A Related works

A.1 Sparse training signals and representation

The concept of frame-masked RL has distinctive characteristics that set it apart from other related topics in the field:

1. Sparse Reward Problems: Sparse reward problems, as studied in prior work (Fu et al., 2017; Riedmiller et al., 2018), involve scenarios where the environment provides meaningful reward signals only rarely, often resulting in a lack of guidance for learning agents. In contrast, frame-masked RL, as discussed in this paper, focuses on training agents more efficiently by utilizing fewer visual observations (high-dimensional) rather than dealing with the scarcity of reward signals.
2. Partially Observable Markov Decision Process (POMDP; Monahan (1982); Ghosh et al. (2021)): POMDP is a variant of the regular MDP where the agent receives observations from the observation space ($o \in \Omega$) instead of directly sensing the states ($s \in \mathcal{S}$). While POMDP addresses the issue of partial observability, frame-masked RL emphasizes the loss of temporal information and the approximation of the original transition structure. It is distinct from focusing solely on the mismatch between state and observation spaces.
3. Sparse Representation Learning for RL: Sparse representation learning for RL, explored in earlier studies (Liu et al., 2019; Le et al., 2017), involves leveraging techniques such as dictionary learning or sparsity-inducing regularization terms to obtain compact representations within neural networks. In contrast, frame-masked RL's primary objective is not sparse representation learning; instead, it aims to learn a meaningful latent space using sparse state signals, focusing on the quality of the representation rather than its sparsity.

In summary, while there are overlapping themes in these related areas, frame-masked RL stands out by addressing the challenges associated with training RL agents efficiently when they have limited access to high-dimensional visual observations while preserving the quality of the learned latent space.

A.2 Clustering methods

SelfDreamer introduces a novel clustering approach to group actions for the purpose of state space regularization. In this section, we compare our method to established clustering methods that address similar challenges:

1. Centroid-based Clustering: Centroid-based clustering, as introduced by MacQueen (1967), partitions data into k non-overlapping clusters by minimizing the sum of squared distances between each point and the centroid of its assigned cluster. While this method minimizes distances within the same cluster and maximizes distances between different clusters, it can struggle with entangled boundaries when centroids are initialized closely. In contrast, SelfDreamer focuses on learning robust prototypes by enforcing greater distance between each pair of prototypes.
2. Density-based Clustering: Density-based clustering, as outlined by Ester et al. (1996), groups data points that are spatially close with high density while identifying noise points. However, in the context of RL, the density of action groups can vary significantly, making clustering based solely on density less flexible in accommodating the distribution.
3. Distribution-based Clustering: Distribution-based clustering, following the work of McLachlan & Basford (1988), assumes that each cluster follows a specific probability distribution and uses statistical methods to estimate distribution parameters that best fit the data. This approach typically requires a predefined number of clusters, which can be challenging to determine without domain knowledge or extensive trial-and-error. In contrast, SelfDreamer employs a larger number of prototypes and automatically retires redundant ones, offering more flexibility.
4. Connectivity-based Clustering (Hierarchical Clustering): Connectivity-based clustering, often referred to as hierarchical clustering and introduced by Defays (1977), builds a hierarchy of clusters by iteratively merging or dividing clusters based on a distance metric.

However, in RL, the distribution of actions dynamically evolves throughout training, and action groups must be consistent across batches. Static results produced by hierarchical clustering may not be as applicable as the learnable prototypes in SelfDreamer.

In summary, SelfDreamer's approach to clustering actions and coupling them with transition prototypes provides a distinct advantage in addressing the challenges of action grouping and state space regularization compared to traditional clustering methods, offering greater flexibility, adaptability, and robustness in the RL context.

B DETAILED ALGORITHM OF SELFDREAMER

We present our proposed algorithm, SelfDreamer, in Algorithm 2.

Algorithm 2: SelfDreamer

Initialize the dataset $\mathcal{D}$ with S random seed episodes.
Initialize model parameters θ, ϕ, ψ, c^A, c^T randomly.
while *not converged* **do**
  `/* Collect Experience */`
  Reset environment $o_1 = env.reset()$
  for *time step* $t = 1..T$ **do**
    Compute current state $s_t \sim p_\theta(s_t|s_{t-1}, a_{t-1}, o_t)$
    Sample action $a_t \sim q_\phi(a_t|s_t)$
    Conduct sampled action $r_t, o_{t+1} \leftarrow env.step(a_t)$
  Store experience to the dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)\}_{t=1}^{T}$.
  for *update step* $c = 1..C$ **do**
    Sample training sequence $\{(o_t, a_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    `/* Representation Learning */`
    Compute model states $s_t \sim p_\theta(s_t|s_{t-1}, a_{t-1}, o_t)$
    Update θ by SelfDreamer loss described in Sec. 3.
    `/* Behavior Learning */`
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ from each state s_t
    Predict rewards and values $\{q_\theta(s_\tau), v_\psi(s_\tau)\}_{\tau=t}^{t+H}$
    Compute the λ-return $V_\lambda(s_\tau)$
    Update actor $\phi \leftarrow \phi + \alpha \nabla_\phi \sum_{\tau=t}^{t+H} V_\lambda(s_\tau)$
    Update critic $\psi \leftarrow \psi - \alpha \nabla_\psi \sum_{\tau=t}^{t+H} \frac{1}{2} ||v_\psi(s_\tau) - V_\lambda(s_\tau)||^2$

