Appendix

A Behavior Learning

We utilize the actor-critic learning scheme of DreamerV2 [15]. Specifically, we introduce a stochastic actor and a deterministic critic as below:

$$\text{Actor:} \quad \hat{a}_t \sim p_\psi(\hat{a}_t \mid \hat{s}_t) \quad \text{Critic:} \quad v_\xi(\hat{s}_t) \approx \mathbb{E}_{p_\theta}\left[\textstyle\sum_{i \le t} \gamma^{i-t} \hat{r}_i\right], \tag{9}$$

where $\{\hat{s}_t, \hat{a}_t, \hat{r}_t\}$ is imagined future states, actions, and rewards which are recursively obtained by conditioning on a initial state $\hat{s}_0$ and utilizing the transition model and the reward model in Equation 1, and the actor in Equation 9. Note that the initial state $\hat{s}_0$ is the model state obtained from the representation model in Equation 1 using the samples from the replay buffer. Then the critic is trained to regress the λ-target [42, 55] as follows:

$$\mathcal{L}^{\texttt{critic}}(\xi) \doteq \mathbb{E}_{p_\theta}\left[\sum_{t=1}^{H-1} \frac{1}{2}\left(v_\xi(\hat{s}_t) - \text{sg}(V_t^\lambda)\right)^2\right], \tag{10}$$

$$V_t^\lambda \doteq \hat{r}_t + \gamma \begin{cases} (1-\lambda) v_\xi(\hat{s}_{t+1}) + \lambda V_{t+1}^\lambda & \text{if } t < H \\ v_\xi(\hat{s}_H) & \text{if } t = H, \end{cases} \tag{11}$$

where sg is a stop gradient function. Then we train the actor that maximizes the imagined return by back propagating the gradients through the learned world models as follows:

$$\mathcal{L}^{\texttt{actor}}(\psi) \doteq \mathbb{E}_{p_\theta}\left[-V_t^\lambda - \eta \,\text{H}\left[a_t | \hat{s}_t\right]\right], \tag{12}$$

where the entropy of actor $\text{H}\left[a_t|\hat{s}_t\right]$ is maximized to encourage exploration, and η is a hyperparameter that adjusts the strength of entropy regularization. We refer to Hafner et al. [15] for more details.

B Extended Qualitative Analysis

We provide our qualitative analysis in videos on our project website:

`https://sites.google.com/view/mwm-rl`

which contains videos for (i) reconstructions from masked autoencoders (MAE) [13] and (ii) predictions from latent dynamics models. To be self-contained, we also provide reconstructions from masked autoencoders with images in Appendix B.1.

B.1 Reconstructions from Masked Autoencoders

In this section, we provide motivating examples for introducing convolutional feature masking. Specifically, we provide reconstructions from MAE [13] trained on Coffee-Pull and Peg-Insert-Side tasks from Meta-world [16] in Figure 8. We find that reconstruction with pixel patch masking can be an extremely difficult objective, which makes it difficult for the model to learn the fine-grained details such as object positions. For instance, in Figure 8, MAE struggles to predict the position of objects (*e.g.,* a cup or a block) within masked patches, making it difficult to learn such details.

C Extended Related Work

Vision transformers with early convolution Introducing convolutional layers into a ViT architecture is not new. Dosovitskiy et al. [14] investigated a hybrid ViT architecture that utilizes a modified version of ResNet [56] to obtain a convolutional feature map. This has been followed by a series of works that investigate the architecture design to introduce convolutions for improved performance [45, 57]. While these works mostly consider deep convolutional networks to maximize the performance on downstream tasks, we introduce a lightweight convolution stem consisting of a few convolution layers, following the design of Xiao et al. [40]. This is because our motivation for introducing the convolution stem is to avoid the pitfall of reconstruction objective with masked pixel patches, but not to investigate the optimal hybrid ViT architecture that maximizes the performance.

