in Sec. 4.2). Finally, a decoder reconstructs the frame from the quantized feature:

$$\begin{aligned} z &= \texttt{Encoder}(o), \\ z_q, c &= \texttt{FSQ}(z), \\ \hat{o} &= \texttt{Decoder}(z_q). \end{aligned}$$

The VQ-VAE is trained by minimizing the following reconstruction loss:

$$\mathcal{L}_{\texttt{VQ-VAE}}(o) = ||o - \hat{o}||^2. \tag{1}$$

4.2 Factored Latent Action Model (FLAM)

As shown in Fig. 3 (b), our latent action model contains four key components: 1) a factorizer that decomposes the scene z_q into a set of independent slots s, 2) an inverse dynamics model that infers a separate latent action a_t^i for each slot, 3) a forward dynamics model that, given the current slot value and latent action, predicts the next-frame value for each slot, and 4) an aggregator that maps the predicted slots back to the feature space. These four components are jointly trained to minimize the feature prediction error.

Factorizer To decompose the scene into a set of factors with independent actions, FLAM uses slot attention [20]. For each frame, K slots $s_t \in \mathbb{R}^{K \times d_s}$ are initialized from learned embeddings and then compete to bind to different regions in the frame through iterative slot attention:

$$s_t = \texttt{SlotAttention}(z_t). \tag{2}$$

Algorithm 1 FLAM Latent Action World Model

Prepare a dataset of (o_t, o_{t+1}).
Initialize the VQ-VAE (`Encoder`, `FSQ`, `Decoder`) and the latent action model (`Factorizer`, `IDM`, `FDM`, `Aggregator`).
Pretrain the VQ-VAE with the reconstruction loss in Eq. (1). ▷ Sec. 4.1
// Train the latent action model ▷ Sec. 4.2
Extract features with the VQ-VAE encoder: $z_t = \texttt{Encoder}(o_t)$, $z_{t+1} = \texttt{Encoder}(o_{t+1})$.
Extract slots using Eq. 2: $s_t^{1:K} = \texttt{Factorizer}(z_t)$, $s_{t+1}^{1:K} = \texttt{Factorizer}(z_{t+1})$.
for each slot $i = 1, \ldots, K$ **do**
    Infer the latent action: $a_t^i = \texttt{IDM}(s_t^{1:K}, s_{t+1}^i)$.
    Predict the next-frame slot: $\hat{s}_{t+1}^i = \texttt{FDM}(s_t^{1:K}, a_t^i)$.
Map predicted slots back to the feature space: $\hat{z}_{t+1} = \texttt{Aggregator}(z_t, \hat{s}_{t+1}^{1:K})$.
Optimize the latent action model with the prediction loss in Eq. 5.

Note that although FLAM employs slot attention, a technique that is also widely used for learning object-centric representations, FLAM differs from prior object-centric representation work in its learning signal. While object-centric representation methods typically optimize reconstruction and extract objects based on visual features, FLAM instead optimizes prediction and extracts factors based on action independence, as detailed in the Training Objective paragraph.

Inverse Dynamics Model (IDM) After extracting independent slots, the IDM aims to infer a latent action a_t^i for each slot i, based on all current slots $s_t^{1:K}$ and its next-frame value s_{t+1}^i:

$$a_t^i = \texttt{IDM}(s_t^{1:K}, s_{t+1}^i), \tag{3}$$

where for a variable x, $x^{1:K}$ denotes the set $\{x^i\}_{i=1}^K$, and we use $s_t^{1:K}$ and s_t interchangeably. We use all current slots rather than just s_t^i as inputs to account for interactions between factors, enabling more accurate latent action prediction (e.g., a person in a car moves because of the car rather than by themselves).

To implement the IDM, we adopt the spatio-temporal model introduced in Genie [2]. The spatial block applies self-attention to capture interaction information across $s_t^{1:K}$, while the temporal block applies cross-attention to compare s_{t+1}^i with its current value and encode the most meaningful changes between them:

$$\begin{aligned} s_t^{1:K} &= \texttt{SelfAttention}(s_t^{1:K}), \\ a_t^i &= \texttt{CrossAttention}(\texttt{query} = s_{t+1}^i, \texttt{key} = [s_t^i, s_{t+1}^i]). \end{aligned}$$

Note that although we only use the current and next values in the temporal block, our model can be easily extended to use all prior frames $s_{1:t}^i$ as inputs.

Similar to existing LAM methods, we use vector quantization to compress the latent action into a small set of codes. This compression limits the capacity of the latent action, preventing it from simply copying s_{t+1}^i and bypassing dynamics learning. In our work, we use the same IDM (including the latent action codebook) for all slots.

Forward Dynamics Model (FDM) To provide the learning signals for the IDM, the forward dynamics model takes all current slots $s_t^{1:K}$ together with the latent action a_t^i and predicts the next-frame value for each slot $\hat{s}_{t+1}^i$:

$$\hat{s}_{t+1}^i = \texttt{FDM}(s_t^{1:K}, a_t^i). \tag{4}$$

Similarly to the IDM, we use all current slots instead of just s_t^i as inputs to capture interactions between slots, as they are not intended to be encoded by the latent action. For implementation, we use the same spatio-temporal model as the IDM, except that the temporal block uses a_t^i as the key. We also use the same FDM to predict all slots.