Notations

o_t	Observation
s_t	Model state
a_t	Action
r_t	Reward

Model components

$p_\theta(s_t\|s_{t-1}, a_{t-1}, o_t)$	Representation
$q_\theta(s_t\|s_{t-1}, a_{t-1})$	Transition
$q_\theta(r_t\|s_t)$	Reward
$q_\phi(a_t\|s_t)$	Action
$v_\psi(s_t)$	Value

Hyperparameters

S	Seed episodes
T	Length of interaction
C	Collect interval
L	Sequence length
H	Imagination horizon
α	Learning rate

C HYPERPARAMETERS

Table 5 shows the hyperparameters we adopted in our experiments.

Table 5: Hyperparameters used in our experiments. These hyperparameters are shared by all methods, except for the batch size of TPC, which is set to 150 to have a similar training time as SelfDreamer. We also set the scale of reward head loss to 8 and 64 specifically for SelfDreamer with frame-masking period set to 2 and 3, and keep the scale to 1 for the baseline methods as this setting does not improve their final results.

Name	**Symbol**	**Value**
World Model		
Dataset size	-	$2 * 10^6$
Batch size	B	50
Sequence length	L	50
RSSM number of units	-	200
KL balancing	-	0.8
World model learning rate	-	$3 * 10^{-4}$
Reward loss scale	-	1
Reward loss scale (natural background)	-	1000
SelfDreamer number of prototpyes	-	$2^{dim(\mathcal{A})+1}$
SelfDreamer loss scale	-	0.5
Behavior		
Imagination horizon	H	15
Discount factor	γ	0.99
λ-target parameter	λ	0.95
Actor gradient mixing	ρ	0
Actor entropy loss scale	η	$1 * 10^{-4}$
Actor learning rate	-	$8 * 10^{-5}$
Critic learning rate	-	$8 * 10^{-5}$
Slow critic update interval	-	100
Common		
Policy steps per gradient step	-	5
Optimizer	-	Adam
Gradient clipping	-	100
Epsilon	ϵ	10^{-5}
Weight decay	-	10^{-6}
MLP number of layers	-	4
MLP number of units	-	400
Activation function	-	ELU

D Further empirical results

D.1 Ablation study

In Figure 3, we conduct an ablation study to examine the individual contributions of action and transition prototypes to the overall SelfDreamer algorithm. This analysis involves two scenarios:

1. Fixed Action Prototypes: In this scenario, we keep the randomly initialized action prototypes fixed and do not update them during training. Only the transition prototypes are allowed to learn and adapt.
2. Direct Cosine Distance Minimization: In this scenario, we bypass the use of transition prototypes and directly minimize the pairwise cosine distances between transitions caused by similar actions, as described in Equation 6.

The results of the ablation study reveal the following insights:

- Easier Tasks with Smaller Action Spaces: In tasks such as Cartpole Swingup Sparse and Reacher Easy, which have smaller action spaces, both the fixed action prototypes and direct

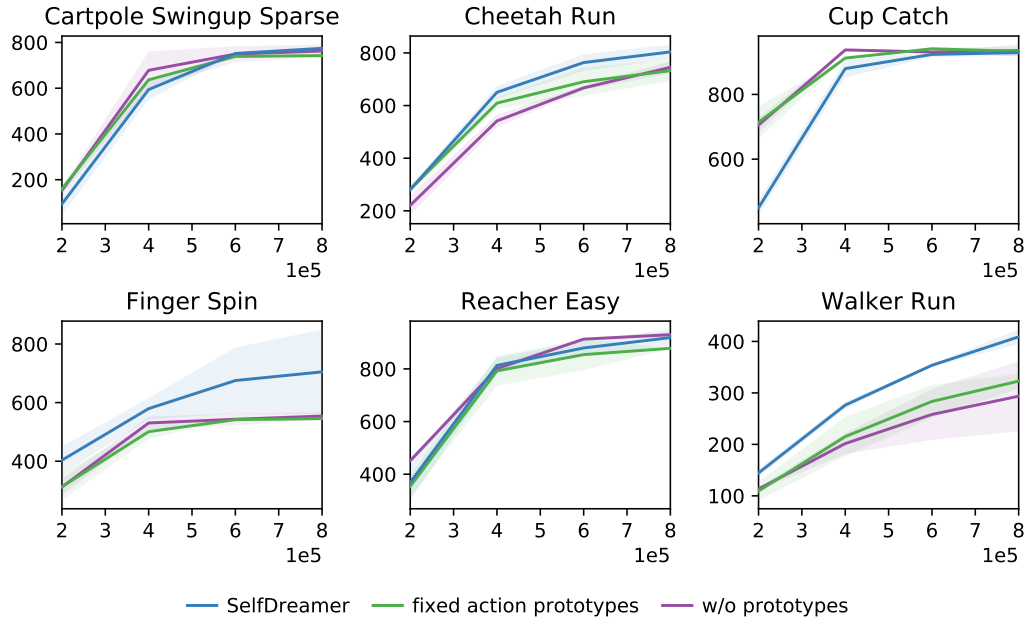

Figure 3: Ablation study under frame-masked DMC (3x less state signals).