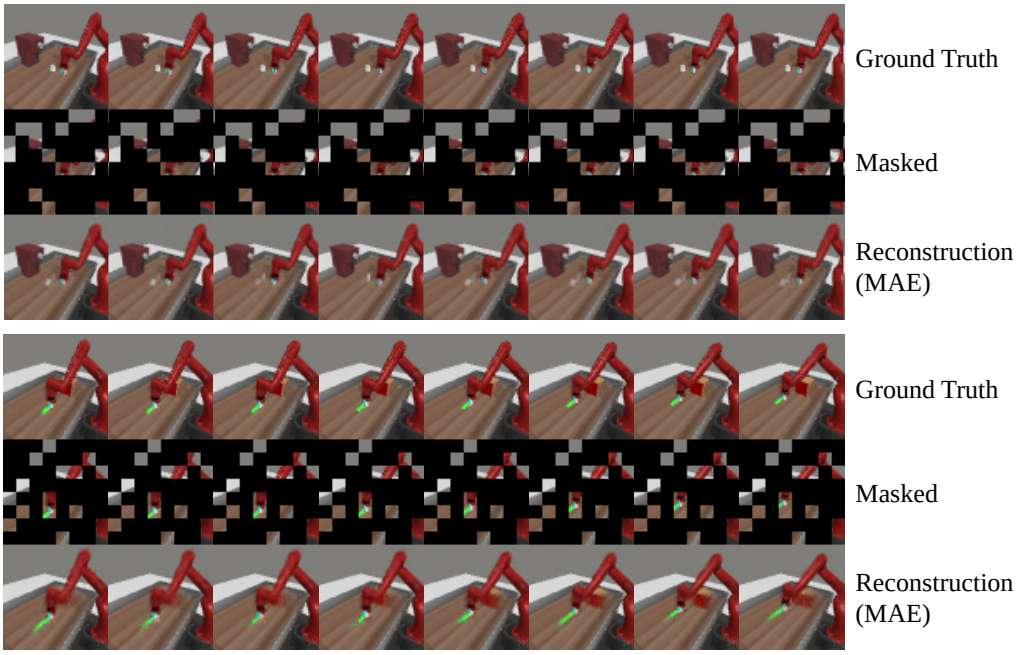


Figure 8: Frames reconstructed with the masked autoencoders (MAE) [13] trained on Meta-world (Top) Coffee Pull and (Bottom) Peg Insert Side. We find that reconstructions are not capturing the detailed object positions within patches. Best viewed as video provided in our website.

Convolutional feature masking In the context of semantic segmentation, Dai et al. [58] proposed to utilize convolutional feature masking instead of pixel masking. But this differs in that their goal is to utilize the masks as inputs to classifiers, unlike our approach that drops convolutional features with masks. More related to our work is the approaches that mask out convolutional features as a regularization technique [59, 60]. For instance, Tompson et al. [59] demonstrated that masking out entire channels for a specific feature from a convolutional feature map can be more effective than Dropout [61] that masks randomly sampled channels. Ghiasi et al. [60] further developed this idea by proposing DropBlock that masks contiguous region of a feature map. In the context of visual control, Park et al. [62] trained a VQ-VAE [63] and proposed to drop convolutional features corresponding to randomly sampled discrete latent codes. Our work extends the idea of masking out convolutional features to self-supervised learning with a ViT and demonstrates its effectiveness for representation learning in visual model-based RL.

Unsupervised representation learning for visual control Following the work of Jaderberg et al. [64] that demonstrated the effectiveness of auxiliary unsupervised objectives for RL, a variety of unsupervised learning objectives have been studied, including future latent reconstruction [27, 65, 66, 67], bisimulation [68, 69], contrastive learning [70, 71, 72, 73, 74, 30, 75, 29, 76, 77], keypoint extraction [78, 79, 80, 81], world model learning [8, 9, 10, 54] and reconstruction [82, 83]. Recent approaches have also demonstrated that simple data augmentations can sometimes be effective even without such representation learning objectives [84, 85, 86]. The work closest to ours is Xiao et al. [87], which demonstrated that frozen representations from MAE pre-trained on real-world videos can be used for training RL agents on visual manipulation tasks. Our work differs in that we demonstrate that training a self-supervised ViT with reconstruction and convolutional feature masking can be more effective for visual control tasks when compared to MAE that masks pixel patches. We also note that our work is orthogonal to Xiao et al. [87] in that our framework can also initialize the autoencoder with parameters pre-trained using real-world videos.

D Pseudocode

For clarity, we define the optimization objectives for autoencoder and latent dynamics model, and describe the pseudocode for our method. Specifically, given a random batch $\{(o_j, r_j, a_j)\}_{j=1}^{B}$, visual representation learning and dynamics learning objectives are defined as:

$$\mathcal{L}^{\texttt{vis}}(\phi) = \frac{1}{B}\sum_{j=1}^{B}\Big(-\ln p_\phi(o_j|z_j^{c,m}) - \ln p_\phi(r_j|z_j^{c,m})\Big) \tag{13}$$

$$\mathcal{L}^{\texttt{dyn}}(\theta) = \frac{1}{B}\sum_{j=1}^{B}\Big(-\ln p_\theta(z_j^{c,0}|s_j) - \ln p_\theta(r_j|s_j) + \beta\,\mathrm{KL}\Big[q_\theta(s_j|s_{j-1}, a_{j-1}, z_j^{c,0}) \,\|\, p_\theta(\hat{s}_j|s_{j-1}, a_{j-1})\Big]\Big) \tag{14}$$

