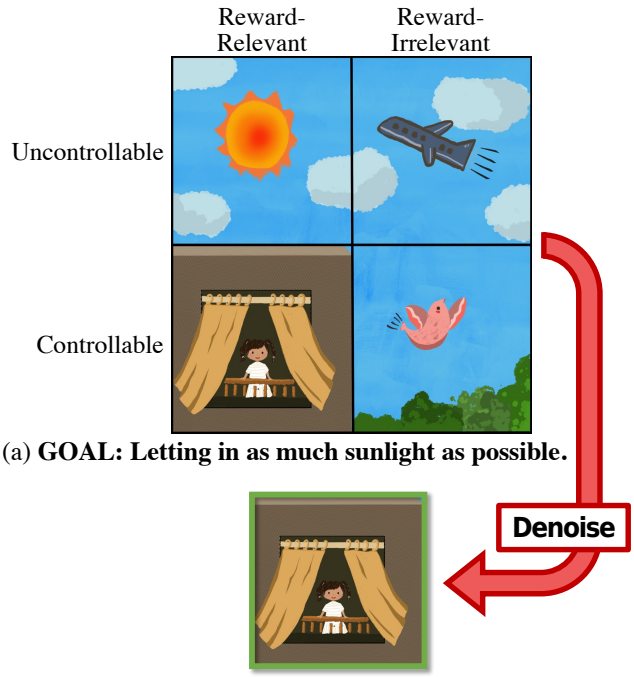
Denoised MDPs: Learning World Models Better Than the World Itself

Tongzhou Wang [1] **Simon S. Du** [2] **Antonio Torralba** [1] **Phillip Isola** [1] **Amy Zhang** [3 4] **Yuandong Tian** [4]

Abstract

The ability to separate signal from noise, and reason with clean abstractions, is critical to intelligence. With this ability, humans can efficiently perform real world tasks without considering all possible nuisance factors. How can artificial agents do the same? What kind of information can agents safely discard as noises? In this work, we categorize information out in the wild into four types based on controllability and relation with reward, and formulate useful information as that which is both *controllable* and *reward-relevant*. This framework clarifies the kinds information removed by various prior work on representation learning in reinforcement learning (RL), and leads to our proposed approach of learning a *Denoised MDP* that explicitly factors out certain noise distractors. Extensive experiments on variants of DeepMind Control Suite and `RoboDesk` demonstrate superior performance of our denoised world model over using raw observations alone, and over prior works, across policy optimization control tasks as well as the non-control task of joint position regression.

Project Page: `ssnl.github.io/denoised_mdp`
Code: `github.com/facebookresearch/denoised_mdp`



(a) **GOAL: Letting in as much sunlight as possible.**

(b) Optimal control only relies on information that is **both controllable and reward-relevant**. Good world models should ignore other factors as noisy distractors.

Figure 1: **Illustrative example: (a)** Four distinct kinds of information in the scenario described in Section 1, where the person desires to increase the amount of sunlight let into the room. Their opening of the curtain scares away the bird. **(b)** A denoised world model only includes a small subset of all information.

1. Introduction

The real world provides us a plethora of information, from microscopic physical interactions to abstracted semantic signals such as the latest COVID-19 news. Fortunately, processing each and every signal is unnecessary (and also impossible). In fact, any particular reasoning or decision often only relies on a small portion of information.

Imagine waking up and wanting to embrace some sunlight. As you open the curtain, a nearby resting bird is scared away and you are pleasantly met with a beautiful sunny day. Far away, a jet plane is slowly flying across the sky.

This may seem a simple activity, but in fact highlights four distinct types of information (see Figure 1), with respect to the goal of letting in as much sunlight as possible:

- **Controllable and reward-relevant**: curtain, influenced by actions and affecting incoming sunlight;
- **Controllable and reward-irrelevant**: bird, influenced by actions but not affecting sunlight;
- **Uncontrollable and reward-relevant**: weather, independent with actions but affecting sunlight;
- **Uncontrollable and reward-irrelevant**: plane, independent with both actions and the sunlight.

Our optimal actions towards the goal, however, only in fact depend on information that is **controllable and reward-**

[1]MIT CSAIL [2]University of Washington [3]UC Berkeley [4]Meta AI. Correspondence to: Tongzhou Wang <tongzhou@mit.edu>. Work done while Tongzhou Wang was an intern at Meta AI.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

relevant, and the three other kinds of information are merely *noise distractors*. Indeed, no matter how much natural sunlight there is outside, or how the plane and the bird move, the best plan is always to open up the curtain.

When performing a particular task, we humans barely think about the other three types of information, and usually only plan on how our actions affect information that is **controllable and reward-relevant**. Our mental model is an abstract and condensed version of the real world that is actually *better* suited for the task.

The notion of better model/data is ubiquitous in data science and machine learning. Algorithms rarely perform well on raw noisy real data. The common approach is to perform data cleaning and feature engineering, where we manually select the useful signals based on prior knowledge and/or heuristics. Years of research have identified ways to extract good features for computer vision (Lowe, 1999; Donahue et al., 2014), natural language processing (Elman, 1990; Mikolov et al., 2013), reinforcement learning (RL) (Mahadevan & Maggioni, 2007; Bellemare et al., 2019), etc. Similarly, system identification aligns real observation with a predefined set of abstract signals/states. Yet for tasks in the wild (in the general form of (partially observable) Markov Decision Processes), there can be very little prior knowledge of the optimal set of signals. In this work, we ask: can we infer and extract these signals automatically, in the form of a learned world model?

The general idea of a mental world model have long been under active research in philosophy and social science (Craik, 1952; Dennett, 1975), cognitive science, where an intuitive physics model is hypothesized to be core in our planning capabilities (Spelke & Kinzler, 2007), and in reinforcement learning, where various methods investigate state abstractions for faster and better learning (Sutton, 1991; 1981).

In this work, we explore this idea within the context of machine learning and reinforcement learning, where we aim to make concrete the different types of information in the wild, and automatically learn a world model that removes noise distractors and is beneficial for both control (i.e., policy optimization) and non-control tasks. Toward this goal, our contributions are

- We categorize information into four distinct kinds as in Figure 1, and review prior approaches under this framework (Section 2).
- Based on the above framework, we propose Denoised MDPs, a method for learning world models with certain distractors removed (Section 3).
- Through experiments in DeepMind Control Suite and `RoboDesk` environments, we demonstrate superior performance of policies learned our method, across many distinct types of noise distractors (Sections 5.1 and 5.2).
- We show that Denoised MDP is also beneficial beyond control objectives, improving the supervised task of robot joint position regression (Section 5.1).

2. Different Types of Information in the Wild

In Section 1, we illustrated the four types of information available in the wild w.r.t. a task. Here we make these notions more concrete, and relate them to existing works.

For generality, we consider tasks in the form of Markov Decision Processes (MDPs), described in the usual manner: $\mathcal{M} \triangleq (\mathcal{S}, \mathcal{A}, R, P, p_{s_0})$ (Puterman, 1994), where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $R\colon \mathcal{S} \to \Delta([0, r_{\max}])$ defines the reward random variable $R(s')$ received for arriving at state $s' \in \mathcal{S}$, $P\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition dynamics, and $p_{s_0} \in \Delta(\mathcal{S})$ defines the distribution of initial state. We use $\Delta(A)$ to denote the set of all distributions over A. P and R define the most important components of a MDP: the transition dynamics $\mathbb{P}[s' \mid s, a]$ and the reward function $\mathbb{P}[r \mid s']$. Usually, the objective is to find a policy $\pi\colon \mathcal{S} \to \Delta(\mathcal{A})$ acting based on current state, that maximizes the expected cumulative (discounted) reward.

Indeed, MDPs provide a general formulation that encompasses many tasks. In fact, the entire real world may be viewed as an MDP with a rich state/observation space $\mathcal{S}$ that contains all possible information/signal. For an artificial agent to successfully perform real world tasks, it must be able to process observations that are incredibly rich and high-dimensional, such as visual or audio signals.

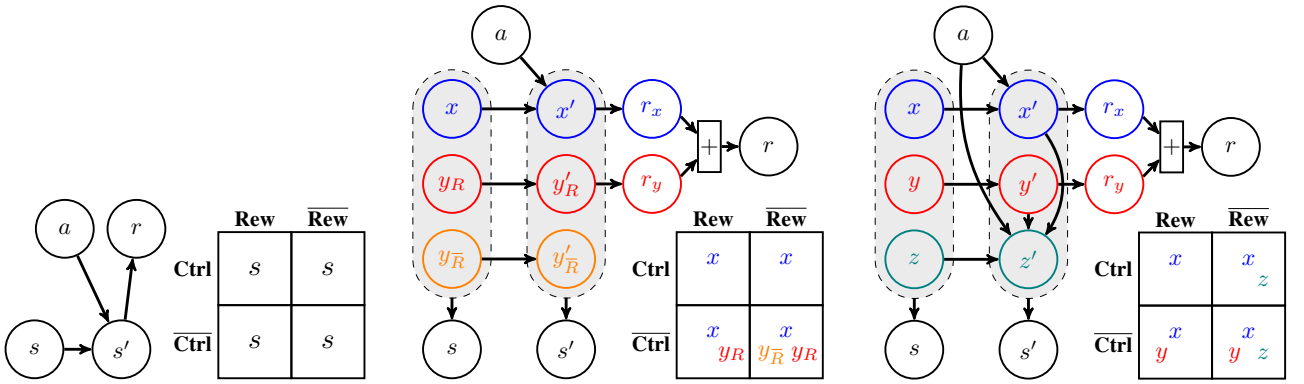
We characterize different types of information in such observations by considering two intuitive notions of "noisy and irrelevant" signals: (1) uncontrollable information and (2) reward-irrelevant information. Such factors can often be ignored without affecting optimal control, and are referred to as *noise distractors*.

To understand their roles in MDPs, we study different formulations of the transition dynamics and reward functions, and show how different structures naturally leads to decompositions that may help identify such distractors. Removing these distractors can thus *transform the original noisy MDP to a clean denoised one*, to be used in downstream tasks.

For starters, the most generic transition model in Figure 2a has little to no structure. The state s can contain both the useful signals and noise distractors. Therefore, it is not directly useful for extracting important information.

2.1. Controllability

Intuitively, if something is not controllable, an agent might be able to do well without considering it. Yet it is not enough to only require some variable to be unaffected by actions (e.g., wind directions should not be ignored while sailing).



(a) Transition without useful structure. s may contain any type of information.

(b) Transition that factorizes out uncontrollable information in $y_{\overline{R}}$ and y_R.

(c) Transition that factorizes out uncontrollable y and reward-irrelevant z.

Figure 2: MDP transition structures consisting of dynamics and reward functions. Unlike the regular structure of **(a)**, **(b, c)** factorized (yet still general) structures inherently separate information into controllable (**Ctrl**) versus uncontrollable (**$\overline{\text{Ctrl}}$**), and reward-relevant (**Rew**) versus reward-irrelevant (**$\overline{\text{Rew}}$**). Presence of a variable in a cell means *possible* containing of respective information. E.g., in **(c)**, z can only contain reward-irrelevant information. In **(b, c)**, the x dynamics form an MDP with less noise and sufficient for optimal planning. Our Denoised MDP (see Section 3) is based on these two factorizations.

Instead, we focus on factors that simply evolve on their own, without influencing or being influenced by others.

Not all such information can be safely ignored, as they still may affect reward (e.g., traffic lights when driving). Fortunately, in the usual objective of maximizing expected return, we can ignore ones that only additively affect reward.

Concretely, if an MDP transition can be represented in the form of Figure 2b, we say variables $y_{\overline{R}}$ and y_R are *uncontrollable* information, as they evolve independently of actions and do not affect *controllable* x. Here y_R (additively) affects reward, but can be ignored. One can safely discard both $y_{\overline{R}}$ and y_R as *noise distractors*. Operating with the compressed MDP of only x is sufficient for optimal control.

