Model-Based Reinforcement Learning under Random Observation Delays

Armin Karamzade, Kyungmin Kim, JB Lanier, Davide Corsi, Roy Fox
Department of Computer Science
University of California, Irvine

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

Delays frequently occur in real-world environments, yet standard reinforcement learning (RL) algorithms often assume instantaneous perception of the environment. We study random sensor delays in POMDPs, where observations may arrive out-of-sequence, a setting that has not been previously addressed in RL. We analyze the structure of such delays and demonstrate that naive approaches, such as stacking past observations, are insufficient for reliable performance. To address this, we propose a model-based filtering process that sequentially updates the belief state based on an incoming stream of observations. We then introduce a simple delay-aware framework that incorporates this idea into model-based RL, enabling agents to effectively handle random delays. Applying this framework to the Dreamer world-modeling scheme, our method consistently outperforms delay-aware baselines developed for MDPs and demonstrates robustness to delay distribution shifts during deployment. Additionally, we present experiments on simulated robotic tasks, comparing our method to common practical heuristics and emphasizing the importance of explicitly modeling observation delays.
Keywords: Reinforcement Learning, Model Based, Delays, POMDPs

1. Introduction

Despite the remarkable success of reinforcement learning (RL) across a wide range of domains, standard RL algorithms rely on the assumption of delay-free interaction with the environment. In practice, however, delays are pervasive and often unavoidable, particularly in real-world systems such as robotics, autonomous driving, and distributed control (Abadía et al., 2021; Mahmood et al., 2018; Fagundes-Junior et al., 2023). These delays can occur at different stages of the pipeline, such as sensing, processing, and communication. They are generally divided into two types: (i) *feedback delays*, the time lag in receiving observations, and (ii) *execution delays*, describing the delay between selecting an action and its execution in the environment. While these are highly common in practical applications, they are typically ignored or oversimplified in the RL literature.

When delays are present, a common workaround in robotics is to issue "no-op" actions, effectively instructing the agent to wait until the delayed observation arrives (Walsh et al., 2007). However, this approach is often impractical or even unsafe. For instance, an autonomous vehicle detecting a sudden obstacle via a low-latency sensor cannot wait for other sensors, as delay may cause a collision. Even defaulting to braking may not be viable, and the vehicle must infer the safest evasive motion under the uncertainty caused by delayed perception.

Even when delays are explicitly considered, they are often addressed with simplifying assumptions that fail to capture their full complexity in real-world tasks. Existing approaches assume either a fully observable environment, as in Markov Decision Processes (MDPs) (Katsikopoulos and Engelbrecht,

2003; Liotet et al., 2021; Derman et al., 2021; Liotet et al., 2022; Wu et al., 2024a), or fixed delays in Partially Observable MDPs (POMDPs) (Kim and Jeong, 1987; Karamzade et al., 2024). However, real-world systems often involve partial observability and random delays. Unlike MDPs, where each observation fully represents the state, POMDPs require integrating past observations to maintain, perhaps implicitly, a belief over it. With random delays, observations may arrive out-of-sequence (OOS), a phenomenon that does not arise in fixed-delay settings and is innocuous in MDPs, but critical in POMDPs, where relying only on the latest observation is insufficient for optimal control.

In this work, we consider random observation delays in POMDPs. To address the OOS phenomenon, we propose a latent-space filtering approach that enables effective learning in the presence of OOS observations imposed by random delays. By leveraging model-based approaches, our method forms a belief over the current latent state, given the set of available observations. In particular, the filtering process exploits a world model trained in the delayed environment to sequentially update the belief based on received observations. This belief state then serves as a sufficient statistic for policy learning under an incomplete set of observations to ensure actions are informed by only and all available inputs.

We evaluate on both synthetic control tasks and realistic simulated robotic environments. Our method outperforms existing approaches in fully observable MDP settings and is the only one capable of handling more realistic, partially observable scenarios. Notably, our approach also generalizes well to test-time delay distribution shifts: when trained on a wider delay distribution, it performs significantly better under shorter test-time delays and shows minimal performance degradation under longer ones. These results highlight its potential for real-world deployment, where delay patterns are often unknown in advance and nonstationary.

Here, we summarize the contributions of this paper as follows. (i) We study random observation delays in POMDPs and propose a framework that connects this setting to standard POMDP formulations. (ii) We introduce a filtering procedure for processing OOS observations within model-based RL. (iii) We present the first method designed for this setting and describe how to integrate the filtering process into existing model-based RL algorithms. (iv) We conduct extensive experiments across diverse environments, demonstrating superior performance over baselines and strong generalization to unseen delay distributions.

2. Preliminaries

A *Partially Observable Markov Decision Process (POMDP)* is a tuple $\mathcal{M} = \langle S, A, \mathcal{T}, r, \Omega, O, \gamma \rangle$, where S, A, and Ω denote the sets of states, actions, and observations, respectively. The transition dynamics is defined by $\mathcal{T}(s' \mid s, a)$, the reward function by $r(s, a)$, and the observation (emission) probabilities by $O(o \mid s)$. At each timestep t, the environment is in state $s_t \in S$, the agent receives an observation $o_t \sim O(o_t \mid s_t)$, selects an action $a_t \in A$, receives reward $r_t = r(s_t, a_t)$, and transitions according to $s_{t+1} \sim \mathcal{T}(s_{t+1} \mid s_t, a_t)$. The objective is to select actions that maximize the expected return $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where $\gamma \in [0, 1)$ is the discount factor.

2.1. Model-Based RL

Recent model-based RL (MBRL) approaches focus on learning a latent dynamics model, or world model, that captures the environment's behavior and enables long-term prediction (Ha and Schmidhuber, 2018; Hansen et al., 2022; Micheli et al., 2022; Hafner et al., 2025). In this framework, the agent maintains a latent state x_t governed by a parametrized transition model $p_\theta(x_t \mid x_{t-1}, a_{t-1})$,

and generates observations through a decoder $p_\theta(o_t \mid x_t)$. Note that rewards are usually part of the model, but here we omit them for brevity. Since the latent state is not directly observable in training data, a variational posterior $q_\theta(x_{\leq T} \mid o_{\leq T}, a_{<T}) = \prod_t q_\theta(x_t \mid x_{t-1}, a_{t-1}, o_t)$, first proposed by Hafner et al. (2019), can be used for the distribution of the latent state sequence of a particular observed episode $(o_{\leq T}, a_{<T})$ of length T. The model is trained by maximizing an evidence lower bound (ELBO) on the sequence log-likelihood, leading to the objective (Hafner et al., 2019):

$$\mathcal{L} = \sum_{t=0}^{T} \mathbb{E}_{q_\theta}[\ln p_\theta(o_t \mid x_t)] - \mathbb{E}_{q_\theta}[\mathbb{D}[q_\theta(x_t \mid x_{t-1}, a_{t-1}, o_t) \,\|\, p_\theta(x_t \mid x_{t-1}, a_{t-1})]], \tag{1}$$

where $\mathbb{D}$ is the Kullback–Leibler (KL) divergence. This objective encourages the latent state to retain sufficient information for reconstructing observations, while remaining consistent with the prior dynamics p_θ. Throughout the text, we omit the dependence on θ whenever it is clear from context.

Many existing works exploit this or similar models for reinforcement learning and planning (Ha and Schmidhuber, 2018; Hafner et al., 2019; Micheli et al., 2022; Zhang et al., 2023). One notable example is Dreamer (Hafner et al., 2025), which trains policies entirely within a learned Recurrent State Space Model (RSSM). Dreamer alternates between three key stages: (1) training the world model, (2) learning the policy through imagined trajectories, and (3) collecting new experiences.