Algorithm 1 Masked World Models

Initialize parameters of autoencoder ϕ, latent dynamics model θ, actor ψ, and critic ξ
Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
for each timestep t **do**
  // COLLECT TRANSITIONS
  Get autoencoder representation $z_t^{c,0}$
  Update model state $s_t \sim q_\theta(s_t|s_{t-1}, a_{t-1}, z_t^{c,0})$
  Sample action $a_t \sim p_\psi(a_t|s_t)$
  Add transition to replay buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \{(o_t, a_t, r_t)\}$
  // VISUAL REPRESENTATION LEARNING WITH REWARD PREDICTION
  Sample random minibatch $\{(o, r)\} \sim \mathcal{B}$
  Update autoencoder by minimizing $\mathcal{L}^{\texttt{vis}}(\phi)$
  // DYNAMICS LEARNING
  Sample random minibatch $\{(o, r, a)\} \sim \mathcal{B}$
  Update latent dynamics model by minimizing $\mathcal{L}^{\texttt{dyn}}(\theta)$ and obtain states $\{s\}$
  // ACTOR CRITIC LEARNING
  Imagine future rollouts $\{\hat{s}, \hat{a}, \hat{r}\}$ from $\{s\}$ using latent dynamics model and actor
  Update actor by minimizing $\mathcal{L}^{\texttt{actor}}(\psi)$
  Update critic by minimizing $\mathcal{L}^{\texttt{critic}}(\xi)$
end for

E Architecture Details

E.1 Autoencoder

Convolution stem and masking We use visual observations of $64 \times 64 \times 3$. For the convolution stem, similar to the design of Xiao et al. [40], we stack 3 convolution layers with the kernel size of 4×4 and stride 2, followed by a convolution layer with the kernel size of 1×1. This convolution stem processes o_t into $8 \times 8 \times 256$, which has the same spatial shape of 8×8 when we use the patchify stem with patch size of 8×8. Then a masking is applied with a masking ratio of $m = 0.75$.

ViT encoder and decoder We use a 4-layer ViT encoder and a 3-layer ViT decoder, which are implemented using tfimm[4] library. The ViT encoder concatenates class token with un-masked convolutional features, embeds inputs into 256-dimensional vectors, and processes them through Transformer layers. Then the ViT decoder takes outputs from the encoder and concatenate learnable mask tokens into them. Here, we use the same learnable mask token for reward prediction, which can be discriminated from other mask tokens because it gets different positional encoding. Finally, two linear output heads for predicting pixels and rewards are used to generate predictions. Unlike MAE, we compute the loss on entire pixels because we do not apply masking to pixels.

[4] https://github.com/martinsbruveris/tensorflow-image-models

Initialization with warm-up schedule We initialize parameters of the autoencoder using 5000 gradients steps with a linear warm-up schedule over initial 2500 steps using the samples collected from initial random exploration. We find this improves sample-efficiency on relatively easy tasks, but does not make significant difference on complex tasks. This is because better visual representations are used for learning latent dynamics models from the beginning. However, we also observe that this initialization is not required when we update the parameters in a more short interval (*e.g.,* every 2 timesteps instead of 5 timesteps), because the autoencoder can be trained quickly without introducing such an initialization period. In our experiments, we use the initialization scheme and update the parameters every 5 timesteps for faster experimentation.

E.2 Latent Dynamics Model

Architecture Our model is built upon the discrete latent dynamics model introduced in DreamerV2. Inputs to our model are representations $z_t^{c,0}$ from the autoencoder, which are of shape $8 \times 8 \times 256$ obtained by processing the visual observations through the convolution stem and ViT encoder of our autoencoder without masking (*i.e.,* $m = 0$). Because our dynamics model does not take visual observations as inputs, we do not utilize CNN encoder and decoder as in the original architecture. Instead, we introduce a shallow 2-layer ViT encoder and decoder with the embedding size of 128, which takes $z_t^{c,0}$ as inputs. Following Seo et al. [54], we increase the hidden size of dense layers and the model state dimension from 200 to 1024.

Prediction visualization While the latent dynamics model is not trained directly to reconstruct raw pixels, its predictions can still be used for visualizing the open-loop predictions. This is because it is trained to reconstruct $z_t^{c,0}$, which can be processed through the ViT decoder of the autoencoder. We use this scheme for visualizing the predictions from the model in Figure 7.

F Experiments Details

Meta-world experiments In order to use a single camera viewpoint consistently over all 50 tasks, we use the modified `corner2` camera viewpoint for all tasks. Specifically, we adjusted the camera position with `env.model.cam_pos[2][:]=[0.75, 0.075, 0.7]`, rendering visual observations as in Figure 2 which enables us to solve non-zero success rate on all tasks. Maximum episode length for Meta-world tasks is 500. We use the action repeat of 2, which we find it easy to solve tasks compared to the action repeat of 1 used in Seo et al. [54].

