DISCRETE CODEBOOK WORLD MODELS FOR CONTINUOUS CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

In reinforcement learning (RL), world models serve as internal simulators, enabling agents to predict environment dynamics and future outcomes in order to make informed decisions. While previous approaches leveraging discrete latent spaces, such as DreamerV3, have achieved strong performance in discrete action environments, they are typically outperformed in continuous control tasks by models with continuous latent spaces, like TD-MPC2. This paper explores the use of discrete latent spaces for continuous control with world models. Specifically, we demonstrate that quantized discrete codebook encodings are more effective representations for continuous control, compared to alternative encodings, such as one-hot and label-based encodings. Based on these insights, we introduce DCWM: **D**iscrete **C**odebook **W**orld **M**odel, a model-based RL method which surpasses recent state-of-the-art algorithms, including TD-MPC2 and DreamerV3, on continuous control benchmarks.

1 INTRODUCTION

In model-based reinforcement learning (RL), world models (Ha & Schmidhuber, 2018) have been introduced in order to simulate or predict the environment's dynamics in a data-driven way. An agent equipped with a world model can make predictions about its environment by "simulating" possible actions within the model and "imagining" the outcomes. This equips the agent with the ability to plan and anticipate outcomes given a (learned) reward function, and the additional ability to envision transitions and outcomes before taking them in the real world can in turn improve sample efficiency.

One of the state-of-the-art world models, DreamerV2/V3 (Hafner et al., 2022; 2023) achieves strong performance in a wide variety of tasks, by "imagining" sequences of future states within a world model and using them to improve their policies. Interestingly, DreamerV2/V3 introduced a discrete latent space, in the form of a one-hot encoding, which offered significant benefits over its predecessor, DreamerV1 (Hafner et al., 2019a). This suggests that discrete latent spaces may have benefits over continuous latent spaces. One possible reason is that turning continuous states into discrete latent representations enables the policy and value learning to harness the benefits of discrete variable processing for efficiency and interoperability. However, in the context of continuous control, TD-MPC2 (Hansen et al., 2023) uses a continuous latent space and significantly outperforms DreamerV3. Whilst there are multiple differences between TD-MPC2 and DreamerV2/V3, in this paper, we are specifically interested in exploring if discrete latent spaces can offer any benefits for continuous control.

Recently, Farebrother et al. (2024) showed that training value functions with classification may have benefits over training with regression. The benefits may arise because *(i)* classification considers uncertainty during training (via the cross-entropy loss), *(ii)* the categorical distribution is multi-modal so can consider multiple modes during training, or *(iii)* learning in discrete spaces is more efficient. In the context of world models, it is natural to ask, what benefits are obtained by *(i)* using discrete vs continuous latent spaces and *(ii)* modelling deterministic vs stochastic transition dynamics. Further to this, when considering stochastic latent transition dynamics, what is the effect of modelling *(i)* unimodal distributions (*e.g.* Gaussian in continuous latent spaces) vs *(ii)* multimodal distributions (*e.g.* categorical in discrete latent spaces). In this paper, we explore these ideas in the context of world models, *i.e.* does learning a discrete latent space using classification have benefits over learning a continuous latent space using regression.

Contributions The main contributions are as follows:

(C1) In the context of continuous control, we show that learning discrete latent spaces with classification does have benefits over learning continuous latent spaces with regression.
(C2) We show that formulating a discrete latent state using codebook encodings has benefits over alternatives, such as one-hot (like DreamerV2/V3) and label encodings.
(C3) Based on our insights, we introduce Discrete Codebook World Model (DCWM): a world model with a discrete latent space where each latent state is a discrete code from a codebook. DCWM obtains strong performance in the difficult locomotion tasks from DeepMind Control suite (Tassa et al., 2018) and the hard manipulation tasks from Meta-World (Yu et al., 2019).

2 RELATED WORK

In this section, we recap world models in the context of model-based RL. We introduce two competing methods for learning latent spaces *(i)* those using observation reconstruction and *(ii)* those using latent state consistency losses. We then compare methods that learn continuous latent spaces using regression and those that learn discrete latent spaces using classification.

World models Model-based RL is often said to be more sample-efficient than model-free methods. This is because it learns a model in which it can reason about the world, instead of simply trying to learn a policy or a value function to maximize the return (Ha & Schmidhuber, 2018). The world model can be used for planning (Allen & Koomen, 1983; Basye et al., 1992). A prominent idea has been to optimize the evidence lower bound of observation and reward sequences to learn world models that operate on the latent space of a learned Variational Autoencoder (VAE, Kingma & Welling (2014); Igl et al. (2018)). These models rely on maximizing the conditional observation likelihood $p(\boldsymbol{o}_t|\boldsymbol{z}_t)$, which is the reconstruction objective. The latent space of the model can then be used for both policy learning in the imagination of the world model, known as offline planning, *e.g.* Dreamer (Hafner et al., 2019a), or for decision-time planning with model-predictive control (Rubinstein, 1997; Hafner et al., 2019b) and exploration (Luck et al., 2019).

Latent-state consistency Using the reconstruction loss for learning latent state representations is unreliable (Lutter et al., 2021) and can have a detrimental effect on the performance of model-based methods in various benchmarks (Kostrikov et al., 2020; Yarats et al., 2021a). To this end, TD-MPC (Hansen et al., 2022) and its successor, TD-MPC2 (Hansen et al., 2023), use a consistency loss to learn representations for planning with Model Predictive Path Integral control together with reward and value functions learned through temporal difference methods (Williams et al., 2015). Given the success of learning representations without observation reconstruction in continuous control tasks, we predominantly focus on this class of methods, *i.e.* methods that use latent-state consistency losses.

Discrete latent spaces DreamerV1 (Hafner et al., 2019a), DreamerV2 (Hafner et al., 2022), and DreamerV3 (Hafner et al., 2023), are world model methods which learn policies using imagined transitions from their world models. They utilize observation reconstruction when learning their world models and perform well across a wide variety of tasks. However, they are significantly outperformed by TD-MPC2 in continuous control tasks, which does not reconstruct observations. Of particular interest in this paper, is that DreamerV2/V3 introduced a discrete latent space, in the form of a one-hot encoding, and trained it with a classification objective, significantly improving performance. In contrast, TD-MPC2 learns a continuous latent space with mean squared error regression. In this paper, we are interested in learning discrete latent spaces with classification, however, in contrast to DreamerV2/V3, we seek to avoid observation reconstruction and instead learn the latent space using a self-supervised latent-state consistency loss. That is, we investigate using a classification loss, *i.e.* the cross-entropy loss, for the self-supervised latent-state consistency loss.

3 PRELIMINARIES

In this section, we recap different types of discrete encodings and compare their pros and cons. First, let us assume we have three discrete categories: A, B, C, and D.

- **One-hot encoding** Given categories A, B, C, and D, a one-hot encoding would take the form $A = [1, 0, 0, 0]$, $B = [0, 1, 0, 0]$, $C = [0, 0, 1, 0]$ and $D = [0, 0, 0, 1]$, respectively.

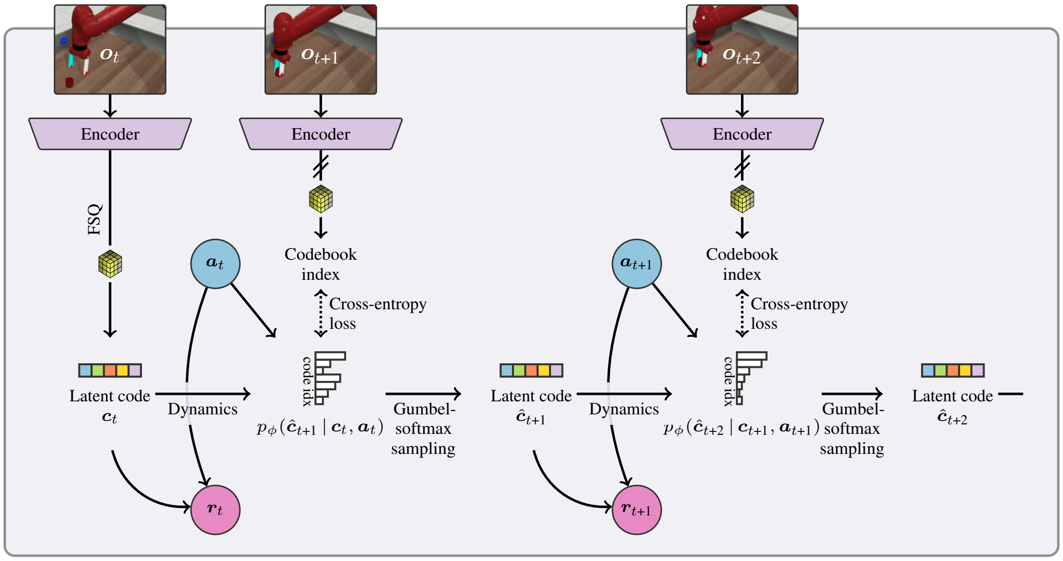


Figure 1: **World model training** DCWM is a world model with a discrete latent space where each latent state is a discrete code $\boldsymbol{c}$ (🟦🟩🟧🟨🟪) from a codebook $\mathcal{C}$. Observations $\boldsymbol{o}$ are first mapped through the encoder and then quantized (▨) into one of the discrete codes. We model probabilistic latent transition dynamics $p_\phi(\boldsymbol{c}' \mid \boldsymbol{c}, \boldsymbol{a})$ as a classifier such that it captures a potentially multimodal distribution over the next state $\boldsymbol{c}'$ given the previous state $\boldsymbol{c}$ and action $\boldsymbol{a}$. During training, multi-step predictions are made using Gumbel-softmax sampling such that gradients backpropagate to the encoder. Given this discrete formulation, we train the latent space using a classification objective, *i.e.* cross-entropy loss. Making the latent representation discrete with a codebook contributes to the very high sample efficiency of DCWM.

- **Label encoding** Given categories A, B, C, and D, label encoding would result in $A = 1$, $B = 2$, $C = 3$, and $D = 4$, respectively.
- **Codebook encoding** Given categories A, B, C, and D, a codebook might encode them as $A = [0.5, 0.5]$, $B = [0.5, -0.5]$, $C = [-0.5, 0.5]$, and $D = [-0.5, -0.5]$, respectively.

Ordinal relationships Mathematically, if we have an ordinal relationship $A < B < C$, label and codebook encodings can ensure $|e(A) - e(B)| < |e(A) - e(C)|$, where $e(x)$ is the encoding function. One-hot encoding, however, results in $|e(A) - e(B)| = |e(A) - e(C)| = \sqrt{2}$ for all distinct pairs, eliminating any notion of ordering. Whilst this may be beneficial in some scenarios, *e.g.*, when modelling distinct categories like fruits, it means that they cannot capture the inherent ordering in continuous data. In contrast, label and codebook encodings can capture ordinal relationships.

Sparsity Another downside of one-hot encodings is that they create sparse data (*i.e.*, data with many zero values), which can have a negative impact on neural network training. In contrast, label and codebook encodings create dense data (*i.e.* many non-zero values).

Dimensionality Finally, it is worth noting that one-hot encodings have high dimensionality, especially when there are many categories. This makes them memory-intensive and slow to train when using a large number of categories.

In this work, we show that discrete codebook encodings resulting from quantization (Mentzer et al., 2023) offer benefits over both one-hot and label encodings, when learning discrete latent spaces for continuous control. This is because they preserve ordinal relationships in multiple dimensions whilst being simpler, much lower-dimensional and having less memory requirements.

4 METHOD

In this section, we detail our method, named *Discrete Codebook World Model* (DCWM). DCWM is a model-based RL algorithm which *(i)* learns a world model with a discrete latent space and then, *(ii)* performs decision-time planning with model-predictive path integral control (MPPI). The paper's main contribution is formulating a discrete latent space using quantization, allowing us to train the latent representation using classification, in a self-supervised manner. See Fig. 1 for an overview of DCWM, Alg. 1 for details of world model training and Alg. 2 for details on the MPPI planning procedure.

We consider Markov Decision Processes (MDPs, Bellman (1957)) $\mathcal{M} = (\mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where agents receive observations $\boldsymbol{o}_t \in \mathcal{O}$ at time step t, perform actions $\boldsymbol{a}_t \in \mathcal{A}$, and then obtain the next observation $\boldsymbol{o}_{t+1} \sim \mathcal{P}(\cdot \mid \boldsymbol{o}_t, \boldsymbol{a}_t)$ and reward $r_t = \mathcal{R}(\boldsymbol{o}_t, \boldsymbol{a}_t)$. The discount factor is denoted $\gamma \in [0, 1)$.

4.1 WORLD MODEL

Learning world models with discrete latent spaces (*e.g.* DreamerV2) has proven powerful in a wide variety of domains. However, these approaches generally perform poorly in continuous control tasks when compared to algorithms like TD-MPC2 and TCRL (Zhao et al., 2023), which use continuous latent spaces. Rather than representing a discrete latent space using a one-hot encoding, as was done in DreamerV2, DCWM aims to construct a more expressive representation which is effective for continuous control. More specifically, DCWM represents discrete latent states as codes from a discrete codebook, obtained via finite scalar quantization (FSQ, Mentzer et al. (2023)). The world model can subsequently benefit from the advantages of discrete representations, *e.g.* efficiency and training with classification, whilst performing well in continuous control tasks.