Aggregator Finally, the aggregator maps the predicted slots back into the feature space, enabling decoding into the next-frame prediction. Instead of relying solely on $\hat{s}_{t+1}^i$ to predict the feature, the

aggregator also conditions on the current feature z_t. This design allows $\hat{s}^i_{t+1}$ to focus on capturing changes between current and next time steps, rather than redundantly encoding static visual details. Concretely, the aggregator uses cross-attention to update each patch feature based on the predicted slots:

$$\hat{z}_{t+1} = \texttt{CrossAttention}(\texttt{query} = z_t, \texttt{key} = \hat{s}^{1:K}_{t+1}).$$

Training Objective The four components of the latent action model (Factorizer, IDM, FDM, and Aggregator) are trained jointly to minimize the feature prediction error:

$$\mathcal{L}_{\texttt{LAM}}(z_t, z_{t+1}) = \|z_{t+1} - \hat{z}_{t+1}\|. \tag{5}$$

As mentioned earlier, although the factorizer and the aggregator use implementations similar to prior object-centric representation methods, the prediction loss, together with the design of independent latent actions for each slot in the IDM and FDM, enables FLAM to separate entities in a scene according to their independent dynamics. This design differs from object-centric representation methods, which rely on reconstruction loss and separate objects based on visual appearance.

4.3 Learned Latent Actions Utilization

The latent actions learned from FLAM implicitly incorporate the action information of each entity in the world, thus offering a degree of freedom to manipulate the video generation. We can let human specify a latent action value from the learned latent action codebook, and use that as a control variable to generate video accordingly. Conditioned on different latent action chosen, the video generation will be diverse.

The latent actions can also be used as action labels for policy learning on video frames. Because the dimensions of the latent action do not necessarily align with the dimensions of the real action, a latent action decoder f that maps latent action a to real action u is needed. We first collect a small dataset of latent action-action pairs and train a latent action decoder f offline through supervised learning. We can then learn a policy π_u through behavior cloning on the observation-action pairs $(o, f(a))$ obtained by applying FLAM and a latent action decoder on expert demonstration videos D_E:

$$\mathcal{L}_{\pi_u} = \mathbb{E}_{(o,a)\sim D_E} \|\pi_u(o) - f(a)\|^2 . \tag{6}$$

5 Experiments

Our central hypothesis is that our factorized representation and latent action learning approach can better capture the features and dynamics of each entity separately, thus leading to more accurate world modeling on complex, noisy, multi-entity videos. These more accurate world models can then be used for planning and policy learning, leading to higher performance on downstream tasks. To this end, we evaluate our method in both clean simulation settings as well as more noisy real-world settings. We focus on designing our experiments to answer the following key questions:

1. Does factorization truly help learning more accurate world models?
2. Can FLAM work well on noisy real-world data?

5.1 Settings

We conducted experiments on 2 simulation datasets and 3 real-world datasets. The simulation datasets include a Multigrid dataset and a Procgen dataset. The real-world datasets include an autonomous driving dataset, a sports dataset, and an egocentric activity dataset. We pick these datasets because they all include multiple independent entities, such as cars, players, or two human arms. Please refer to Appendix B for more details. The hyperparameters are listed in Tab. 4.

5.2 Baselines

We compare our method with three baseline algorithms. Latent Action Model (LAM) is the vanilla variant that learns latent actions without factorization. LAM using Object-Centric representation (OC+LAM) is the variant that adds a reconstruction loss on top of the prediction loss of the vanilla

LAM. World Model (WM) is the variant without factorization and uses ground truth actions rather than latent actions for forward dynamics prediction. While LAM and LAM+OC run on all datasets, WM only runs on the simulation datasets because ground-truth action labels are not available within the real-world datasets.

5.3 Encoder-Decoder Pre-training

As introduced in Section 4, the encoder-decoder architecture feature extractor is pre-trained and frozen in the latent action model learning stage. Instead of training a universal feature extractor for all data, we separately train a feature extractor for each dataset. This is because the focus of our work is to prove the superiority of factorization in world modeling, rather than a universal world model. Feature extractors customized for data can ensure that this part is not the bottleneck of our model, for the sake of fair comparison of multiple latent action model variants. Fig. 5 and Fig. 6 in Appendix C are the illustrations of the original video frames and the reconstructed video frames using the trained feature extractor. We can tell from the small difference between the original frames and reconstructed frames that the feature extractors are trained very well to capture useful information.

5.4 Do we need factor tracking?

When using slot attention as the factorizor and inferring a separate latent action for each slot, a question would naturally arise that whether we need one-on-one matching between slots on two different frames. We tried multiple ways of slot matching and find that recursive slot initialization - using the final slots of previous frame as the initialization for iterative slot updates on current frame would lead to the best results. This reveals that it is too hard for the factorized world model to implicitly learn the tracking of factors and there is the necessity of offering priors through explicit slot matching. For details of various slot matching methods we tried and results comparison, please refer to Appendix D.

5.5 How does FLAM perform compared to the vanilla latent action model?

We examine the performance of FLAM from two perspectives, the dynamics modeling accuracy, i.e. how good the model is in future video frame prediction, and controllability, i.e. how much adjustment users can apply onto video generation through the latent actions. For dynamics modeling accuracy, after scaling the pixel values into the [0,1] space, we calculate the Peak Signal-to-Noise Ratio (PSNR) as the evaluation metric, using the mean squared error (MSE) between the 1-step ahead ground-truth frame and predicted frame:

$$\text{PSNR}_{t=1}(x_t, \hat{x}_t) = 10 * \log_{10}\left(\frac{1}{\text{MSE}(x_t, \hat{x}_t)}\right),$$

where x_t represents the ground-truth frame at time t and $\hat{x}_t$ represents the predicted frame at time t using latent actions $a_{1:t}$ inferred from ground-truth frames. The bigger the PSNR is, the more accurate the model is on dynamics modeling. For controllability, we use a metric called the PSNR difference devised by [2]:

$$\Delta_t\text{PSNR} = \text{PSNR}_t(x_t, \hat{x}_t) - \text{PSNR}_t(x_t, \hat{x}'_t).$$

It measures the difference between the video frames $\hat{x}_t$ generated conditioned on a sequence of latent actions inferred from ground-truth frames, and the video frames $\hat{x}'_t$ generated conditioned on a sequence of latent actions randomly sampled from a categorical distribution. The greater Δ_tPSNR, the higher level of controllability the latent actions offer during the video generation.