2.2. Reward-Relevance

Among controllable information, there can still be some that is completely unrelated to reward. In Figure 1, the bird is affected by the opening curtain, but is irrelevant to the task of letting in sunlight. In such cases, the information can be safely discarded, as it does not affect the objective.

If an MDP transition can be represented in the form of Figure 2c, we say z is *reward-irrelevant* because it evolves by potentially using everything (i.e., all latent variables and actions), but crucially *does not affect anything but itself*.

Similar to uncontrollable information, z (and y) is a *noise distractor* that can be discarded. The compressed MDP of only x contains all signals needed for optimal control.

2.3. Which Information Do Existing Methods Learn?

In RL, many prior work have explored state abstractions in some form. Here we cast several representative ones under the framework described above, and show which kinds of information they learn to remove, summarized in Figure 3, together with our proposed method (explained in Section 3). Below we discuss each prior work in detail.

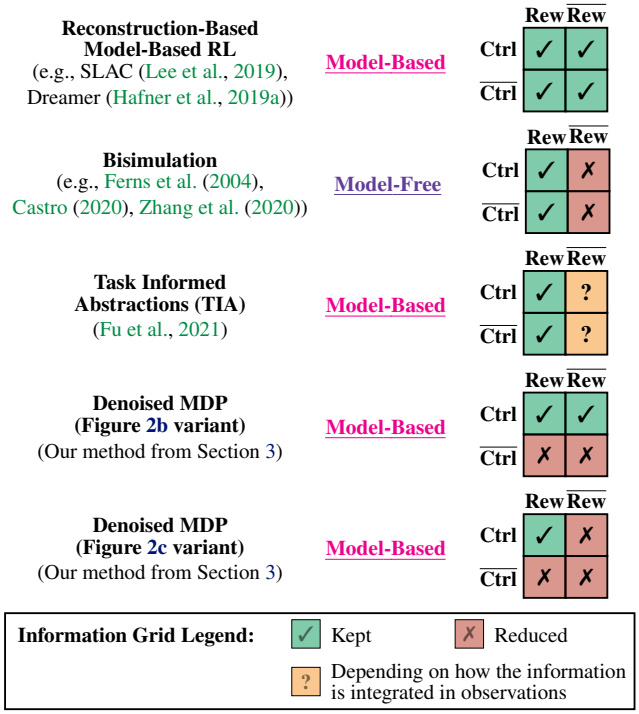


Figure 3: Categorization of information learned and removed by various methods with distinct formulations.

Reconstruction-Based Model-Based RL. Many model-based RL methods learn via reconstruction from a single latent code, often as a result of a variational formulation (Hafner et al., 2019a;b; Lee et al., 2019). The latent code

must try to compress all information present in the observation, and necessarily contains all types of information.

Bisimulation. Bisimulation defines a state abstraction where states aggregated together must have the same expected return and transition dynamics up to the abstraction (Givan et al., 2003), and is known to optimally ignore reward-irrelevant information (Ferns et al., 2004). While its continuous version, bisimilation metric, is gaining popularity, learning them is computationally difficult (Modi et al., 2020). Even with many additional assumptions, it is generally only possible to learn an on-policy variant that loses the above guarantee (Castro, 2020; Zhang et al., 2020).

Task Informed Abstractions (TIA). TIA (Fu et al., 2021) extends Dreamer by modelling two independent latent MDPs, representing signal and noise. The noise latent is enforced to be independent with reward and reconstruct the observation as well as possible. Reconstructions from each latent are composed together using an inferred mask in pixel-space, to form the full reconstruction for the reconstruction loss. Because of its special structure, TIA can remove *reward-irrelevant* noise distractors that are present via pixel-wise composing two images from *independent* processes (e.g., agent moving on a noisy background), but not general ones (e.g., a shaky camera affecting both the agent and the noisy background).

Predictive Information, Data Augmentation, etc. Another set of researches learn state representation that only contains information useful for predicting future states (e.g., CPC (Oord et al., 2018) and PI-SAC (Lee et al., 2020)) or augmented views of the current state (e.g., CURL (Laskin et al., 2020b)). These methods *do not guarantee* removal of any of the three redundant piece of information identified above. Non-i.i.d. noises (e.g., people moving in background) are predictive of future and may be kept by CPC and PI-SAC. The performance of augmentation-based methods can critically rely on specific types of augmentation used and relevance to the tasks. As we show in experiments (see Section 5), indeed they struggle to handle certain noise types.

2.4. Possible Extensions to Further Factorizations

The above framework is sufficient for characterizing most prior work and related tasks, and can also be readily extended with further factorized transition structures. E.g., if an independent process confounds a signal process and a noise process, fitting the Figure 2c structure must group all three processes into x (to properly model the dependencies). However, a further factorization shows that only considering the signal and the confounding processes is theoretically sufficient for control. We leave such extensions as future work.

3. Denoised MDPs

Figures 2b and 2c show two special MDP structures that automatically identify certain information that can be ignored, leaving x as the useful information (which also forms an MDP). This suggests a naïve approach: directly fitting such structures to collected trajectories, and then extract x.

However, the same MDP dynamics and rewards can be decomposed as Figures 2b and 2c in many different ways. In the extreme case, x may even contain all information in the raw state s, and such extraction may not help at all. Instead, we desire a fit with the *minimal* x, defined as being least informative of s (so that removal of the other latent variables discards the most information possible). Concretely, we aim for a fit with least $I(\{x_t\}_{t=1}^T; \{s_t\}_{t=1}^T \mid \{a_t\}_{t=1}^T)$, the mutual information x contains about s over T steps. Then from this fit, we can extract a minimal *Denoised MDP* of only x. For notation simplicity, we use bold symbols to denote variable sequences, and thus write, e.g., $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$.

Practically, we consider regularizing model-fitting with $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$. As we show below, this amounts to a modification to the well-established variational objective (Hafner et al., 2019a). The resulting method is easy-to-implement yet effective, enabling clean removal of various noise distractors the original formulation cannot handle (see Section 5).

We instantiate this idea with the structure in Figure 2c. The Figure 2b formulation can be obtained by simply removing the z components and viewing y as combined y_R and $y_{\overline{R}}$.

The transition structure is modeled with components:

$$
\begin{aligned}
p_\theta^{(x_t)} &\triangleq p_\theta(x_t \mid x_{t-1}, a) && (x \text{ dynamics}) \\
&\quad p_\theta(r_x \mid x_t) && (x \text{ reward}) \\
p_\theta^{(y_t)} &\triangleq p_\theta(y_{t-1} \mid y_{t-1}) && (y \text{ dynamics}) \\
&\quad p_\theta(r_y \mid y_t) && (y \text{ reward}) \\
p_\theta^{(z_t)} &\triangleq p_\theta(z_t \mid x_t, y_t, z_{t-1}, a) && (z \text{ dynamics}) \\
&\quad p_\theta(s_t \mid x_t, y_t, z_t). && (\text{obs. emission})
\end{aligned}
$$

Consider training data in the form of trajectory segments $\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}$ sampled from some data distribution p_{data} (e.g., stored agent experiences from a replay buffer). We perform model learning by minimizing the negative log likelihood:

$$
\mathcal{L}_{\mathsf{MLE}}(\theta) \triangleq -\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{r}\sim p_{\mathsf{data}}}\left[\log p_\theta\left(\boldsymbol{s}, \boldsymbol{r} \mid \boldsymbol{a}\right)\right].
$$

To obtain a tractable form, we jointly learn three variational posterior components (i.e., encoders):

$$
\begin{aligned}
q_\psi^{(x_t)} &\triangleq q_\psi(x_t \mid x_{t-1}, y_{t-1}, z_{t-1}, s_t, a_t) && (x \text{ posterior}) \\
q_\psi^{(y_t)} &\triangleq q_\psi(y_t \mid x_{t-1}, y_{t-1}, z_{t-1}, s_t, a_t) && (y \text{ posterior}) \\
q_\psi^{(z_t)} &\triangleq q_\psi(z_t \mid x_t, y_t, s_t, a_t), && (z \text{ posterior})
\end{aligned}
$$

whose product defines the posterior $q_\psi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{s}, \boldsymbol{a})$[1]. We choose this factorized form based on the forward (prior) model structure of Figure 2c.

Then, the model can be optimized w.r.t. the standard variational bound on log likelihood:

$$\begin{aligned}\mathcal{L}_{\mathsf{MLE}}(\theta) = \min_{\psi} \mathop{\mathbb{E}}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{r}} \mathop{\mathbb{E}}_{\substack{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\sim \\ q_\psi(\cdot\mid \boldsymbol{s},\boldsymbol{a},\boldsymbol{r})}} \Bigg[\underbrace{-\log p_\theta(\boldsymbol{s},\boldsymbol{r},\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\mid \boldsymbol{a})}_{\triangleq \mathcal{L}_{\mathsf{recon}}(\theta,\psi)} \\ + \underbrace{\sum_{t=1}^{T} D_{\mathsf{KL}}\big(q_\psi^{(x_t)} \,\big\|\, p_\theta^{(x_t)}\big)}_{\triangleq \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta,\psi)} + \underbrace{\sum_{t=1}^{T} D_{\mathsf{KL}}\big(q_\psi^{(y_t)} \,\big\|\, p_\theta^{(y_t)}\big)}_{\triangleq \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta,\psi)} \\ + \underbrace{\sum_{t=1}^{T} D_{\mathsf{KL}}\big(q_\psi^{(z_t)} \,\big\|\, p_\theta^{(z_t)}\big)}_{\triangleq \mathcal{L}_{\mathsf{KL}\text{-}z}(\theta,\psi)} \Bigg], \end{aligned} \tag{1}$$

where equality is attained by optimal q_ψ that is compatible with p_θ, i.e., q_ψ is the exact posterior of p_θ.

The mutual information regularizer $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$, using a variational formulation, can be written as

$$I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) = \min_{\theta} \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta, \psi), \tag{2}$$

with equality attained when q_ψ and p_θ are compatible. The appendix describes this derivation in detail.

Therefore, for a regularizer weight of $c \geq 0$, we can optimize Equations (1) and (2) together as

$$\begin{aligned} &\min_{\theta} \mathcal{L}_{\mathsf{MLE}}(\theta) + c \cdot I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) \\ &= \min_{\theta,\psi} \mathcal{L}_{\mathsf{recon}}(\theta,\psi) + (1+c) \cdot \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta,\psi) \\ &\qquad + \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta,\psi) + \mathcal{L}_{\mathsf{KL}\text{-}z}(\theta,\psi). \end{aligned} \tag{3}$$

Recall that we fit to the true MDP with the structure of Figure 2c, which inherently guarantees all useful information in the x latent variable. As the regularizer ensures learning the *minimal* x latents, the learned model extracts an MDP of condensed useful information with $\mathcal{X}$ as the *denoised* state space, $p_\theta(x' \mid x, a)$ as the transition dynamics, $p_\theta(r_x \mid x')$ as the reward function. This MDP is called the *Denoised MDP*, as it discards the noise distractors contained in y and z. Additionally, we also obtain $q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$ as the encoder mapping from raw noisy observation s to the denoised x.

A loss variant for improved stability. When using a large $c \geq 0$ (e.g. when the environment is expected to be very noisy), Equation (3) contains to a term with a large weight. Thus Equation (3) often requires learning rates to be tuned for different c. To avoid this, we use the following loss form that empirically has better training stability and does not require tuning learning rates w.r.t. other hyperparameters:

$$\min_{\theta,\psi} \mathcal{L}_{\mathsf{recon}} + \alpha \cdot (\mathcal{L}_{\mathsf{KL}\text{-}x} + \beta \mathcal{L}_{\mathsf{KL}\text{-}y} + \beta \mathcal{L}_{\mathsf{KL}\text{-}z}), \tag{4}$$

where θ, ψ in arguments are omitted, and the hyperparameters are $\alpha > 0$ and $0 < \beta \leq 1$. Here β is bounded, where $\beta = 1$ represents no regularization. α is also generally small and simply chosen according to the state-space dimensionality (see the appendix; $\alpha \in \{1, 2\}$ in our experiments). This form is justified from the observation that in practice we use isotropic Gaussians with fixed variance to parameterize the distributions of observation $p_\theta(s \mid \dots)$ and reward $p_\theta(r \mid \dots)$, where scaling log likelihoods is essentially changing the variance hyperparameter. Thus, Equation (4) is effectively a scaled Equation (3) with different variance hyperparameters.