In our experiments, we classify 50 tasks into `easy`, `medium`, `hard`, and `very hard` tasks where experiments are run over 500K, 1M, 2M, 3M environments steps with action repeat of 2, respectively.

Difficulty	Tasks
`easy`	Button Press, Button Press Topdown, Button Press Topdown Wall, Button Press Wall, Coffee Button, Dial Turn, Door Close, Door Lock, Door Open, Door Unlock, Drawer Close, Drawer Open, Faucet Close, Faucet Open, Handle Press, Handle Press Side, Handle Pull, Handle Pull Side, Lever Pull, Plate Slide, Plate Slide Back, Plate Slide Back Side, Plate Slide Side, Reach, Reach Wall, Window Close, Window Open, Peg Unplug Side
`medium`	Basketball, Bin Picking, Box Close, Coffee Pull, Coffee Push, Hammer, Peg Insert Side, Push Wall, Soccer, Sweep, Sweep Into
`hard`	Assembly, Hand Insert, Pick Out of Hole, Pick Place, Push, Push Back
`very hard`	Shelf Place, Disassemble, Stick Pull, Stick Push, Pick Place Wall

RLBench experiments We consider two relatively easy tasks (*i.e.,* Reach Target and Push Button) with dense rewards, and utilize an action mode that specifies the delta of joint positions. Because original RLBench repository does not support shaped rewards for Push Button task, we design a shaped rewards for Push Button following the design of rewards in Reach Target. Specifically, the reward is defined as the sum of (i) the L2 distance of gripper to a button and (ii) the magnitude of the button being pushed. We set the maximum episode length to 200, and use the action repeat of 2. Because RLBench is designed to be episodic unlike Meta-world, we use the discount prediction

scheme in DreamerV2 that introduces a linear head that predicts the termination of each rollout. For visual observations, we use the front RGB observation (see Figure 2 for an example).

DeepMind Control Suite experiments We follow the setup of Hafner et al. [21] where the action repeat of 2 is used. We use default camera configurations without modification. We note that direct comparison with the results from Hafner et al. [21] is not possible because our experiments are based on DreamerV2 with larger networks (see Appendix E.2 for architecture details).

Computation In terms of parameter counts, MWM consists of 25.9M parameters while DreamerV2 consists of 33.2M parameters. However, in terms of training time, MWM takes 5.5 hours for training over 500K environment steps, which is 1.57 times slower than DreamerV2 that takes 3.5 hours, because MWM processes visual observations through low-throughput ViT twice with and without masking. Given the improvement in final performances and sample-efficiency on complex tasks as demonstrated in our experiments, we note that it is worth spending additional computational costs.

Hyperparameters We report the hyperparameters used in our experiments in Table 1.

Table 1: Hyperparameters used in our experiments. Unless otherwise specified, we use the same hyperparameters used in DreamerV2 [15]. DMC is an abbreviation of DeepMind Control Suite.

Hyperparameter	**Value**
Image observation	$64 \times 64 \times 3$
Image normalization	Mean: $(0.485, 0.456, 0.406)$, Std: $(0.229, 0.224, 0.225)$
Action repeat	2
Max episode length	500 (Meta-world), 200 (RLBench), 1000 (DMC)
Early episode termination	True (RLBench), False otherwise
Random exploration	5000 environment steps
Reward normalization	False (DMC), True otherwise
World model batch size	16 (DMC), 50 otherwise
World model sequence length	50
World model tradeoff (β)	0.1 (RLBench), 1.0 otherwise
World model tradeoff free-bits	0.1 (RLBench), 0.01 otherwise
World model ViT encoder size	2 layers, 4 heads, 128 units
World model ViT decoder size	2 layers, 4 heads, 128 units
Autoencoder batch size	1024
Autoencoder initialization steps	5000
Autoencoder warm-up steps	2500
Autoencoder learning rate	$3 \cdot 10^{-4}$
Autoencoder masking ratio	0.75
Autoencoder ViT encoder size	4 layers, 4 heads, 256 units
Autoencoder ViT decoder size	3 layers, 4 heads, 256 units