The results are shown in Tab. 1. The results show that FLAM achieves lower predection error than other baselines without access to ground truth actions. This validates our assumption that in environments with multiple independent entities, factorizing the states and learning a dynamics model separately for each entity would help modeling the world dynamics more accurately. Although in Procgen environment, FLAM gets slightly worse controllability than LAM. This leads us to carefully choose the number of factors in the following experiments, considering the complexity of the scenes. Meanwhile, because FLAM outperforms OC+LAM, we can claim that without learning object-centric representation through an auxiliary reconstruction task, we can still do world modeling well by learning dynamics-aware representations, especially in the multi-entity settings. In fact,

OC+LAM appears to suffer from a performance degradation compared to LAM, which reveals that object-centric representations are not always beneficial to world modeling, if learned from superficial visual appearance. Due to space limit, please refer to Appendix E for visualized examples of reconstruction and prediction rollout samples.

Table 1: Video prediction performance of all methods across the simulation datasets.

	Dataset	WM	LAM	OC+LAM	FLAM
PSNR(↑)	Multigrid	48.90	31.04	26.79	**35.49**
	Procgen	24.74	25.28	20.87	**26.28**
Δ_t PSNR(↑)	Multigrid	-	7.84	0.11	**11.24**
	Procgen	-	**5.02**	2.18	3.99

5.6 When do we need FLAM?

To investigate whether FLAM indeed help with dynamics modeling in complex multi-entity videos, we conduct an ablation study over the number of entities in the multigrid environment. As we can see from 2, when the number of moving entities increases in the scene, FLAM is more robust than non-factored LAM.

Table 2: Video prediction performance across Multigrid environments with various number of entities.

	# Entities	WM	LAM	FLAM
PSNR(↑)	1	57.46	32.04	**35.39**
	2	56.56	33.39	**34.47**
	4	48.90	31.04	**35.49**
	8	39.85	26.18	**28.36**
	16	34.03	24.61	**25.24**

5.7 Is FLAM robust to real-world scenes?

We now apply FLAM to real-world datasets, with results shown in Tab. 3. On all three dataset, FLAM outperforms OC+LAM, which confirms the significance of learning dynamics-aware representations instead of visual representations. While FLAM and LAM shows similar capability in accurate prediction, FLAM demonstrate superior controllability in video generation. Due to space limitations, please refer to Appendix E for visualized examples of reconstruction and prediction rollout samples.

Table 3: Video prediction performance of all methods across the real-world datasets.

	Dataset	LAM	OC+LAM	FLAM
PSNR(↑)	nuPlan	**18.55**	16.59	17.70
	Sports	**17.96**	15.68	17.47
	EGTEA	**19.41**	13.47	19.35
Δ_t PSNR(↑)	nuPlan	2.07	0.26	**5.97**
	Sports	1.58	0.63	**4.94**
	EGTEA	4.58	0.73	**5.13**

5.8 Controllable Video Generation

The factored latent actions learned by FLAM not only help with accurate world modeling, but can also serve as a manipulation surrogate to guide the video generation. After an action codebook is learned, we let human user specify a choice from the codebook, and then rollout multiple steps. As shown in Fig. 4, latent actions can be used as a control variable to generate various video frames even with the same initial frame.

6 Conclusion

We present FLAM, a factored latent action model that scales to multi-entity scenarios where the action space grows exponentially with the number of entities. Although the large action space makes learning the action codebook challenging for vanilla LAM, FLAM addresses this issue by decomposing the state and the latent action representation into multiple independent factors, allowing learning to focus on a small action codebook shared across factors. This structure enables us to learn structured state and action representations from action-free video data, leading to more accurate modeling of complex multi-entity dynamics and improved both prediction quality and video controllability.

FLAM makes predictions in latent space with a transformed-based aggregator and a pre-trained decoder for visualization. For future work, exploring more sophisticated decoding methods, such as diffusion models, could further enhance the visual quality of generated rollouts.

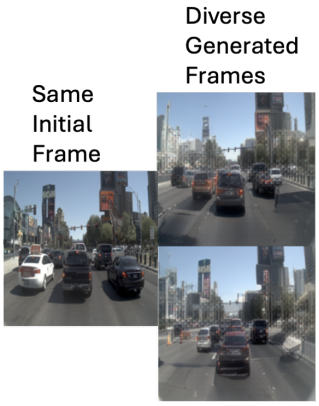

Figure 4: Conditioned on different latent actions, diverse videos can be generated on same initial frame.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by the National Science Foundation (FAIN-2019844, NRT-2125858), the Office of Naval Research (N00014-24-1-2550), Army Research Office (W911NF-17-2-0181, W911NF-23-2-0004, W911NF-25-1-0065), DARPA (Cooperative Agreement HR00112520004 on Ad Hoc Teamwork), Lockheed Martin, and Good Systems, a research grand challenge at the University of Texas at Austin. The views and conclusions contained in this document are those of the authors alone. Peter Stone serves as the Chief Scientist of Sony AI and receives financial compensation for that role. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

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

[2] 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.

[3] Christopher P. Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv: 1901.11390*, 2019.

[4] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles, 2022.

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[6] Chang Chen, Fei Deng, and Sungjin Ahn. Roots: Object-centric representation and rendering of 3d scenes. *Journal of Machine Learning Research*, 22(259):1–36, 2021.

[7] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld:

Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*, December 2023.

