Adapting World Models with Latent-State Dynamics Residuals

JB Lanier Kyungmin Kim Armin Karamzade Yifei Liu [†] Ankita Sinha [†] Kat He [†] Davide Corsi Roy Fox

Abstract

Simulation-to-reality reinforcement learning (RL) faces the critical challenge of reconciling discrepancies between simulated and real-world dynamics, which can severely degrade agent performance. A promising approach involves learning corrections to simulator forward dynamics represented as a residual error function, however this operation is impractical with high-dimensional states such as images. To overcome this, we propose ReDRAW, a latent-state autoregressive world model pretrained in simulation and calibrated to target environments through residual corrections of latent-state dynamics rather than of explicit observed states. Using this adapted world model, ReDRAW enables RL agents to be optimized with imagined rollouts under corrected dynamics and then deployed in the real world. In multiple vision-based MuJoCo domains and a physical robot visual lane-following task, ReDRAW effectively models changes to dynamics and avoids overfitting in low data regimes where traditional transfer methods fail.

1. Introduction

Training robot control policies with reinforcement learning (RL) in real-world environments is inherently expensive, time-consuming, and risky because it requires extensive interactions with physical systems. Simulation provides a promising alternative as it offers a controlled, cost-effective, and parallelizable setting for generating data and training capable policies. However, leveraging simulated environments effectively is challenging due to inaccuracies in their representation of agent observations and dynamics. These inaccuracies create a **sim-to-real gap**, where simulated environments fail to capture every relevant detail of real-world physics. This gap arises when real-world dynamics are only partially understood or are too expensive to model accurately. As a result, agents trained in simulation often struggle to transfer their policies directly to real-world settings without additional adaptation (Kaufmann et al., 2023).

Department of Computer Science, University of California Irvine. [†]Work done while at UCI. Correspondence to: JB Lanier <jblanier@uci.edu>.

One approach to addressing this gap is to use a small amount of real-world data to learn corrections to simulated transition models, known as *residual dynamics corrections*. These corrections adjust the simulated dynamics to better match real-world behavior, allowing for more accurate training of control policies (Kaufmann et al., 2023; Schperberg et al., 2023; Golemo et al., 2018). However, this approach relies on the ability to efficiently learn corrections, which is difficult when the state information is represented in high-dimensional formats such as images. In these cases, significant feature engineering is often required to extract compact and meaningful state representations for learning residuals.

This work introduces a novel method for learning residual dynamics corrections directly in the **latent state space** of world models, eliminating the need for explicit feature engineering. Specifically, we build on latent-state world models such as Dreamer (Hafner et al., 2023) and TD-MPC (Hansen et al., 2023), which encode high-dimensional observations into compact latent states. These latent states can then be used to predict future dynamics, rewards, policy values, and optimal actions. World models enable RL agents to plan and gather experience using synthetic trajectories in latent space, significantly reducing the need for real-world interactions.

We propose a new world-model architecture, **DRAW** (**D**ynamics-**R**esidual **A**daptable **W**orld model), that encodes observations into a **generalizable latent space** and supports efficient residual learning. After pretraining DRAW on simulated data, its weights are frozen to provide a stable latent representation. A small offline dataset of real-world trajectories is then used to learn a residual function in the latent space. This function corrects the world model's dynamics, enabling it to more accurately represent real-world behavior. We refer to this residual-calibrated model as **Re**ctified DRAW (**ReDRAW**). RL agents can be trained with ReDRAW using imagined rollouts, producing policies that perform well in the real environment. Importantly, we do not require reward labels from the real environment to make this calibration, extending ReDRAW's applicability to real scenarios where rewards are only available in simulation.