G Full Meta-world Experiments

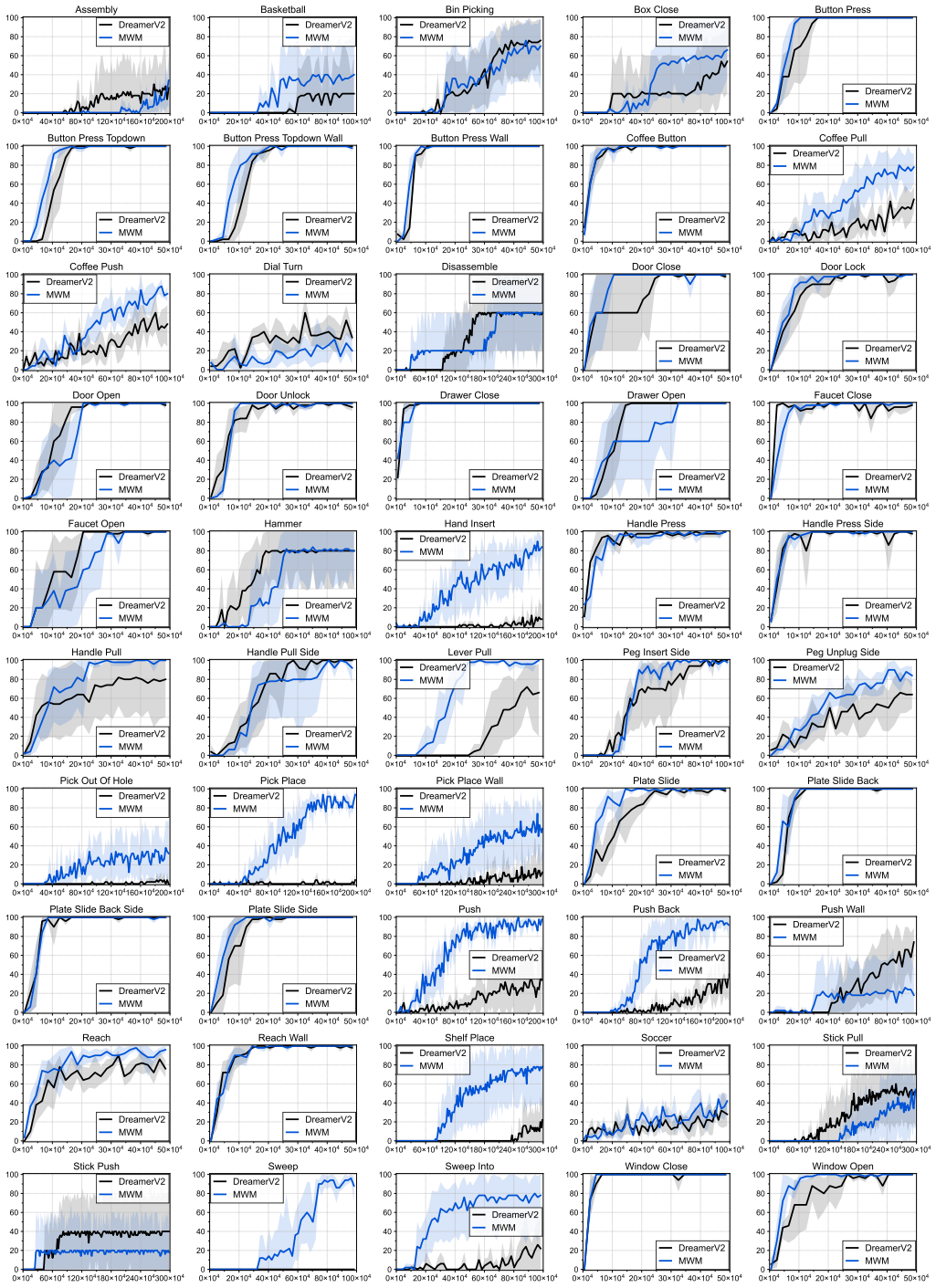


Figure 9: Learning curves on 50 visual robotic manipulation tasks from Meta-world as measured on the success rate. The solid line and shaded regions represent the mean and bootstrap confidence intervals, respectively, across five runs.

H Additional DeepMind Control Suite Experiments

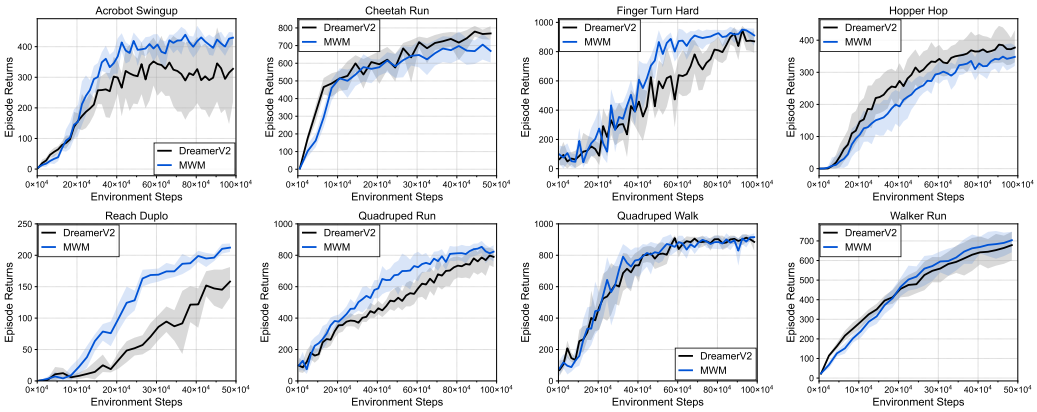


Figure 10: Learning curves on eight visual robot tasks from DeepMind Control Suite as measured on the episode return. The solid line and shaded regions represent the mean and bootstrap confidence intervals, respectively, across eight runs.