[8] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

[9] Yutao Cui, Chenkai Zeng, Xiaoyu Zhao, Yichun Yang, Gangshan Wu, and Limin Wang. Sportsmot: A large multi-object tracking dataset in multiple sports scenes. *arXiv preprint arXiv:2304.05170*, 2023.

[10] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell. Imitating latent policies from observation. In *International conference on machine learning*, pages 1755–1763. PMLR, 2019.

[11] Hayden Faulkner and Anthony Dick. Tenniset: A dataset for dense fine-grained event recognition, localisation and description. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8. IEEE.

[12] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv: 1903.00450*, 2019.

[13] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[15] Jiaheng Hu, Peter Stone, and Roberto Martín-Martín. Slac: Simulation-pretrained latent action space for whole-body real-world rl. *arXiv preprint arXiv:2506.04147*, 2025.

[16] Jiaheng Hu, Zizhao Wang, Peter Stone, and Roberto Martín-Martín. Disentangled unsupervised skill discovery for efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 37:76529–76552, 2024.

[17] Mostafa S. Ibrahim, Srikanth Muralidharan, Zhiwei Deng, Arash Vahdat, and Greg Mori. A hierarchical deep temporal model for group activity recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] Jindong Jiang, Fei Deng, Gautam Singh, and S. Ahn. Object-centric slot diffusion. *Neural Information Processing Systems*, 2023.

[19] Albina Klepach, Alexander Nikulin, Ilya Zisman, Denis Tarasov, Alexander Derevyagin, Andrei Polubarov, Nikita Lyubaykin, and Vladislav Kurenkov. Object-centric latent action learning. *arXiv preprint arXiv:2502.09680*, 2025.

[20] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in neural information processing systems*, 33:11525–11538, 2020.

[21] 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.

[22] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. *arXiv preprint arXiv:2309.15505*, 2023.

[23] Matthias Minderer, Chen Sun, Ruben Villegas, Forrester Cole, Kevin P Murphy, and Honglak Lee. Unsupervised learning of object structure and dynamics from videos. *Advances in Neural Information Processing Systems*, 32, 2019.

[24] Alexander Nikulin, Ilya Zisman, Denis Tarasov, Nikita Lyubaykin, Andrei Polubarov, Igor Kiselev, and Vladislav Kurenkov. Latent action learning requires supervision in the presence of distractors. *arXiv preprint arXiv:2502.00379*, 2025.

[25] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023.

[26] Ian Osband and Benjamin Van Roy. Near-optimal reinforcement learning in factored mdps. *Advances in Neural Information Processing Systems*, 27, 2014.

[27] Yulu Pan, Ce Zhang, and Gedas Bertasius. Basket: A large-scale video dataset for fine-grained skill estimation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 28952–28962, 2025.

[28] Silviu Pitis, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. *Advances in Neural Information Processing Systems*, 33:3976–3990, 2020.

[29] Dominik Schmidt and Minqi Jiang. Learning to act without actions. *arXiv preprint arXiv:2312.10812*, 2023.

[30] Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate dall-e learns to compose. *arXiv preprint arXiv:2110.11405*, 2021.

[31] Shagun Sodhani, Sergey Levine, and Amy Zhang. Improving generalization with approximate factored value functions. *Transactions on Machine Learning Research*, 2022.

[32] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[33] Zizhao Wang, Xuesu Xiao, Zifan Xu, Yuke Zhu, and Peter Stone. Causal dynamics learning for task-independent state abstraction. *arXiv preprint arXiv:2206.13452*, 2022.

[34] Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Sejune Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, et al. Latent action pretraining from videos. *arXiv preprint arXiv:2410.11758*, 2024.

[35] Weirui Ye, Yunsheng Zhang, Pieter Abbeel, and Yang Gao. Become a proficient player with limited data through watching pure videos. In *The Eleventh International Conference on Learning Representations*, 2022.

[36] Qihang Zhang, Zhenghao Peng, and Bolei Zhou. Learning to drive by watching youtube videos: Action-conditioned contrastive policy pretraining. In *European Conference on Computer Vision*, pages 111–128. Springer, 2022.

A Hyperparameter

Table 4: Experiment hyperparameters. Simulation datasets are Multigrid and Procgen; real-world datasets are Sports, EGTEA, and nuPlan.

	Simulation	**Real-world**
Encoder	IMPALA	MAGVIT-v2
Decoder	LAPO	MAGVIT-v2
Image size	128 (Multigrid), 224 (Procgen)	224
Tokenizer quantizer	FSQ	FSQ
Tokenizer codebook size	1024	16384
Tokenizer codebook levels	[4,4,4,4,4]	[4,4,4,4,4,4,4]
z-channels	128	128
Codebook dimension	128	128
d_{model} (LAM/FLAM)	256 / 256	256 / 512
Number of blocks	2	3
Number of factors K	4 (Multigrid), 32 (Procgen)	16 (nuPlan), 32 (Sports & EGTEA)
Action quantizer	FSQ	FSQ
Action codebook size	4 (Multigrid), 16 (Procgen)	32
Action FSQ levels	[2,2] or [2,2,2,2]	[2,2,2,2,2]
Sub-trajectory length	10	10
Prediction steps	5	5
Hierarchical subtokens	[32,32]	[128,128]

B Dataset Details

We summarize here the datasets used in our experiments with additional implementation details. All datasets are split into train/validation/test with an 80-10-10 ratio by frame count, unless otherwise specified. Frames are scaled to $[0, 1]$ and resized to the image resolutions reported in Table 4.

Multigrid (Simulation). We extend the MiniGrid environment [7] to support multiple independently moving agents. Each video consists of 32×32 gridworlds rendered at 128×128 resolution, with between 2–4 agents moving simultaneously. The number of factors K is set to 4 to match the number of independent agents. We use the official `empty-agent_K-v0` (train) and `v1` (validation) splits from the Minari dataset format.