Algorithm 1 Denoised MDP

Input: Model p_θ. Posterior encoder q_ψ. Policy $\pi \colon \mathcal{X} \to \Delta(\mathcal{A})$. Policy optimization algorithm PI-OPT.
Output: Denoised MDP of x in p_θ; Encoder q_ψ; Policy π.

while training **do**
  // Exploration
  Collect trajectories with π acting on q_ψ encoded outputs
  // Model learning
  Sample a batch of $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$ segments from reply buffer
  Train p_θ and q_ψ with Equation (4) on $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$
  // Policy optimization
  Sample $\boldsymbol{x} \sim q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$; Compute $\overline{\boldsymbol{r_x}} = \mathbb{E}\left[p_\theta(\boldsymbol{r_x} \mid \boldsymbol{x})\right]$
  Train π by running PI-OPT on $(\boldsymbol{x}, \boldsymbol{a}, \overline{\boldsymbol{r_x}})$
end while

Online algorithm with policy optimization. The model fitting objective of Equation (4) can be used in various settings, e.g., offline over a collected trajectory dataset. Without assuming existing data, we explore an online setting, where the training process iteratively performs (1) exploration, (2) model-fitting, and (3) policy optimization, as shown in Algorithm 1. The policy $\pi \colon \mathcal{X} \to \Delta(\mathcal{A})$ soley operates on the Denoised MDP of x, which has all information sufficient for control. For policy optimization, the learned posterior encoder $q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$ is used to extract $\boldsymbol{x}$ information from the raw trajectory $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r})$, obtaining transition sequences in $\mathcal{X}$ space. Paired with the $p_\theta(\boldsymbol{r_x} \mid \boldsymbol{x})$ rewards, we obtain $(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{r_x})$ as trajectories collected from the Denoised MDP on x. Any general-purpose MDP policy optimization algorithm may be employed on these data, such as Stochastic Actor-Critic (SAC) (Haarnoja et al., 2018). We can also utilize the learned *differentiable* Denoised MDP, e.g., optimizing policy by backpropagating through additional roll-outs from the model, as is done in Dreamer.

While presented in the fully observable setting, Denoised MDP readily handles partial observability without extra changes. In the appendix, we discuss this point in details, and provide a guideline for choosing hyperparameters α, β.

[1] Following Dreamer (Hafner et al., 2019a), we define posterior of first-step latents $q_\psi(x_1, y_1, z_1 \mid s_1) \triangleq q_\psi(\cdot, \cdot, \cdot \mid \mathbf{0}, \mathbf{0}, \mathbf{0}, s_1, \mathbf{0})$, where $\mathbf{0}$ is the all zeros vector of appropriate size.

4. Related Work

Model-Based Learning for Control jointly learns a world model and a policy. Such methods often enjoy good sample efficiency on RL tasks with rich observations. Some formulations rely on strong assumptions, e.g., deterministic transition in DeepMDP (Gelada et al., 2019) and bilinear transition in FLAMBE (Agarwal et al., 2020). Most general-setting methods learn with a reconstruction-based objective (Hafner et al., 2019b; Kim et al., 2020; Ha & Schmidhuber, 2018; Lee et al., 2019). Among them, Dreamer (Hafner et al., 2019a) trains world models with a variational formulation and optimizes policies by backpropagating through latent-space rollouts. It has proven effective across a variety of environments with image observations. However, such reconstruction-based approaches can struggle with the presence of noise distractors. TIA (Fu et al., 2021) partially addresses this limitation (see Section 2.3) but can not handle general distractors, unlike our method.

Representation Learning and Reinforcement Learning. Our work automates selecting useful signals from noisy MDPs by learning denoised world models, and can be viewed as an approach for learning general representations (Donahue et al., 2014; Mikolov et al., 2013; He et al., 2019; Huh et al., 2016). In model-free RL, various methods learn state embeddings that are related to value functions (Schaul et al., 2015; Bellemare et al., 2019), transition dynamics (Mahadevan & Maggioni, 2007; Lee et al., 2020), recent action (Pathak et al., 2017), bisimulation structure (Ferns et al., 2004; Castro, 2020; Zhang et al., 2020), data augmentations (Laskin et al., 2020b) etc. Recently, Eysenbach et al. (2021) proposes a regularizer similar to ours but for the different purpose of robust compressed policies. The theoretical work by Efroni et al. (2021) is closest to our setting, but does not yield a practical algorithm and only discusses distractors that are both uncontrollable and reward-irrelevant.

System Identification. Our work is related to system identification, where an algorithm infers from real world an abstract state among a predefined limited state space, e.g., pose estimation (Rıza Alp Güler, 2018; Yen-Chen et al., 2021) and material estimation (Hahn et al., 2019). Such results are useful for robotic manipulation (Manuelli et al., 2019), image generation (Gu et al., 2019), etc. Our setting is not limited to a predefined abstract state space, but instead focuses on automatic discovery of such valuable states.

5. Experiments

In this section, we contrast our method with existing approaches on environments with image observations and many distinct types of noise distractors. Our experiments are designed to include a variety of noise distractors and to confirm our analysis on various methods in Section 2.3.

Environments. We choose DeepMind Control (DMC) Suite (Tunyasuvunakool et al., 2020) (Section 5.2) and `RoboDesk` (Kannan et al., 2021) (Section 5.1) with image observations, where we explore adding various noise distractors. Information types in all evaluated environments are categorized in Table 2 of the appendix. Tasks include control (policy optimization) and a non-control task of regressing robot joint position from `RoboDesk` image observations.

Methods. We compare not only model-based RL methods, but also model-free algorithms and general representation learning approaches, when the task is suited:

- **Model Learning**: Denoised MDP (our method), Dreamer (Hafner et al., 2019a), and TIA (Fu et al., 2021);
- **Model-Free**: DBC (Zhang et al., 2020), CURL (Laskin et al., 2020b), PI-SAC (Lee et al., 2020) (without data augmentation for a fair comparison of its core predictive information regularization against other non-augmenting methods), and SAC on true state-space (Haarnoja et al., 2018) (instead of using image observation space, this is a rough "upper bound");
- **General Image Representation Learning for Non-Control Tasks**: Contrastive learning with the Alignment+Uniformity loss (Wang & Isola, 2020) (a form of contrastive loss theoretically and empirically comparable to the popular InfoNCE loss (Oord et al., 2018)).

Model-learning methods can be used in combination with any policy optimization algorithm. For a complete comparison for general control, we compare the models trained with these two policy learning choices: (1) backpropagating via the learned dynamics and (2) SAC on the learned latent space (which roughly recovers SLAC (Lee et al., 2019) when used with an unfactorized model such as Dreamer).

Most compared methods do not apply data augmentations, which is known to strongly boost performance (Yarats et al., 2021; Laskin et al., 2020a). Therefore, for a fair comparison, we run PI-SAC *without augmentation* to highlight its main contribution—representation of only predictive information.

All results are aggregated from 5 runs, showing mean and standard deviations. The appendix contains more details, hyperparameter studies, and additional results. The supplementary video shows clearer video visualizations.

For Denoised MDP, we use the Figure 2b variant. Empirically, the Figure 2c variant leads to longer training time and sometimes inferior performance (perhaps due to having to optimize extra components and fit a more complex model). The appendix provides a comparison between them.

5.1. `RoboDesk` with Various noise distractors

We augment `RoboDesk` environment with many noise distractors that models realistic noises (e.g., flickering lights

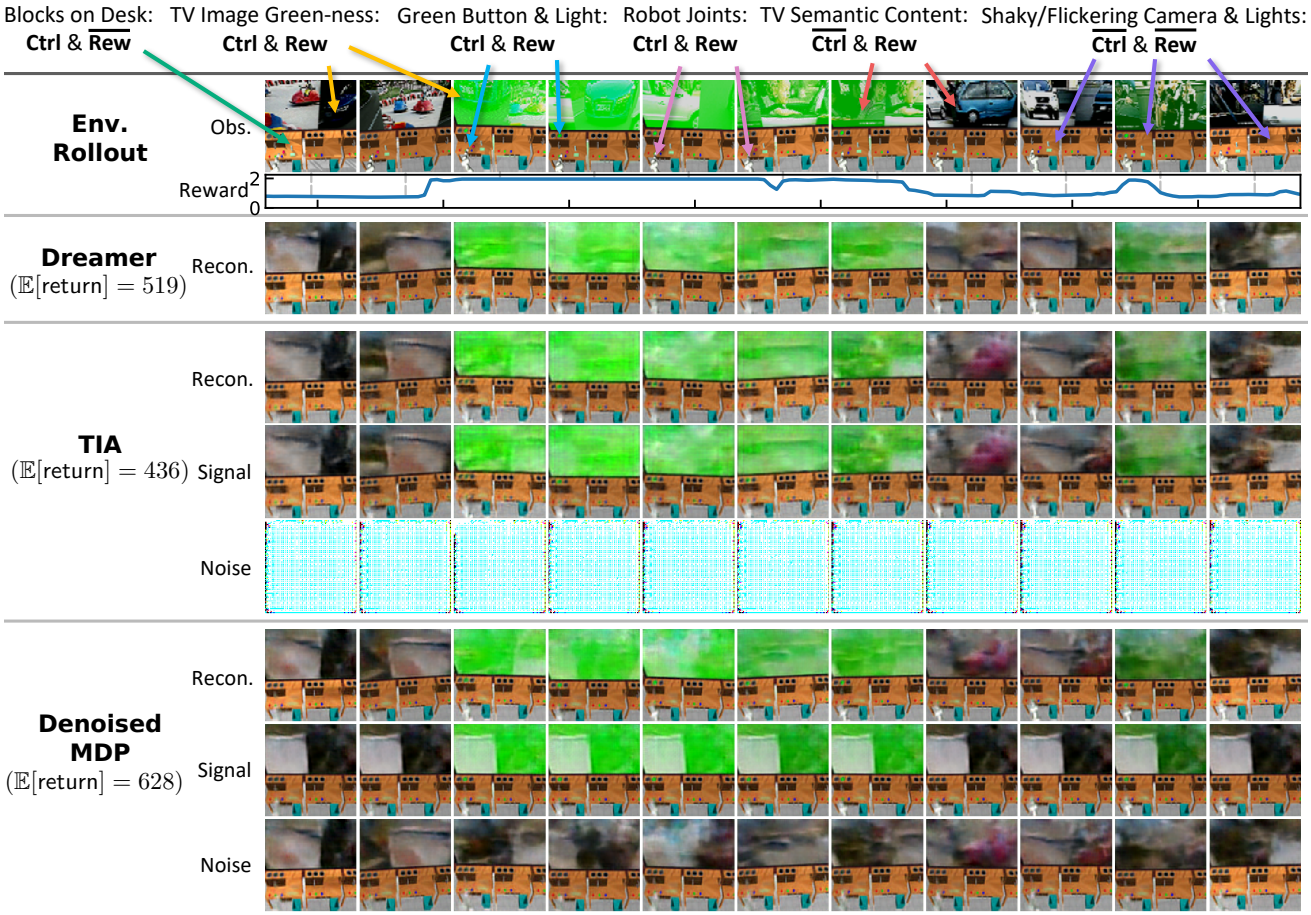


Figure 4: Visualization of learned models for RoboDesk by using decoders to reconstruct from encoded latents. For TIA and Denoised MDP, we visualize how they separate information as signal versus noise. In each row, *what changes over frames is the information modeled by the corresponding latent component.* E.g., in the bottom row, only the TV content, camera pose and lighting condition change, so Denoised MDP considers these factors as noises, while modelling the TV hue as signal. See our website for clearer video visualizations.

and shaky camera). Most importantly, we place a large TV in the scene, which plays natural RGB videos. A green button on the desk controls the TV's hue (and a light on the desk). The agent is tasked with using this button to shift the TV to a green hue. Its reward is directly affected by how green the TV image is. The first row of Figure 4 shows a trajectory with various distractors annotated. All four types of information exist (see Table 2), with the controllable and reward-relevant information being the robot arm, the green button, the light on the desk, and the TV screen green-ness.