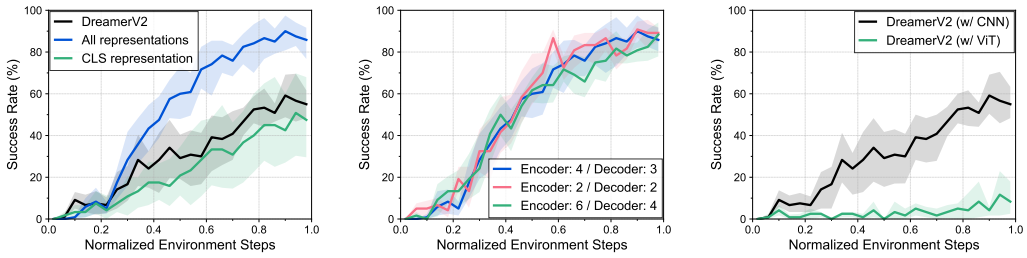
I Extended Ablation Study and Analysis

We provide additional analysis in Figure 11 and learning curves on individual tasks in Figure 12.

Utilizing only CLS representation We ablate our design choice of utilizing all representations of $z_t^{c,0}$ for dynamics learning, instead of using only CLS representation as in MAE. As shown in Figure 11(a), we find that utilizing all representations (*i.e.,* CLS + Conv) outperforms the baseline (*i.e.,* CLS), by encouraging the model to learn spatial information included in all representations.

Model size We report the performance of our method with varying number of layers for autoencoders in Figure 11(b). We find that there are no significant differences between three models sizes we consider, which might be because Meta-world visual observations are not too complex. It would be interesting to investigate the effect of model sizes in more complex environments.

DreamerV2 with ViT In order to demonstrate that performance gain from our approach does not solely come from employing ViT instead of CNN, we evaluate DreamerV2 w/ ViT that replaces CNN encoder and decoder with ViT encoder and decoder, in Figure 11(c). We find DreamerV2 w/ ViT exhibits severe instability during training, and sometimes becomes completely unable to solve the tasks. We conjecture this might be because ViT suffers from unstable training without large data and regularization [14], which makes it difficult to learn world models end-to-end.



(a) Inputs to world models (b) Model size (c) DreamerV2 with ViT

Figure 11: Learning curves on three manipulation tasks from Meta-world that investigate the effect of (a) inputs to world models and (b) autoencoder model sizes. (c) We also report the performance of DreamerV2 with CNNs and ViT. The solid line and shaded regions represent the mean and stratified bootstrap confidence interval across 12 runs.