Procgen (Simulation). We adopt the Procgen benchmark [8] with background rendering disabled for consistency. Videos are rendered at 224×224 resolution. We set the number of factors $K = 32$ to capture the diverse moving entities.

nuPlan (Real-world). We use the nuPlan benchmark [4], restricting to front-facing camera streams only. Videos are downsampled from 10 fps to 5 fps (frame skip = 2). Frames are resized to 224×224, and the number of factors is set to $K = 16$ to capture vehicles and pedestrians in each scene. We use the `train_CAM_F0.hdf5` and `valid_CAM_F0.hdf5` splits.

Sports (Real-world). The sports dataset combines five sub-datasets: (1) soccer, basketball, and volleyball sequences from SportsMOT [9]; (2) volleyball data from [17]; (3) tennis videos from Playable Video Generation [21]; (4) the TenniSet dataset [11]; and (5) basketball clips from BASKET [27]. All videos are sampled at 5 fps (frame skip = 3 from original 15 fps) and resized to 224×224. Because multiple players are present in each scene, we set $K = 32$.

EGTEA (Real-world). We additionally evaluate on EGTEA, an egocentric video dataset of hand-object interactions. Following prior work, we use the official train/validation split with frame skip 2 and resize frames to 224×224. We set $K = 32$ to reflect the diversity of object-level factors.

C Examples of Video Frame Reconstruction of the Encoder-Decoder Feature Extractor

Figure 5: The reconstruction results on nuPlan dataset. The images on top are the original frames, the images at the bottom are reconstructed images using the extracted features.

Figure 6: The reconstruction results on Sports and dataset. The images on top are the original frames, the images at the bottom are reconstructed images using the extracted features.

D Explicit slot matching

While slot attention is good at factorizing the state features into K slots, each attending to a subarea in the frame, there is no guarantee of entity tracking. To perform separate dynamics learning for each slot-based factor is highly based on the assumption that the slot s_t^k at time t and slot s_{t+1}^k at time $t+1$ we used for each IDM and FDM are corresponded to the same entity (subarea). Therefore, we tried two categories of slot matching, one is slot consistency loss and the other is recursive slot initialization.

Because there is no constraint to enforce the same distribution for the slot representations of different frames, which can cause problem for the following dynamics learning, we applied a slot distribution loss

$$\mathcal{L}_{\texttt{slot_dist}} = D_{KL}(s_t^{1:K} \| s_{t+1}^{1:K})$$

to encourage the slot representations across consecutive frames follow the same distribution. To enforce the consistent ordering of the slots, we also tried hard 1-to-1 Hungarian matching with MSE loss and greedy top-1 matching with contrastive infoNCE loss. For Hungarian matching, we find the permutation σ^* (one-to-one assignment) that minimizes total cost:

$$\sigma^* \;=\; \arg\min_{\sigma \in \mathfrak{S}_K} \sum_{i=1}^{K} C_{i,\sigma(i)},$$

where $\mathfrak{S}_K$ is the set of permutations of $\{1, \ldots, K\}$, and $C_{i,\sigma(i)}$ is the distance between the paired slots s_t^i and $s_{t+1}^{\sigma(i)}$. Then Hungarian matching loss is the matched cost:

$$\mathcal{L}_{\texttt{slot_Hungarian}} \;=\; \sum_{i=1}^{K} C_{i,\sigma^*(i)}.$$

For greedy top-1 matching, we treat each s_t^i as a query, and use s_{t+1}^j for all j as keys. Then top-1 slot matching loss for slot s_t^i is:

$$\mathcal{L}_i \;=\; -\log\left(\frac{\exp\big(\,\mathrm{sim}(s_t^i,\, s_{t+1}^{j^*})/\tau\big)}{\sum_{j=1}^{K} \exp\big(\,\mathrm{sim}(s_t^i,\, s_{t+1}^{j})/\tau\big)}\right),$$

where j^* is the index of the most similar slot (top-1 match):

$$j^* \;=\; \arg\max_{j}\; \mathrm{sim}(s_t^i,\, s_{t+1}^j).$$

Then contrastive matching loss is

$$\mathcal{L}_{\texttt{slot_contrastive}} \;=\; \frac{1}{K}\sum_{i=1}^{K} \mathcal{L}_i.$$

We augment the FLAM training objective $\mathcal{L}_{\texttt{LAM}}$ with each of the slot consistency loss above, but there is no significant improvement in the accuracy of future frame prediction. Therefore, we use recursive slot initialization instead. For the slot attention learning at time t, we initialize the slot embeddings with the values from slots of previous time $t-1$ rather then random initialization, and then start the iterative slot updates. The results in Fig. 7 show that recursive slot initialization helps learn better slots, thus we use this implementation as the default slot initialization method for our FLAM model.