Components DCWM's world model has six main components:

$$\text{Encoder:} \qquad \boldsymbol{x} = e_\theta(\boldsymbol{o}) \in \mathbb{R}^{|\mathcal{L}| \times d} \tag{1}$$

$$\text{Latent quantization:} \qquad \boldsymbol{c} = f(\boldsymbol{x}) \in \mathcal{C} \tag{2}$$

$$\text{Dynamics:} \qquad \boldsymbol{c}' \sim \text{Categorical}\left(p_1, \dots, p_{|\mathcal{C}|}\right) \quad \text{with } p_i = P_\phi(\boldsymbol{c}' = \boldsymbol{c}^i \mid \boldsymbol{c}, \boldsymbol{a}) \tag{3}$$

$$\text{Reward:} \qquad r = R_\xi(\boldsymbol{c}, \boldsymbol{a}) \in \mathbb{R} \tag{4}$$

$$\text{Value:} \qquad \boldsymbol{q} = \mathbf{q}_\psi(\boldsymbol{c}, \boldsymbol{a}) \tag{5}$$

$$\text{Policy prior:} \qquad \boldsymbol{a} = \pi_\eta(\boldsymbol{c}) \tag{6}$$

The encoder $e_\theta(\cdot)$ first maps observations $\boldsymbol{o}$ to continuous latent vectors $\boldsymbol{x} \in \mathbb{R}^{b \times d}$, where the number of channels b and the latent dimension d are hyperparameters. This continuous latent vector $\boldsymbol{x}$ is then quantized $f(\cdot)$ into one of the discrete latent codes $\boldsymbol{c} \in \mathcal{C}$ from the (fixed) codebook $\mathcal{C}$, using finite scalar quantization (FSQ, Mentzer et al. (2023)). As we have a discrete latent space, we formulate the transition dynamics to model the distribution over the next latent state $\boldsymbol{c}'$ given the previous latent state $\boldsymbol{c}$ and action $\boldsymbol{a}$. That is, we model stochastic transition dynamics in the latent space. We denote the probability of the next latent state $\boldsymbol{c}'$ taking the value of the i^{th} code $\boldsymbol{c}^i$ as $p_i = P_\phi(\boldsymbol{c}' = c^i \mid \boldsymbol{c}, \boldsymbol{a})$. This results in the next latent state following a categorical distribution $\boldsymbol{c}' \sim \text{Categorical}\left(p_1, \dots, p_{|\mathcal{C}|}\right)$. We use a standard classification setup, where we use an MLP to predict the logits $\boldsymbol{l} = \{l_1, \dots, l_{|\mathcal{C}|}\}$. Note that logits are the raw outputs from the final layer of the neural network (NN), which represent the unnormalized probabilities of the next latent state $\boldsymbol{c}'$ taking the value of each discrete code in the codebook $\mathcal{C}$. The logit for the i^{th} code is given by $l_i = d_{\phi,i}(\boldsymbol{c}, \boldsymbol{a}) \in \mathbb{R}$. We then apply the softmax operation to obtain the probabilities p_i of the next latent state taking each discrete code in the codebook $\mathcal{C}$, *i.e.*, $p_i = \text{softmax}_i(\boldsymbol{l})$. DCWM uses the discrete codes $\boldsymbol{c}$ as its latent state to make dynamics, reward $R_\xi(\boldsymbol{c}, \boldsymbol{a})$, value $\boldsymbol{q}_\psi(\boldsymbol{c}, \boldsymbol{a})$, and policy $\pi_\eta(\boldsymbol{c})$ predictions. Our hypothesis is that learning these components with a discrete latent space will be more efficient than with a continuous latent space.

Quantized latent space Unlike previous approaches, DCWM uses a discretized latent space where world states are encoded as discrete codes from a codebook $\mathcal{C}$. We use latent quantization (Mentzer et al., 2023) to enforce data compression and encourage organization (Hsu et al., 2023). However, we implement this using finite scalar quantization (FSQ, Mentzer et al. (2023)) instead of dictionary learning (van den Oord et al., 2017). As a result, our codebook is fixed and we obviate two codebook learning loss terms, which stabilizes early training. In this section, we will give an overview of our discretization method which utilizes codebooks. First, let us assume the output of the encoder is a matrix[1] $\mathbf{x} \in \mathbb{R}^{b \times d}$, with d dimensions and b as the number of channels.

Each latent dimension is quantized into a codebook $\mathcal{C}$. That is, we have d independent codebooks, one for each latent dimension. Our first step is to define the size of the codebook for each dimension,

---

[1] For simplicity, we omit here the batch dimension.

i.e. to define the ordered set of quantization levels $\mathcal{L} = \{L_1, L_2, \cdots, L_b\}$. Each quantization level L_i corresponds to the i-th channel, *e.g.* L_1 defines the number of discrete values in the first channel, L_2 for the second and so on. In short, a quantization level of *e.g.* $L_i = 11$ would mean that we discretize each dimension in the i-th channel into 11 distinct values/symbols. We use integers as symbols, which would mean that the code for dimension d in channel i would be a symbol from the set $\{-5, -4, \cdots, 0, \cdots, 4, 5\}$. In practice, for fast conversion from continuous values to codes we use a similar discretization scheme as FSQ and apply the function

$$f : \boldsymbol{x}, \mathcal{L}, i \to \text{round}\left(\left\lfloor \frac{L_i}{2} \right\rfloor \cdot \tanh(\boldsymbol{x}_{i,:})\right), \tag{7}$$

to each channel, taking the output $\boldsymbol{x}$ of the encoder and the channel quantization level L_i. This approach results in a codebook with $|\mathcal{C}| = \prod_{i=1}^{b} L_i$ unique codes for each dimension d, each code being made of b symbols, *i.e.* an b-dimensional vector.

Intuitively, this results in a Voronoi partition of the b-dimensional space[2] in each dimension d, where any point in space is assigned to one of the equidistantly placed centroids via Eq. (7). A visualization of this can be found in Fig. 2. In effect, this leads to an efficient and fast discretization of the latent embedding space.

In practice, Eq. (7) is not differentiable. To solve this for training and implementing this discretization approach in our world model using standard deep learning libraries, we use the straight-through gradient estimation (STE) approach with $\text{round\_ste}(\boldsymbol{x}) : x \to x + \text{sg}(\text{round}(\boldsymbol{x}) - \boldsymbol{x})$, where the function $\text{sg}(\cdot)$ stops the gradient flow. Furthermore, we normalize codes to be in the range $[-1, 1]$ after the discretization step for improved performance as improved performance was reported in Mentzer et al. (2023). The hyperparameters of this approach are the number of channels b and the number of code symbols per channel (i.e. quantization levels) L_i. In our experiments, we found two channels and the quantization levels $\mathcal{L} = \{5, 3\}$ to be sufficient.

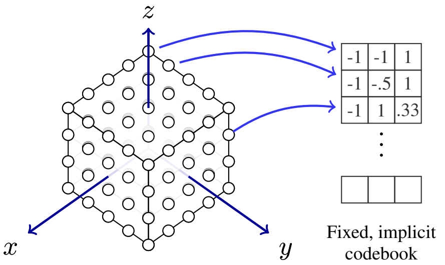


Figure 2: **Illustration of Codebook** ($\mathcal{C}$) FSQ's codebook can be considered a b-dimensional hypercube (left). This figure illustrates an b=3-dimensional codebook, where each axis of the 3-dimensional hypercube (left) corresponds to one dimension of the codebook (right). The i^{th} dimension of the hypercube is discretized into L_i values, *e.g.*, the x and y-axis are discretized into $L_0 = L_1 = 5$ and the z-axis into $L_3 = 4$. Code symbols (here integers) are normalized to the range [-1,1].

It is worth highlighting that our codebook encoding preserves ordinal relationships between observations. This contrasts one-hot encodings which were used by DreamerV2 (Hafner et al., 2022). See Sec. 3 for a comparison of the different discrete encodings.

World model training We train our world model components e_θ, d_ϕ, R_ξ jointly to minimize the following objective

$$\mathcal{L}(\theta, \phi, \xi; \mathcal{D}) = \mathbb{E}_{(\boldsymbol{o},\boldsymbol{a},\boldsymbol{o}',r)_{0:H} \sim \mathcal{D}} \left[\sum_{h=0}^{H-1} \left(\gamma^h \text{CE}(\underbrace{p_\phi(\hat{\boldsymbol{c}}_{h+1} \mid \hat{\boldsymbol{c}}_h, \boldsymbol{a}_h), \boldsymbol{c}_{h+1}}_{\text{Latent-state consistency}}) + \underbrace{\| R_\xi(\boldsymbol{c}_h, \boldsymbol{a}_h) - r_h \|_2^2}_{\text{Reward prediction}} \right) \right] \tag{8}$$

$$\text{with } \underbrace{\hat{\boldsymbol{c}}_0 = f(e_\theta(\boldsymbol{o}_0))}_{\text{First latent state}} \quad \underbrace{\hat{\boldsymbol{c}}_{h+1} \sim p_\phi(\hat{\boldsymbol{c}}_{h+1} \mid \hat{\boldsymbol{c}}_h, \boldsymbol{a}_h)}_{\text{Stochastic dynamics}} \quad \underbrace{\boldsymbol{c}_h = \text{sg}(f(e_\theta(\boldsymbol{o}_h)))}_{\text{Latent code}}, \tag{9}$$

where H denotes the multi-step prediction horizon and γ is the discount factor. The first predicted latent code $\hat{\boldsymbol{c}}_0$ is obtained by passing the observation $\boldsymbol{o}_0$ through the encoder and then quantizing the output. At subsequent time steps, the dynamics model predicts the probability mass function over the next latent code $p_\phi(\hat{\boldsymbol{c}}_{h+1} \mid \hat{\boldsymbol{c}}_h, \boldsymbol{a}_h)$. Given this probabilistic dynamics model, we must consider how to make H-step predictions in the latent space. In practice, we propagate uncertainty by sampling and we use the Gumbel-Softmax trick so that gradients backpropagate through our samples to the encoder.

---

[2] Remember that b is the number of channels of $\boldsymbol{x}$.

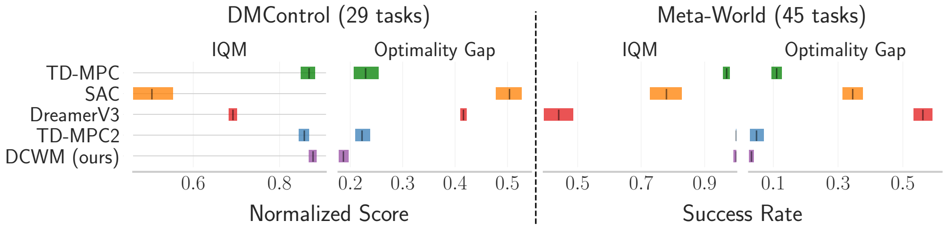


Figure 3: **DMControl (left) & Meta-World manipulation (right) results** DCWM outperforms TD-MPC2 and DreamerV3 in DMControl tasks across all metrics. This is due to DCWM's strong performance in the hard Dog and Humanoid tasks. DCWM performs well in Meta-World, generally matching TD-MPC2, whilst significantly outperforming DreamerV3 and SAC. Results are for 1M environment steps and error bars represent 95% stratified bootstrap confidence intervals.

Note that gradients must flow back to the encoder at the first time step when it was used to obtain the first latent code $\hat{\boldsymbol{c}}_0$, as the target codes $\boldsymbol{c}$ are obtained by passing the next observation $\boldsymbol{o}'$ through the encoder and using the stop gradient operator sg. We then train our dynamics "classifier" using the cross-entropy (CE) loss. Finally, we note that our reward model R_ξ is trained jointly with the encoder e_θ and dynamics model p_ϕ to ensure that the world model can accurately predict rewards in the latent space.

Policy and value learning We learn the policy $\pi_\eta(\boldsymbol{c})$ and action-value functions $\boldsymbol{q}_\psi(\boldsymbol{c}, \boldsymbol{a})$ in the latent space using the actor-critic RL method TD3 (Fujimoto et al., 2018). However, we follow Yarats et al. (2021b); Zhao et al. (2023) and augment the loss with N-step returns. The main difference to TD3 is that instead of using the original observations $\boldsymbol{o}$, we map them through the encoder $\boldsymbol{c} = f(e_\theta(\boldsymbol{o}))$ and learn the actor/critic in the discrete latent space $\boldsymbol{c}$. We also reduce bias in the TD target by following REDQ (Chen et al., 2021) and learning an ensemble of $N_q = 5$ critics, as was done in TD-MPC2. When calculating the TD target we randomly subsample two of the critics and use the minimum of these two. Let us denote the indices of the two randomly subsampled critics as $\mathcal{M}$. The critic is then updated by minimizing the following objective:

$$\mathcal{L}_q(\psi; \mathcal{D}) = \mathbb{E}_{(\boldsymbol{o},\boldsymbol{a},\boldsymbol{o}',r)_{n=1}^{N} \sim \mathcal{D}} \left[\frac{1}{N_q} \sum_{k=1}^{N_q} (q_{\psi_k}(f(e_\theta(\boldsymbol{o}_t)), \boldsymbol{a}_t) - y)^2 \right], \tag{10}$$

$$y = \sum_{n=0}^{N-1} \gamma^n r_{t+n} + \gamma^N \min_{k \in \mathcal{M}} q_{\bar{\psi}_k}(f(e_\theta(\boldsymbol{o}_{t+n+1})), \boldsymbol{a}_{t+n+1}), \quad \text{with } \boldsymbol{a}_{t+n} = \pi_{\bar{\eta}}(\boldsymbol{c}_{t+n}) + \epsilon_{t+n},$$

where we use policy smoothing by adding clipped Gaussian noise $\epsilon_{t+n} \sim \text{clip}\left(\mathcal{N}(0, \sigma^2), -c, c\right)$ to the action $\boldsymbol{a}_{t+n} = \pi_{\bar{\eta}}(\boldsymbol{c}_{t+n}) + \epsilon_{t+n}$. We then use the target action-value functions $\boldsymbol{q}_{\bar{\psi}}$ and the target policy $\pi_{\bar{\eta}}$ to calculate the TD target y. Note that the target networks use an exponential moving average, *i.e.* $[\bar{\psi}, \bar{\eta}] \leftarrow (1 - \tau)[\bar{\psi}, \bar{\eta}] + [\psi, \eta]$. We follow REDQ and learn the actor's by minimizing

$$\mathcal{L}_\pi(\eta; \mathcal{D}) = -\mathbb{E}_{\boldsymbol{o}_t \sim \mathcal{D}} \left[\frac{1}{N_q} \sum_{k=1}^{N_q} q_{\psi_k}(\underbrace{f(e_\theta(\boldsymbol{o}_t))}_{\boldsymbol{c}_t}, \pi_\eta(\underbrace{f(e_\theta(\boldsymbol{o}_t))}_{\boldsymbol{c}_t})) \right]. \tag{11}$$

That is, we train the actor to maximize the average action value over the ensemble of critics.