cosine distance minimization methods exhibit similar performance to the fully equipped SelfDreamer. Additionally, in the Cup Catch task, despite SelfDreamer's initial cold start, after the convergence of dual prototypes, the final returns become nearly identical for all three methods. SelfDreamer even achieves near-optimal behavior on one of the seeds, while the other two methods converge to sub-optimal policies on all seeds.

- Tasks with Larger Action Spaces: In contrast, for tasks featuring larger action spaces, such as Cheetah Run and Walker Run, SelfDreamer provides significant performance improvements. In particular, SelfDreamer outperforms the ablated methods by 9% and 33% in Cheetah Run and Walker Run, respectively. This emphasizes the essential role played by both action and transition prototypes in SelfDreamer's success, especially in more complex tasks with larger action spaces.

These results underscore the significance of both action and transition prototypes in SelfDreamer's ability to effectively capture and leverage action-transition relationships, particularly in challenging RL scenarios with larger action spaces.

D.2 Frame-masked DMC (4x and 5x less state signals)

In Tables 6 and 7, we present the results for frame-masked DMC with larger masking periods set to 4 and 5, respectively. These experiments aim to evaluate the impact of even greater frame masking on performance compared to the smaller masking periods presented in Table 2 and Table 3.

As observed in the tables, increasing the frame-masking period leads to significant performance drops across all tasks compared to the smaller masking periods. In some cases, the performance degradation is severe, resulting in unacceptable or trivial solutions, such as in the case of Walker Run.

It's important to note that our method, SelfDreamer, struggles to benefit from the dual-prototypical design in these scenarios where the number of padded frames greatly outnumbers the genuine frames. This imbalance in frame distribution makes it challenging for SelfDreamer to create a meaningful representation with action-consistent transitions, leading to the observed performance drops.

These results serve as references for future research in the context of frame-masked reinforcement learning, highlighting the challenges and limitations associated with larger frame-masking periods.

Table 6: Final performance in frame-masked DMC (**4x** less state signals).

Task	Dreamer	DreamerPro	SelfDreamer
Cartpole Swingup Sparse	816 ± 34	751 ± 23	773 ± 27
Cheetah Run	680 ± 163	765 ± 11	738 ± 92
Cup Catch	946 ± 12	951 ± 7	943 ± 7
Finger Spin	658 ± 229	644 ± 147	$\mathbf{696 \pm 191}$
Reacher Easy	702 ± 21	926 ± 47	894 ± 97
Walker Run	71 ± 10	265 ± 43	$\mathbf{271 \pm 32}$

Table 7: Final performance in frame-masked DMC (**5x** less state signals).

Task	Dreamer	DreamerPro	SelfDreamer
Cartpole Swingup Sparse	832 ± 13	640 ± 79	694 ± 101
Cheetah Run	568 ± 126	686 ± 68	581 ± 69
Cup Catch	888 ± 48	924 ± 5	$\mathbf{949 \pm 11}$
Finger Spin	613 ± 77	624 ± 108	567 ± 74
Reacher Easy	950 ± 5	810 ± 38	879 ± 68
Walker Run	52 ± 23	192 ± 38	$\mathbf{209 \pm 21}$

D.3 VISUALIZATION

In Figures 4 to 7, we provide visualizations of the final policies trained by four different algorithms: Dreamer, DreamerPro, TPC, and our proposed algorithm, SelfDreamer. These visualizations are focused on the Walker Run environment, where we have applied a frame-masking period of 2, resulting in significant differences in performance.

Here are the observations from the visualizations:

- Figure 4: The agent trained by Dreamer sometimes exhibits a running behavior with its knees on the ground, which may not resemble natural walking or running.
- Figure 5: The agent trained by TPC encounters difficulty and gets stuck at the starting point, indicating challenges in learning an effective policy for the task.
- Figure 6: The agent trained by DreamerPro occasionally stumbles harshly during its run, suggesting instability in its learned policy.
- Figure 7: In contrast, the agent trained by SelfDreamer demonstrates a smooth and natural running behavior, which appears more consistent with the desired locomotion task.

These visualizations provide qualitative evidence of SelfDreamer's ability to learn policies that result in more natural and stable behaviors in the Walker Run environment compared to the other baselines, emphasizing the efficacy of our proposed approach.

Figure 4: Visualization for Dreamer on Walker Run, trained with frame-masking period set to 2.

Figure 5: Visualization for TPC on Walker Run, trained with frame-masking period set to 2.

Figure 6: Visualization for DreamerPro on Walker Run, trained with frame-masking period set to 2.

Figure 7: Visualization for SelfDreamer on Walker Run, trained with frame-masking period set to 2.