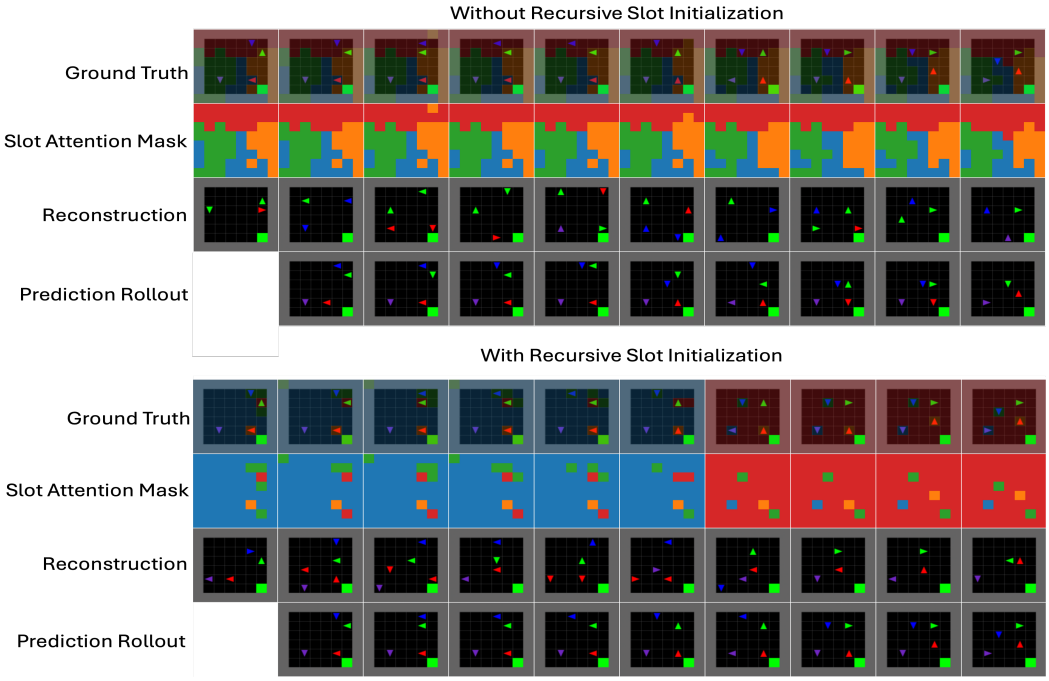

Figure 7: Comparison of results with and without recursive slot initialization on Multigrid dataset. The first row is the ground truth frames from the dataset. The second row is the attention mask from the factorizor module, in which each color represent a factor. The third row is the reconstructed frames,

E Examples of Reconstruction and Prediction Rollout Results

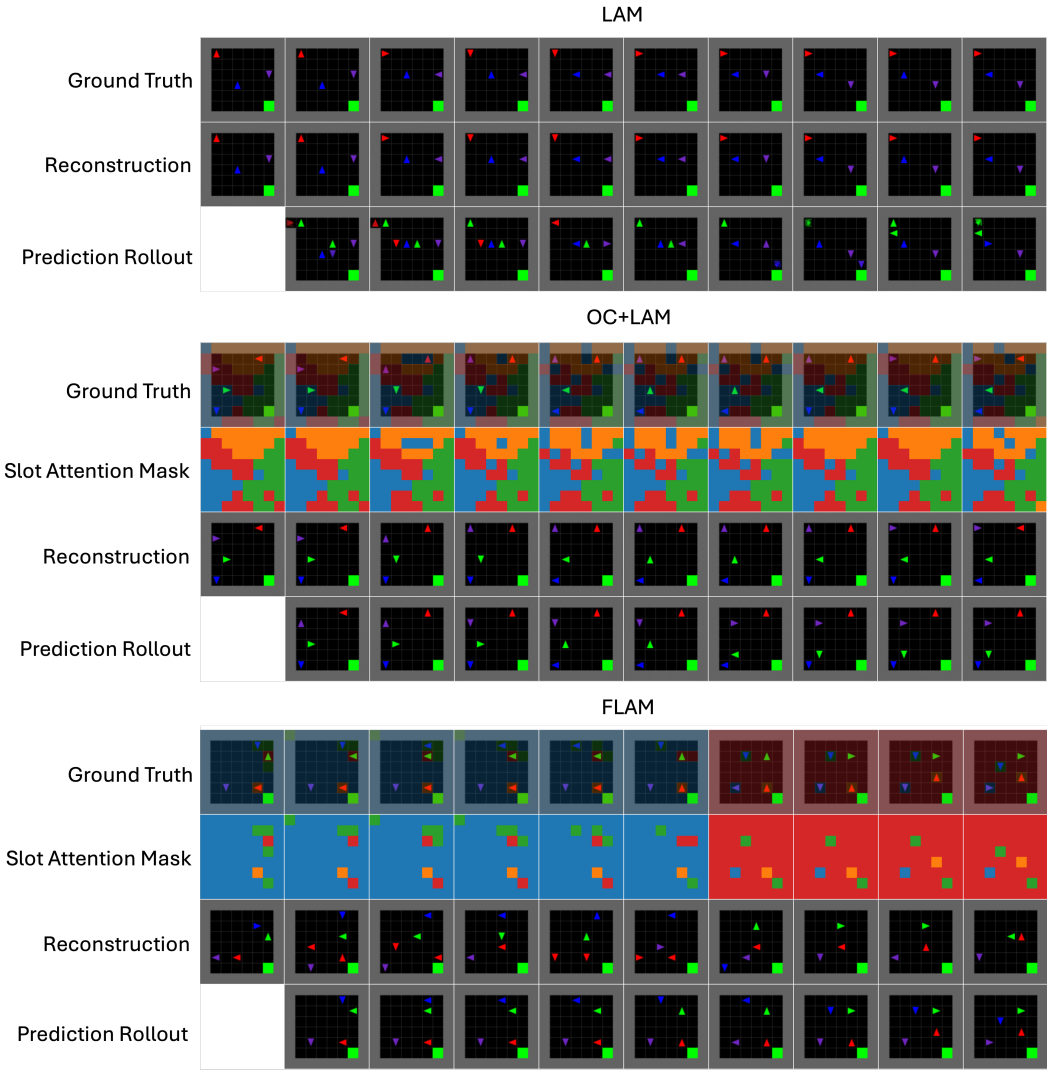

Figure 8: Example of reconstruction and prediction rollout results of LAM, OC+LAM and FLAM on Multigrid dataset.