3. Random Observation Delay Environments

We define a *Random Observation Delay Environment* as a pair $\langle \mathcal{M}, \mathcal{D} \rangle$, where $\mathcal{M}$ is a standard POMDP and $\mathcal{D}$ is a distribution over non-negative integers representing stochastic delays. At each time step t, the observation generated by the environment is not revealed immediately, but is instead delivered after a random delay $d_t \sim \mathcal{D}$[1]. That is, o_t becomes available at time $t + d_t$. The agent's actual observation at time t consists of all information scheduled to arrive at that step. We denote this (possibly empty) set by $\tilde{o}_t = \{(o_\tau, \tau) : \tau + d_\tau = t\}$. We assume that the agent observes the timestamp of delivered observations, but not d_τ for undelivered observations. We further assume that all delays are bounded, with D denoting the maximum possible delay.

In the fully observable setting, an equivalent delay-free MDP can be constructed by augmenting the state with the sequence of actions taken since the most recently observed state (Katsikopoulos and Engelbrecht, 2003). Similarly, in partially observable environments with constant delays, one can construct an equivalent delay-free POMDP by augmenting the state, while preserving the original observation space (Karamzade et al., 2024). In both cases, the augmentation only needs to track a sequence of past actions. However, with random delays, this structure is no longer sufficient, and reducing a delayed environment to a standard POMDP becomes more complex.

Reducing to a standard POMDP. Under random observation delays, the agent may potentially receive any nonnegative number of observations at each timestep. The new observation space becomes $\widetilde{\Omega} = \bigcup_{k=0}^{D+1} (\Omega \times \mathbb{N})^k$, where each new observation $\tilde{o}_t$ is a set of observations delivered at time t, each paired with its original emission timestamp. To encode this within a standard POMDP, we augment the state with a buffer u_t that stores all observations not previously received:

$$u_t = \{(o_\tau, \tau, d_\tau) : t - d_\tau \leq \tau \leq t\} \in U,$$

1. Here we assume that delays are i.i.d for simplicity. However, our analysis holds for non-stationary or state-dependent delays with some adjustments.

where U is the space of sets of observation–timestamp–delay tuples. The augmented state space is $\widetilde{S} = S \times U$, and the equivalent delay-free POMDP is given by the tuple $\langle \widetilde{S}, A, \widetilde{\mathcal{T}}, \widetilde{r}, \widetilde{\Omega}, \widetilde{O}, \gamma \rangle$, with:

$$\begin{aligned} \widetilde{r}((s_t, u_t), a_t) &= r(s_t, a_t), \\ \widetilde{O}(\tilde{o}_t \mid (s_t, u_t)) &= \delta(\tilde{o}_t = \{(o_\tau, \tau) : (o_\tau, \tau, t-\tau) \in u_t\}), \\ \widetilde{\mathcal{T}}((s_{t+1}, u_{t+1}) \mid (s_t, u_t), a_t) &= \mathcal{T}(s_{t+1} \mid s_t, a_t) \cdot \mathbb{P}_{\mathcal{D}}(u_{t+1} | u_t, s_{t+1}), \end{aligned}$$

where the observation function $\widetilde{O}$ deterministically returns all entries in the buffer that are scheduled for delivery at time t, and each new observation is drawn and stored in the buffer with its associated delay $\mathbb{P}_{\mathcal{D}}(u_{t+1} | u_t, s_{t+1}) = O(o_{t+1} | s_{t+1}) \mathcal{D}(d_{t+1})$, when $u_{t+1} = \{(o_\tau, \tau, d_\tau) \in u_t : \tau + d_\tau > t\} \cup \{(o_{t+1}, t+1, d_{t+1})\}$ for any o_{t+1} and d_{t+1}, and assigning probability 0 to any u_{t+1} that does not have this form.

This construction enables the use of standard POMDP algorithms, but at the cost of exponentially increased state and observation space sizes. In particular, the agent's belief must capture uncertainty not only over the latent state but also over pending, undelivered observations. This added complexity is fundamental to POMDPs with random delays, as observations may arrive out of sequence and cannot be ignored or replaced by the most recent one, unlike in MDPs.

World Model for the Reduced POMDPs. In the reduced POMDP formulation, the latent state at time t must be inferred from the set of observations received up to that point. Therefore, o_t would be replaced by $\tilde{o}_t$ in Eq. (1) and the resulting ELBO becomes:

$$\sum_{t=0}^{T} \mathbb{E}[\ln p(\tilde{o}_t \mid x_t)] - \mathbb{E}[\mathbb{D}(q(x_t \mid x_{t-1}, a_{t-1}, \tilde{o}_t) \,\|\, p(x_t \mid x_{t-1}, a_{t-1}))]. \tag{2}$$

While this formulation enables the use of standard MBRL tools in the delayed setting, it treats the set of received observations as a generic input and does not explicitly model the delay process. As a result, the model may learn unnecessarily complex dynamics to compensate for the partial and OOS nature of the observations, instead of exploiting the structure imposed by the delays.

4. Delay-Aware Model-Based RL

We introduce, in contrast to Section 3, a structured latent-space filtering approach to address random delays and OOS inputs within the context of MBRL. By maintaining a belief over the current latent state using only received observations, the agent can act effectively under incomplete information. This section presents the belief update formulation and its integration into the MBRL framework.

4.1. Out-of-Sequence Filtering via a World Model

In the scheme of Section 2.1, a standard model-based agent processes its observations as $q_\theta(x_t \mid x_{t-1}, a_{t-1}, o_t)$ and then bases its actions on x_t as if it were the state of a fully observable MDP. An agent operating under delayed observations, on the other hand, cannot reproduce the same latent process in time for the relevant decisions. Instead, our agent can form a belief over the x_t of an undelayed agent, and base its actions on this belief-state, as in a belief-state representation of a POMDP (Kaelbling et al., 1998). In this case, the exact belief $\phi_t \coloneqq \Pr_q(x_t \mid \tilde{o}_{\leq t}, a_{<t})$ is a sufficient

statistic of the available information, namely the received observations $\tilde{o}_{\leq t}$ and the past actions $a_{<t}$, for the latent state x_t.

It is worth highlighting two aspects of the temporal structure of this belief. First, unlike standard belief-states, it may not be possible to compute ϕ_t fully sequentially only from ϕ_{t-1}, a_{t-1}, and $\tilde{o}_t$. The reason is that ϕ_{t-1} can lose information of past observations that only becomes informative once delayed inputs arrive. For example, consider a state feature defined as the cumulative parity of a Bernoulli observation: if any observations are missing from $\tilde{o}_{\leq t-1}$, then ϕ_{t-1} will have a uniform belief of their parity. This will prevent recovering that state feature in ϕ_t from ϕ_{t-1} and $\tilde{o}_t$, even if all remaining observations arrive by time t. For sequential computation of ϕ_t, we therefore consider the belief sequence $\phi_{\tau,t} \coloneqq \Pr_q(x_\tau | \tilde{o}_{\leq t}, a_{<t})$, which may need partial recomputation at each step t.

The second temporal insight starts with noting that the "causal" variational structure of $q_\theta(x_{\leq t}|o_{\leq t}, a_{<t}) = \prod_\tau q_\theta(x_\tau|x_{\tau-1}, a_{\tau-1}, o_\tau)$ in Section 2.1 may be unable to represent the exact posterior of the generative process p_θ, because in the latter x_τ generally depends on all observations $o_{\leq t}$, rather than only on $o_{\leq \tau}$ in q_θ. Despite creating a variational gap, in which the ELBO is generally not a tight lower bound, this structure of q_θ is justified by the need to use it online in deployment, as mentioned above. A similar rationale reappears in the sequential computation of $\phi_{\tau,t}$, motivated here by greater sample efficiency and faster execution.