Only Denoised MDP learns a clean denoised model. Using learned decoders, Figure 4 visualizes how the models captures various information. As expected, Dreamer model captures all information. TIA also fails to separate any noise distractors out (the Noise row fails to capture anything), likely due to its limited ability to model different noises. In contrast, Denoised MDP cleanly extracts all controllable and reward-relevant information as signals—the Signal row only *tracks changes* in robot arms, green button and light, and the TV screen green-ness. All other information is modeled as noises (see the Noise row). We recommend viewing video visualizations on our website.

Denoised models improve policy learning. Figure 4 also shows the total episode return achieved by policies learned with each of the three models, where the cleanest model from Denoised MDP achieves the best performance. Aggregating over 5 runs, the complete comparison in Figure 5 shows that Denoised MDP (with backpropagating via dynamics) generally outperforms all baselines, suggesting that its clean models are helpful for control.

Denoised models benefit non-control tasks. We evaluate the learned representations on a *supervised non-control* task—regressing the robot arm joint position from observed images. Using various pretrained encoders, we finetune on a labeled training set, and measure mean squared error (MSE) on a heldout test set. In addition to RL methods, we compare encoders learned via general contrastive learning on the same amount of data. In Figure 6, Denoised MDP representations lead to best converged solutions across a wide range of training set sizes, achieve faster training, and

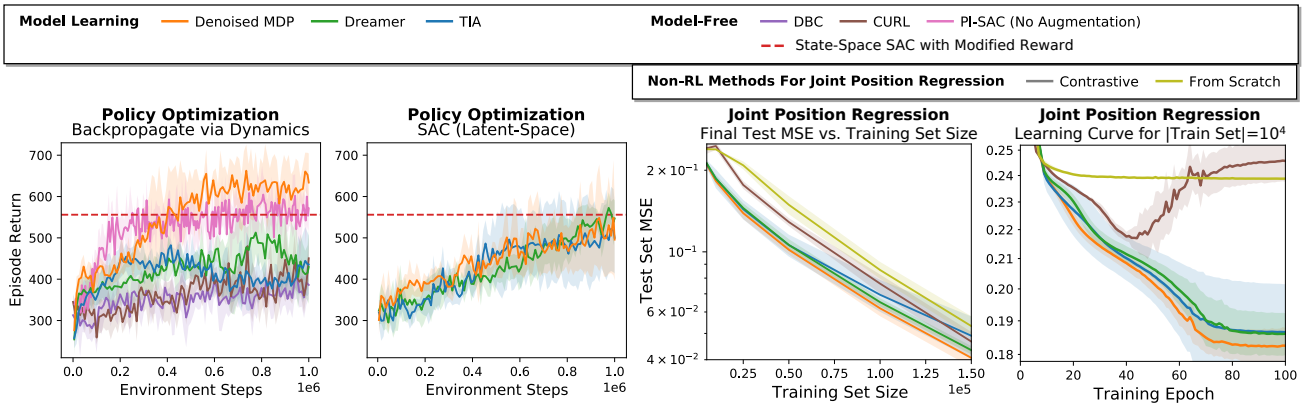


Figure 5: Policy optimization on RoboDesk. We give state-space SAC a less noisy reward so it can learn (see appendix).

Figure 6: Performance of finetuning various encoders to infer joint position from RoboDesk image observation.

	Policy Learning: Backprop via Dynamics			**Policy Learning:** SAC (Latent-Space)			DBC	PI-SAC (No Aug.)	CURL (Use Aug.)	State-Space SAC (Upper Bound)
	Denoised MDP	Dreamer	TIA	Denoised MDP	Dreamer	TIA				
Noiseless	801.4 ± 96.6	**848.6 ± 137.1**	769.7 ± 97.1	**587.1 ± 98.7**	575.4 ± 146.2	480.2 ± 125.5	297.4 ± 72.5	246.4 ± 56.6	417.3 ± 183.2	910.3 ± 28.2
Video Background	**597.7 ± 117.8**	227.8 ± 102.7	407.1 ± 225.4	309.8 ± 153.0	188.7 ± 78.2	**318.1 ± 123.7**	188.0 ± 67.4	131.7 ± 20.1	478.0 ± 113.5	910.3 ± 28.2
Video Background + Noisy Sensor	**563.1 ± 143.0**	212.4 ± 89.7	261.2 ± 200.4	**288.2 ± 123.4**	218.2 ± 58.1	197.3 ± 124.2	79.9 ± 36.0	152.5 ± 12.6	354.3 ± 119.9	919.8 ± 100.7
Video Background + Camera Jittering	**254.1 ± 114.2**	98.6 ± 27.7	151.7 ± 160.5	**186.8 ± 47.7**	105.2 ± 33.8	126.5 ± 125.6	68.0 ± 38.4	91.6 ± 7.6	**390.4 ± 64.9**	910.3 ± 28.2

Table 1: DMC policy optimization results. For each variant, we aggregate performance across three tasks (Cheetah Run, Walker Walk, Reacher Easy) by averaging. Denoised MDP performs well across all four variants with distinct noise types. **Bold numbers** show the best model-learning result for specific policy learning choices, or the best overall result. On **Camera Jittering**, Denoised MDP greatly outperforms all other methods except for CURL, which potentially benefits from its specific data augmentation choice (random crop) on this task, and can be seen as using extra information (i.e., knowing the noise distractor form). In fact, Denoised MDP is the only method that consistently performs well across all tasks and noise variants, which can be seen from the full results in the appendix.

avoid overfitting when the training set is small. DBC, CURL and PI-SAC encoders, which take in stacked frames, are not directly comparable and thus absent from Figure 6. In the appendix, we compare them with running Denoised MDP encoder on each frame and concatenating the output features, where Denoised MDP handily outperforms both DBC and CURL by a large margin.

5.2. DeepMind Control Suite (DMC)

To evaluate a diverse set of noise distractors, we consider four variants for each DMC task (see Figure 7 top row):

- **Noiseless**: Original environment without distractors.
- **Video Background**: Replacing noiseless background with natural videos (Zhang et al., 2020) ($\overline{\textbf{Ctrl}} + \overline{\textbf{Rew}}$).
- **Video Background + Sensor Noise**: Imperfect sensors sensitive to intensity of a background patch ($\overline{\textbf{Ctrl}} + \textbf{Rew}$).
- **Video Background + Camera Jittering**: Shifting the observation by a smooth random walk ($\overline{\textbf{Ctrl}} + \overline{\textbf{Rew}}$).

Denoised MDP consistently removes noise distractors. In Figure 7, TIA struggles to learn clean separations in many settings. Consistent with analysis in Section 2.3, it cannot handle **Sensor Noise** or **Camera Jittering**, as the former is reward-relevant noise that it cannot model, and the latter (although reward-irrelevant) cannot be represented by masking. Furthermore, it fails on Reacher Easy with **Video Background**, where the reward is given by the distance between the agent and a randomly-located ball. TIA encourages its noise latent to be independent of reward, but does not prevent it from capturing the controllable agent. These failures lead to either TIA trying to model everything as useful signals, or a badly-fit model (e.g., wrong agent pose in the last column). In contrast, Denoised MDP separates out noise in all cases, obtaining a clean and accurate MDP (its Signal rows only have the agent moving).

Denoised models consistently improve policy learning. We evaluate the learned policies in Table 1, where results are aggregated by the noise distractor variant. Other methods, while sometimes handling certain noise types well, struggle to deal with all four distinct variants. TIA, as expected, greatly underperforms Denoised MDP under **Noisy Sensor** or **Camera Jittering**. CURL, whose augmentation choice potentially helps handling **Camera Jittering**, underperforms in other three variants. In contrast, Denoised MDP policies *consistently* perform well for all noisy variants and also the noiseless setting, regardless of the policy optimizer.

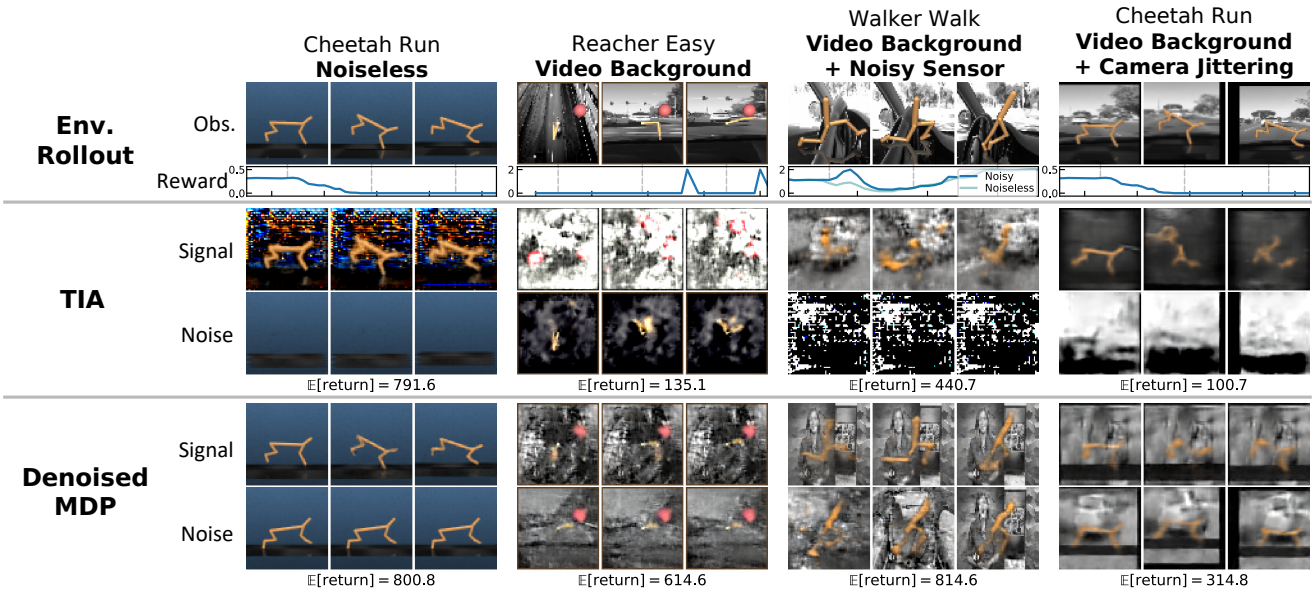


Figure 7: Visualization of the different DMC variants and factorizations learned by TIA and Denoised MDP. E.g., bottom Noise row often shows a static agent but varying background, indicating that only the background is modeled as noises in Denoised MDP. Visualizations of full reconstructions are in appendix. See our website for clearer video visualizations.

Model-based approaches have a significant lead over the model-free ones, as seen from the DBC results in Table 1 and the well-known fact that direct model-free learning on raw image observations usually fails (Laskin et al., 2020a; Kostrikov et al., 2020; Yarats et al., 2021). These results show that learning in a world model is useful, and that learning in a denoised world model is even better.

6. Implications

In this work we explore learning denoised and compressed world models in the presence of environment noises.