Summary Whilst this world model resembles TD-MPC2, there are some important distinctions. First, the latent space is represented as a discrete codebook which enables DCWM to train the dynamics model using the cross-entropy loss. Importantly, the cross-entropy loss considers a (potentially multimodal) distribution over the predicted latent codes during training. In contrast, TD-MPC2 considers deterministic dynamics and uses mean squared error regression. Interestingly, our experiments suggest that our stochastic dynamics model and classification objective combination offers benefits in deterministic environments. Second, DCWM does not use value prediction when training the encoder. Instead, we follow the insight from Zhao et al. (2023) that value prediction is not necessary for obtaining a good latent representation and instead, train the action-value function separately.

Importantly, our discrete latent space is parameterized as a set of discrete codes from a codebook. This contrasts alternative discrete encodings such as one-hot encodings, which were used in DreamerV2/V3, label encodings, and binary encodings. We hypothesize that this will offer significant improvements when representing continuous state vectors in a discrete space.

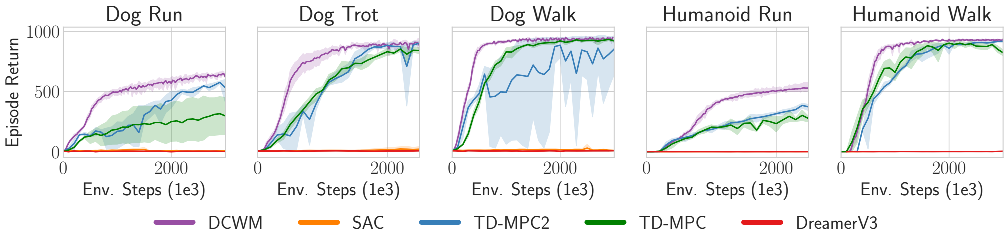


Figure 4: **High-dimensional locomotion** DCWM (purple) significantly outperforms TD-MPC2 (blue) and DreamerV3 (red) in the complex, high-dimensional locomotion tasks from DMControl.

4.2 DECISION-TIME PLANNING

DCWM follows TD-MPC2 and leverages the world model for decision-time planning. It uses MPC to obtain a closed-loop controller and uses model-predictive path integral control (MPPI) (Williams et al., 2015) as the underlying trajectory optimization algorithm. MPPI is a sampling-based trajectory optimization method which does not require any gradients. See Alg. 2 for full algorithm details. At each environment step, we estimate the parameters $\boldsymbol{\mu}^*_{0:H}, \boldsymbol{\sigma}^*_{0:H}$ of a diagonal multivariate Gaussian over a H-step action sequence that maximizes the following objective

$$\boldsymbol{\mu}^*_{0:H}, \boldsymbol{\sigma}^*_{0:H} = \operatorname*{arg\,max}_{\boldsymbol{\mu}_{0:H}, \boldsymbol{\sigma}_{0:H}} \mathbb{E}_{\boldsymbol{a}_{0:H} \sim \mathcal{N}(\boldsymbol{\mu}_{0:H}, \mathrm{diag}(\boldsymbol{\sigma}^2_{0:H}))} \left[J_{\text{MPPI}}(\boldsymbol{a}_{0:H}, \boldsymbol{o}) \right] \tag{12a}$$

$$J_{\text{MPPI}}(\boldsymbol{a}_{0:H}, \boldsymbol{o}) = \sum_{h=0}^{H-1} \gamma^h R_\xi(\boldsymbol{c}_h, \boldsymbol{a}_h) + \gamma^H \frac{1}{N_q} \sum_{k=1}^{N_q} q_{\psi_k}(\boldsymbol{c}_H, \boldsymbol{a}_H) \tag{12b}$$

$$\text{s.t.} \quad \boldsymbol{c}_0 = f(e_\theta(\boldsymbol{o})) \quad \text{and} \quad \boldsymbol{c}_{>0} = \sum_{i=1}^{|\mathcal{C}|} \Pr(\boldsymbol{c}_{h+1} = \boldsymbol{c}^i \mid \boldsymbol{c}_h, \boldsymbol{a}_h) \boldsymbol{c}^i, \tag{12c}$$

where H is the planning horizon and γ is a discount factor. MPPI solves Eq. (12) in an iterative manner. It starts by sampling candidate action sequences and evaluating them using the objective $J_{\text{MPPI}}(\boldsymbol{a}_{0:H}, \boldsymbol{o})$. It then refits the sampling distribution's parameters $\boldsymbol{\mu}_{0:H}$, $\boldsymbol{\sigma}^2_{0:H}$ based on a weighted average. After several iterations, we select the mean of the first action $\boldsymbol{\mu}_0$ and apply it in the environment. Note that during training we promote exploration by adding Gaussian noise. Importantly, Eq. (12) uses the action-value function $\boldsymbol{q}_\psi(\boldsymbol{c}, \boldsymbol{a})$ to bootstrap the planning horizon such that it estimates the full RL objective. DCWM follows TD-MPC2 and warm starts the planning process with a fraction of action sequences originating from the prior policy π_η, and we warm start by initializing $\boldsymbol{\mu}_{0:H}$, $\boldsymbol{\sigma}^2_{0:H}$ as the solution to the previous time step. See Alg. 2 for further details.

Note that at planning time, we do not sample from the transition dynamics $p(\boldsymbol{c}_{h+1} \mid \boldsymbol{c}_h, \boldsymbol{a}_h)$ because this would introduce unwanted stochasticity into the planning procedure. Instead, we take the expected code, which is a weighted sum over the codes in the codebook. Whilst the expected value of a discrete variable does not necessarily take a valid discrete value, we find it effective in our setting. We hypothesize that this is because our discrete codes have an ordering such that expected values simply interpolate between the codes in the codebook.

5 EXPERIMENTS

In this section, we experimentally evaluate DCWM in a variety of continuous control tasks from the DeepMind Control (DMC) Suite (Tassa et al., 2018) and Meta-World (Yu et al., 2019) against a number of baselines and ablations. Our experiments seek to answer the following research questions:

RQ1 How does DCWM compare to state-of-the-art model-based RL algorithms leveraging latent state embeddings, especially in the hard DMC and Meta-World tasks?
RQ2 Does DCWM's discrete latent space offer benefits over a continuous latent space?
RQ3 Does DCWM's codebook offer benefits for dynamics/value/policy learning over alternative discrete encodings such as *(i)* one-hot encoding (similar to DreamerV2) and *(ii)* label encoding?
RQ4 What is important for learning a latent space *(i)* classification loss, *(ii)* discrete codebook, *(iii)* stochastic dynamics or *(iv)* multimodal dynamics?

Experimental Setup We used the standard RL tasks from the DeepMind Control Suite (DMControl) (Tassa et al., 2018) and Meta-World (Yu et al., 2019). We compared our proposed approach, DCWM, against two state-of-the-art model-based RL baselines, namely DreamerV3 (Hafner et al., 2023) which utilizes a discrete one-hot encoding as its latent state and TD-MPC2 (Hansen et al., 2023) using a continuous latent space. We also compare against soft actor-critic (SAC) (Haarnoja et al., 2018) and TD-MPC (Hansen et al., 2022). Our proposed approach utilized a latent space with $d = 512$ dimensions and $b = 2$ channels, with 15 code symbols per dimension by using $\mathcal{L} = \{L_1 = 5, L_2 = 3\}$.

5.1 PERFORMANCE OF DCWM

We compare our proposed approach, DCWM, using a discrete codebook encoding latent space against SOTA baselines using one-hot encoding and continuous latent spaces in the world model in a range of tasks from the DeepMind Control Suite, manipulation tasks in Meta-World, and musculoskeletal tasks from MyoSuite. In Figs. 3, 8 and 9, we compare the aggregate performance of DCWM against TD-MPC, TD-MPC2, DreamerV3, and SAC, in 29 DMControl tasks 45 Meta-World tasks, and 5 MyoSuite respectively, with 3 seeds per task. Many tasks in DeepMind Control Suite are particularly high-dimensional. For instance, the observation space of the Dog tasks is $\mathcal{O} \in \mathbb{R}^{223}$ and the action space is $\mathcal{A} \in \mathbb{R}^{38}$, and for Humanoid, the observation space is $\mathcal{O} \in \mathbb{R}^{67}$ and the action space $\mathcal{A} \in \mathbb{R}^{24}$. Fig. 4 shows that DCWM excels in the high dimensional Dog and Humanoid environments when compared to the baselines. We hypothesize that our discretized representations are particularly beneficial for simplifying learning the transition dynamics in high-dimensional spaces, making DCWM highly sample efficient in these tasks.

Similarly, we can find that DCWM outperforms DreamerV3 in simulated manipulation tasks in the Meta-World task suite (Figs. 9 and 14). We also see that DCWM either outperforms or matches the performance of TD-MPC2. Comparing the results at a global level (Fig. 7), we can find that our proposed method performs well across both benchmarks.

5.2 COMPARISON OF DIFFERENT LATENT SPACES

We now evaluate how different latent dynamics formulations affect the performance. We seek to answer the following: *(i)* do discrete latent spaces offer benefits over continuous latent spaces? *(ii)* does training with classification (cross-entropy) offer benefits over mean squared error regression? and *(iii)* does modelling stochastic (and potentially multimodal) transition dynamics offer benefits?

In our experiments, we consider both continuous and discrete latent spaces to investigate the impact of discretizing the latent space of the world model. In Figs. 5 and 12, the experiments with discrete latent spaces are labelled with “Discrete” (red, green, and purple) whilst continuous latent spaces are labelled “Continuous” (orange). We also evaluate DCWM using the simplical normalization used in TD-MPC2 – which bounds latent space – labelled “SimNorm” (blue) . Experiments labelled with “MSE” were trained with mean squared error regression whilst those labelled “CE” were trained with the cross-entropy classification loss. The experiment labelled “Discrete+CE+det” used FSQ to get a discrete latent space and trained with the cross-entropy loss, where the logits were obtained as the MSE between the dynamics prediction and each code in the codebook. This experiment enabled us to test if DCWM’s performance boost resulted from training with the cross-entropy loss or from making the dynamics stochastic. In Fig. 12, experiments labelled with “log-lik.” were trained by maximizing the log-likelihood, *i.e.* cross-entropy for “FSQ-log-lik.” (purple), Gaussian log prob. for “Gaussian+log-lik.” (blue), and Gaussian mixture model log prob. for “GMM+log-lik.” (green).

Discrete vs continuous latent spaces The experiments using discrete latent spaces (red and purple) significantly outperform the ones with continuous latent spaces in terms of sample efficiency. This suggests that our discrete codebook encoding offers significant benefits over continuous latent spaces.

Classification vs regression Interestingly, training a deterministic discrete latent space using MSE regression (red) never performs as well as training a stochastic discrete latent space using classification (purple). However, our experiment with the deterministic discrete latent space using classification (green) confirms that the benefit arises from the stochasticity of our latent space. This suggests that using Gumbel-softmax sampling when making multi-step dynamics during training boosts performance. Our results extending TD-MPC2 to use DCWM’s discrete stochastic latent space in Fig. 21 support this conclusion.

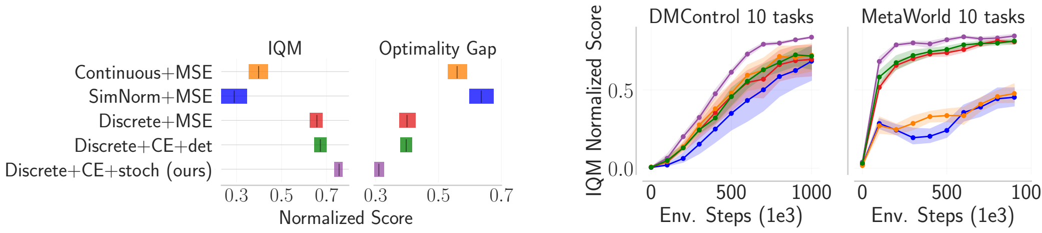


Figure 5: **Latent space comparison** Evaluation of *(i)* discrete (Discrete) vs continuous (Continuous) latent spaces, *(ii)* using cross-entropy (CE) vs mean squared error (MSE) for the latent-state consistency loss, and *(iii)* formulating a deterministic (det) vs stochastic (stoch) dynamics model. Discretizing the latent space (red) improves sample efficiency over the continuous latent space (orange) and formulating a stochastic dynamics model and training this with cross-entropy (purple) improves performance further. Aggregate metrics are for 3 seeds in 10 DMControl and 10 Meta-World tasks.