With these considerations in place, we define an auxiliary transition distribution $\psi_{\tau,t}$ over latent states that retroactively incorporates information available at time t into the filtering process at time $\tau \leq t$, using the learned models q_θ and p_θ:

$$\psi_{\tau,t}(x_\tau \mid x_{\tau-1}, a_{\tau-1}, [\tilde{o}_{\leq t}]_\tau^\tau) = \begin{cases} q_\theta(x_\tau \mid x_{\tau-1}, a_{\tau-1}, o_\tau) & \text{if } (o_\tau, \tau) \in [\tilde{o}_{\leq t}]_\tau^\tau, \\ p_\theta(x_\tau \mid x_{\tau-1}, a_{\tau-1}) & \text{otherwise,} \end{cases}$$

where $[\tilde{o}_{\leq t}]_{t_1}^{t_2}$ is the restriction of $\tilde{o}_{\leq t}$ to timestamps in $[t_1, t_2]$, with $[\tau, \tau]$ representing a single timestamp. This auxiliary kernel ψ serves as a time-dependent transition function that updates the state based on whether an observation at time τ becomes available by time t. It uses the variational posterior when o_τ is observed, and otherwise defaults to the prior dynamics model. This lets us write an approximate $\phi_t = \phi_{t,t}$ recursively as

$$\begin{aligned} \phi_{\tau,t} &\overset{\text{def}}{=} \Pr(x_\tau \mid \tilde{o}_{\leq t}, a_{<t}) \\ &= \mathbb{E}_{x_{\tau-1}|\tilde{o}_{\leq t}, a_{<t}}[\Pr(x_\tau \mid x_{\tau-1}, \tilde{o}_{\leq t}, a_{<t})] && (3) \\ &= \mathbb{E}_{x_{\tau-1}|\tilde{o}_{\leq t}, a_{<t}}[\Pr(x_\tau \mid x_{\tau-1}, a_{\tau-1,\ldots,t-1}, [\tilde{o}_{\leq t}]_\tau^t)] && (4) \\ &\approx \mathbb{E}_{x_{\tau-1}|\tilde{o}_{\leq t}, a_{<t}}[\Pr(x_\tau \mid x_{\tau-1}, a_{\tau-1}, [\tilde{o}_{\leq t}]_\tau^\tau)] && (5) \\ &\approx \mathbb{E}_{(x_{\tau-1}|\tilde{o}_{\leq t}, a_{<t}) \sim \phi_{\tau-1,t}}[\psi_{\tau,t}(x_\tau \mid x_{\tau-1}, a_{\tau-1}, [\tilde{o}_{\leq t}]_\tau^\tau)], && (6) \end{aligned}$$

where $\phi_{0,t}$ is the initial latent state distribution, Eq. (3) follows from simple conditioning on $x_{\tau-1}$, Eq. (4) omits past observations and actions since they are separated by $x_{\tau-1}$, Eq. (5) omits future observations and actions as a "causal" variational approximation, and in Eq. (6) the learned q_θ and p_θ approximate the corresponding distributions, and $\phi_{\tau-1,t}$ indicates that the same approximations are repeated in each step.

The belief-state we approximate here is a sufficient statistic for control with respect to the information actually available at time t as no policy can exploit observations that have not yet arrived. Thus, conditioning the policy on ϕ_t is optimal for that information set (Kaelbling et al., 1998). Note that, in the undelayed case, where $\tilde{o}_{\leq t}$ includes all observations and ψ always uses q_θ, this process

Algorithm 1: Delay-Aware MBRL

Input: $\mathcal{A}$: Model-Based RL algorithm optimizing objective (1)

```
// Inference Mode
```

Initialize $\kappa_{-1} = 0$ and $\hat{o}_{-1} = \emptyset$;

for *time t in episode* **do**

  Receive $\tilde{o}_t$;

  Compute ϕ_t from $\phi_{\kappa_{t-1}-1,t-1}$ and $\hat{o}_{t-1} \cup \tilde{o}_t$ using Eq. (6) with the model (p, q);

  Find the time κ_t of the first unreceived observation and checkpoint $\phi_{\kappa_t-1,t}$;

  Store $\hat{o}_t = [\hat{o}_{t-1} \cup \tilde{o}_t]_{\kappa_t}^{t}$, discard $[\hat{o}_{t-1} \cup \tilde{o}_t]_{\kappa_{t-1}}^{\kappa_t-1}$ that will no longer be needed; Execute action $a_t \sim \pi(\cdot \mid \phi_t)$;

end

```
// Training Mode
```

foreach *update step* **do**

  Collect data with π using inference mode and store it in replay buffer B;

  Use $\mathcal{A}$ with sample $(o_{\leq T}, a_{<T}, r_{\leq T}, d_{<T}) \sim B$ to update world model (p, q);

  Compute beliefs $\phi_{<T}$ using Eq. (6);

  Use $\mathcal{A}$ to update the policy $\pi(\cdot \mid \phi_t)$ (and critic $V(\phi_t)$, if applicable);

end

reduces to the standard latent state process. More generally, we can represent an approximate belief distribution using particle filtering (Ma et al., 2020b) to capture uncertainty. Each step of the recursion in Eq. (6) is then represented by K particles $\{x_\tau^k\}_{k=1}^K$ propagated independently through the model.

4.2. Incorporating into RL

We present a general training procedure for incorporating belief inference into MBRL under delayed observations. This framework, outlined in Algorithm 1, can be applied to any algorithm employing an RSSM-style world model. Modifications to the standard pipeline are highlighted in blue. The key idea is to train the world model on complete, ordered trajectories as in undelayed settings, while training the policy on belief states inferred from partially observed sequences using Eq. (6).

During inference, the agent maintains a timestamped buffer of observations that may be needed for belief recomputation, when earlier observations eventually arrive. Let κ_t be the earliest time of an observation unreceived by time t. Then $\phi_{\tau,t}$ for all $\tau < \kappa_t$ already uses all observations, and will thus never need recomputation. $\hat{o}_t := [\tilde{o}_{\leq t}]_{\kappa_t}^{t}$ is therefore a sufficient buffer for future recomputation. Importantly, $t - \kappa_t < d_{\kappa_t} \leq D$, so this buffer has at most D observations. In practice, one could set this maximum delay to some pre-defined constant that trades-off inference memory with performance.

In each inference step t, the agent adds new observations $\tilde{o}_t$ to the buffer $\hat{o}_{t-1}$, recomputes ϕ_t from the latest checkpoint $\phi_{\kappa_{t-1}-1,t-1}$ using available observations $[\tilde{o}_{\leq t}]_{\kappa_{t-1}}^{t}$ using Eq. (6), and stores $\hat{o}_t = [\tilde{o}_{\leq t}]_{\kappa_t}^{t}$. The agent then selects an action from the policy defined on the belief space $\pi(\cdot \mid \phi_t)$.

While training happens in a delayed environment, world model training remains identical to the standard setting. This is possible because learning and data collection processes are decoupled. In particular, after each episode terminates, we wait until all pending observations arrive before storing the ordered trajectory in the replay buffer. To make policy training consistent with deployment, i.e., delay-aware, the replay buffer is augmented with the delays d_t. Replaying these indices allows us to reconstruct the same partial buffers, recompute beliefs, and provide them as inputs to the downstream policy learning algorithm.

Training in Imagination. MBRL algorithms that model prior dynamics train the policy purely on imagined trajectories to improve sample efficiency. In Algorithm 1, we noted that policy learning is identical to the delay-free case, except that the policy input is now the belief ϕ_t inferred from received observations. In delayed environments, an optimal policy requires simulating the belief dynamics under the delay distribution. However, because policy training relies on imagined trajectories, ground-truth observations needed to model belief dynamics are unavailable during imagination. As a result, training in imagination corresponds to a scenario where the agent receives no further observations. Despite this distribution shift, our experiments in Section 5.1.2 indicate that the policy still generalizes well to unseen delay distributions.