As a step towards better understanding of such noises, we categorize of information in the wild into four types (Section 2). This provides a framework to contrast and understand various methods, highlighting where they may be successful and where they will suffer (Section 2.3). Insights gained this way empirically agrees with findings from extensive experiments (Section 5). It can potentially assist better algorithm design and analysis of new MDP representation methods, as we have done in designing Denoised MDP (Section 3). We believe that this categorization will be a useful framework for investigation on learning under noises, revealing not just the (conceptual) success scenarios, but also the failure scenarios at the same time. Additionally, the framework can be readily extended with more sophisticated factorizations (Section 2.4), which can lead to corresponding Denoised MDP variants and/or new algorithms.

Based on the framework, our proposed Denoised MDP novelly can remove *all* noise distractors that are *uncontrollable or reward-irrelevant*, in distinction to prior works. Empirically, it effectively identifies and removes a diverse set of noise types, obtaining clean denoised world models (Section 5). It may serve as an important step towards efficient learning of general tasks in the noisy real world. Our experiments also highlight benefits of cleanly denoised world models on both standard control tasks as well as non-control tasks. The success in both cases highlights the general usefulness of such models. Given the generality of MDPs, this opens up the possibility of casting non-RL tasks as MDPs and automatically learn representations from denoised world models, as an alternative to manual feature engineering.

Acknowledgements

We thank Jiaxi Chen for the beautiful Figure 1 illustration. We thank Daniel Jiang and Yen-Chen Lin for their helpful comments and suggestions. We are grateful to the following organizations for providing computation resources to this project: IBM's MIT Satori cluster, MIT Supercloud cluster, and Google Cloud Computing with credits gifted by Google to MIT.

References

Agarwal, A., Kakade, S., Krishnamurthy, A., and Sun, W. FLAMBE: Structural complexity and representation learning of low rank mdps. *arXiv preprint arXiv:2006.10814*, 2020.

Bellemare, M., Dabney, W., Dadashi, R., Ali Taiga, A., Castro, P. S., Le Roux, N., Schuurmans, D., Lattimore, T., and Lyle, C. A geometric perspective on optimal representations for reinforcement learning. *Advances in neural information processing systems*, 32:4358–4369, 2019.

Castro, P. S. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 10069–10076, 2020.

Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.

Craik, K. J. W. *The nature of explanation*, volume 445. CUP Archive, 1952.

Dennett, D. C. Why the law of effect will not go away. *Journal for the Theory of Social Behaviour*, 1975.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pp. 647–655. PMLR, 2014.

Du, S., Krishnamurthy, A., Jiang, N., Agarwal, A., Dudik, M., and Langford, J. Provably efficient rl with rich observations via latent state decoding. In *International Conference on Machine Learning*, pp. 1665–1674. PMLR, 2019.

Efroni, Y., Misra, D., Krishnamurthy, A., Agarwal, A., and Langford, J. Provable rl with exogenous distractors via multistep inverse dynamics. *arXiv preprint arXiv:2110.08847*, 2021.

Elman, J. L. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Eysenbach, B., Salakhutdinov, R., and Levine, S. Robust predictable control. *arXiv preprint arXiv:2109.03214*, 2021.

Ferns, N., Panangaden, P., and Precup, D. Metrics for finite markov decision processes. In *UAI*, volume 4, pp. 162–169, 2004.

Fu, X., Yang, G., Agrawal, P., and Jaakkola, T. Learning task informed abstractions. In *International Conference on Machine Learning*, pp. 3480–3491. PMLR, 2021.

Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pp. 2170–2179. PMLR, 2019.

Givan, R., Dean, T., and Greig, M. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

Gu, S., Bao, J., Yang, H., Chen, D., Wen, F., and Yuan, L. Mask-guided portrait editing with conditional gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3436–3445, 2019.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019b.

Hahn, D., Banzet, P., Bern, J. M., and Coros, S. Real2sim: Visco-elastic parameter estimation from dynamic motion. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

Huh, M., Agrawal, P., and Efros, A. A. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

Kannan, H., Hafner, D., Finn, C., and Erhan, D. RoboDesk: A multi-task reinforcement learning benchmark. https://github.com/google-research/robodesk, 2021.

Kim, S. W., Zhou, Y., Philion, J., Torralba, A., and Fidler, S. Learning to Simulate Dynamic Environments with GameGAN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems*, 33: 19884–19895, 2020a.

Laskin, M., Srinivas, A., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020b.

Lee, A. X., Nagabandi, A., Abbeel, P., and Levine, S. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

Lee, K.-H., Fischer, I., Liu, A., Guo, Y., Lee, H., Canny, J., and Guadarrama, S. Predictive information accelerates learning in rl. *Advances in Neural Information Processing Systems*, 33:11890–11901, 2020.

Lowe, D. G. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pp. 1150–1157. Ieee, 1999.

Mahadevan, S. and Maggioni, M. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(10), 2007.

Manuelli, L., Gao, W., Florence, P., and Tedrake, R. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Modi, A., Jiang, N., Tewari, A., and Singh, S. Sample complexity of reinforcement learning using linearly combined model ensembles. In *International Conference on Artificial Intelligence and Statistics*, pp. 2010–2020. PMLR, 2020.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Poole, B., Ozair, S., Van Den Oord, A., Alemi, A., and Tucker, G. On variational bounds of mutual information. In *International Conference on Machine Learning*, pp. 5171–5180. PMLR, 2019.

Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

Rıza Alp Güler, Natalia Neverova, I. K. Densepose: Dense human pose estimation in the wild. 2018.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

Smaira, L., Carreira, J., Noland, E., Clancy, E., Wu, A., and Zisserman, A. A short note on the kinetics-700-2020 human action dataset. *arXiv preprint arXiv:2010.10864*, 2020.

Spelke, E. S. and Kinzler, K. D. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

Sutton, R. S. An adaptive network that constructs and uses and internal model of its world. *Cognition and Brain Theory*, 4(3):217–246, 1981.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

Wang, T. and Isola, P. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9929–9939. PMLR, 13–18 Jul 2020.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Yen-Chen, L., Florence, P., Barron, J. T., Rodriguez, A., Isola, P., and Lin, T.-Y. iNeRF: Inverting neural radiance fields for pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

Zhang, A., McAllister, R., Calandra, R., Gal, Y., and Levine, S. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

A. Denoised MDP Discussions

A.1. Loss Derivation

To apply our mutual information regularizer $I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a})$, we can consider a form using another variational distribution ρ (see, e.g., Poole et al. (2019)),

$$\begin{aligned} I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) &= \min_{\rho} \mathbb{E}_{\boldsymbol{a}} \mathbb{E}_{p_\theta(\boldsymbol{s}|\boldsymbol{a})} \left[D_{\mathsf{KL}}(p_\theta(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a}) \parallel \rho(\boldsymbol{x} \mid \boldsymbol{a})) \right] \\ &\approx \min_{\rho} \mathbb{E}_{\boldsymbol{a}} \mathbb{E}_{q_\psi(\boldsymbol{s}|\boldsymbol{a})} \left[D_{\mathsf{KL}}(q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a}) \parallel \rho(\boldsymbol{x} \mid \boldsymbol{a})) \right] \qquad \text{(assume } q_\psi \text{ is roughly the posterior of } p_\theta) \\ &= \min_{\theta'} \mathcal{L}_{\mathsf{KL}\text{-}x}(\psi, \theta'). \end{aligned} \tag{5}$$

The assumption that q_ψ is roughly the posterior of p_θ is acceptable because it is the natural consequence of optimizing the variational MLE objective in Equation (1) over θ, ψ.

Alternatively, we can consider the MI defined by a joint conditional distribution $P(\boldsymbol{x}, \boldsymbol{s} \mid a)$ not from the forward model p_θ, but from the data distribution and posterior model $q_\psi(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{a})$. This is also sensible because the variational MLE objective in Equation (1) optimizes for compatible p_θ and q_ψ that both fit data and consistently describe (conditionals of) the same underlying distribution. Thus regularizing either can encourage a low MI. This approach leads to exactly Equation (5), without approximation.

Then, the total loss in Equation (3) from combining Equations (1) and (5) is given by

$$\begin{aligned} \min_{\theta} \mathcal{L}_{\mathsf{MLE}}(\theta) + c \cdot I(\boldsymbol{x}; \boldsymbol{s} \mid \boldsymbol{a}) &= \min_{\theta, \theta', \psi} \mathcal{L}_{\mathsf{recon}}(\theta, \psi) + \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta, \psi) + \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta, \psi) + \mathcal{L}_{\mathsf{KL}\text{-}z} + c \cdot + \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta', \psi) \\ &= \min_{\theta, \psi} \mathcal{L}_{\mathsf{recon}}(\theta, \psi) + (1 + c) \cdot \mathcal{L}_{\mathsf{KL}\text{-}x}(\theta, \psi) + \mathcal{L}_{\mathsf{KL}\text{-}y}(\theta, \psi) + \mathcal{L}_{\mathsf{KL}\text{-}z}(\theta, \psi). \end{aligned}$$

A.2. Discussions

Posterior distributions of r_x and r_y. The p_θ reward distributions $p_\theta(r_x \mid x_t)$ and $p_\theta(r_y \mid y_t)$ are modelled via Gaussians (as is done in usual world models, such as Dreamer (Hafner et al., 2019a)). By the transition structure of Denoised MDPs, these distributions are inherently independent. Recall that $r = r_x + r_y$. Therefore, we can easily compute the distribution of $p_\theta(r \mid x_t, y_t)$ and its log likelihoods. This enables easy optimization of the variational MLE objective, without letting the posterior model to also infer r_x and r_y.

Partial observability. Sections 2 and 3 discussions are mostly based in the fully observable setting. Yet most benchmarks and real-world tasks are partially observable, e.g., robot joint speeds that can not be inferred from a single frame. Fortunately, the transition models used in Denoised MDP are fully capable of handle such cases, as long as the encoder q_ψ is not deterministic and the observation model $p_\theta(s \mid \dots)$ does not have the block structure (Du et al., 2019) (which would make x, y, z fully determined from s). In practice, we let both components to be generic conditional distributions (parameterized by regular deep neural networks). Therefore, Denoised MDP does not require full observability.

Hyperparameter choice. The loss in Equation (4) has two hyperparameters $\alpha \in (0, \infty), \beta \in (0, 1)$. To maintain relative ratio with the observation reconstruction loss, we recommend scaling α roughly proportionally with dimensionality of observation space. A smaller β means stronger regularization. Therefore, β can be chosen based on training stability and the level of noise distractors in the task.

B. Experiment Details

All code (including code for our environment variants and code for our Denoised MDP method) will be released upon publication.

B.1. Implementation Details

B.1.1. Environments and Tasks

In all environments, trajectories are capped at 1000 timesteps. Table 2 shows a summary of what kinds of information exist in each environment.

		Ctrl + Rew	**Ctrl + $\overline{\text{Rew}}$**	**$\overline{\text{Ctrl}}$ + Rew**	**$\overline{\text{Ctrl}}$ + $\overline{\text{Rew}}$**
DMC	**Noiseless**	Agent	—	—	—
	Video Background	Agent	—	—	Background
	Video Background + Noisy Sensor	Agent	—	Background	—
	Video Background + Camera Jittering	Agent	—	—	Background, Jittering camera
	`RoboDesk`	Agent, Button, Light on desk, Green hue of TV	Blocks on desk, Handle on desk, Other movable objects	TV content, Button sensor noise	Jittering and flickering environment lighting, Jittering camera

Table 2: Categorization of various information in the environments we evaluated with.