We evaluate ReDRAW on four vision-based DeepMind Con-

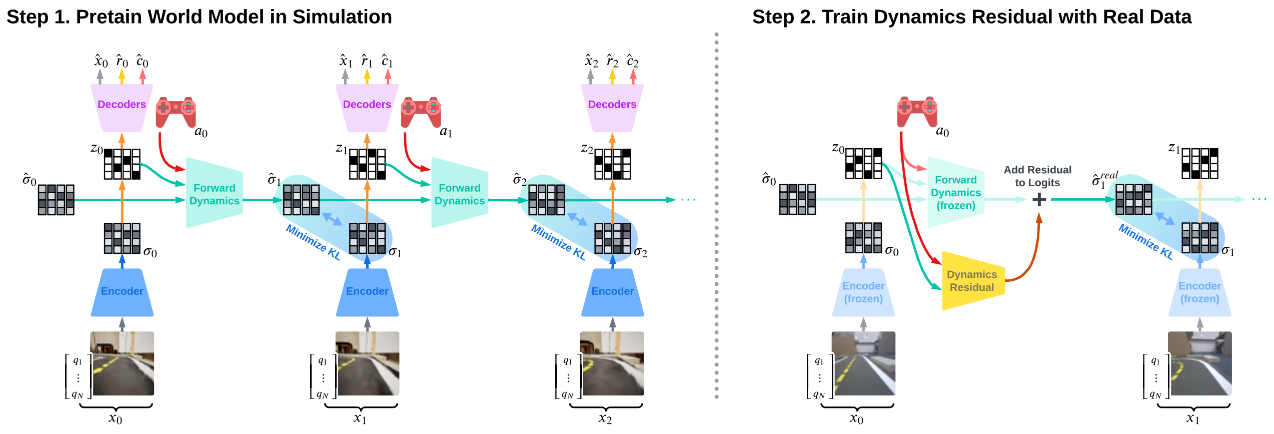


Figure 1. (**Left**) The DRAW world model is trained to encode states into a latent representation, from which states, rewards, terminations, and future latent states are predicted. An RL agent is trained in the world model via synthetic rollouts. (**Right**) World model dynamics can be calibrated to a target environment by training a residual error correction on latent state dynamics predictions, allowing the RL agent to be trained under rectified dynamics.

trol Suite (DMC) environments and further demonstrate the real-world usability of ReDRAW in sim-to-real applications by adapting from simulation to a physical real-time visual-navigation task on the Duckiebot low-cost robot (Paull et al., 2017). Our experimental results suggest that ReDRAW outperforms traditional transfer learning methods in small data regimes to adapt to mismatched dynamics and avoids overfitting without early stopping. In real robot experiments, ReDRAW successfully performs simulation-to-reality dynamics adaptation with only 10K real steps ($\sim$17-minute demonstration), transferring from simulation with synthetic visual inputs to real-world images collected on the robot.

Contributions

1. We propose a new world-model architecture for dynamics adaptation in fully observable visual-control domains. DRAW encodes all state information into a single generalizable latent space suitable for transfer in low-data regimes.

2. We demonstrate that the ReDRAW architectural extension can learn residual corrections in the latent space of DRAW to efficiently transfer between domains with mismatched dynamics, using only a small amount of offline target-domain data without reward labels.

3. We show that our method adapts dynamics from simulation to reality while also zero-shot transferring latent-state encoders from synthetic to real robot images.

4. Additionally, we open-source the code for our Unreal Engine (Epic Games, 2024) Duckiebot visual-control simulator to help facilitate further sim-to-real transfer research. Code and videos are available at `https://redraw.jblanier.net`.

2. Related Work

This research lies at the intersection of sim-to-real dynamics transfer and RL with latent-state world models.

2.1. Transferring Dynamics with Explicit Representations

Sim-to-real transfer of dynamics aims to adapt existing simulators or dynamics models used for planning and policy optimization to better match real-world environments. One way to transfer dynamics from simulation to reality is to calibrate predefined simulator physics parameters to match the target environment, either directly from real data (Si et al., 2022) or as a correction to existing parameters (Allevato et al., 2020). However, doing so can be insufficient if no good approximation of the real environment exists in the space spanned by the allowed range of these parameters. In such cases, a more expressive modification of the simulator state transition function may be needed.

Along these lines, Ball et al. (2021) and Arcari et al. (2023) calibrate linear error models on simulator transition dynamics using real data for policy adaptation and, respectively, model predictive control. Similarly, Mallasto et al. (2021) use affine transport to adapt simulator state dynamics models to real domains. Golemo et al. (2018) train an LSTM conditioned on state–action history to predict a state transition residual, and Schperberg et al. (2023) efficiently adapt a neural-network state-dynamics residual by using Unscented Kalman Fitlering. Kaufmann et al. (2023) employ k-nearest neighbor regression and Gaussian process residuals on transition dynamics and state encodings to calibrate their simulator for drone racing at an expert human level.

Each of these methods relies on the assumption that the environment state can be represented with a compact vector representation with which a generalizable dynamics correction can be learned with a relatively low-complexity model and small real-data requirements. We consider the case where the components of the state are instead in a *high-dimensional* format like images and we do not have a predefined mapping from these states to such a necessary compact vector representation. To adapt simulation transition dynamics under these more difficult conditions, we propose to learn a latent-state world model of the simulation and then train a residual correction on the world model's dynamics to match transitions in the real environment.