5. Experiments

We evaluate our method through two sets of experiments. Section 5.1 covers fully observable MuJoCo environments (Todorov et al., 2012), comparing against delayed-MDP baselines. As no prior work addresses our setting, we adopt MDP-based methods for fair comparison. Section 5.2 evaluates our method on four Meta-World environments (Yu et al., 2019) with visual inputs, which are inherently partially observable. In Meta-World, we compare against practical heuristics commonly adopted in the presence of delays, as naïve alternatives in the absence of existing delay-aware methods.

Our method[2] builds on Dreamer-v3 (Hafner et al., 2025), an MBRL algorithm that learns an RSSM-based world model (Section 2.1), modified according to Algorithm 1. Most experiments use a single particle ($K = 1$) to approximate the belief, for computational efficiency, as we observed no significant performance differences across different values of K (Appendix B.4). Dreamer's world model consists of both deterministic and stochastic paths, preserving information over long horizons, which likely contributes to making single particle sufficient. The version of Dreamer that treats the received observations $\tilde{o}_t$ as generic inputs, represented by stacked frames, is referred to as `Stack-Dreamer` (Eq. (2)), and the version using the **D**elay-**A**ware Algorithm 1 as `DA-Dreamer`. Each experiment is repeated for 5 random seeds.

5.1. Main Comparison with Baselines

For MuJoCo environments, we compare with `DCAC` (Bouteiller et al., 2020) and the best-performing method of Wang et al. (2023), referred to as "detach Encoding" in their work and, for clarity, as `Encoding` here. Throughout, we denote the uniform distribution by $\mathcal{U}\{a, b\}$ and rounded truncated Gaussian by $\mathcal{N}^+(\mu, \sigma^2)$, where we round each sample of a truncated Gaussian to the nearest integer.

5.1.1. Results

Table 1 reports the expected returns of all methods under two uniform delay distributions. `DA-Dreamer` achieves better performance than other methods in more environments, despite lacking the prior knowledge that the environments are fully observable, by inheriting this property from the underlying world model. In contrast, `Stack-Dreamer` performs well in simpler settings but fails to scale to environments with larger observation spaces, such as Humanoid and HumanoidStandup. This supports our earlier hypothesis that a generic PODMP reduction that treats delayed observations as generic inputs leads to an unnecessarily complex latent space and fails to exploit the structure of delays. `DCAC` shows a sharp performance drop in Hopper and Humanoid under $\mathcal{U}\{0, 20\}$. These

2. The code is available at https://github.com/indylab/DA-Dreamer.

Table 1: Expected return of methods for different delay distributions across MuJoCo environments. Results represent the mean and 95% confidence interval. Bold values indicate statistical significance.

Delay	Environment	DCAC	Encoding	Stack-Dreamer	DA-Dreamer
$\mathcal{U}\{0, 10\}$	HalfCheetah-v4	3841.43 ± 802.96	4189.26 ± 401.81	1959.96 ± 839.58	**4985.40 ± 253.10**
	Hopper-v4	2394.67 ± 721.94	2373.70 ± 206.46	2694.22 ± 416.69	2251.36 ± 413.62
	Humanoid-v4	1062.81 ± 248.87	552.21 ± 31.89	522.13 ± 41.48	**1854.26 ± 205.04**
	HumanoidStandup-v4	145293.09 ± 4314.40	113240.31 ± 11024.15	93154.06 ± 17839.01	**220017.11 ± 23671.63**
	Reacher-v4	−6.36 ± 0.55	−6.79 ± 0.30	−6.81 ± 0.23	−6.37 ± 0.13
	Swimmer-v4	40.53 ± 1.55	121.56 ± 23.92	**346.58 ± 2.51**	**347.00 ± 5.64**
$\mathcal{U}\{0, 20\}$	HalfCheetah-v4	2144.86 ± 526.30	**4242.77 ± 213.47**	1045.15 ± 179.75	2958.56 ± 54.19
	Hopper-v4	6.48 ± 0.83	1707.41 ± 147.40	**2981.87 ± 275.14**	1713.85 ± 343.09
	Humanoid-v4	112.08 ± 76.92	545.76 ± 17.19	418.24 ± 42.17	**855.97 ± 95.77**
	HumanoidStandup-v4	139806.95 ± 20078.74	118456.86 ± 7541.70	101241.22 ± 4591.26	**195611.38 ± 14140.51**
	Reacher-v4	−6.59 ± 0.44	−7.17 ± 0.24	−6.71 ± 0.21	−7.06 ± 0.16
	Swimmer-v4	34.19 ± 3.38	117.84 ± 20.27	**348.32 ± 3.21**	**349.60 ± 3.53**

environments may terminate early due to unsafe joint configurations, exposing the limitations of augmentation-based methods under longer delays. `Encoding` demonstrates more stable performance but quite consistently underperforms `DA-Dreamer` . While the baselines rely on the MDP assumption and are not burdened by integrating past information, `DA-Dreamer` consistently outperforms them.

5.1.2. Evaluation on Unseen Delay Distribution

In this section, we evaluate how well each method generalizes in deployment to delay distributions different from the one used during training. All methods are trained with delay distribution $\mathcal{U}\{0, 20\}$, and evaluated on three test distributions: $\mathcal{U}\{0, 10\}$ representing shorter delays, $\mathcal{U}\{10, 20\}$ for longer one, and $\mathcal{N}^+(10, 1)$ a narrow Gaussian around the training mean. Note that policy values may change significantly under a delay distribution shift, even if the training distribution supports the test one.

Figure 1 reports the normalized expected return for each method, averaged across the same 6 environments. Normalization follows $\nu(R) = \frac{R - R_{\min}}{R_{\max} - R_{\min}}$, where $R_{\min}$ and $R_{\max}$ are the expected returns of a random policy and Dreamer trained without delay, respectively. A well designed method for random delays should generally perform better under shorter delays. Otherwise, one could simply pad the delay to be longer.

Both `DCAC` and `Stack-Dreamer` show little improvement with shorter delays. `DCAC` underperforms even within its training distribution, and its insensitivity to test-time shifts likely reflects more the weak overall performance rather than robustness. Its fixed-size state augmentation also prevents its applicability to delays longer than those seen during training. `Stack-Dreamer` degrades sharply under longer delays, revealing the limits of treating observations as generic inputs without modeling temporal structure. In contrast, `DA-Dreamer` generalizes well across all delay distributions, achieving much higher performance under shorter delays while remaining stable under longer ones. This is a desirable property, as delay distributions are often unknown or non-stationary in real-world settings, and methods trained on a distribution with wide support should ideally remain reliable at deployment. `Encoding` exhibits a similar pattern to a lesser extent.

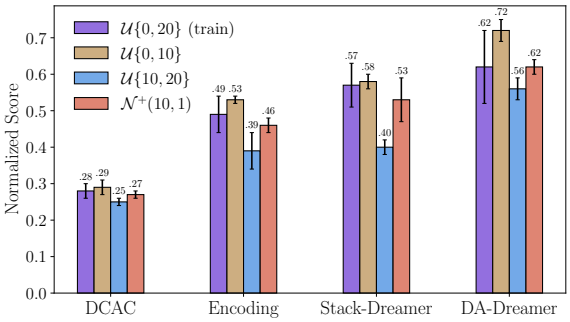


Figure 1: Normalized expected return under different test-time delay distributions. Bars and caps represent the mean and 95% confidence interval.

5.2. Importance of Addressing Delays

In Meta-World, because existing baselines are not applicable, we used two alternatives commonly employed in practice: (1) the `Memoryless` agent uses the latest available observation in place of the current one; and (2) the `Wait` method pauses to receive a new observation, where pausing is implemented by issuing no-op (zero) actions during waiting steps. While `Wait` is not always feasible in practice, we include it for comparison purposes. To evaluate whether methods can complete the task *in-time* while making delays more impactful, we reduce the default episode length in Meta-World environments to 50 steps, yet sufficient for the tasks considered.