DeepMind Control Suite (DMC). Our **Video Background** implementation follows Deep Bisimulation for Control (Zhang et al., 2020) on most environments, using `Kinetics-400` grayscale videos (Smaira et al., 2020), and replacing pixels where blue channel is strictly the greatest of three. This method, however, does not cleanly remove most of background in the Walker Walk environment, where we use an improved mask that replaces all pixels where the blue channel is *among the greatest* of three. For **Camera Jittering**, we shift the observation image according to a smooth random walk, implemented as, at each step, Gaussian-perturbing acceleration, decaying velocity, and adding a pulling force if the position is too far away from origin. For **Sensor Noise**, we select one sensor, and perturb it according to intensity of a patch of the natural video background (i.e., adding average patch value -0.5). We perturb the `speed` sensor for Cheetah Run, the `torso_height` sensor for Walker Walk, and the normalized `finger_to_target_dist` sensor for Reacher Easy. These sensor values undergo non-linear (mostly piece-wise linear) transforms to compute rewards. While they can not be perfectly modelled by additive reward noise, such a model is usually sufficient in most cases when the sensor values are not too extreme and stay in one linear region.

`RoboDesk`. We modify the original `RoboDesk` environment by adding a TV screen and two neighboring desks. The TV screen places (continuously horizontally shifting) natural RGB videos from the `Kinetics-400` dataset (Smaira et al., 2020). The environment has three light sources from the above, to which we added random jittering and flickering. The viewing camera is placed further to allow better view of the noise distractors. Resolution is increased from 64×64 to 96×96 to compensate this change. Camera jittering is implemented by a 3D smooth random walk. Finally, the button sensor (i.e., detected value of how much the button is pressed) is also offset by a random walk. Each of the three button affects the corresponding light on the desk. Additionally, pressing the green button also shifts the TV screen content to a green hue. Following `RoboDesk` reward design, we reward the agent for (1) placing arm close to the button, (2) pressing the button, and (3) how green the TV screen content is.

`RoboDesk` Joint Position Regression Datasets. To generate training and test set, we use four policies trained by state-space SAC at different stages of training (which is not related to any of the compared methods) and a uniform random actor, to obtain five policies of different qualities. For each policy, we sample 100 trajectories, each containing 1001 pairs (from 1000 interactions) of image observation and groundtruth joint position (of dimension 9). This leads to a total of 500.5×10^3 samples from each policy. From these, 100×10^3 samples are randomly selected as test set. Training sets of sizes $5 \times 10^3, 10 \times 10^3, 25 \times 10^3, 50 \times 10^3, 100 \times 10^3, 150 \times 10^3$ are sampled from the rest. For all test sets and training sets, we enforce each policy to strictly contribute an equal amount.

B.1.2. Model Learning Methods

For all experiments, we let the algorithms use 10^6 environment steps. For PI-SAC and CURL, we follow the original implementations (Laskin et al., 2020b; Lee et al., 2020) and use an action repeat of 4 for Cheetah Run and Reacher Easy, and an action repeat of 2 for Walker Walk. For Denoised MDP, Dreamer, TIA and DBC, we always use an action repeat of 2, following prior works (Hafner et al., 2019a; Fu et al., 2021; Zhang et al., 2020).

Operator	Input Shape	Kernel Size	Stride	Padding
Input	$[3, 96, 96]$	—	—	—
Conv. + ReLU	$[k, 47, 47]$	4	2	0
Conv. + ReLU	$[2k, 22, 22]$	4	2	0
Conv. + ReLU	$[4k, 10, 10]$	4	2	0
Conv. + ReLU	$[8k, 4, 4]$	4	2	0
Conv. + ReLU	$[8k, 2, 2]$	3	1	0
Reshape + FC	$[m]$	—	—	—

Table 3: Encoder architecture for 96×96-resolution observation. The output of this encoder is then fed to other network for inferring posteriors. m and k are two architectural hyperparameters. m controls the output size (unrelated to the actual latent variable sizes). k controls the network width.

Operator	Input Shape	Kernel Size	Stride	Padding
Input	`[input_size]`	—	—	—
FC + ReLU + Reshape	$[m, 1, 1]$	—	—	—
Conv. Transpose + ReLU	$[4k, 3, 3]$	5	2	0
Conv. Transpose + ReLU	$[4k, 9, 9]$	5	2	0
Conv. Transpose + ReLU	$[2k, 21, 21]$	5	2	0
Conv. Transpose + ReLU	$[k, 46, 46]$	6	2	0
Conv. Transpose + ReLU	$[3, 96, 96]$	6	2	0

Table 4: Decoder architecture for 96×96-resolution observation. m and k are two architectural hyperparameters. m controls width the fully connected part. k controls width of the convolutional part. They are the same values as in Table 3.

	DMC				RoboDesk			
	Latent Sizes	m	k	Total Number of Parameters	Latent Sizes	m	k	Total Number of Parameters
Dreamer	$(220 + 33)$	1024	32	7,479,789	$(220 + 33)$	1024	32	6,385,511
TIA	$(120 + 20) + (120 + 20)$	490	24	7,475,567	$(120 + 20) + (120 + 20)$	490	24	6,384,477
Denoised MDP	$(120 + 20) + (120 + 20)$	1024	32	7,478,826	$(120 + 20) + (120 + 20)$	1024	32	6,384,248

Table 5: The specific architecture parameters for model learning methods. Since RSSM uses a deterministic part and a stochastic part to represent each latent variable, we use (`deterministic_size` + `stochastic_size`) to indicate size of a latent variable. TIA and Denoised MDP have more than one latent variable. Note that while TIA has lower m and k, it has multiple encoder and decoders, whereas Dreamer and Denoised MDP only have one encoder and one decoder. The total number of parameters is measured with the actor model, but without any additional components from policy optimization algorithm (e.g., critics in SAC). Total number of parameters is lower for `RoboDesk` as the encoder and decoder architecture is narrower than those of DMC for the purpose of reducing memory usage, despite with a higher resolution.

Denoised MDP, Dreamer, TIA. Both Dreamer and TIA use the same training schedule and the Recurrent State-Space Model (RSSM) as the base architecture (Hafner et al., 2019b). Following them, Denoised MDP also uses these components, and follow the same prefilling and training schedule (see Dreamer (Hafner et al., 2019b) for details). These three model learning methods take in 64×64 RGB observations for DMC, and 96×96 RGB observations for `RoboDesk`. Dreamer only implements encoder and decoder for the former resolution. To handle the increased resolution, we modify the 64×64 architectures and obtain convolutional encoder and decoder shown in Tables 3 and 4. For fair comparison, we ensure that each method has roughly equal number of parameters by using different latent variable sizes, encoder output sizes (m of Table 3) and convolutional net widths (k of Table 4). Details are shown in Table 5. For Denoised MDP, we also follow Hafner et al. (2019b;a) and allow 3 free nats for the $\mathcal{L}_{\mathsf{KL}\text{-}x}$ term. However, we do not allow this for the $\mathcal{L}_{\mathsf{KL}\text{-}y}$ and $\mathcal{L}_{\mathsf{KL}\text{-}z}$ terms, as these variables are to be discarded and information is not allowed to hide in them unless permitted by the structure.

B.1.3. Policy Optimization Algorithms Used with Model Learning

Backpropagate via Dynamics. We use the same setting as Dreamer (Hafner et al., 2019a), optimizing a λ-return over 15-step-long rollouts with $\lambda = 0.95$, clipping gradients with norm greater than 100. TIA uses the same strategy, except that it groups different models together for gradient clipping. We strictly follow the official TIA implementation.

Latent-Space SAC. We use the regular SAC with automatic entropy tuning, without gradient clipping. This works well for almost all settings, except for Walker Walk variant of DMC, where training often collapses after obtaining good return, regardless of the model learning algorithm. To address instability in this case, we reduce learning rates from 3×10^{-4} to 1×10^{-4} and clip gradients with norm greater than 100 for all latent-space SAC run on these variants.

B.1.4. Model-Free Methods

DBC. For DMC, we used 84×84-resolution observation following original work (even though other methods train on 64×64-resolution observations). For `RoboDesk`, DBC uses the encoder in Table 3 for 96×96-resolution observation, for fair comparison with other methods. Following the original work, we stack 3 consecutive frames to approximate the required full observability. In the robot arm joint position regression experiment Section 5.1, DBC encoders also see stacked observations. For DMC evaluations, we use the data provided by Zhang et al. wherever possible, and run the official repository for other cases.

State-Space SAC. The state space usually contains robot joint states, including position, velocity, etc. For DMC, when **Sensor Noise** is present, this is not the true optimal state space, as we do not supply it with the noisy background that affects the noisy reward. However, it still works well in practice. For `RoboDesk`, the TV's effect on reward is likely stronger and direct state-space SAC fails to learn. Since this evaluation is to obtain a rough "upper bound", we train state-space SAC with a modified reward with less noise— the agent is rewarded by pressing the button, independent of the TV content. This still encourages the optimal strategy of the task allows achieving good policies.

B.1.5. Non-RL methods

Contrastive Learning. We used the Alignment+Uniformity contrastive learning loss from Wang & Isola (2020). The hyperparameters and data augmentations strictly follow their experiments on STL-10 (Coates et al., 2011), which also is of resolution 96×96. The exact loss form is $\mathcal{L}_{\mathsf{align}}(\alpha = 2) + \mathcal{L}_{\mathsf{uniform}}(t = 2)$, a high-performance setting for STL-10.

B.2. Compute Resources

All our experiments are run on a single GPU, requiring 8GB memory for DMC tasks, and 16GB memory for `RoboDesk` tasks. We use NVIDIA GPUs of the following types: 1080 Ti, 2080 Ti, 3080 Ti, P100, V100, Titan XP, Titan RTX. For `MuJoCo` (Todorov et al., 2012), we use the `EGL` rendering engine. Training time required for each run heavily depends on the CPU specification and availability. In general, a Denoised MDP run needs $12 \sim 36$ hours on DMC and $24 \sim 50$ hours on `RoboDesk`. TIA uses about $1.5\times$ of these times, due to the adversarial losses. For a comparison between the two Denoised MDP variants, running the same DMC task on the same machine, the Figure 2b variant used 23 hours while the Figure 2c variant used 26 hours.

B.3. Visualization Details

Visualizations of components in learned models. We use different methods to visualize signal and noise information learned by TIA and Denoised MDP in Figures 4 and 7. For TIA, we used the reconstructions from the two latent (before mask-composing them together as the full reconstruction). For Denoised MDP, we only have one decoder (instead of three for TIA), and thus we decode $(x_t, \texttt{const})$ and $(\texttt{const}, y_t)$ to visualize information contained in each variable, with `const` chosen by visual clarity (usually as value of the other variable at a fixed timestep). Due to the fundamental different ways to obtain these visualizations, in DMC, TIA can prevent the agent from showing up in noise visualizations, while Denoised MDP cannot. However, as stated in Section 5.2, our focus should be on what evolves/changes in these images, rather than what is visually present, as static components are essentially not modelled by the corresponding transition dynamics. Visualizations in Figures 4 and 7 use trajectories generated by a policy trained with state-space SAC. To obtain diverse behaviors, policy outputs are randomly perturbed before being used as actions. From the same trajectory, we use the above described procedure to obtain visualizations. The specific used trajectory segments are chosen to showcase both the modified environment and representative behavior of each method. Please see the supplementary video for clearer visualizations.

B.4. `RoboDesk` Result Details

Environment modifications. The agent controls a robotic arm placed in front of a desk and a TV, and is tasked to push down the green button on the desk, which turns on a small green light and makes the TV display have a green hue. The intensity of the TV image's green channel is given to the agent as part of their reward, in addition to distance between the arm to the button, and how much the button is pressed. Additionally, the environment contains other noise distractors, including moveable blocks on the desk (**Ctrl** + $\overline{\textbf{Rew}}$), flickering environment light and camera jittering ($\overline{\textbf{Ctrl}}$ + $\overline{\textbf{Rew}}$), TV screen hue (**Ctrl** + **Rew**), TV content ($\overline{\textbf{Ctrl}}$ + **Rew**), and noisy button sensors ($\overline{\textbf{Ctrl}}$ + **Rew**).