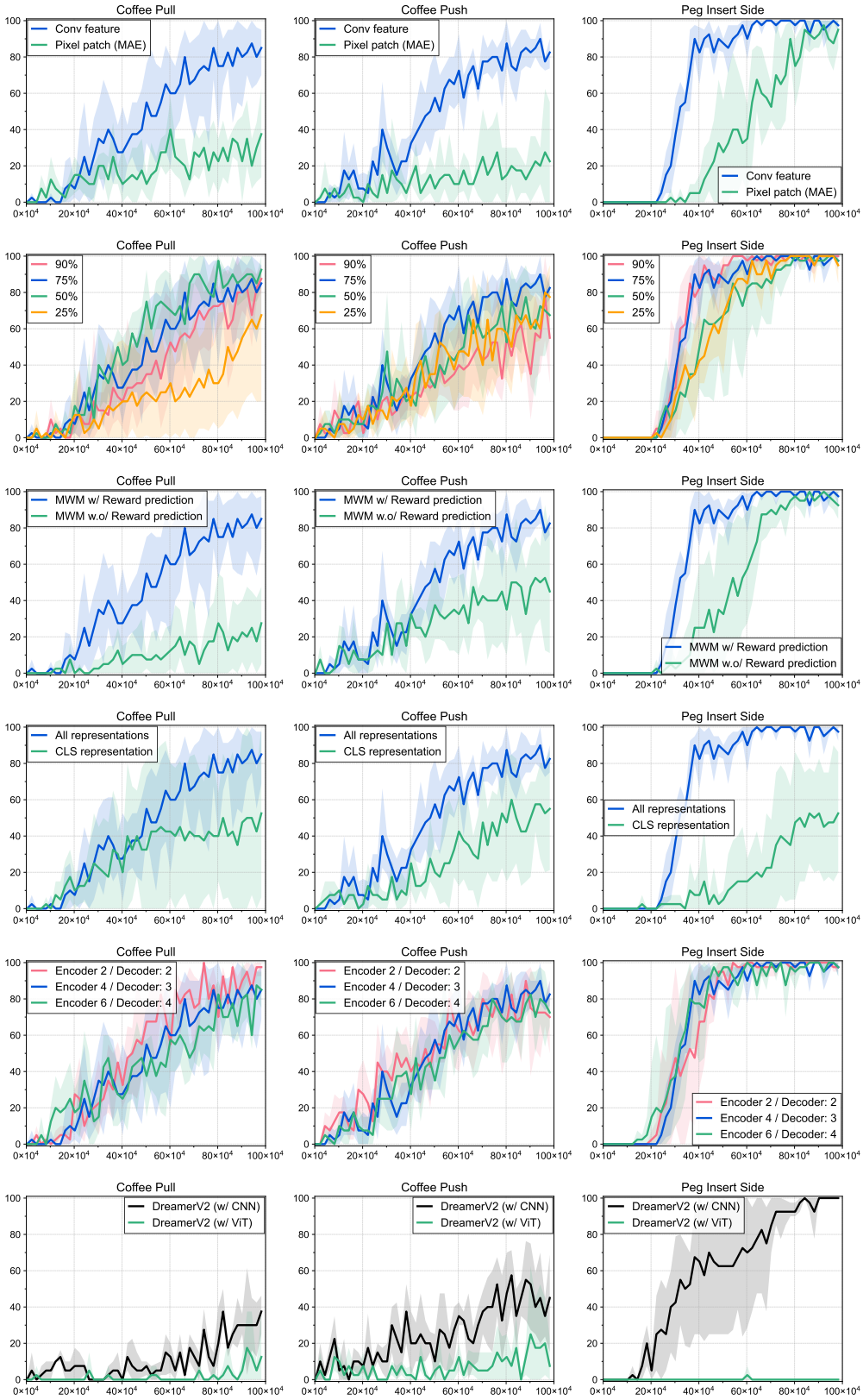
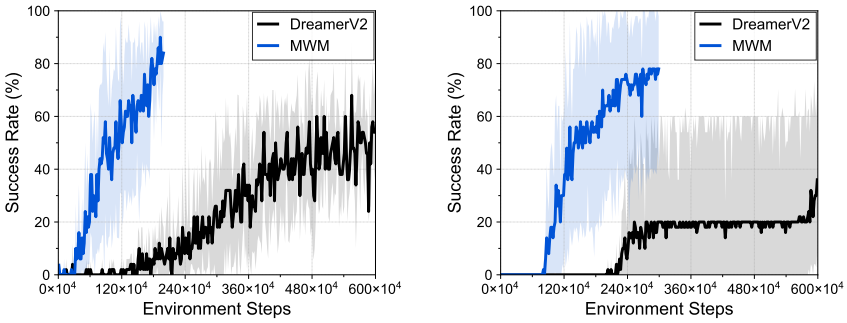


Figure 12: Learning curves on individual tasks used in ablation studies and analysis. The solid line and shaded regions represent the mean and bootstrap confidence interval across 4 runs.

J Additional Meta-world Experiments with Longer Environment Steps

We provide additional experiments that investigate the performance of DreamerV2 and MWM with more samples in Figure 13. Specifically, we report the performance of DreamerV2 over 6M environment steps, and find that DreamerV2 cannot outperform MWM even with much larger number of samples. This shows our approach allows for solving the tasks in a sample-efficient way with better asymptotic performance.



(a) Hand Insert (b) Shelf Place

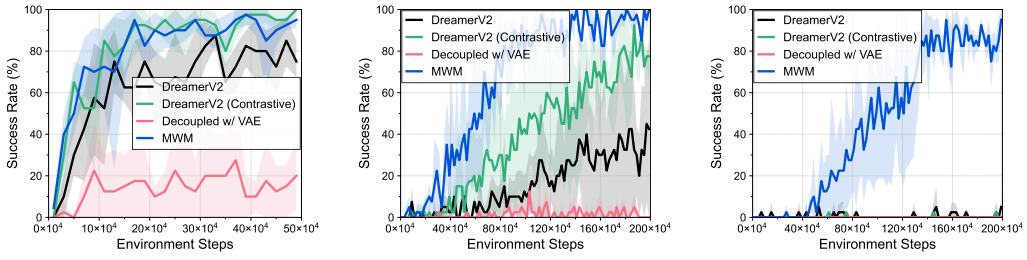
Figure 13: Learning curves on (a) Hand Insert and (b) Shelf Place tasks from Meta-world as measured on the success rate. The solid line and shaded regions represent the mean and bootstrap confidence intervals, respectively, across five runs.