5.2.1. RESULTS

Table 2 reports the final success rate, defined as the average proportion of successful episodes. The `Wait` agent is only effective in the simplest task under short delays; as it fails entirely in most tasks, we exclude it from evaluations under longer delays. `Memoryless` performs well under short delays but degrades quickly as delays increase, failing on more complex tasks. In contrast, `DA-Dreamer` consistently outperforms both baselines and maintains a high success rate even under long delays relative to the episode length. In stochastic environments, both `Memoryless` and `DA-Dreamer` experience performance drops on harder tasks, which is expected given the increased task complexity when partial observations are delayed.

Additionally, we include experimental details and further results in the appendix, including stochastic environments, ablations on `Stack-Dreamer` and `DA-Dreamer` , a study on the effect of number of the particles, visualizations of reconstructed observations, and training curves.

6. Related Work

Research on delayed RL began with foundational works establishing unified frameworks for MDPs with observation and action delays (Altman and Nain, 1992; Katsikopoulos and Engelbrecht, 2003). Early methods like dSARSA (Schuitema et al., 2010) used memoryless policies based on the last observation, but performance deteriorates quickly with increasing delays. Augmentation-based methods extend states using the last observation and subsequent actions; for instance, DCAC resamples trajectory fragments in hindsight (Bouteiller et al., 2020; Haarnoja et al., 2018), though such methods

Table 2: Success rate of methods for different delay distributions in Meta-World environments. Results represent the mean and 95% confidence interval.

Delay	Environment	Wait	Memoryless	DA-Dreamer
$\mathcal{U}\{0,5\}$	button-press-wall-v2	0.0 ± 0.0	**0.99 ± 0.01**	**0.90 ± 0.09**
	drawer-close-v2	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	plate-slide-v2	0.0 ± 0.0	0.69 ± 0.22	**0.97 ± 0.05**
	reach-v2	0.05 ± 0.01	**1.00 ± 0.00**	**1.00 ± 0.00**
$\mathcal{N}^+(10,3)$	button-press-wall-v2 ($\alpha = 0.25$)	-	0.47 ± 0.15	0.49 ± 0.26
	drawer-close-v2 ($\alpha = 0.25$)	0.14 ± 0.04	**0.97 ± 0.03**	**1.00 ± 0.00**
	plate-slide-v2 ($\alpha = 0.25$)	-	0.39 ± 0.13	0.47 ± 0.16
	reach-v2 ($\alpha = 0.25$)	0.0 ± 0.0	0.83 ± 0.12	**1.00 ± 0.00**
$\mathcal{N}^+(20,1)$	button-press-wall-v2	-	0.12 ± 0.14	**0.91 ± 0.13**
	drawer-close-v2	-	1.00 ± 0.01	1.00 ± 0.00
	plate-slide-v2	-	0.05 ± 0.06	**0.45 ± 0.33**
	reach-v2	-	0.25 ± 0.13	**1.00 ± 0.00**

suffer from the curse of dimensionality. Model-based approaches infer the current state from extended states (Walsh et al., 2007; Derman et al., 2021) or learn compact belief representations (Liotet et al., 2021). Recent works explore design heuristics in deep RL (Wang et al., 2023) or use world models to predict the state (Karamzade et al., 2024; Valensi et al., 2024). Other efforts apply imitation learning from undelayed experts (Liotet et al., 2022) or reformulate delayed RL as variational inference solved via behavior cloning (Wu et al., 2024a), though these face policy mismatch. Auxiliary-task methods (Wu et al., 2024b) introduce shorter delays to support training under long ones. BPQL (Kim et al., 2023) avoids full augmentation by projecting critic evaluations onto the original state space but struggles under high stochasticity. Most prior methods assume constant delays. Random delays remain underexplored, with few works (Bouteiller et al., 2020; Wang et al., 2023; Valensi et al., 2024) tackling them—only in fully observable MDPs without OOS observations. In the partially observable setting, Kim and Jeong (1987) studied lagged observations without proposing a learning method, while Karamzade et al. (2024) addressed constant delays without OOS inputs. Our work fills this gap by targeting stochastic observation delays in POMDPs.

7. Conclusion

We addressed random observation delays in POMDPs by proposing a model-based framework that effectively processes OOS observations for RL. Unlike prior methods, our approach does not assume full observability or fixed delays, making it applicable to more realistic scenarios. Experiments on synthetic and simulated robotic environments show that our method outperforms baselines even in MDP settings, remains effective under partial observability, and generalizes well to unseen delay distributions, an essential feature for real-world deployment.

A key limitation is the reliance on recursive filtering, which can accumulate one-step prediction errors and hinder scalability. Future work could explore more scalable architectures, such as Transformer-based models. Also, the filtering procedure incurs additional inference overhead that scales linearly with the maximum delay, increasing memory and update costs for large delays.

Acknowledgments

Authors AK and KK were supported by Hasso Plattner Foundation Fellowship. Author DC was supported by DARPA ARC Award HR0011-24-3-0148. This work was supported by BSF Grant 2024079 and NSF Award 2321786.

References

Ignacio Abadía, Francisco Naveros, Eduardo Ros, Richard R Carrillo, and Niceto R Luque. A cerebellar-based solution to the nondeterministic time delay problem in robotic control. *Science Robotics*, 6(58):eabf2756, 2021.

Eitan Altman and Philippe Nain. Closed-loop control with delayed information. *ACM sigmetrics performance evaluation review*, 20(1):193–204, 1992.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International conference on learning representations*, 2020.

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. *arXiv preprint arXiv:2101.11992*, 2021.

Leonardo Alves Fagundes-Junior, Andre Fialho Coelho, Daniel Khede Dourado Villa, Mario Sarcinelli-Filho, and Alexandre Santos Brandão. Communication delay in uav missions: A controller gain analysis to improve flight stability. *IEEE Latin America Transactions*, 21(1):7–15, 2023.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

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.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pages 1–7, 2025.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Armin Karamzade, Kyungmin Kim, Montek Kalsi, and Roy Fox. Reinforcement learning from delayed observations via world models. *arXiv preprint arXiv:2403.12309*, 2024.

Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE transactions on automatic control*, 48(4):568–574, 2003.

Jangwon Kim, Hangyeol Kim, Jiwook Kang, Jongchan Baek, and Soohee Han. Belief projection-based reinforcement learning for environments with delayed feedback. *Advances in Neural Information Processing Systems*, 36:678–696, 2023.

Soung Hie Kim and Byung Ho Jeong. A partially observable markov decision process with lagged information. *Journal of the Operational Research Society*, 38(5):439–446, 1987.

Pierre Liotet, Erick Venneri, and Marcello Restelli. Learning a belief representation for delayed reinforcement learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

Pierre Liotet, Davide Maran, Lorenzo Bisi, and Marcello Restelli. Delayed reinforcement learning by imitation. In *International conference on machine learning*, pages 13528–13556. PMLR, 2022.

Xiao Ma, Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter recurrent neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5101–5108, 2020a.

Xiao Ma, Peter Karkus, David Hsu, Wee Sun Lee, and Nan Ye. Discriminative particle filter reinforcement learning for complex partial observations. *arXiv preprint arXiv:2002.09884*, 2020b.

A Rupam Mahmood, Dmytro Korenkevych, Brent J Komer, and James Bergstra. Setting up a reinforcement learning task with a real-world robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4635–4640. IEEE, 2018.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3226–3231. IEEE, 2010.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

David Valensi, Esther Derman, Shie Mannor, and Gal Dalal. Tree search-based policy optimization under stochastic execution delay. *arXiv preprint arXiv:2404.05440*, 2024.

Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Planning and learning in environments with delayed feedback. In *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18*, pages 442–453. Springer, 2007.

Wei Wang, Dongqi Han, Xufang Luo, and Dongsheng Li. Addressing signal delay in deep reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2023.