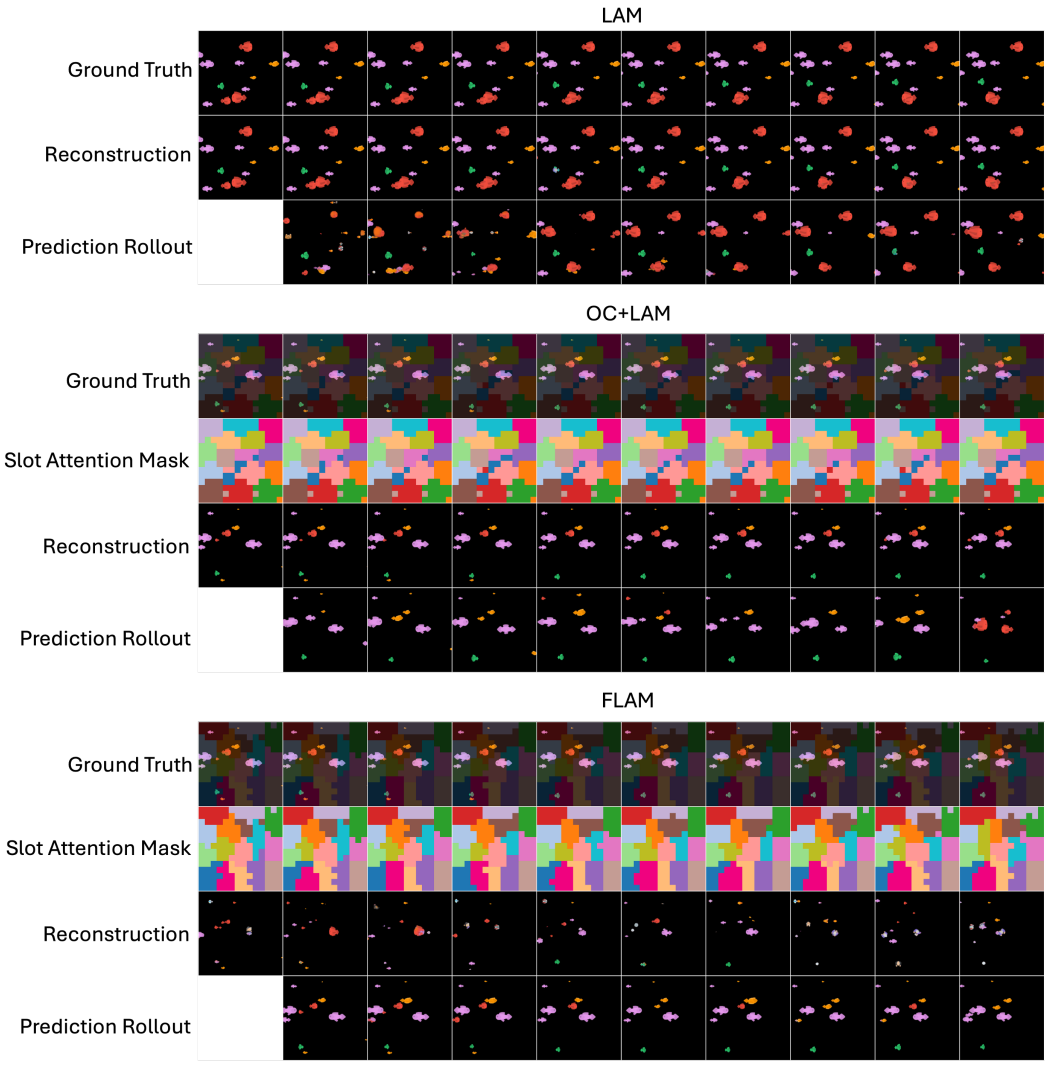

Figure 9: Example of reconstruction and prediction rollout results of LAM, OC+LAM and FLAM on Procgen dataset.