K Comparison with Additional Baselines

In this section, we provide additional experiments that compare MWM with additional baselines. Specifically, we compare MWM against (i) a baseline that decouples visual representation and dynamics learning by learning the model on top of frozen VAE [12] representations and (ii) a baseline that learns representations via contrastive learning [9] instead of reconstruction in Figure 14.

Comparison with VAE We find that the decoupled baseline based on VAE fails to solve most of the tasks, while MWM solves the target tasks. This shows that our representation learning method is crucial for performance improvement from the decoupling approach. It would be an interesting investigation to consider additional baselines that extracts keypoints and learns a dynamics model on top of them [78, 79, 80, 81].

Comparison with contrastive baseline Moreover, we also observe that DreamerV2 with contrastive representation learning outperforms DreamerV2 with reconstruction, which shows that contrastive learning could be better in capturing fine-grained details from small objects. But importantly we find MWM still outperforms DreamerV2 (Contrastive) especially on Pick Place task, which demonstrates the effectiveness of our decoupled approach for solving challenging manipulation tasks.



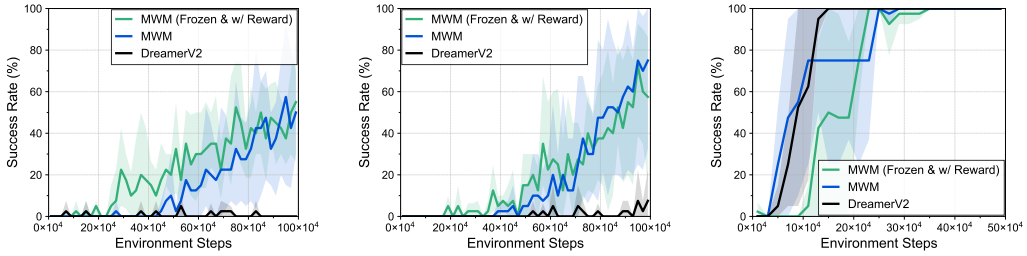
(a) Reach (b) Push (c) Pick Place

Figure 14: Learning curves on (a) Reach, (b) Push, and (c) Pick Place tasks from Meta-world as measured on the success rate. The solid line and shaded regions represent the mean and bootstrap confidence intervals, respectively, across four runs.

L Generalizability of Representations Learned with Reward Prediction

As we mentioned in Section 6, representation learning only with task-irrelevant information would be an interesting direction to further improve the applicability of our approach to diverse setups. Nevertheless, we remark that reward prediction on a specific task can also encourage visual representations to capture task-irrelevant information useful for solving various manipulation tasks. For instance, rewards are often designed to contain information about robot arm movements (*e.g.,* translations and rotations) and fine-grained details about objects, which can be shared across various tasks.

Transfer to unseen tasks In order to empirically support this, we report the experimental results where we utilize frozen representations trained on Push task for solving manipulation tasks with a difference to push task: (i) Push Back that requires moving a block to a different position, (ii) Pick Place that requires picking up the block, and (iii) Drawer Open that contains unseen drawer object in the observation. As shown in Figure 15, we observe that performance with frozen representations from unseen task can be similar to or better than the performance of MWM trained from scratch, which shows that representations learned with reward prediction can be versatile.



(a) Pick Place (b) Push Back (c) Drawer Open

Figure 15: Learning curves on (a) Pick Place, (b) Push Back, and (c) Drawer Open tasks from Meta-world as measured on the success rate. The solid line and shaded regions represent the mean and bootstrap confidence intervals, respectively, across four runs.

State regression analysis To further investigate how reward prediction affects the quality of learned representations, we provide additional experiments where we train a regression model to predict proprioceptive states available from the simulator. Specifically, we first train MWM on Meta-world Push task with and without reward prediction, and train a regression model to predict the states in Push (seen) and Pick-place (unseen) tasks on top of frozen autoencoder representations from Push task. In Figure 16, we observe that representations trained with reward prediction consistently achieve low prediction error. This supports that reward prediction can indeed provide useful information for robotic manipulation.

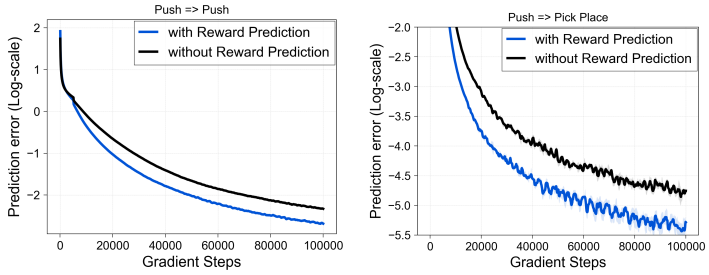


Figure 16: Learning curves of regression models that predicts proprioceptive states using the frozen representations pre-trained from Push task on (a) Push and (b) Pick Place tasks. The solid line and shaded regions represent the mean and bootstrap confidence intervals, respectively, across four runs.