Deterministic vs Stochastic Given that modelling a stochastic latent space and training with maximum log-likelihood is beneficial for discrete latent spaces, we now test if this holds in continuous latent spaces. To this end, we formulate two stochastic, continuous latent spaces. The first models a unimodal Gaussian distribution (blue) whilst the second models a multimodal Gaussian mixture model (GMM) (green). Interestingly, these stochastic transition models sometimes increase sample efficiency on DMControl tasks when compared to their deterministic counterpart (orange). However, they drastically underperform on Meta-World tasks.

Our method (purple) has a discrete latent space, is trained by maximum log-likelihood (*i.e.* cross-entropy), and models a (potentially multimodal) distribution over the latent transition dynamics during training. These factors, combined with using Gumbel-softmax sampling when making multi-step dynamics predictions, offer improved sample efficiency over continuous latent spaces.

5.3 IMPACT OF LATENT SPACE ENCODING

In this work, we leveraged quantization so that our discrete encoding took the form of code vectors $\boldsymbol{c}$ from a discrete codebook $\mathcal{C}$. Our world model consists of NNs for the dynamics $P_{\phi}(\mathbf{c}' \mid \mathbf{c}, \mathbf{a})$, reward $R_{\xi}(\mathbf{c}, \mathbf{a})$, critic $Q_{\psi}(\mathbf{c}, \mathbf{a})$, and prior policy $\pi_{\eta}(\mathbf{c})$, which all make predictions given the discrete codebook encoding $\mathbf{c} = \mathbf{e}_{\text{codes}}$. We compare DCWM's codebook encoding $\boldsymbol{e}_{\text{codes}} = \boldsymbol{c}^{i} \in \mathcal{C}$ to *(i)* label encoding $e_{\text{labels}} = i \in \{1, \dots |\mathcal{C}|\}$ and *(ii)* one-hot encoding $\boldsymbol{e}_{\text{one-hot}} = \boldsymbol{v} \in \{0,1\}^{|\mathcal{C}|}$ given $\sum_{i=1}^{|\mathcal{C}|} v_i = 1$. In Fig. 6, we evaluate what happens when we replace the codebook encoding $\mathbf{c}$ with either one-hot $\mathbf{e}_{\text{one-hot}}$ or label $\mathbf{e}_{\text{label}}$ encodings. In these experiments, we did not modify the dynamics $P_{\phi}(\mathbf{c}' \mid \mathbf{c}, \mathbf{a})$, that is, the dynamics continued to make predictions using the codebook encoding $\mathbf{c}$ and did not use the one-hot or label encodings. This is because we found that using one-hot and label encodings for the dynamics led to the agent being unable to learn. This suggests that our codebook encoding is needed in our world model setup. Nevertheless, we evaluated the performance when changing the encoding for the other components. We evaluated the following experiment configurations:

1. **Codes (ours)**: All components used codes: $P_{\phi}(\mathbf{c}' \mid \mathbf{c}, \mathbf{a})$, reward $R_{\xi}(\mathbf{c}, \mathbf{a})$, critic $Q_{\psi}(\mathbf{c}, \mathbf{a})$ and prior policy $\pi_{\eta}(\mathbf{c})$.
2. **Labels**: Dynamics model used codes $P_{\phi}(\mathbf{c}' \mid \mathbf{c}, \mathbf{a})$ whilst reward $R_{\xi}(\mathbf{e}_{\text{labels}}, \mathbf{a})$, critic $Q_{\psi}(\mathbf{e}_{\text{labels}}, \mathbf{a})$ and prior policy $\pi_{\eta}(\mathbf{e}_{\text{labels}})$ used labels $\mathbf{e}_{\text{labels}}$ obtained from the code's index i in the codebook.
3. **One-hot**: Dynamics model used codes $P_{\phi}(\mathbf{c}' \mid \mathbf{c}, \mathbf{a})$ whilst reward $R_{\xi}(\mathbf{e}_{\text{one-hot}}, \mathbf{a})$, critic $Q_{\psi}(\mathbf{e}_{\text{one-hot}}, \mathbf{a})$ and prior policy $\pi_{\eta}(\mathbf{e}_{\text{one-hot}})$, used one-hot $\mathbf{e}_{\text{one-hot}}$ obtained by applying PyTorch's torch.nn.functional.one_hot to the label encoding.

The label encoding (blue) struggles to learn in the Humanoid Walk task and is often less sample efficient than the alternative encodings. This is likely because the label encoding is not expressive enough to model the multi-dimensional ordinal structure of our codebook. Let us provide intuition via a simple example. Our codebook has $b = 2$ channels, so two different codes may take the form $e_{\text{codes}}(A) = [0.5, -0.5]$ and $e_{\textbf{codes}}(B) = [0, 0.5]$. As a result, our codebook encoding can model ordinal structure in both of its channels, *i.e.*, $e_{\text{codes}}(A)_1 > e_{\text{codes}}(B)_1$ whilst $e_{\text{codes}}(A)_2 < e_{\text{codes}}(B)_2$.

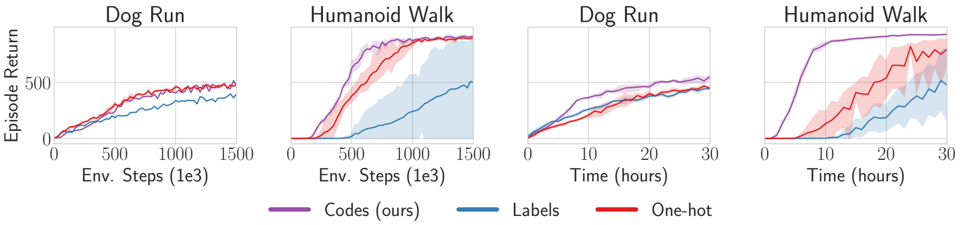


Figure 6: **Discrete encodings comparison** DCWM with its discrete codebook encoding (purple) outperforms using DCWM with one-hot encoding (red) and label encoding (blue), in terms of both sample efficiency (left) and computational efficiency (right). Dynamics model used codes $P_\phi(\mathbf{c}' \mid \mathbf{c}, \mathbf{a})$ whilst reward $R_\xi(\mathbf{e}, \mathbf{a})$, critic $Q_\psi(\mathbf{e}, \mathbf{a})$ and prior policy $\pi_\eta(\mathbf{e})$ used the respective encoding $\mathbf{e}$.

The corresponding label encoding would encode this as $e_{\text{labels}}(A) = 1$ and $e_{\text{labels}}(B) = 2$, which incorrectly implies that $B > A$. In short, the label encoding cannot model the multi-dimensional ordinal structure of the codebook $\mathcal{C}$. In contrast, the one-hot encoding (red) matches the codebook encoding in terms of sample efficiency in all tasks except Humanoid Walk. However, the one-hot encoding introduces an extremely large input dimension for the reward, value and policy networks, and this significantly slows down training. See Sec. 3 for further details on why this is the case.

5.4 ABLATION OF CODEBOOK SIZE $|\mathcal{C}|$ AND LATENT DIMENSION d

As a final set of experiments, we evaluate how the size of the codebooks $|\mathcal{C}|$ and the number of latent dimensions d influences training. We indirectly configure different codebook sizes via the FSQ levels $\mathcal{L} = \{L_1, \dots, L_b\}$ hyperparameter. This is because the codebook size is given by $|\mathcal{C}| = \prod_{i=1}^{b} L_i$. The top row of Fig. 10 compares the training curves for different codebook sizes. The algorithm's performance is not particularly sensitive to the codebook size. However, a codebook that is too large can result in slower learning. Note that DCWM has similar runtime to TD-MPC2 when using our default hyperparameters. The best codebook size varies between environments. Given that a codebook has a particular size, we can gain insights into how quickly DCWM's encoder starts to activate all of the codebook. The connection between the codebook size and the activeness of the codebook is intuitive: the bottom row of Fig. 10 shows that the smaller the codebook, the larger the active proportion. To gain further insights, we evaluate how the dimension of the latent space d impacts DCWM's performance. We find that DCWM is fairly robust to different latent dimensions. We find that a latent dimension of $d = 1024$ with FSQ levels $\mathcal{L} = [5, 3]$, which corresponds to a codebook size $|\mathcal{C}| = 5 \times 3 = 15 \approx 2^4$, performs best in the harder DMC tasks. See App. B.3 for more details.

6 CONCLUSION

We have presented DCWM, a world model that learns a discrete latent space using codebook encodings and a cross-entropy based self-supervised loss for model-based RL. DCWM demonstrates strong performance in continuous control tasks, including Meta-World and the complex DMC Humanoid and Dog tasks, where it exceeds or matches the performance of SOTA baselines. In our experiments, we demonstrated the benefit of a discrete latent space with codebook encodings over a standard continuous latent embedding or classical discrete spaces such as labels and one-hot encoding. These findings open up a new interesting avenue for future research into discrete embeddings for world models.

Limitations and Future Work We have demonstrated the benefit of discretizing the latent space and training with the cross-entropy loss in a self-supervised manner. In order to do this, we had to model stochastic latent transition dynamics using a classification setup. However, we have only evaluated DCWM in deterministic environments. Therefore, extending DCWM to stochastic environments is an interesting direction for future work. Also, as we have modelled the aleatoric uncertainty associated with the latent transition dynamics, a natural extension is to consider modelling the epistemic uncertainty, arising from learning from limited data. This could equip DCWM with a more principled exploration mechanism like Chua et al. (2018); Scannell et al. (2024); Daxberger et al. (2021); Scannell et al. (2023). Finally, we have not explored the scaling capabilities of our method. Given that training with classification unlocks scalability an exciting direction for future work is to test how well DCWM scales to larger NN architectures.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320. Curran Associates, Inc., 2021.

James F Allen and Johannes A Koomen. Planning using a temporal world model. In *Proceedings of the Eighth international joint conference on Artificial intelligence-Volume 2*, pp. 741–747, 1983.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization, July 2016.

Kenneth Basye, Thomas Dean, Jak Kirman, and Moises Lejter. A decision-theoretic approach to planning, perception, and control. *IEEE Expert*, 7(4):58–65, 1992.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957. ISSN 0095-9057.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. In *International Conference on Learning Representations*, 2021. Comment: Published as a conference paper at ICLR 2021.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace Redux - Effortless Bayesian Deep Learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20089–20103. Curran Associates, Inc., 2021.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop Regressing: Training Value Functions via Classification for Scalable Deep RL, March 2024.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1587–1596. PMLR, July 2018.

David Ha and Jürgen Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 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*, pp. 1861–1870. PMLR, July 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

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*, pp. 2555–2565. PMLR, May 2019b.

Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*, February 2022.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, Robust World Models for Continuous Control. In *The Twelfth International Conference on Learning Representations*, October 2023.

Nicklas A. Hansen, Hao Su, and Xiaolong Wang. Temporal Difference Learning for Model Predictive Control. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 8387–8406. PMLR, June 2022.

Kyle Hsu, William Dorrell, James Whittington, Jiajun Wu, and Chelsea Finn. Disentanglement via Latent Quantization. *Advances in Neural Information Processing Systems*, 36:45463–45488, December 2023.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pp. 2117–2126. PMLR, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Diederik P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *ICLR*, 2014.

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

Kevin Sebastian Luck, Mel Vecerik, Simon Stepputtis, Heni Ben Amor, and Jonathan Scholz. Improved exploration through latent trajectory optimization in deep deterministic policy gradient. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3704–3711. IEEE, 2019.

Michael Lutter, Leonard Hasenclever, Arunkumar Byravan, Gabriel Dulac-Arnold, Piotr Trochim, Nicolas Heess, Josh Merel, and Yuval Tassa. Learning dynamics models for model predictive agents. *arXiv preprint arXiv:2109.14311*, 2021.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite Scalar Quantization: VQ-VAE Made Simple, September 2023.

Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

Aidan Scannell, Riccardo Mereu, Paul Chang, Ella Tami, Joni Pajarinen, and Arno Solin. Sparse function-space representation of neural networks. In *ICML Workshop on Duality Principles for Modern Machine Learning*, 2023.

Aidan Scannell, Riccardo Mereu, Paul Edmund Chang, Ella Tamir, Joni Pajarinen, and Arno Solin. Function-space parameterization of neural networks for sequential learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=2dhxxIKhqz`.

R.S. Sutton and A.G. Barto. *Reinforcement Learning, Second Edition: An Introduction*. Adaptive Computation and Machine Learning Series. MIT Press, 2018. ISBN 978-0-262-35270-3.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural Discrete Representation Learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021a.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning. In *International Conference on Learning Representations*, October 2021b.

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`.

Yi Zhao, Wenshuai Zhao, Rinu Boney, Juho Kannala, and Joni Pajarinen. Simplified Temporal Consistency Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 42227–42246. PMLR, July 2023.

APPENDICES

This appendix is organized as follows. In App. A we provide further details on our method. App. B provides further experimental results, including an ablation of the codebook size in App. B.2, further details on the latent space ablation in App. B.4, full DeepMind control suite results in App. B.5, Meta-World results in App. B.6, a comparison of DCWM using VQ instead of FSQ in App. B.7, and a comparison of DCWM's ensemble critic approach vs the standard double Q approach in App. B.8. In App. C, we provide further implementation details, including default hyperparameters, hardware, *etc.* In App. D, we provide further details of the baselines and in App. E we detail the different DeepMind control and Meta-World tasks used throughout the paper.