Qingyuan Wu, Simon S Zhan, Yixuan Wang, Yuhui Wang, Chung-Wei Lin, Chen Lv, Qi Zhu, and Chao Huang. Variational delayed policy optimization. *Advances in Neural Information Processing Systems*, 37:54330–54356, 2024a.

Qingyuan Wu, Simon Sinong Zhan, Yixuan Wang, Yuhui Wang, Chung-Wei Lin, Chen Lv, Qi Zhu, Jürgen Schmidhuber, and Chao Huang. Boosting reinforcement learning with strongly delayed feedback through auxiliary short delays. *arXiv preprint arXiv:2402.03141*, 2024b.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

Weipu Zhang, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang. Storm: Efficient stochastic transformer based world models for reinforcement learning. *Advances in Neural Information Processing Systems*, 36:27147–27166, 2023.

Appendix A. Experimental Details

All methods and baselines were trained for 10^6 environment steps. We used an action repeat of 2 for the Meta-World environments, resulting in 5×10^5 decision steps during training. For the baselines, we used the exact same hyperparameters as reported in their papers. For Dreamer-v3, we disabled the replay value loss, which prevents training the critic on data stored in the replay buffer; thus, the critic is only trained on generated trajectories during the policy learning phase. Additionally, we increased the number of classes in the discrete latent state representation to 32 and the number of neurons per layer to 512 for the Gym MuJoCo tasks only. In Meta-World, we used dense reward signals and (64×64) RGB images from `camera_id=1` as observations.

Appendix B. Additional Experiments

B.1. Stochastic environments

While MuJoCo environments are deterministic apart from the initial state, we introduce added Gaussian noise with variance α into the normalized action space to evaluate the robustness of each method under stochastic environments. Table 3 reports the performance of all methods under delay distribution $\mathcal{U}\{0, 10\}$ in the HalfCheetah environment across different noise levels. `DA-Dreamer` and `Encoding` maintain competitive performance as noise increases, demonstrating greater robustness. In high-noise settings, `DA-Dreamer` achieves the best performance. `DCAC` shows high performance in the noise-free setting for this environment (Table 1), but its performance degrades sharply under high stochasticity. Similarly, `Stack-Dreamer` struggles even under mildly stochastic conditions. In such regimes, the agent must estimate a belief over latent states to act reliably. By explicitly computing this belief, `DA-Dreamer` effectively accounts for uncertainty, which is reflected in its strong performance.

Table 3: Exptected return under different levels of stochasticity of the environment(α) in HalfCheetah-v4. Results represent the mean and 95% confidence interval.

Delay	Environment	`DCAC`	`Encoding`	`Stack-Dreamer`	`DA-Dreamer`
$\mathcal{U}\{0, 10\}$	HalfCheetah-v4 ($\alpha = 0.2$)	2061.48 ± 726.16	**3538.54 ± 244.46**	1608.72 ± 385.88	**3399.84 ± 149.52**
	HalfCheetah-v4 ($\alpha = 0.4$)	1735.57 ± 110.66	**2598.16 ± 130.39**	1252.71 ± 406.18	2347.83 ± 58.20
	HalfCheetah-v4 ($\alpha = 0.6$)	1177.43 ± 394.54	**1768.95 ± 186.91**	720.30 ± 247.73	**1707.84 ± 48.65**
	HalfCheetah-v4 ($\alpha = 0.8$)	623.95 ± 303.60	855.34 ± 45.21	336.37 ± 146.56	**1093.30 ± 80.53**
	HalfCheetah-v4 ($\alpha = 1$)	147.47 ± 218.97	383.75 ± 110.70	65.02 ± 24.13	**587.99 ± 60.00**

B.2. Delay Aware Inference without Training

Table 4 shows the final expected return of `DA-Dreamer` when trained in a standard (non-delayed) environment but deployed in a delayed environment. Compared to training directly in the delayed setting, it underperforms in half of the environments, shows similar performance in two, and achieves higher scores in HumanoidStandup. We speculate, based on Figure 6, that the original method has not yet converged in HumanoidStandup, and with additional training, the performance gap may close. Overall, this approach appears promising for scenarios where the delay distribution is unknown during training, or when the agent must be trained once and deployed under varying, unknown delays.

Table 4: Expected return of the `DA-Dreamer` ablation on Gym environments. An asterisk (*) indicates similar performance. Results represent the mean and 95% confidence interval.

Delay	Environment	`DA-Dreamer` (inference only)
$\mathcal{U}\{0, 10\}$	HalfCheetah-v4	2642.02 ± 644.15
	Hopper-v4	1692.25 ± 1603.35
	Humanoid-v4	1808.95 ± 631.46
	HumanoidStandup-v4	**267659.35 ± 52512.41**
	Reacher-v4	−6.13 ± 0.23*
	Swimmer-v4	349.99 ± 2.53*
$\mathcal{U}\{0, 20\}$	HalfCheetah-v4	1384.41 ± 233.52
	Hopper-v4	1430.39 ± 1695.49
	Humanoid-v4	722.37 ± 264.09
	HumanoidStandup-v4	**207792.76 ± 25154.57**
	Reacher-v4	−7.08 ± 0.28*
	Swimmer-v4	350.35 ± 3.43*

B.3. Stacking Observations

In Figure 2, we experiment with different ways of stacking delayed information in `Stack-Dreamer`. The default version, reported in the main text, inputs the previous D observations and actions, with missing observations filled with zeros. We also evaluate a variant that removes actions from the input, as they are already represented in the latent state from the previous time step, and another variant that additionally includes a mask indicating which observations have been received. As shown, all variants perform similarly, with no significant differences observed between them.

B.4. Number of Particles

Figure 3 shows the performance of `DA-Dreamer` with different numbers of particles K. In our implementation, we simply concatenate the K particles, though various alternatives exist for combining them (Ma et al., 2020a). As shown, increasing the number of particles generally improves performance, but the gains are not substantial. We hypothesize that this is because the Dreamer-v3 world model includes both deterministic and stochastic components, with the deterministic part capable of retaining information along a trajectory. To balance performance with computational cost, we used $K = 1$ in the main experiments.

B.5. Visualizing Reconstructed Frames

Figure 4 shows reconstructed observations from delayed information in `DA-Dreamer`. As seen, for shorter delays (a), the reconstructed frames are sharper compared to longer delays (b). This is expected, as longer delays increase uncertainty in the belief state about the current environment state. Consequently, the belief assigns more weight to nearby states, resulting in blurrier reconstructed frames, as depicted.