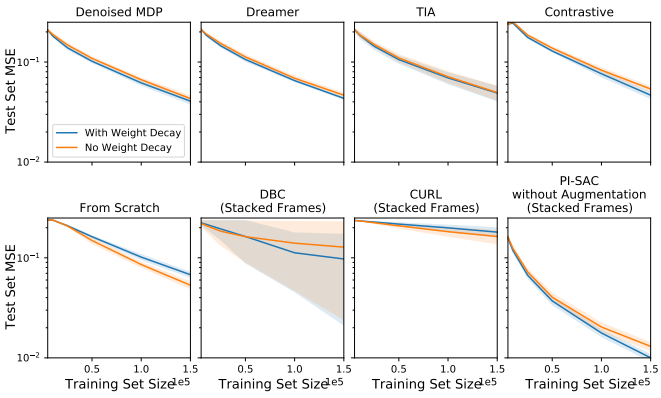


Figure 8: Effect of weight decay on RoboDesk joint position regression. The curves show final test MSE for various training set sizes. Weight decay generally helps when finetuning from a pretrained encoder, but hurts when training from scratch.

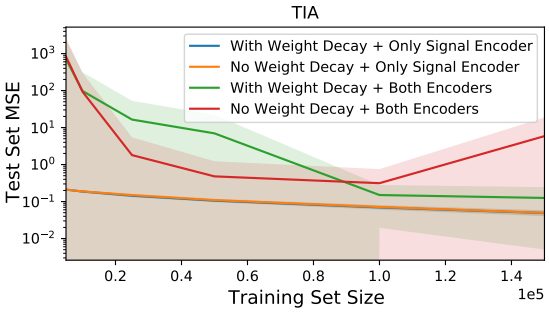


Figure 9: Performance of all TIA settings on RoboDesk joint position regression. Only using the signal encoder is necessary for good performance.

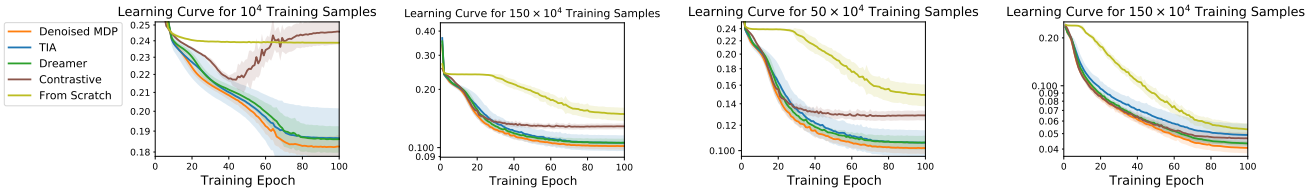


Figure 10: Training curve comparisons for the RoboDesk joint position regression task across many training set sizes.

Denoised MDP hyperparameters. RoboDesk has roughly twice as many pixels as DMC has. For Denoised MDP, we scale α with the observation space dimensionality (see Section 3) and use $\alpha = 2$, with a fixed $\beta = 0.125$.

TIA hyperparameters. We follow recommendations in the TIA paper, setting $\lambda_{\text{Radv}} = 25{,}000$ to match reconstruction loss in magnitude, and setting $\lambda_{O_s} = 2$ where training is stable.

B.4.1. Robot Arm Joint Position Regression.

Training details. For this task, we jointly train the pre-trained backbone and a three-layer MLP head that has 256 hidden units at each layer, with a learning rate of 8×10^{-5}. For finetuning from pretrained encoders, we follow common finetuning practice and apply a weight decay of 3×10^{-5} whenever it is helpful (all cases except CURL and training from scratch). See Figure 8 for comparisons for weight decay options over all methods.

- For model-based RL, we take encoders trained with backpropagating via dynamics as the policy optimization algorithm.
- In training the contrastive encoder, for a (more) fair comparison with RL-trained encoders that are optimized over 10^6 environment steps, we train contrastive encoders on 10^6 samples, obtained in the exact same method of the training sets of this task. In a sense, these contrastive encoders have the advantage of training on the exact same distribution, and seeing more samples (since RL-trained encoders use action repeat of 2 and thus only ever see 0.5×10^6 samples).
- TIA has two sets of encoders. Using concatenated latents from both unfortunately hurts performance greatly (see Figure 9). So we use only the encoder for the signal latent.

We also compare training speeds over a wide range of training set sizes in Figure 10. Denoised MDP encoders lead to faster and better training in all settings.

Additional comparison with frame-stacking encoders. Other pretrained encoders (DBC, CURL and PI-SAC) take in stacked 3 consecutive frames, and are not directly comparable with the other methods. To compare, we also try running Denoised MDP encoders on the 3 consecutive frames, whose feature vector is concatenated before feeding into the head. The result in Figure 11 shows that our encoder outperforms all but PI-SAC encoders. Finally, for DBC, CURL and PI-SAC, we attempted evaluating intermediate features, features before the final layer normalization, and the output space, and find the last option best-performing for DBC and CURL, and the second option best-performing for PI-SAC (see Figures 12

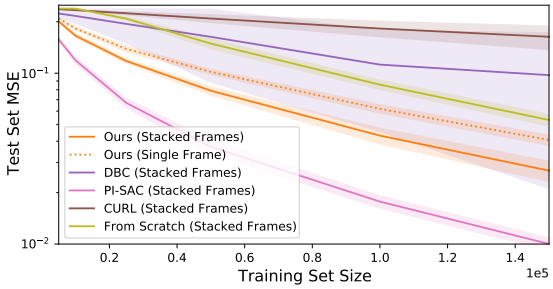


Figure 11: Performance comparison of finetuning from Denoised MDP encoders and frame-stacked encoders that take in 3 consecutive frames. For Denoised MDP and training from scratch, the encoders *take in only a single frame* and are applied for each of the frame, with output concatenated together before feeding to the prediction head.

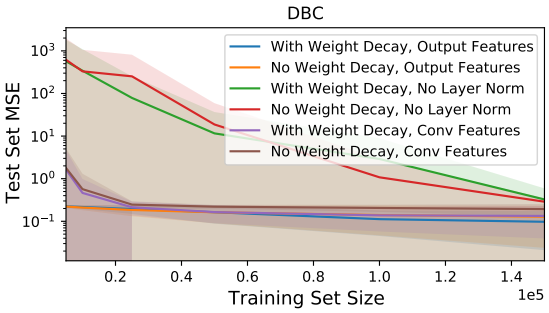


Figure 12: Performance of all DBC settings on RoboDesk joint position regression. Using the output features (after layer normalization) is necessary for good performance.

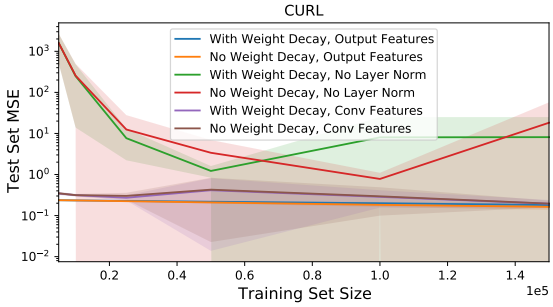


Figure 13: Performance of all CURL settings on RoboDesk joint position regression. Using the output features (after layer normalization) is necessary for good performance.

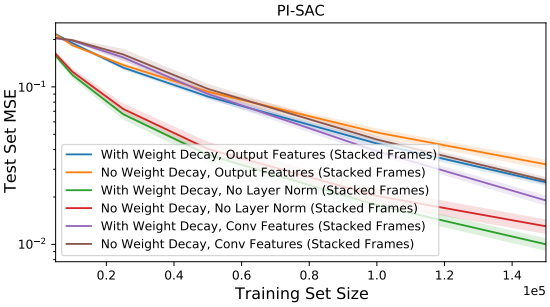


Figure 14: Performance of all PI-SAC settings on RoboDesk joint position regression. Using the activations *before layer normalization* gives best performance.

to 14). Therefore, we use these respective spaces, which arguably gives a further edge to these methods, as we essentially tune this additional option on test results. Notably, these respective choices are often the only one achieving relatively good performance, highlighting the necessity of tuning for these methods.

B.5. DeepMind Control Suite (DMC) Result Details

Full policy optimization results. Figure 15 presents the full results on each DMC environment (task + variant). For environment, a comparison plot is made based on which policy learning algorithm is used with the model learning method (with model-free baselines duplicated in both). Such separation is aimed to highlight the performance difference caused by model structure (rather than policy learning algorithm). Across most noisy environments, Denoised MDP performs the best. It also achieves high return on noiseless environments.

Visualization of learned models. Figure 16 is the extended version of Figure 7 in main text, with full reconstructions from all three models. Please see the supplementary video for clearer visualizations.

Comparison between Denoised MDP variants. We compare the two Denoised MDP variants based Figures 2b and 2c on Cheetah Run environments with policy trained by packpropagating via learned dynamics. The comparison is shown in the top row of Figure 15, where we see the Figure 2b variant often performing a bit better. We hypothesize that this may due to the more complex prior and posterior structure of Figure 2c, which may not learn as efficiently. This also makes Figure 2c variant needing longer (wall-clock) time to optimize, as mentioned above in Appendix B.2.

TIA hyperparameters and instability. We strictly follow recommendations of the original paper, and use their suggested value for each DMC task. We also note that TIA runs sometimes collapse during training, leading to sharp drops in rewards. After closely inspecting the models before and after collapses, we note that in many cases, such collapses co-occur with sudden spikes in TIA's reward disassociation loss, which is implemented as an adversarial minimax loss, and the noise latent

space instantly becomes degenerate (i.e., not used in reconstruction). We hypothesize that this adversarial nature can cause training instability. However, a few collapses do not co-occur with such loss spikes, which maybe alternatively due to that TIA model structure cannot model the respective noise types and that better fitting the model naturally means a degenerate noise latent space.

PI-SAC hyperparameters. For each task, we use the hyperparameters detailed in the original paper (Lee et al., 2020). PI-SAC is usually run with augmentations. However, unlike CURL, augmentation is not an integral part of the PI-SAC algorithm and is completely optional. For a fair comparisons with other methods and to highlight the effect of the predictive information regularizer, the main mechanism proposed by PI-SAC, we do not use augmentations for PI-SAC.

Denoised MDP hyperparameters. For DMC, we always use fixed $\alpha = 1$. β can be tune according to amount of noises in environment, and to training stability. In Figure 17, we compare effects of choosing different β's. On noiseless environments, larger β (i.e., less regularization) performs often better. Whereas on noisy environments, sometimes stronger regularization can boost performance. However, overall good performance can be obtained by usually several β values. In Table 6, we summarize our β choices for each environment in Table 6.