A METHOD DETAILS 15
A.1 ALGORITHMS 15

B FURTHER RESULTS 16
B.1 AVERAGE DMCONTROL AND META-WORLD METRICS 16
B.2 ABLATION OF CODEBOOK SIZE 18
B.3 ABLATION OF LATENT DIMENSION d 19
B.4 ABLATION OF LATENT SPACE 20
B.5 DEEPMIND CONTROL RESULTS 21
B.6 META-WORLD MANIPULATION RESULTS 22
B.7 ABLATION OF FSQ VS VQ-VAE 23
B.8 ABLATION OF REDQ CRITIC VS STANDARD DOUBLE Q APPROACH 24
B.9 RECONSTRUCTION LOSS HAS A DETRIMENTAL IMPACT 25
B.10 MYOSUITE MUSCULOSKELETAL RESULTS 26
B.11 IMPROVING TD-MPC2 WITH DCWM 27

C IMPLEMENTATION DETAILS 28

D BASELINES 30

E TASKS 31

–appendices continue on next page–

A METHOD DETAILS

A.1 ALGORITHMS

Alg. 1 outlines DCWM's world model training procedure. Alg. 2 outlines how we perform trajectory optimization using MPPI (Williams et al., 2015), closely following the formulation of MPPI by Hansen et al. (2022), with two key modifications. First, during rollout evaluation, we update our latent states by taking the expected code as a weighted sum over the codes in the codebook, rather than sampling from the transition dynamics $p(\boldsymbol{c}_{h+1}|\boldsymbol{c}_h, \boldsymbol{c}_a)$. This approach reduces the variance in state transitions, resulting in more stable trajectory evaluations. Second, while the MPPI standard deviation $(\boldsymbol{\sigma}^2_{0:H})^j$ is used for action sampling during the optimization process, the final returned action is computed by adding noise, sampled from a separate noise schedule, to the MPPI mean $\boldsymbol{\mu}^j_0$ for the first step. This method, inspired by TD3 (Fujimoto et al., 2018), strikes a better balance between exploration and exploitation, leading to more stable training performance.

Algorithm 1 DCWM's world model training

```
Input: Encoder $e_\theta$, dynamics $d_\phi$, reward $R_\xi$, critics $\{q_{\psi_i}\}_{i=1}^{N_q}$, policy $\pi_\eta$, learning rate $\alpha$, target network update rate $\tau$
for $i$ to $N_{\text{episodes}}$ do
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1}, r_t\}_{t=0}^{T}$    ▷ Collect data in environment
    for $i = 1$ to $T$ do
        $[\theta, \phi, \xi] \leftarrow [\theta, \phi, \xi] + \alpha \nabla \left( \mathcal{L}(\theta, \phi, \xi; \mathcal{D}) \right)$    ▷ Update world model, Eq. (8)
        $\psi \leftarrow \psi + \alpha \nabla \left( \mathcal{L}_q(\psi; \mathcal{D}) \right)$    ▷ Update critic, Eq. (10)
        if $i$ % 2 == 0 then
            $\eta \leftarrow \eta + \alpha \nabla \left( \mathcal{L}_\pi(\eta; \mathcal{D}) \right)$    ▷ Update actor less frequently than critic, Eq. (11)
        end if
        $[\bar{\psi}, \bar{\eta}] \leftarrow (1 - \tau)[\bar{\psi}, \bar{\eta}] + \tau[\psi, \eta]$    ▷ Update target networks
    end for
end for
```

Algorithm 2 DCWM's MPPI

```
Input: current observation $\boldsymbol{o}$, planning horizon $H$, iterations $J$, population size $N_p$, prior population size $N_\pi$, number of elites $K$,
$\boldsymbol{c}_0 \leftarrow e_\theta(\boldsymbol{o})$    ▷ Encode state into discrete code
Initialize $\boldsymbol{\mu}^0_{0:H}$, $(\boldsymbol{\sigma}^2_{0:H})^0$ with the solution from the last time step shifted by one.
for each iteration $j = 1, \dots, J$ do
    Sample $N_p$ action trajectories of length $H$ from $\{a_h \sim \mathcal{N}(\boldsymbol{\mu}^{j-1}_h, (\boldsymbol{\sigma}^2_h)^{j-1})\}_{h=0}^{H}$    ▷ Sample action candidates
    Sample $N_\pi$ action trajectories of length $H$ using $\pi$ and $d_\phi$    ▷ Prior policy samples
    for all $N_p + N_\pi$ action sequences $\tau_i = (\boldsymbol{a}_0, \dots, \boldsymbol{a}_H)$ do    ▷ Trajectory evaluation
        $\Phi_i \leftarrow 0$
        for step $h = 0, \dots, H-1$ do
            $\Phi_i \leftarrow \Phi_i + R_\xi(\boldsymbol{c}_h, \boldsymbol{a}_h)$    ▷ Compute immediate reward
            $\hat{\boldsymbol{c}}_{h+1} = \sum_{i=1}^{|\mathcal{C}|} \Pr(\hat{\boldsymbol{c}}_{h+1} = \boldsymbol{c}^i \mid \hat{\boldsymbol{c}}_h, \boldsymbol{a}_h) \boldsymbol{c}^i$    ▷ Compute next state
        end for
        $\Phi_i \leftarrow \Phi_i + \gamma^H \frac{1}{N_q} \sum_{k=1}^{N_q} q_{\psi_k}(\boldsymbol{c}_H, \boldsymbol{a}_H)$    ▷ Bootstrap with ensemble of Q-functions
    end for
    $\boldsymbol{\mu}^j_{0:H}, (\boldsymbol{\sigma}^2_{0:H})^j = \text{fit}(\{\boldsymbol{a}_{0:H}\}_0^{N_p+N_\pi}, \Phi)$    ▷ Fit updated action distribution using top-$K$ samples
end for
return $\boldsymbol{\mu}^j_0 + \epsilon$   with   $\epsilon \sim \mathcal{N}(0, \sigma^2_{\text{noise}})$    ▷ Final output with exploration noise
```

–appendices continue on next page–

B FURTHER RESULTS

In this section, we include further ablations and results. In Figs. 3, 8 and 9, we compare the aggregate performance of DCWM against TD-MPC, TD-MPC2, DreamerV3, and SAC, in 29 DMControl tasks, 45 Meta-World tasks, and 5 MyoSuite tasks respectively, with 3 seeds per task. Following Agarwal et al. (2021), we report the median, interquartile mean (IQM), mean, and optimality gap at 1M environment steps, with error bars representing 95% stratified bootstrap confidence intervals. For DMControl, we use min-max normalization as the maximum possible return in an episode is 1000 whilst the minimum is 0, *i.e.* Normalized Return $=$ Return$/(1000-0)$. For Meta-World, we report the success rate which does not require normalization as it is already between 0 and 1.

In Figs. 5 and 21 we report aggregate metrics over 10 DMControl and 10 Meta-World tasks. The tasks are as follows:

- **DMControl 10**: Acrobot Swingup, Dog Run, Dog Walk, Dog Stand, Dog Trot, Humanoid Stand, Humanoid Walk, Humanoid Run, Reacher Hard, Walker Walk.
- **Meta-World 10**: Button Press, Door Open, Drawer Close, Drawer Open, Peg Insert Side, Pick Place, Push, Reach, Window Open, Window Close.

B.1 AVERAGE DMCONTROL AND META-WORLD METRICS

In Fig. 7, we report the average performance at 1M environment steps.

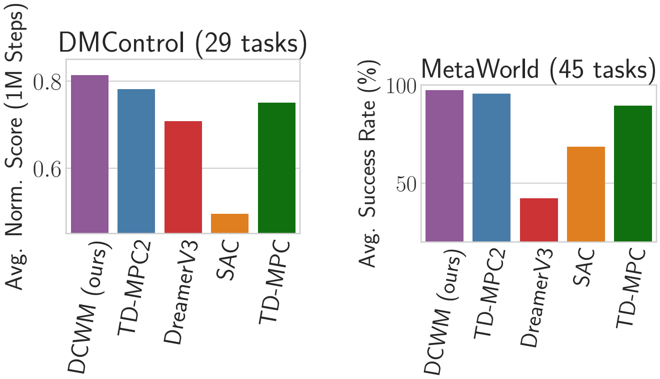


Figure 7: **Overview** We evaluate DCWM (purple) in 29 DMControl and 45 Meta-World tasks and report aggregated metrics at 1M environment steps. See Fig. 13 for all DMControl results and Fig. 14 for all Meta-World results.

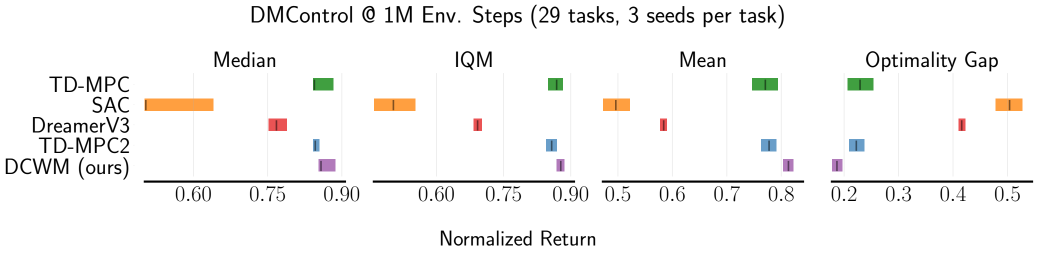


Figure 8: **DMControl results** DCWM outperforms TD-MPC2 and DreamerV3 in DMControl tasks across all metrics. This is due to DCWM's strong performance in the hard Dog and Humanoid tasks. Error bars represent 95% stratified bootstrap confidence intervals.

–appendices continue on next page–

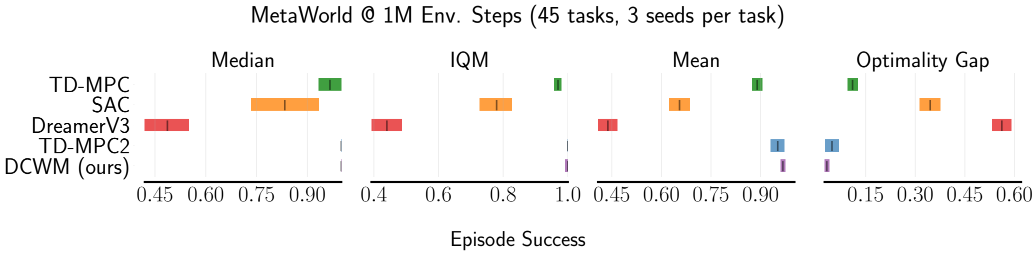


Figure 9: **Meta-World results** DCWM performs well in Meta-World, generally matching TD-MPC2, whilst significantly outperforming DreamerV3 and SAC. Error bars represent 95% stratified bootstrap confidence intervals.

B.2 ABLATION OF CODEBOOK SIZE

In this section, we evaluate how the size of the codebook $|\mathcal{C}|$ influences training. We indirectly configure different codebook sizes via the FSQ levels $\mathcal{L} = \{L_1, \dots, L_b\}$ hyperparameter. This is because the codebook size is given by $|\mathcal{C}| = \prod_{i=1}^{b} L_i$. The top row of Fig. 10 compares the training curves for different codebook sizes. The algorithm's performance is not particularly sensitive to the codebook size. A codebook that is too large can result in slower learning. The best codebook size varies between environments.

Given that a codebook has a particular size, we can gain insights into how quickly DCWM's encoder starts to activate all of the codebook. The connection between the codebook size and the activeness of the codebook is intuitive: the bottom row of Fig. 10 shows that the smaller the codebook, the larger the active proportion.

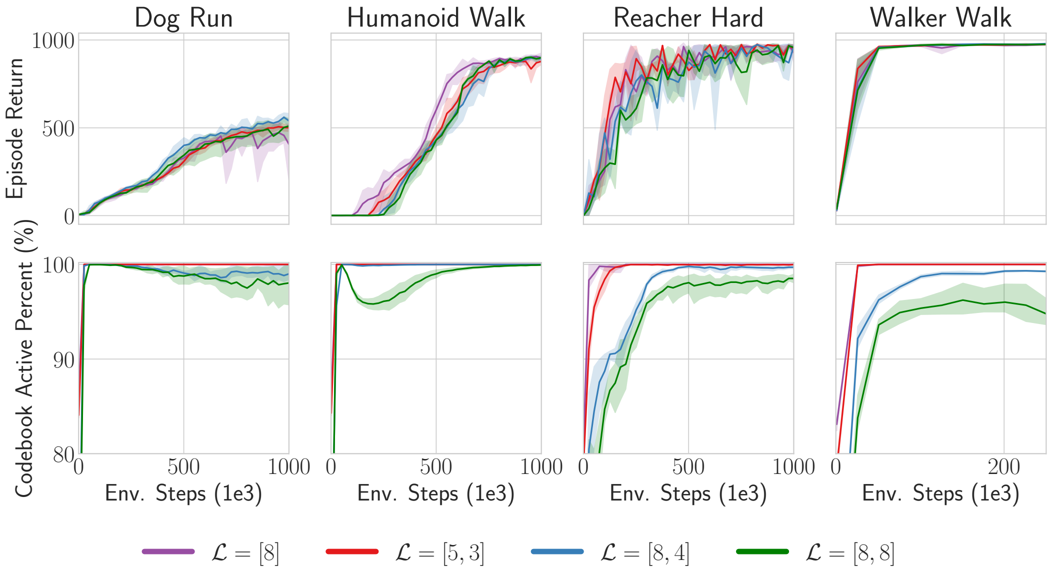


Figure 10: **Codebook size ablation** We compare how the codebook size affects the performance of DCWM (top), the percentage of the codebook that is active during training (middle), and how the different codebook sizes affect the encoder's ability to preserve the rank of the representation (bottom). In general, smaller codebooks become fully active faster than larger codebooks, and the rank of the representation is maintained for all codebook sizes. We plot the mean and the 95% confidence intervals (shaded) across 3 random seeds for all environments.