2.2. World Models with Latent State Spaces

World models (Ha & Schmidhuber, 2018) with latent state spaces are environment models in which planning and policy learning can be more efficient than with environment states due to a succinct representation of environment states and dynamics. Dreamer (Hafner et al., 2019; 2020; Wu et al., 2022; Hafner et al., 2023) models environments in the stochastic POMDP (Cassandra, 1998) by encoding observations as latent states and reconstructing future latent states, rewards, and observations. The Dreamer architecture allows agents to then train on synthetic experience by rolling out "imagined" trajectories inside of the world model. TD-MPC (Hansen et al., 2022; 2023) models deterministic fully observable MDPs by similarly reconstructing future latent states and rewards, as well as task value functions. TD-MPC2 (Hansen et al., 2023) has shown good results learning shared features from a suite of environments to quickly transfer to new ones, while we focus on transferring from a single environment to a similar target environment by avoiding overfitting to limited data.

Concerning exploration with world models, collecting diverse source trajectories was crucial in our experiments for learning transferable features and dynamics. To achieve this, we use Plan2Explore (Sekar et al., 2020), a method compatible with both Dreamer and our proposed DRAW architecture, which trains an auxiliary RL agent alongside the exploit policy to maximize model uncertainty in latent dynamics predictions, promoting wide-reaching exploration.

2.3. Domain Randomization

Domain randomization is widely used for sim-to-real transfer by exposing policies to diverse variations in images (Tobin et al., 2017; James et al., 2019) or dynamics (Peng et al., 2018; Mehta et al., 2020). However, training on a broad distribution of environment conditions can yield an overly conservative policy or require system identification at test time, potentially degrading test-time performance.

We apply limited camera-parameter randomization in Duckiebots to address perception gaps but focus on adapting to unforeseen dynamics. Rather than relying solely on randomization, we use a residual method to correct dynamics mismatches, achieving more flexible adaptation beyond predefined perturbations.

3. Preliminaries and Problem Definition

In this work, we consider two Markov Decision Processes (MDPs), denoted as M_{sim} and M_{real}, which share the same state space, action space, and reward function, but differ in their transition dynamics. Formally, each MDP is defined by a tuple $M_i = (X, A, R, \gamma, P_i)$, with a shared state space X, action space A, reward function upon entering a state $R : X \to \mathbb{R}$, discount factor $\gamma \in [0, 1)$, and stochastic transition function P_i for $i \in \{\text{sim}, \text{real}\}$.

Our objective is to find a policy π_{real} that achieves good expected discounted cumulative reward in M_{real}, $J_{\pi,\text{real}} = \mathbb{E}_{\pi,P_{\text{real}}}[\sum_{t=0}^{\infty} \gamma^t R(x_t)]$. To capture logistic challenges common in real robot settings, we have access to a limited amount of offline reward-free data (x_t, a_t, x_{t+1}) from M_{real}, and to make up for this, we can collect a large amount of online experience $(x_t, a_t, x_{t+1}, r_{t+1})$ in M_{sim}.

Our work addresses the problem of rectifying one-step dynamics predictions to align them with actual outcomes in the real environment. In this work, we consider MDPs where all of the information necessary to predict future dynamics and rewards can be extracted from the agent's immediate observation of the world state, x_t. A world model will then encode x_t into a latent state z_t, which in an MDP can also be a state of the latent dynamics. To maintain this assumption of full observability in our experiments on image-based domains, we supplement visual inputs with vectors of otherwise absent complementary information such as velocity values. We leave relaxing ReDRAW's limitation to MDPs only and addressing the partially observable case for future research.

Additionally, our proposed residual method assumes that only transition dynamics vary between M_{sim} and M_{real}, and that the agent's perceptual processing itself can effectively transfer zero-shot between the two environments. Independent of our latent dynamics residual method, we demonstrate that it is possible to overcome image distribution shifts with world models between our Duckiebot simulation and real robot domains by using image augmentations with an asymmetric state encoder/decoder objective similar to Kim et al. (2024) and a digital-twin simulator that leverages Gaussian splatting (Kerbl et al., 2023) with mild visual domain randomization. We note that the assumption of a shared state space does not lose generality, in principle, because the dynamics can be modeled as leading to state subspaces that are disjoint between the environments; but in practice, visual transfer does help to keep the residual simple.