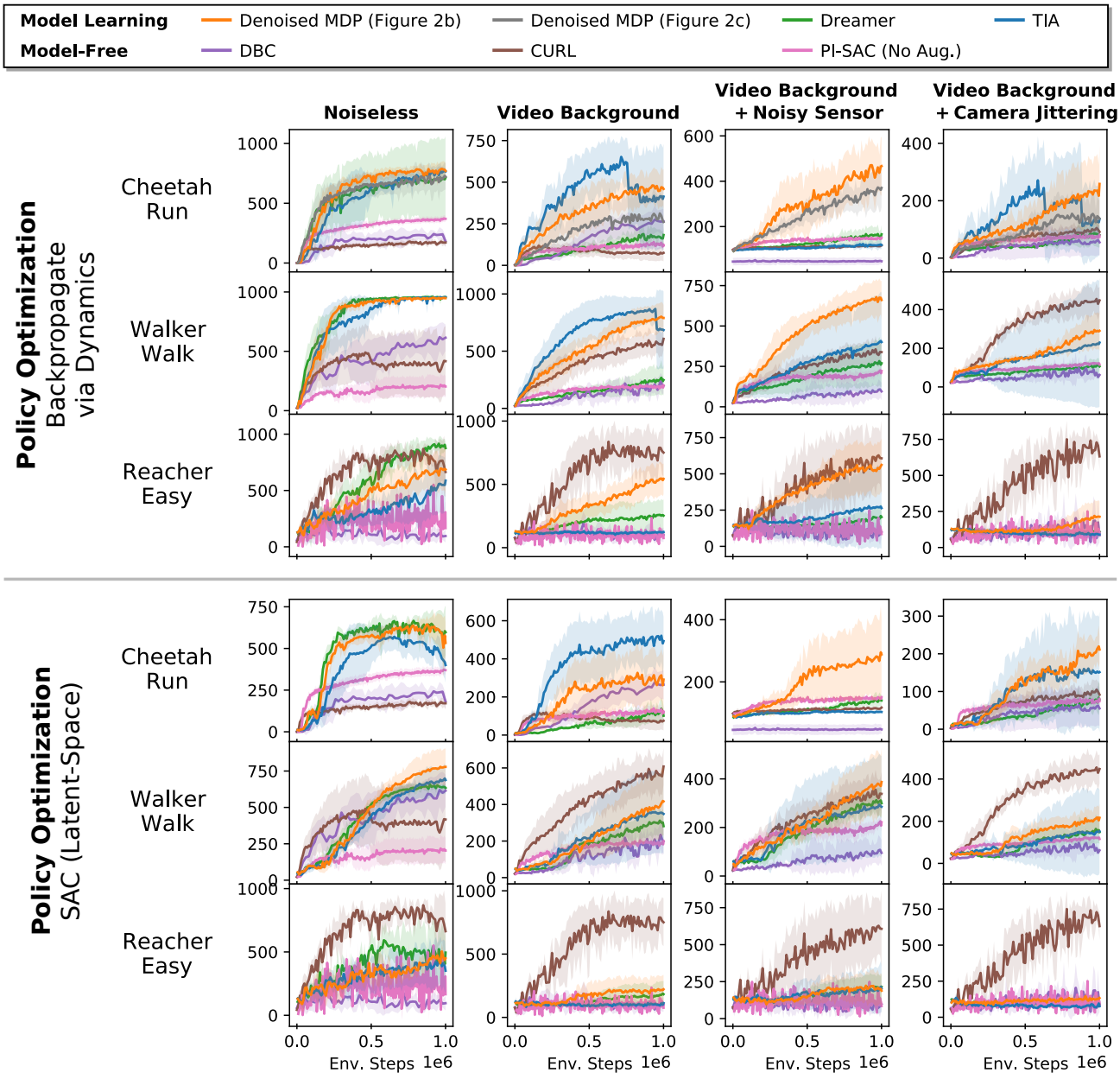


Figure 15: Policy optimization results on DMC. Each plot focuses on a single task variant, showing total episode return versus environment steps taken. For three model-based approaches, we use two policy optimization choices to train on the learned model: **(top half)** backpropagate via learned dynamics and **(bottom half)** SAC on the learned MDP. We also compare with DBC, a model-free baseline. For an "upper bound" (not plotted due to presentation clarity), SAC on true state-space (i.e., optimal representation) in 10^6 environment steps reaches episode return ≈ 800 on Cheetah Run variants, ≈ 980 on Walker Walk variants, and ≈ 960 on Reacher Easy variants. CURL's specific augmentation choice (random crop) potentially helps significantly for Reacher Easy (where the reacher and the target appear in random spatial locations) and **Camera Jittering**. However, unlike Denoised MDP, it does not generally perform well across all environments and noise variants.

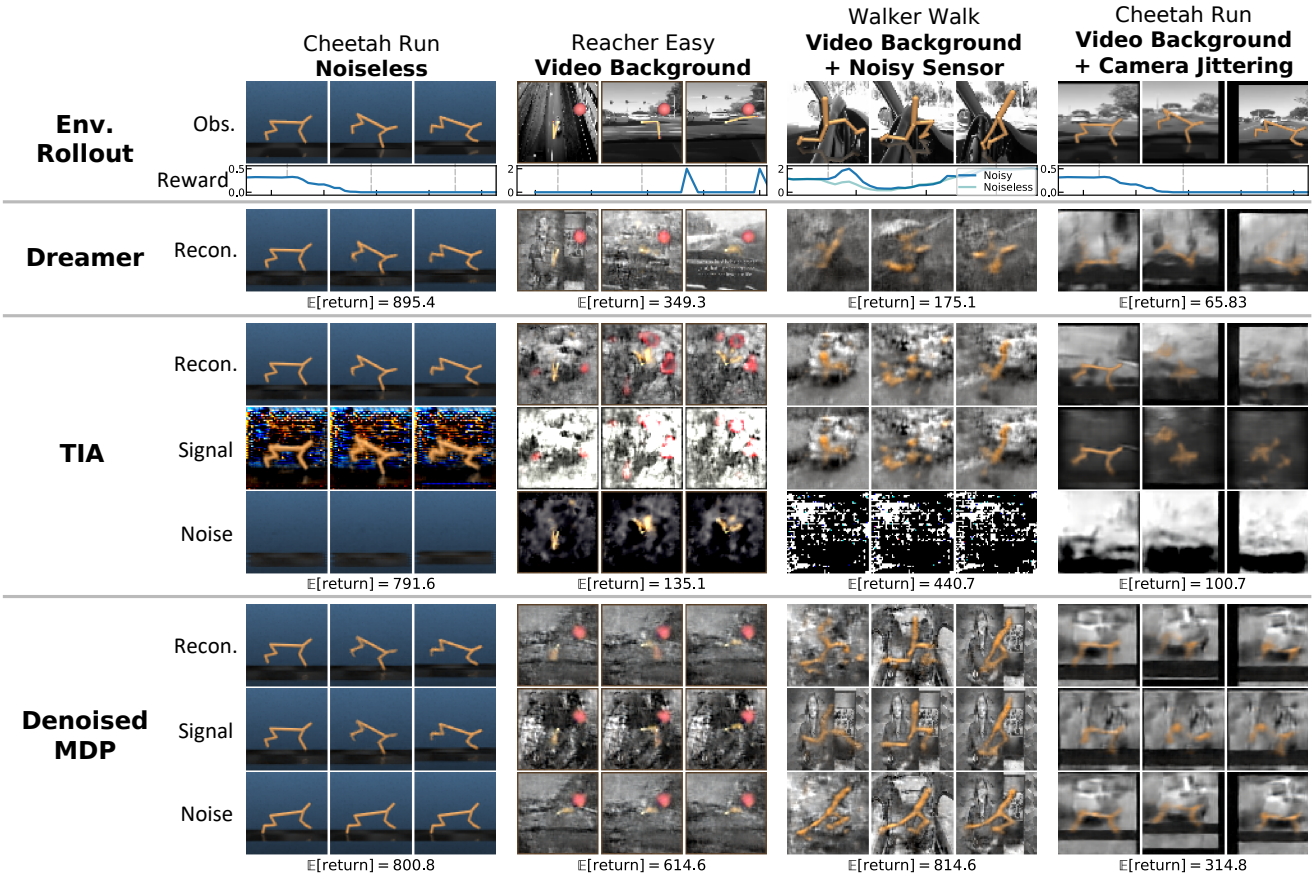


Figure 16: Complete visualization of the different DMC variants and factorizations learned by TIA and Denoised MDP. In addition to visualizations of Figure 7, we also visualize full reconstructions from Dreamer, TIA, and Denoised MDP.

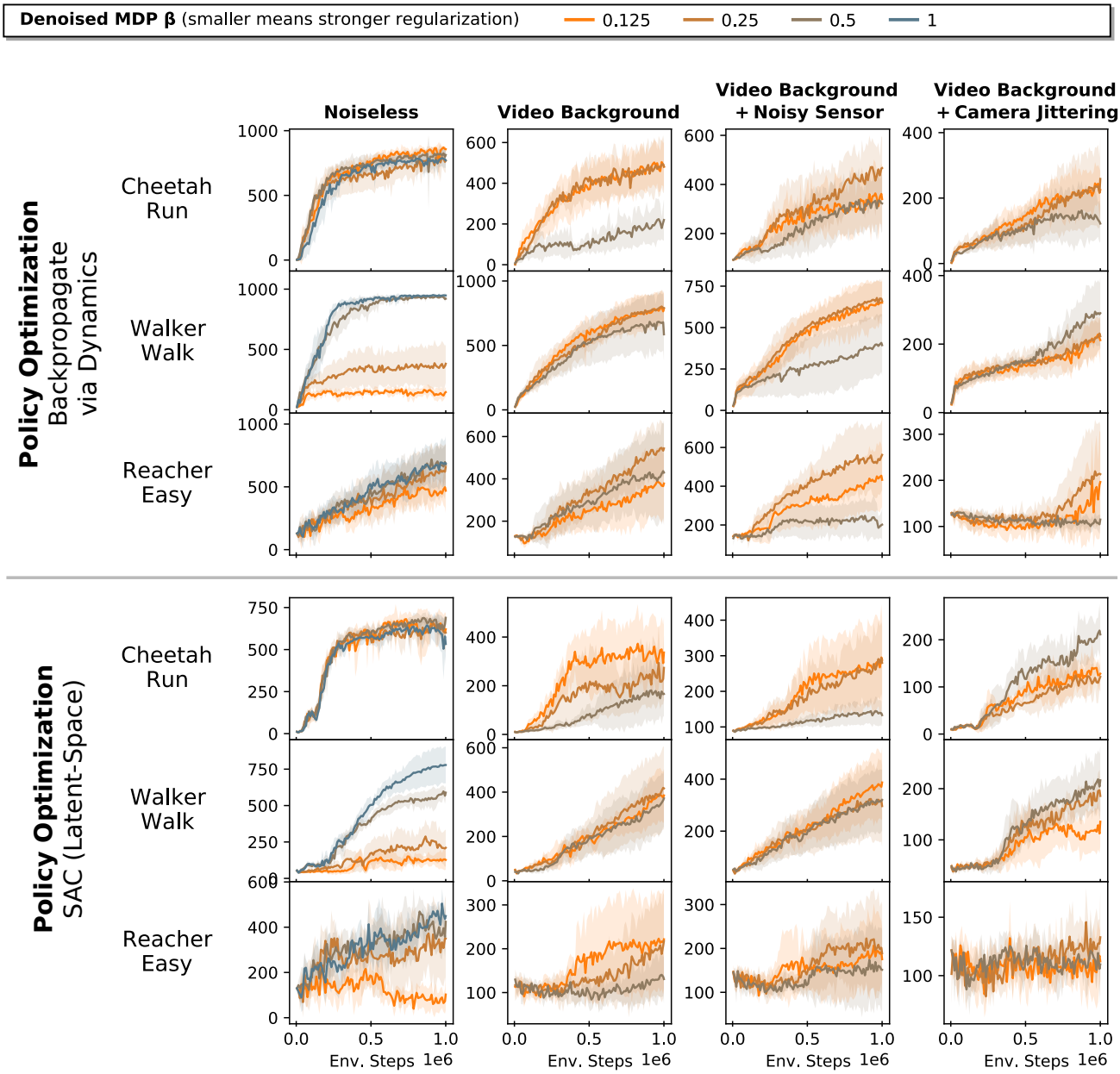


Figure 17: Effect of choosing β in Denoised MDP on DMC policy optimization results. Setting $\beta = 1$ disables regularization and is *only run on noiseless variants*.

		Noiseless	Video Background	Video Background + Noisy Sensor	Video Background + Camera Jittering
Policy Learning: Backprop via Dynamics	Cheetah Run	1	0.125	0.25	0.25
	Walker Walk	1	0.25	0.25	0.5
	Reacher Easy	1	0.25	0.25	0.25
Policy Learning: SAC (Latent-Space)	Cheetah Run	1	0.125	0.125	0.25
	Walker Walk	1	0.25	0.125	0.5
	Reacher Easy	1	0.125	0.25	0.25

Table 6: β choices for Denoised MDP results shown in Table 1 and Figure 15. We choose $\beta = 1$ (i.e., disabling regularization) for all noiseless environments, and tuned others. However, as seen in Figure 17, the results often are not too sensitive to small β changes.