–appendices continue on next page–

B.3 ABLATION OF LATENT DIMENSION d

This section investigates how the latent dimension d affects the behavior and performance of DCWM in four different environments. In the top row of Fig. 11, we see that the performance of our algorithm is robust to the latent dimension d, although a latent dimension too small can result in inferior performance, especially in the more difficult environments. The bottom row of Fig. 11 demonstrates that DCWM learns to use the complete codebook irrespective of the latent dimension.

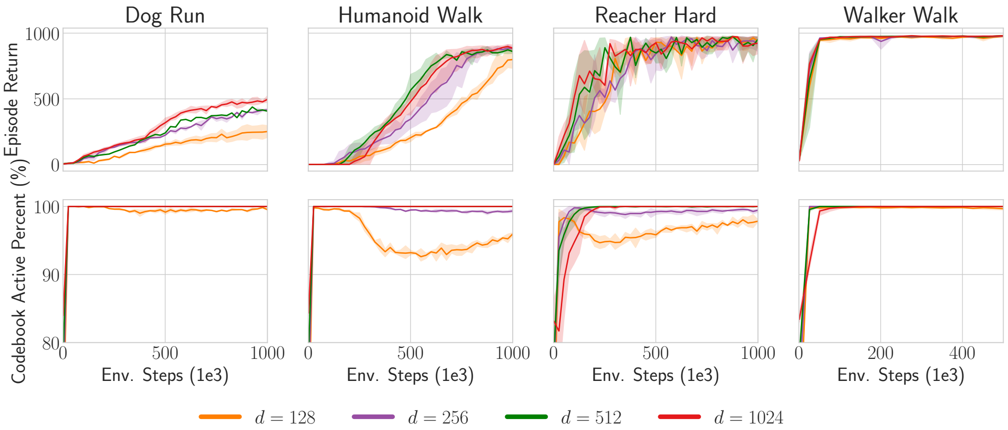


Figure 11: **Latent dim d ablation.** We compare how the latent dimension d affects the performance of DCWM (top) and the percentage of the codebook that is active during training (bottom). In general, our algorithm is robust to the latent dimension of the representation, although in more difficult environments, such as Humanoid Walk, a d too small can harm the agent's performance. We plot the mean and the 95% confidence intervals (shaded) across 3 random seeds for all environments.

–appendices continue on next page–

B.4 ABLATION OF LATENT SPACE

In this section, we provide further details on the comparison of different latent spaces experiments in Sec. 5.2. To validate our method, we test the importance of quantizing the latent space and training the world model with classification instead of regression. In Fig. 12, we compare DCWM to world models with different latent spaces formulations, which we now detail.

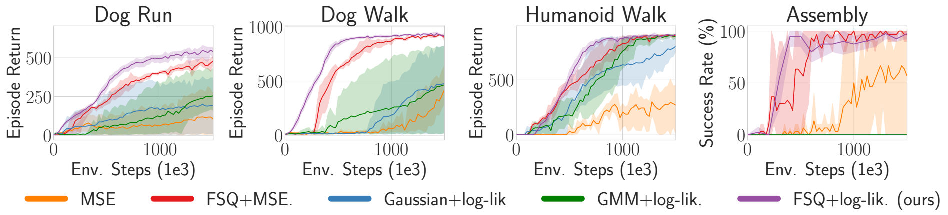


Figure 12: **Latent space comparison** Comparison of different latent space formulations. Continuous and deterministic latent space trained with MSE regression (orange), deterministic and discrete trained with MSE (red), continuous and unimodal Gaussian latent space trained with maximum log-likelihood (blue), continuous and multimodal GMM trained with maximum log-likelihood (green), and discrete trained with classification (purple). Discretizing the latent space with FSQ (red) improves sample efficiency and training this with classification (purple) improves performance further.

MSE (orange) First, we consider a continuous latent space with deterministic transition dynamics trained by minimizing the mean squared error between predicted next latent states and target next latent states.

FSQ+MSE (red) Next, we consider quantization of the latent space and training based on mean squared error regression. This experiment allows us to analyze the importance of quantization.

Gaussian+log-lik. (blue) To consider stochastic continuous dynamics, we configure the transition dynamics to model a Gaussian distribution over predictions of the next state. During training, we sample from the Gaussian distribution using the reparameterization trick. The world model is then trained to maximize the log-likelihood of the next latent state targets. This allows us to investigate if modelling stochastic transition dynamics offers benefits when using continuous latent spaces.

GMM+log-lik. (green) To consider continuous multimodal transitions, we consider a Gaussian mixture with three components. During training, we sample a Gaussian from the mixture with the Gumbel-softmax trick and then we sample from the selected Gaussian using the reparameterization trick. The world model is then trained to maximize the log-likelihood of next latent state targets.

–appendices continue on next page–

B.5 DEEPMIND CONTROL RESULTS

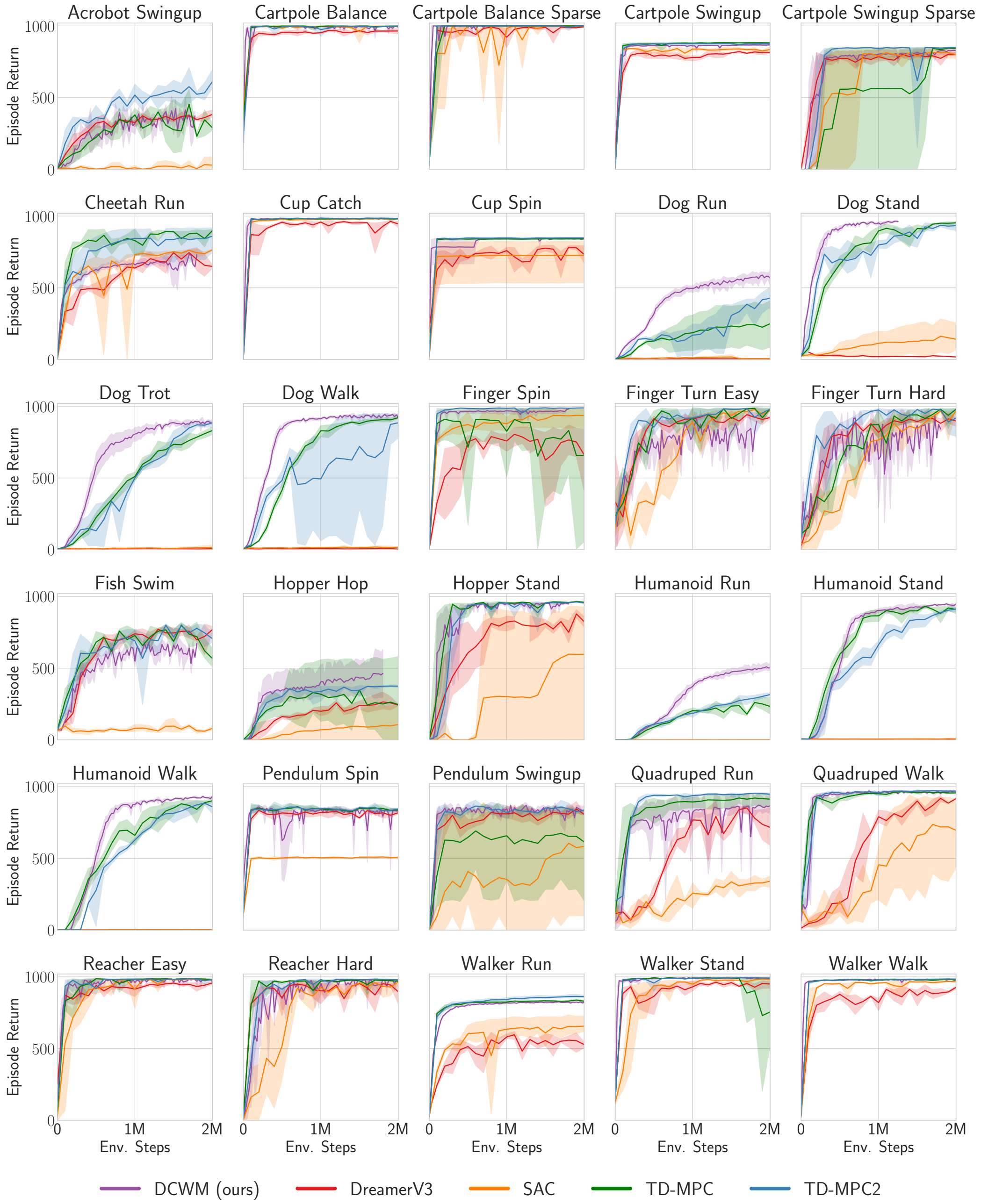


Figure 13: **DeepMind Control results.** DCWM performs well across a variety of DMC tasks. We plot the mean (solid line) and the 95% confidence intervals (shaded) across 5 seeds (DCWM) or 3 seeds (TD-MPC2/TD-MPC/DreamerV3/SAC), where each seed averages over 10 evaluation episodes.

–appendices continue on next page–

B.6 META-WORLD MANIPULATION RESULTS

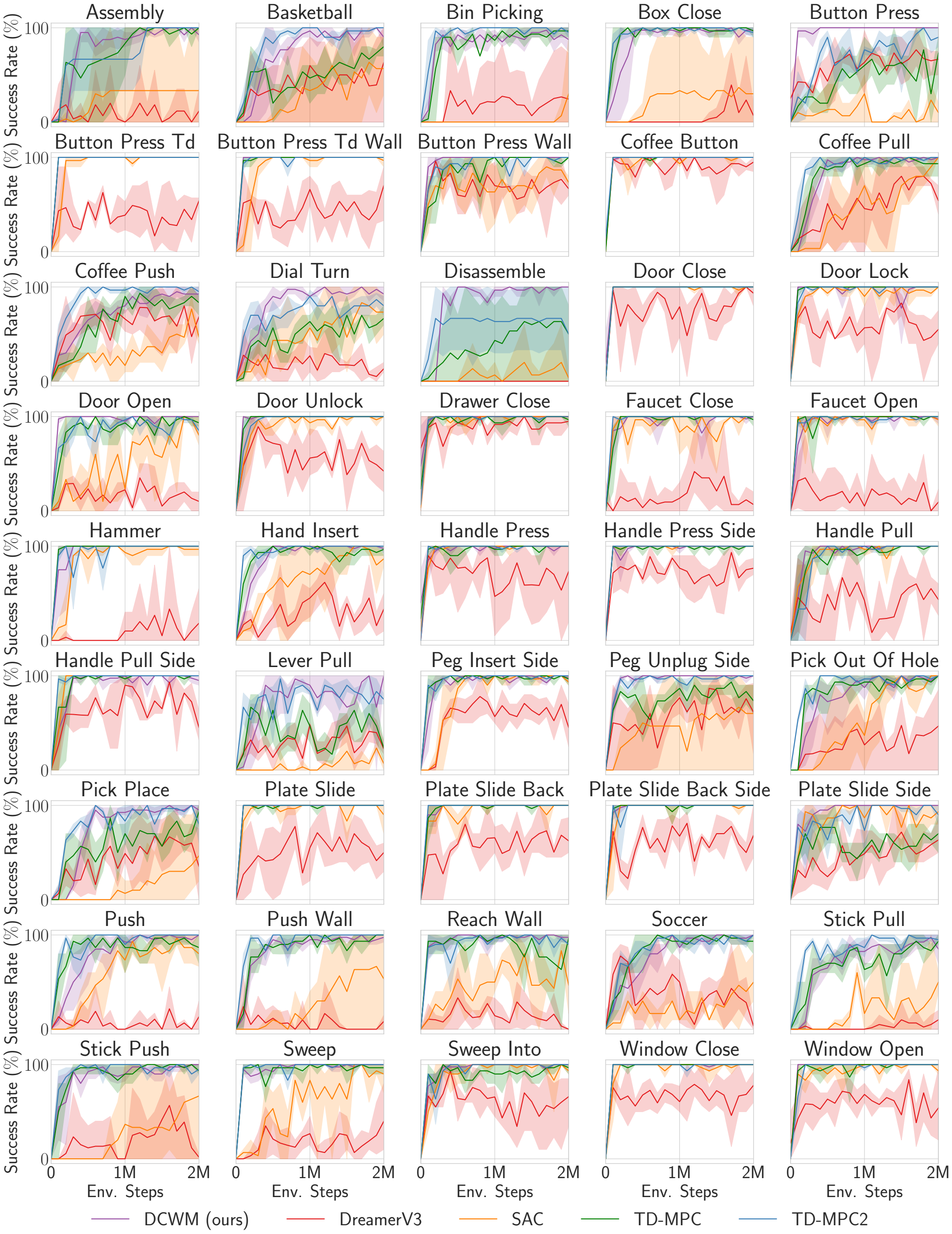


Figure 14: **Meta-World manipulation results.** DCWM performs well across Meta-World tasks. We plot the mean (solid line) and the 95% confidence intervals (shaded) across 5 seeds (DCWM) or 3 seeds (TD-MPC2/TD-MPC/DreamerV3/SAC), where each seed averages over 10 evaluation episodes.