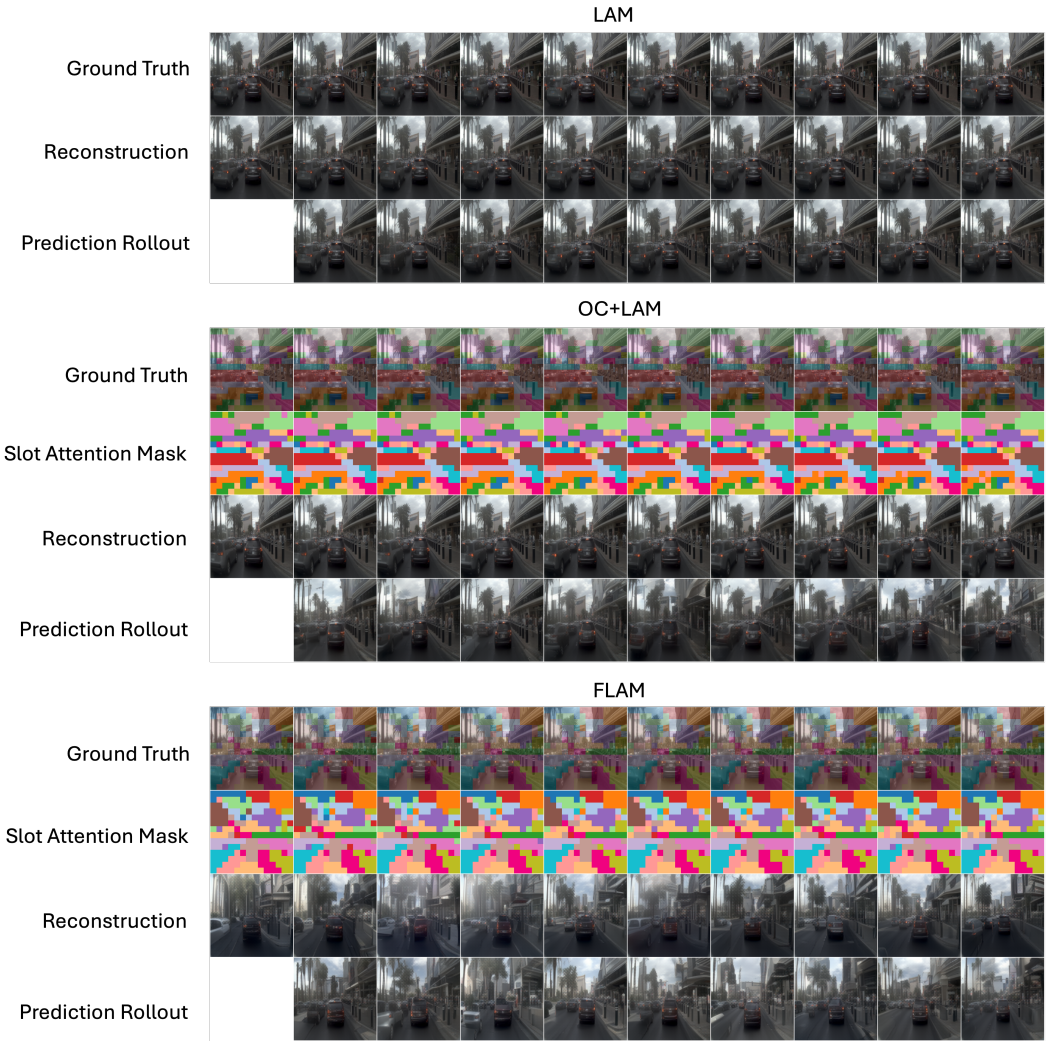

Figure 10: Example of reconstruction and prediction rollout results of LAM, OC+LAM and FLAM on nuPlan dataset.

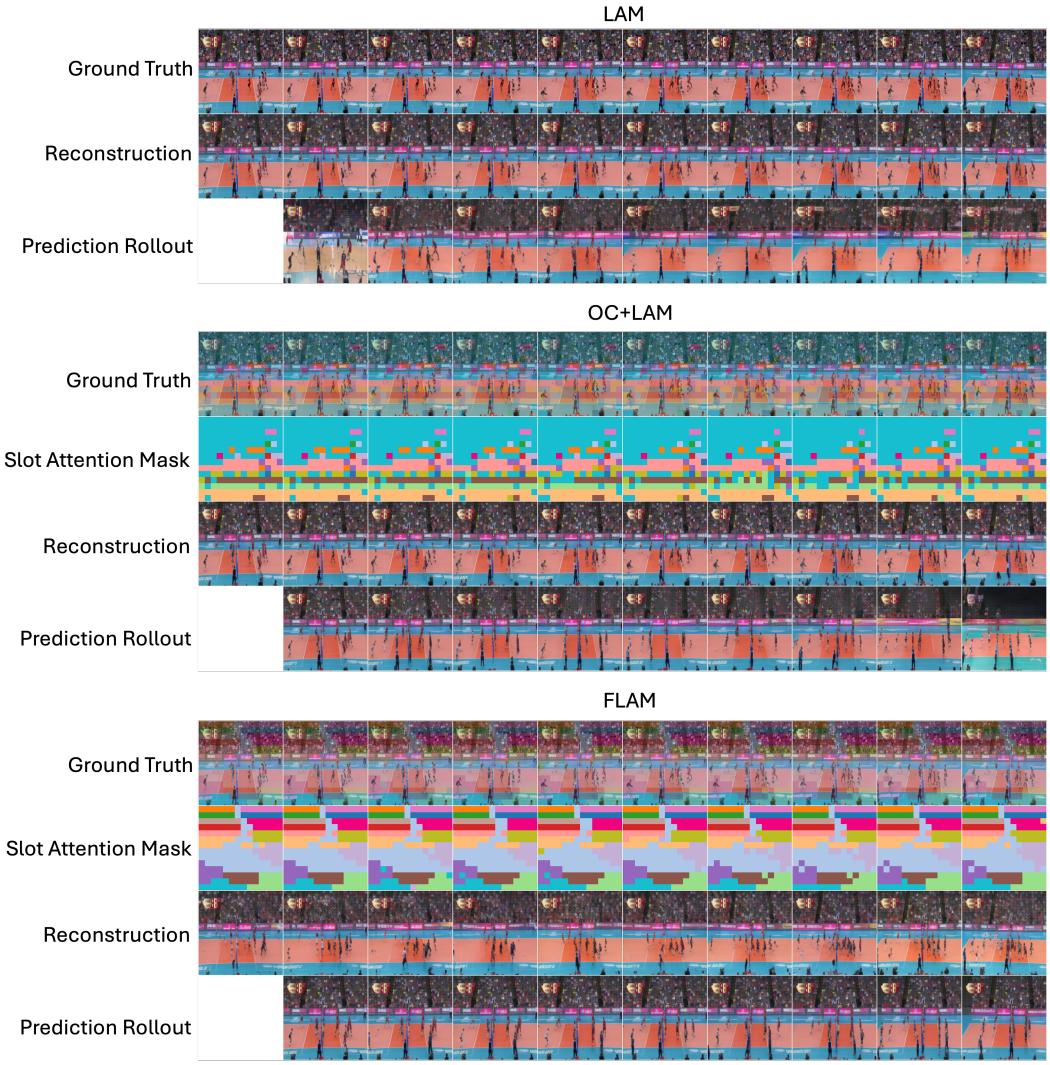

Figure 11: Example of reconstruction and prediction rollout results of LAM, OC+LAM and FLAM on Sports dataset.