4. Method

In this section, we describe our MDP world model architecture DRAW (Figure 1, Left) and its counterpart with calibrated dynamics, ReDRAW (Figure 1, Right). We first define the DRAW model, how it represents latent states and dynamics, and how it is trained. Then we describe how we facilitate sample efficient transfer learning of dynamics by training a residual error correction on latent-state transitions, creating the ReDRAW world model.

4.1. DRAW Architecture and Pretraining

We use DRAW to model an MDP by encoding state inputs into a compressed stochastic latent representation using variational inference. Similar to DreamerV3 (Hafner et al., 2023), our latent representation is trained via objectives for state and reward reconstruction along with future latent-state prediction. We then train an actor–critic reinforcement-learning agent on latent-state inputs by autoregressively rolling out synthetic trajectories as experience and using reconstructed rewards as a learning signal. Finally, the actor can be deployed to the environment by encoding immediate state inputs as latent states and providing these encodings to the actor. Figure 1 (Left) depicts connections during DRAW world model training, while Figure 5 in Appendix A depicts actor–critic training and deployment.

We model the latent state purely as a single stochastic multi-categorical discrete variable $z_t \in \mathcal{Z}$. z_t is a K-tuple of conditionally independent categorical variables, each represented as a 1-hot vector of length N. We denote z_t as the latent state encoded from the immediate state x_t (1) and $\hat{z}_t$ as the latent state predicted via world model dynamics from the previous latent state and action (5). We denote $\hat{u}_t \in \mathbb{R}^{K\times N}$ as the logits for the multi-categorical distribution of $\hat{z}_t$ and $\hat{\sigma}_t = \mathrm{softmax}(\hat{u}_t)$ as the K concatenated normalized probability vectors. To estimate gradients in the sampling step for z_t or $\hat{z}_t$, we use the straight-through estimator (Bengio et al., 2013; Hafner et al., 2020).

By compressing all state information into a single discrete representation z_t, we aim to provide a well-structured encoding of the underlying state x_t, enabling the learning of generalizable functions, such as residual corrections, from limited data using z_t as input. As illustrated in Figure 1 (Left), we define our DRAW world model and actor–critic RL agent, respectively parameterized by θ and ϕ, as follows:

$$\begin{aligned}
\text{State Encoder} \quad & z_t \sim q_\theta(z_t|x_t) && (1)\\
\text{Forward Dynamics} \quad & \hat{u}_t = f_\theta(z_{t-1}, \hat{\sigma}_{t-1}, a_{t-1}) && (2)\\
\text{Forward Belief} \quad & \hat{\sigma}_t = p_\theta(\hat{z}_t|z_{t-1}, \hat{\sigma}_{t-1}, a_{t-1}) && (3)\\
& \quad = \mathrm{softmax}(\hat{u}_t) && (4)\\
\text{Forward Sample} \quad & \hat{z}_t \sim \mathrm{MultiCategorical}(\hat{\sigma}_t) && (5)\\
\text{Reward} \quad & \hat{r}_t \sim p_\theta(\hat{r}_t|z_t) && (6)\\
\text{Continue} \quad & \hat{c}_t \sim p_\theta(\hat{c}_t|z_t) && (7)\\
\text{State Decoder} \quad & \hat{x}_t \sim p_\theta(\hat{x}_t|z_t) && (8)\\
\text{Policy} \quad & a_t \sim \pi_\phi(a_t|z_t) && (9)\\
\text{Value Function} \quad & v_t = V^\pi_\phi(z_t). && (10)
\end{aligned}$$

We represent all functions in DRAW as multi-layer perceptrons (MLPs) except for image components of the state encoder and decoder, which are convolutional (CNNs). Interestingly, we found that providing $\hat{\sigma}_t$ as an input to the forward dynamics function f_θ significantly increased our downstream adaptation performance. We speculate that this is because $\hat{\sigma}_t$ helps provide a gradient signal for learning features relevant for long-term dynamics predictions without adding additional dimensionality to state prediction outputs. We provide ablations on this design choice in Appendix D.