–appendices continue on next page–

B.7 ABLATION OF FSQ VS VQ-VAE

To understand how the choice of using FSQ for discretization contributes to the performance of our algorithm, we tried replacing the FSQ layer with a standard Vector Quantization layer. We evaluated the methods in Walker Walk, Dog Run, Humanoid Walk, and Reacher Hard. We used standard hyperparameters, $\beta = 0.25$, and an EMA-updated codebook with a size of 256 and either 256 (dog) or 128 (other tasks) channels per dimension. We did not change other hyperparameters from DCWM. However, we found that to approach the performance of standard FSQ, VQ-VAE needs environment-dependent adjusting of the planning procedure. In Humanoid Walk, the performance of FSQ aligns closely with the VQ-VAE with a weighted sum over the codes in the codebook for planning (expected code) but significantly outperforms sampled VQ-VAE. Conversely, standard sampling is superior in Reacher Hard, which is unsurprising, as the discrete codes in VQ-VAE have not been ordered like in FSQ. The necessary environment-specific adjustments for VQ-VAE undermine its general applicability compared to FSQ.

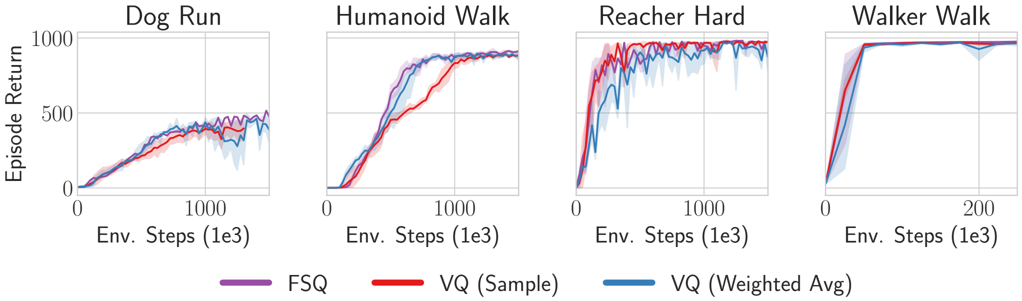


Figure 15: **Ablation of FSQ vs VQ-VAE**

–appendices continue on next page–

B.8 ABLATION OF REDQ CRITIC VS STANDARD DOUBLE Q APPROACH

In this section, we compare the ensemble of Q-functions approach, used by DCWM, REDQ (Chen et al., 2021) and TD-MPC2 (Hansen et al., 2023), to the standard double Q approach. In Fig. 16, we evaluate how our default ensemble size of $N_q = 5$ (purple) compares with the standard double Q approach, which is obtained by setting the ensemble size to $N_q = 2$ (blue). Note that we always sample two critics so the $N_q = 2$ result reduces to the standard double Q approach. Fig. 16 shows that DCWM works fairly well with both approaches but the ensemble approach offers benefits in the harder Dog Run and Humanoid Walk tasks.

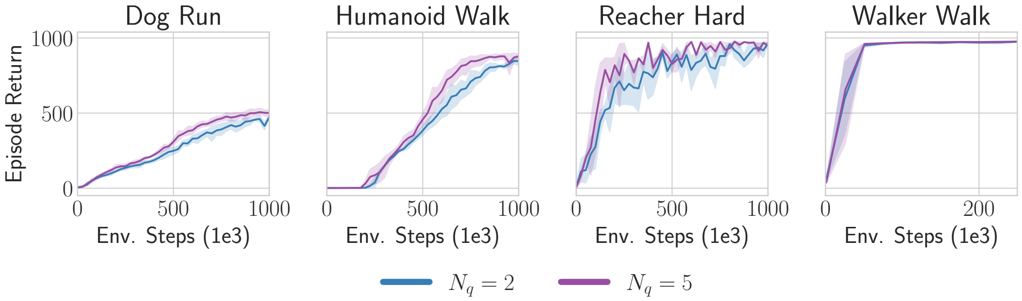


Figure 16: **Ablation of REDQ critic vs standard double Q** DCWM uses a Q ensemble, similar to REDQ, of size $N_q = 5$ (purple) and sub samples two critics when calculating the mean or minimum Q-value. We compare this approach to the standard double Q approach by setting $N_q = 2$ (blue) and we see that the ensemble approach offers a slight benefits in the harder Dog Run and Humanoid Walk tasks.

–appendices continue on next page–

B.9 RECONSTRUCTION LOSS HAS A DETRIMENTAL IMPACT

In this section, we seek to evaluate what happens when we replace DreamerV3's one-hot discrete encoding with the codebook encoding used in DCWM. Fig. 17 shows that in the easy Reacher Hard and Walker Walk environments, FSQ (blue) and one-hot (orange) perform similarly. However, in the difficult Dog Run and Humanoid Walk tasks, no discrete encoding can enable DreamerV3 to perform as well as DCWM (purple). We hypothesize that DreamerV3's poor performance in the Dog Run and Humanoid Walk tasks results from its decoder struggling to reconstruct the states.

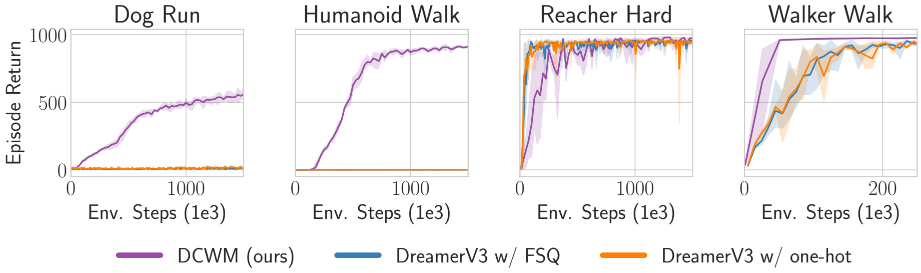


Figure 17: **DreamerV3 with FSQ** Replacing DreamerV3's one-hot encoding (orange) with DCWM's codebook encoding (blue) does not improve performance. Moreover, DreamerV3 is not able to learn in the hard Dog Run and Humanoid Walk tasks and is significantly outperformed by DCWM (purple).

Learning to minimize the observation reconstruction error has been widely applied in model-based RL (Sutton & Barto, 2018; Ha & Schmidhuber, 2018; Hafner et al., 2019b), and an observation decoder has been a component of many of the most successful RL algorithms to date (Hafner et al., 2023). However, recent work in representation learning for RL (Zhao et al., 2023) and model-based RL (Hansen et al., 2022) has shown that incorporating a reconstruction term into the representation loss can hurt the performance, as learning to reconstruct the observations is inefficient due to the observations containing irrelevant details that are uncontrollable by the agent and do not affect the task.

To provide a thorough analysis of DCWM, we include results where we add a reconstruction term to our world model loss in Eq. (8):

$$\mathcal{L}_{\boldsymbol{o}} = \mathbb{E}_{\boldsymbol{o}_t \sim \mathcal{D}}[\|\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t\|_2^2], \quad \hat{\boldsymbol{o}}_t = h_\kappa(\boldsymbol{c}_t), \tag{13}$$

where h_κ is a learned observation decoder that takes the latent code as the input and outputs the reconstructed observation. The decoder h_κ is a standard MLP. We perform reconstruction at each time step in the horizon. The results in Fig. 18 show that in no environments does reconstruction aid learning, and in some tasks, such as the difficult Dog Run and Humanoid Walk tasks, including the reconstruction term has a significant detrimental effect on the performance, and can even prevent learning completely. Our results support the observations of Zhao et al. (2023) and Hansen et al. (2022) about the lack of need for a reconstruction target in continuous control tasks.

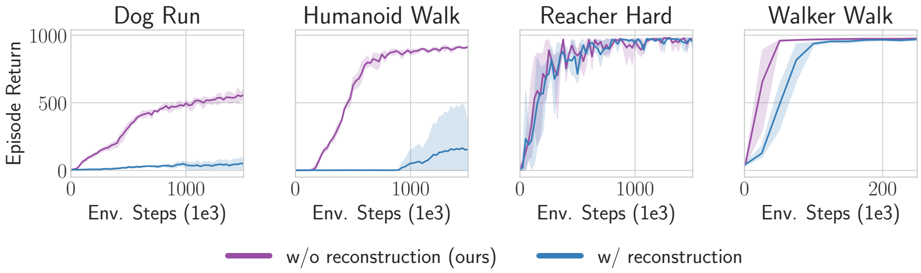


Figure 18: **Reconstruction harms DCWM's performance** Adding observation reconstruction to DCWM (blue) harms the performance of DCWM across a mixture of easy and hard DMControl tasks.

–appendices continue on next page–

B.10 MYOSUITE MUSCULOSKELETAL RESULTS

In this section, we evaluate DCWM in five musculoskeletal tasks from MyoSuite. In Fig. 19, we report aggregate metrics at 1M environment steps over three random seeds in the five tasks. Fig. 20 then shows the training curves for the individual tasks. On average, DCWM performs well, generally matching TD-MPC2 at 1M environment steps and outperforming the other baselines. However, DCWM is initially slower to learn than TD-MPC2. Given that DCWM and TD-MPC2 have several different components, it is hard to say exactly what results in DCWM's lag in performance. It could be because TD-MPC2 utilizes SAC as the underlying model-free RL algorithm which promotes exploration via the maximum entropy framework. It could also be that TD-MPC2's approach of learning the encoder jointly with the Q-function is key to its success. We leave an investigation into this for future work, as our focus is on latent space design. Nevertheless, in App. B.11, we investigate using DCWM's latent space inside TD-MPC2, which shows that DCWM's latent space can in fact improve TD-MPC2.

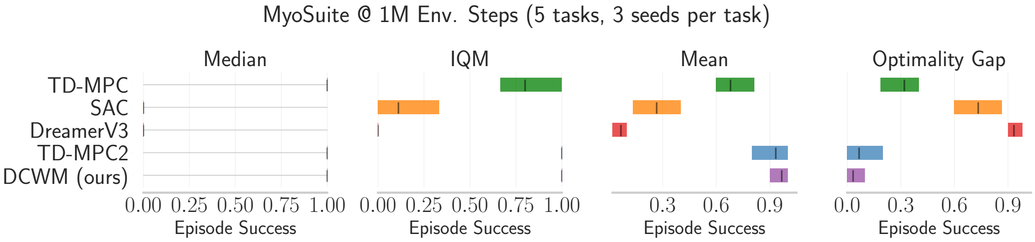


Figure 19: **MyoSuite aggregate metrics**

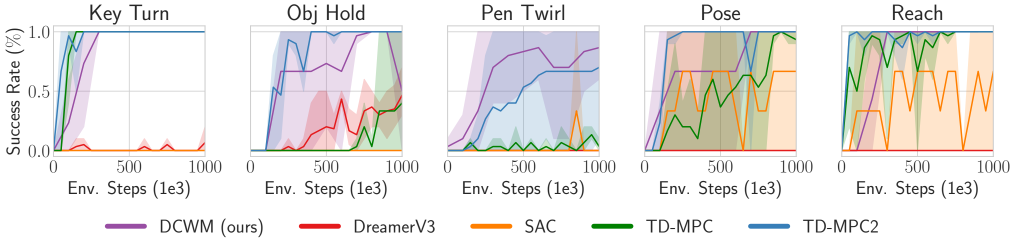


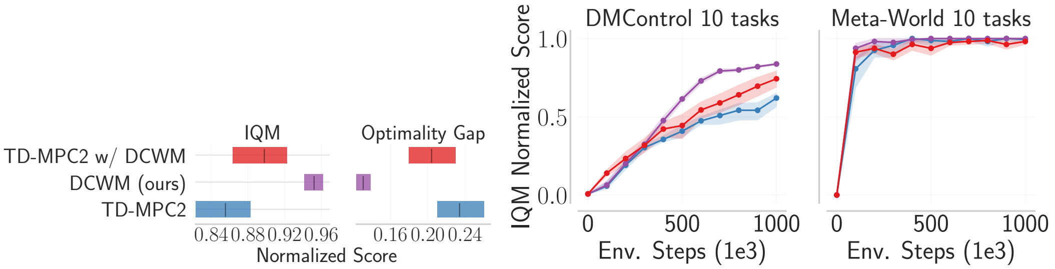
Figure 20: **MyoSuite training curves**

–appendices continue on next page–

B.11 IMPROVING TD-MPC2 WITH DCWM

In this section, we investigate using DCWM's latent space inside TD-MPC2. Note that TD-MPC2's latent space is continuous and trained with MSE regression. It also uses simplical normalization (SimNorm) to make its latent space bounded. In these experiments, we removed SimNorm and replaced it with our discrete and stochastic latent space, and then trained using cross-entropy for the consistency loss. In particular, we made the following changes to the TD-MPC2 codebase: *(i)* removed SimNorm, *(ii)* added FSQ to the encoder, *(iii)* modified the dynamics to predict the logits instead of the next latent state, *(iv)* modified the dynamics to use Gumbel-softmax sampling for multi-step predictions during training and our weighted average approach during planning, and *(v)* changed the world model's loss coefficients for consistency, value, and, reward, to all be 1.