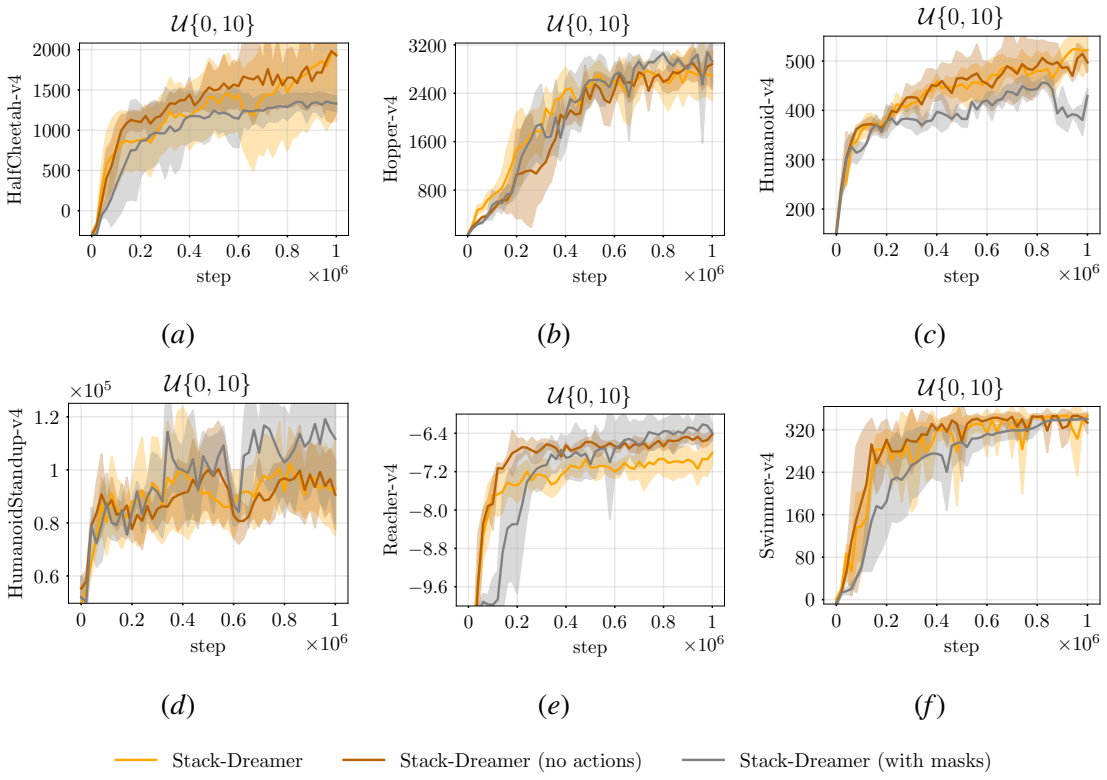


Figure 2: Learning curve of `Stack-Dreamer` variants.

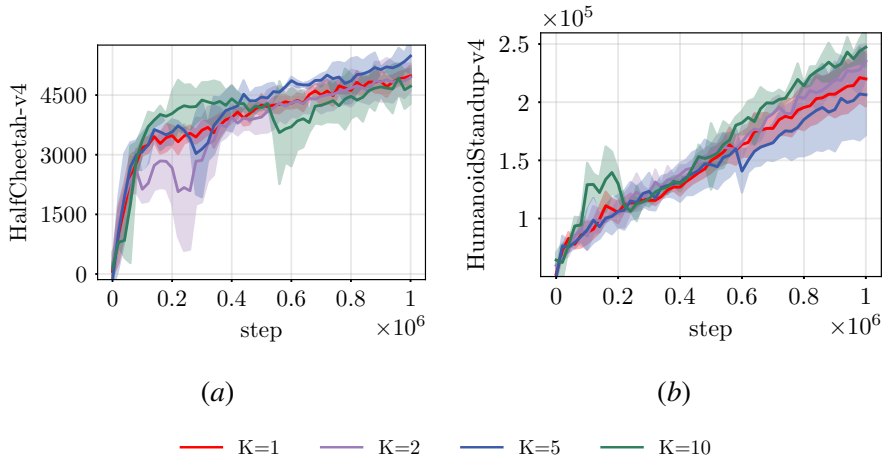
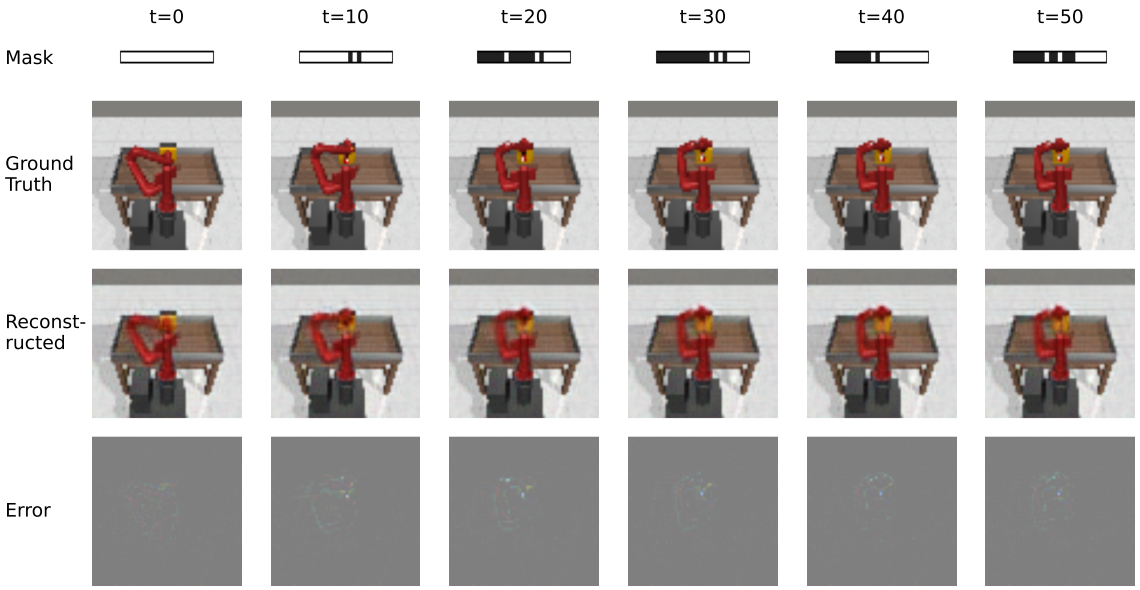
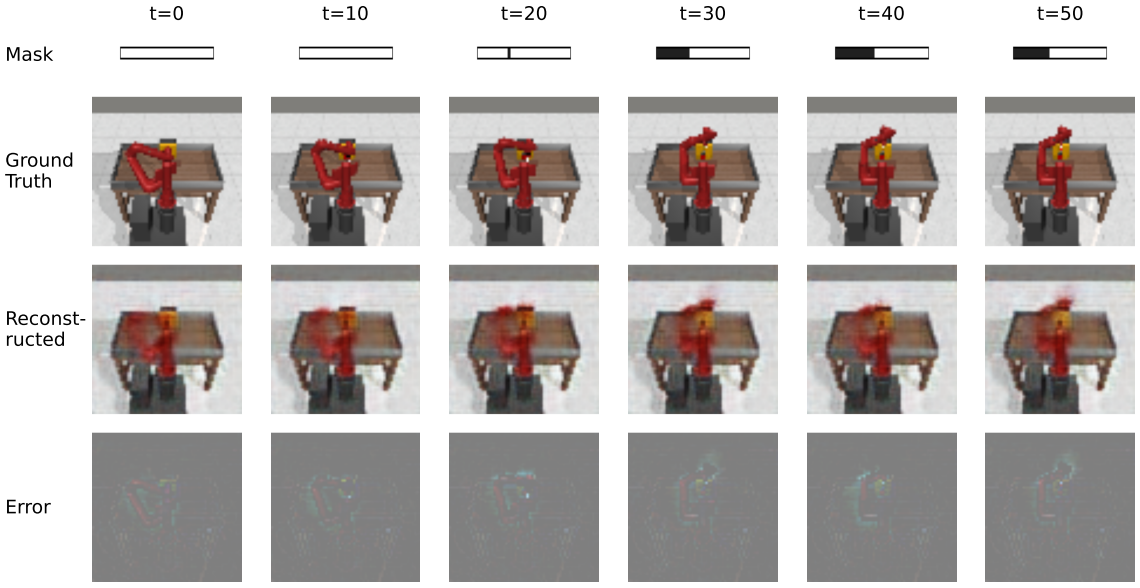


Figure 3: Performance vs different number of particles.



(*a*) $\mathcal{N}^+(10, 3)$ and $\alpha = 0.25$



(*b*) $\mathcal{N}^+(20, 1)$

Figure 4: Reconstructing the current observation in delayed button-press-v2 from the computed belief state. The mask indicates the arrival of past observations (black cells denote received observations), with the rightmost cell representing the current timestep.

Appendix C. Training Curves

Below, we include the training plots for the experiments presented in Table 1 and Table 2, respectively. Shaded regions represent 95% confidence interval.