In Fig. 21, we report aggregate metrics over 3 random seeds in 10 DMControl tasks and 10 Meta-World tasks. Fig. 21a shows the IQM and optimality gap at 1M environment steps over the 20 tasks. It shows that adding DCWM's discrete and stochastic latent space to TD-MPC2 offers some improvement. Fig. 21b shows the aggregate training curves (IQM over 10 tasks) for DMControl and Meta-World, respectively. The results show that using DCWM inside TD-MPC2 offers some benefits in the 10 DMControl tasks, whilst in the 10 Meta-World tasks, the performance of all methods seems about equal. This suggests that, in the context of continuous control, discrete and stochastic latent spaces are advantageous for world models. This is an interesting result which we believe motivates further research into discrete and stochastic latent spaces for world models.



(a) **Aggregate statistics at 1M environment steps**

(b) **Training curves**

Figure 21: **TD-MPC2 with DCWM** Adding DCWM's discrete and stochastic latent space to TD-MPC2 improves performance in DMControl tasks and maintains similar performance in Meta-World.

–appendices continue on next page–

C IMPLEMENTATION DETAILS

Architecture We implemented DCWM with PyTorch (Paszke et al., 2019) and used the AdamW optimizer (Kingma & Ba, 2017) for training the models. All components (encoder, dynamics, reward, actor and critic) are implemented as MLPs. Following Hansen et al. (2023) we let all intermediate layers be linear layers followed by LayerNorm (Ba et al., 2016). We use Mish activation functions throughout. Below we summarize the DCWM architecture for our base model.

```
DCWM(
  (model): WorldModel(
    (_fsq): FSQ(levels=[5, 3])
    (_encoder): ModuleDict(
      (state): Sequential(
        (0): NormedLinear(in_features=obs_dim, out_features=256, act=Mish)
        (1): Linear(in_features=256, out_features=latent_dim)
      )
    )
    (_trans): Sequential(
      (0): NormedLinear(in_features=latent_dim+act_dim, out_features=512, act=Mish)
      (1): NormedLinear(in_features=512, out_features=512, act=Mish)
      (2): Linear(in_features=512, out_features=latent_dim)
    )
    (_reward): Sequential(
      (0): NormedLinear(in_features=latent_dim+act_dim, out_features=512, act=Mish)
      (1): NormedLinear(in_features=512, out_features=512, act=Mish)
      (2): Linear(in_features=512, out_features=1)
    )
  )
  (agent): TD3(
    (_pi): Actor(
      (mlp): Sequential(
        (0): NormedLinear(in_features=latent_dim, out_features=512, act=Mish)
        (1): NormedLinear(in_features=512, out_features=512, act=Mish)
        (2): Linear(in_features=512, out_features=act_dim)
      )
    )
    (_pi_tar): Actor(
      (mlp): Sequential(
        (0): NormedLinear(in_features=latent_dim, out_features=512, act=Mish)
        (1): NormedLinear(in_features=512, out_features=512, act=Mish)
        (2): Linear(in_features=512, out_features=act_dim)
      )
    )
    (Q): Critic(
      (qs): Vectorized ModuleList(
        (0-4): 5 x Sequential(
          (0): NormedLinear(in_features=latent_dim+act_dim, out_features=512, act=Mish)
          (1): NormedLinear(in_features=512, out_features=512, act=Mish)
          (2): Linear(in_features=512, out_features=1)
        )
      )
    )
    (Q_tar): Critic(
      (qs): Vectorized ModuleList(
        (0-4): 5 x Sequential(
          (0): NormedLinear(in_features=latent_dim+act_dim, out_features=512, act=Mish)
          (1): NormedLinear(in_features=512, out_features=512, act=Mish)
          (2): Linear(in_features=512, out_features=1)
        )
      )
    )
  )
)
```

where `obs_dim` is the dimensionality of the observation space, `latent_dim` is the dimensionality of the latent space and `act_dim` is the dimensionality of the action space.

Hyperparameters Table 1 lists all of the hyperparameters for training DCWM which were used for the main experiments and the ablations.

Table 1: **DCWM hyperparameters** We kept most hyperparameters fixed across all tasks.

Hyperparameter	Value	Description
Training		
Action repeat	2	
Max episode length	500 in DMControl	Action repeat makes this 1000
	100 in Meta-World	Action repeat makes this 200
Num. eval episodes	10	
Random episodes	10	Num. random episodes at start
MPPI planning		
Planning horizon	3	
Population size (N_p)	512	
Number of elites (K)	64	
Minimum std	0.05	
Maximum std	2	
Temperature	0.5	
TD3		
Actor update freq.	2	Update actor less than critic
Batch size	512	
Buffer size	10^6	
Discount factor γ	0.99	
Exploration noise	$\text{Linear}(1.0, 0.1, 50)$ (easy)	
	$\text{Linear}(1.0, 0.1, 150)$ (medium)	
	$\text{Linear}(1.0, 0.1, 500)$ (hard)	
Learning rate	3×10^{-4}	
MLP dims	$[512, 512]$	For actor/critic/dynamics/reward
Momentum coef. (τ)	0.005	
Number of Q-functions (N_q)	5	
Noise clip (c)	0.3	
N-step TD	1 or 3	
Policy noise	0.2	
Update-to-data (UTD) ratio	1	
World model		
Discount factor γ	0.9	
Encoder learning rate	10^{-4}	
Encoder MLP dims	$[256]$	
FSQ levels	$[5, 3]$	
Horizon (H)	5	For world model training
Latent dimension (d)	512	
	1024 (Humanoid/Dog)	

Statistical significance We used five seeds for DCWM and three seeds for TD-MPC2/DreamerV3 in the main figures, at least three seeds for all ablations, and plotted the 95 % confidence intervals as the shaded area, which corresponds to approximately two standard errors of the mean.

Hardware We used NVIDIA A100s and AMD Instinct MI250X GPUs to run our experiments. All our experiments have been run on a single GPU with a single-digit number of CPU workers.

Open-source code For full details of the implementation, model architectures, and training, please check the code, which is available in the submitted supplementary material and will be made public upon acceptance to guarantee seamless reproducibility.

–appendices continue on next page–

D BASELINES

In this section, we provide further details of the baselines we compare against.

- **DreamerV3 (Hafner et al., 2023)** is a reinforcement learning algorithm that uses a world model to predict outcomes, a critic to judge their value, and an actor to choose actions to maximize value. It uses symlog loss for training and operates on model states from imagination data. The critic is a categorical distribution with exponentially spaced bins, and the actor is trained with entropy regularization and return normalization. The world model is only used for training and there is no planning in online evaluation. In contrast, DCWM learns a deterministic encoder with a discrete latent space and stochastic dynamics in the world model. The world model objective is based on maximizing the log-likelihood, which is much simpler yet enables superior performance and sample efficiency across different continuous control tasks with the same set of hyperparameters. We report the results of DreamerV3 from the TD-MPC2 official repository [3].
- **Temporal Difference Model Predictive Control 2 (TD-MPC2, Hansen et al. (2023))** is a decoder-free model-based reinforcement learning algorithm with a focus on scalability and sample efficiency. It includes an encoder, latent transition dynamics, a reward predictor, a terminal value (critic), and a policy prior (actor). In contrast to DreamerV3, it utilizes a deterministic encoder and transition dynamics implemented with MLPs and layer normalization (Ba et al., 2016) and Mish (Misra, 2019) activation function. To avoid exploding gradients and representation collapse, the latent space is normalized with projection followed by a softmax operation. All components except the policy prior are trained jointly based on predicting the latent embedding, reward prediction, and value prediction, while reward and value predictions are based on discrete classification in log-transformed space. Similarly, we use a deterministic encoder, but we train the transition dynamics with a cross-entropy loss function, which considers multi-modality and uncertainties, and we decouple representation learning from value learning. We report the results from the TD-MPC2 official repository [4].
- **Temporal Difference Model Predictive Control (TD-MPC, Hansen et al. (2022))** is the first version of TD-MPC2. It is also a decoder-free model-based RL algorithm consisting of an encoder, latent transition dynamics, reward predictor, terminal value (critic), and policy prior (actor). In contrast to TD-MPC2, it does not apply simplical normalization (SimNorm) to its latent state, it trains the reward and value prediction using the MSE loss instead of the cross-entropy loss, and it uses SAC as the underlying RL algorithm. We refer the reader to the TD-MPC paper for further details. We report the results from the TD-MPC2 official repository [5].
- **Soft Actor-Critic (SAC, Haarnoja et al. (2018)** is an off-policy model-free RL algorithm based on the maximum entropy RL framework. That is, it attempts to succeed at the task whilst acting as randomly as possible. It is worth highlighting that TD-MPC2 uses SAC as it's underlying model-free RL algorithm. We report the results from the TD-MPC2 official repository [6].

–appendices continue on next page–

---

[3] `https://github.com/nicklashansen/tdmpc2/tree/main/results/dreamerv3`
[4] `https://github.com/nicklashansen/tdmpc2/tree/main/results/tdmpc2`
[5] `https://github.com/nicklashansen/tdmpc2/tree/main/results/tdmpc2`
[6] `https://github.com/nicklashansen/tdmpc2/tree/main/results/tdmpc2`

E TASKS

We evaluate our method in 30 tasks from the DeepMind Control suite (Tassa et al., 2018) and 7 tasks from Meta-World (Yu et al., 2019) benchmark. Table 2 and Table 3 provide details of the environments we used, including the dimensionality of the observation and action spaces.

Table 2: **DMControl** We consider a total of 30 continuous control tasks from the DeepMind Control suite.

TASK	OBSERVATION DIM	ACTION DIM	SPARSE?
ACROBOT SWINGUP	6	1	N
CARTPOLE BALANCE	5	1	N
CARPOLE BALANCE SPARSE	5	1	Y
CARTPOLE SWINGUP	5	1	N
CARTPOLE SWINGUP SPARSE	5	1	Y
CHEETAH RUN	17	6	N
CUP CATCH	8	2	Y
CUP SPIN	8	2	N
DOG RUN	223	38	N
DOG STAND	223	38	N
DOG TROT	223	38	N
DOG WALK	223	38	N
FINGER SPIN	9	2	Y
FINGER TURN EASY	12	2	Y
FINGER TURN HARD	12	2	Y
FISH SWIM	24	5	N
HOPPER HOP	15	4	N
HOPPER STAND	15	4	N
HUMANOID RUN	67	24	N
HUMANOID STAND	67	24	N
HUMANOID WALK	67	24	N
PENDULUM SPIN	3	1	N
PENDULUM SWINGUP	3	1	N
QUADRUPED RUN	78	12	N
QUADRUPED WALK	78	12	N
REACHER EASY	6	2	Y
REACHER HARD	6	2	Y
WALKER RUN	24	6	N
WALKER STAND	24	6	N
WALKER WALK	24	6	N

Table 3: **Meta-World** We consider a total of 9 continuous control tasks from the Meta-World. This benchmark is designed for multitask research and all tasks thus share similar embodiment, observation space, and action space.

TASK	OBSERVATION DIM	ACTION DIM	SPARSE?
ASSEMBLY	39	4	N
BASKETBALL	39	4	N
BUTTON PRESS	39	4	N
DISASSEMBLE	39	4	N
DOOR OPEN	39	4	N
LEVER PULL	39	4	N
SOCCER	39	4	N

–appendices continue on next page–

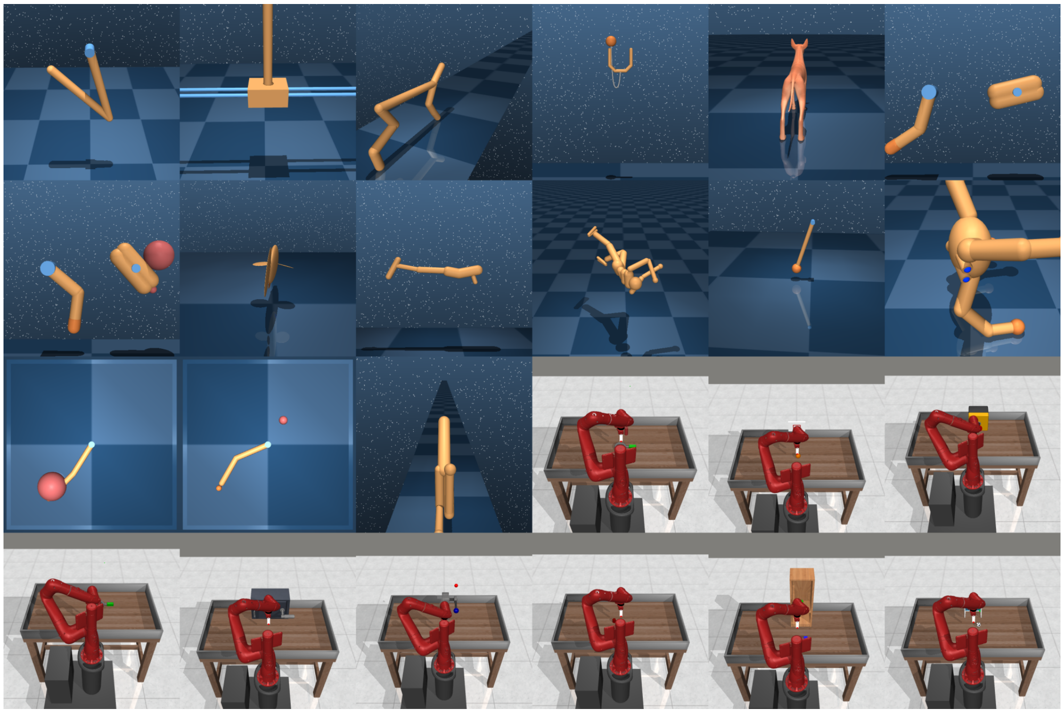


Figure 22: **Tasks visualizations** Visualization of the DMControl tasks and Meta-World tasks used throughout this paper.