C.1. Gym MuJoCo

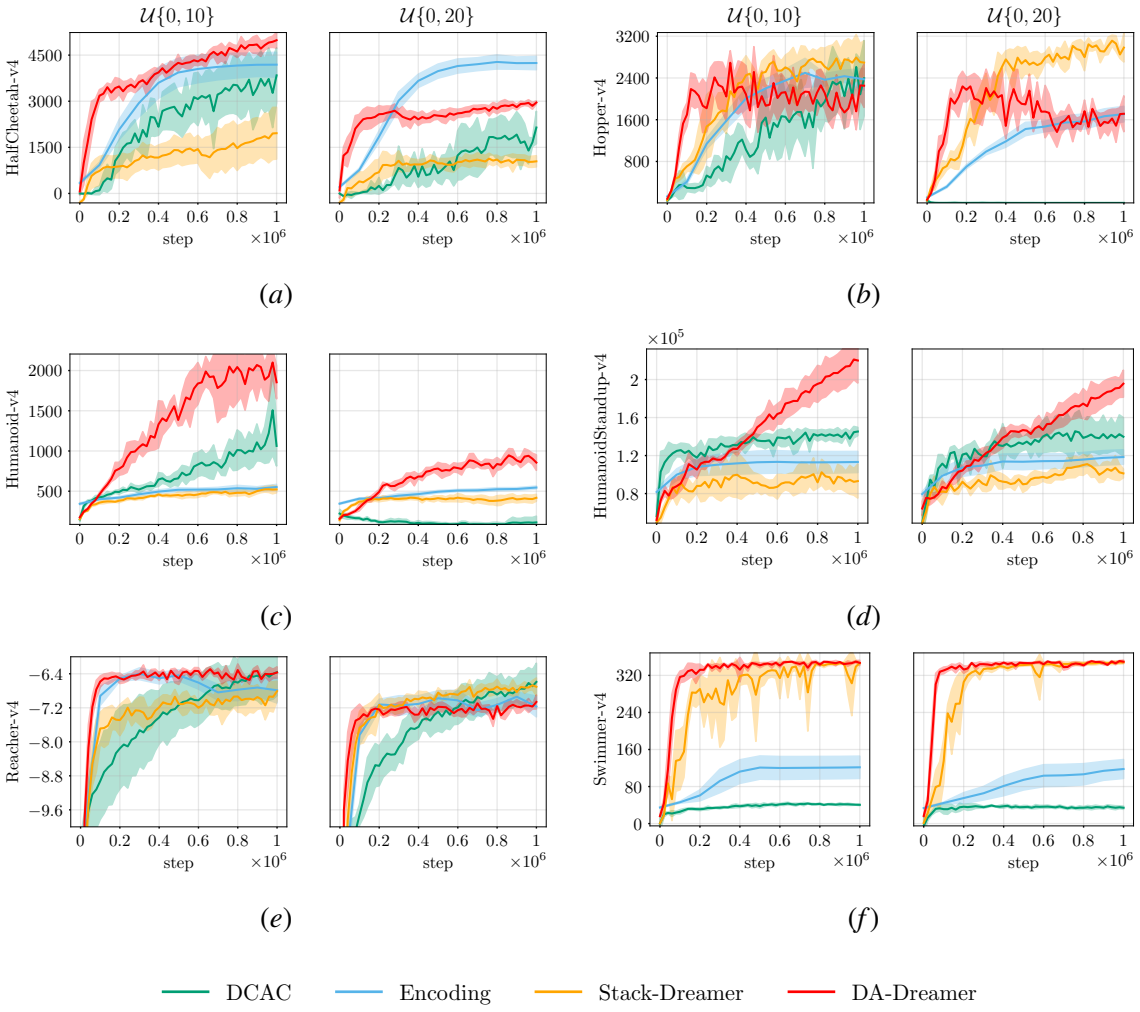


Figure 5: Learning curve of the methods on selected Gym environments.

C.2. Meta-World

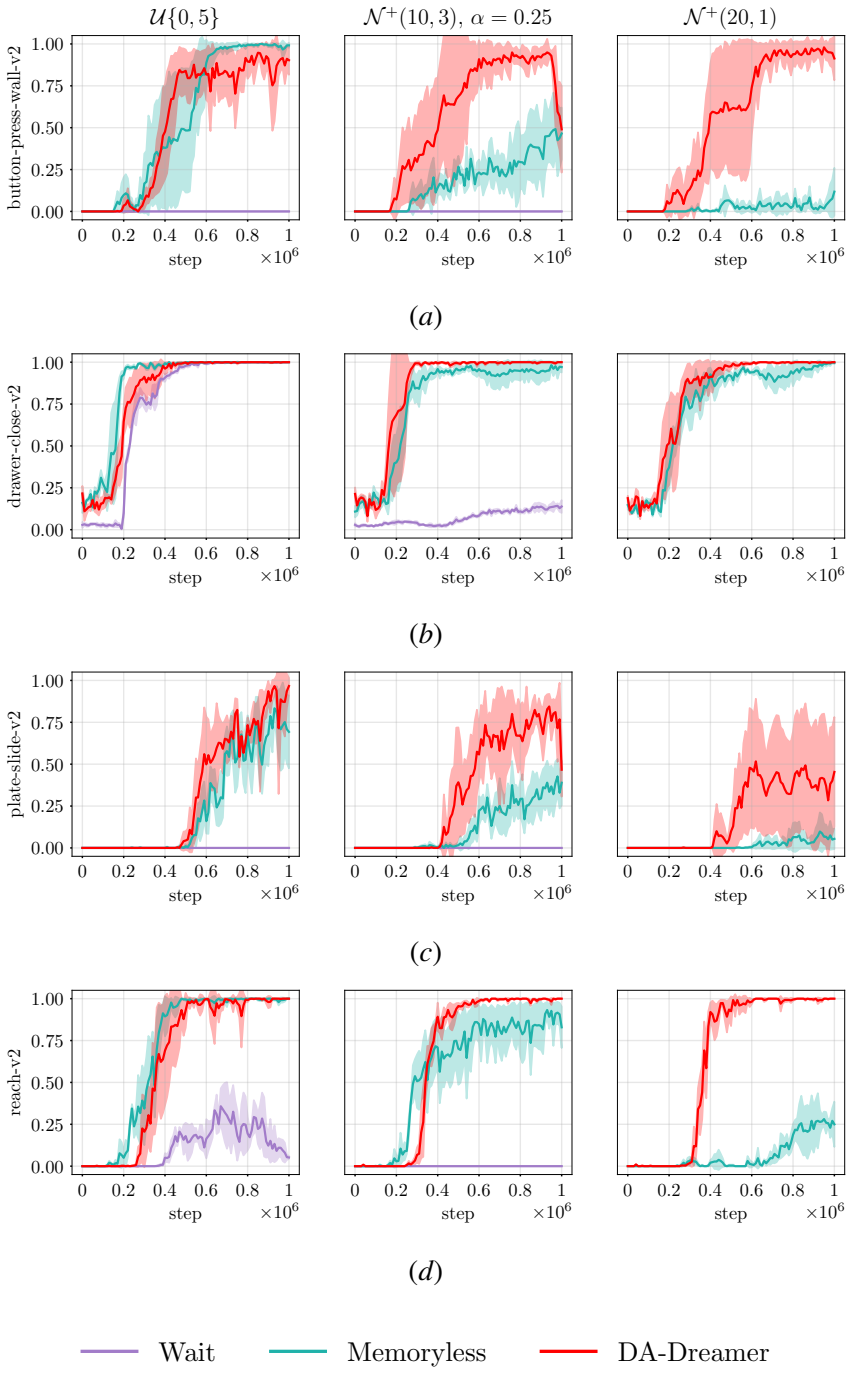


Figure 6: Learning curve of the methods on selected Meta-World environments.