We optimize DRAW on M_{sim} with a prediction loss $\mathcal{L}_{\text{pred}}$ to reconstruct states, rewards, and terminations, as well as a dynamics loss $\mathcal{L}_{\text{dyn}}$ and a representation loss $\mathcal{L}_{\text{rep}}$ to learn latent-state dynamics under a predicable representation. Drawing subtrajectories ζ from a buffer of interaction experience, the world-model loss function $\mathcal{L}(\theta)$ is:

$$\mathcal{L}(\theta) = \mathbb{E}_{q_\theta(z_{1:T}|\zeta)}\left[\sum_{t=1}^{T} \beta_{\text{pred}}\mathcal{L}^t_{\text{pred}}(\theta) + \beta_{\text{dyn}}\mathcal{L}^t_{\text{dyn}}(\theta) + \beta_{\text{rep}}\mathcal{L}^t_{\text{rep}}(\theta)\right], \tag{11}$$

where T is the length of ζ, and for $t = 1, \ldots, T$:

$$\mathcal{L}^t_{\text{pred}}(\theta) \doteq -\ln p_\theta(x_t|z_t) - \ln p_\theta(r_t|z_t) - \ln p_\theta(c_t|z_t) \tag{12}$$

$$\mathcal{L}^t_{\text{dyn}}(\theta) \doteq [\mathbb{D}[q_{\bar{\theta}}(z_t|x_t) || p_\theta(\hat{z}_t|z_{t-1}, \hat{\sigma}_{t-1}, a_{t-1})]]_1 \tag{13}$$

$$\mathcal{L}^t_{\text{rep}}(\theta) \doteq [\mathbb{D}[q_\theta(z_t|x_t) || p_{\bar{\theta}}(\hat{z}_t|z_{t-1}, \hat{\sigma}_{t-1}, a_{t-1}))]]_1, \tag{14}$$

with $[\cdot]_1$ denoting clipping to 1 any value below 1, $\mathbb{D}$ the Kullback–Leibler divergence, and $\bar{\theta}$ a stopped-gradient copy of θ.

We train the actor–critic agent with the same procedure and losses as DreamerV3, providing the DRAW world model state $\hat{z}_t$ as agent inputs during imagined rollouts and z_t during evaluation. When training the actor–critic, we seed synthetic rollouts with starting states x_0 drawn from the same experience buffer as used for world-model training. We do not backpropagate value gradients through dynamics, and we train the policy using the Reinforce objective (Williams, 1992) with normalized returns and critic baselines (Hafner et al., 2023).

We alternate mini-batch updates between the world model and the actor–critic. During source environment pretraining, updates are interleaved with online data collection. Since we cannot fully predict which trajectories in M_{sim} will best facilitate learning transferable features and dynamics for M_{real}, we employ Plan2Explore (Sekar et al., 2020) to provide intrinsically motivated exploration, ensuring that we collect a large and highly diverse set of source-environment trajectories.

4.2. Adaptation via Latent Dynamics Residuals

After pretraining the DRAW world model online in the M_{sim} environment with a large amount of data, we propose the ReDRAW architecture and method to use a small offline dataset of transitions from the target environment to calibrate DRAW's dynamics to match M_{real} using a latent-state error residual.

We model the dynamics residual using an MLP δ_ψ that predicts a correction $\hat{e}_t$ to the forward-dynamics logit vector $\hat{u}_t$. This correction produces a modified transition distribution $\hat{\sigma}_t^{real}$, from which forward latent-state predictions $\hat{z}_t^{real}$ are sampled to approximate M_{real}. That is, we model the calibrated dynamics as:

$$\begin{aligned}
\hat{u}_t &= f_\theta(z_{t-1}, \hat{\sigma}_{t-1}^{real}, a_{t-1}) && (15)\\
\hat{e}_t &= \delta_\psi(z_{t-1}, a_{t-1}) && (16)\\
\hat{\sigma}_t^{real} &= p_{\theta,\psi}(\hat{z}_t^{real} | z_{t-1}, \hat{\sigma}_{t-1}^{real}, a_{t-1}) && (17)\\
&= \text{softmax}(\hat{u}_t + \hat{e}_t) && (18)\\
\hat{z}_t^{real} &\sim \text{MultiCategorical}(\hat{\sigma}_t^{real}). && (19)
\end{aligned}$$

To train the residual on real data, we freeze the world-model weights θ and only optimize the parameters ψ of the residual network δ_ψ. In the transfer phase, we only optimize the actor–critic agent and a new loss $\mathcal{L}_\delta(\psi)$ on the rectified world-model dynamics. Our objective is to predict corrections $\hat{e}_t$ of $\hat{u}_t$ so that our new dynamics predictions $\hat{\sigma}_t^{real}$ match the observed encoder distribution over latent states collected in M_{real}. The loss function for the residual is:

$$\begin{aligned}
\mathcal{L}_\delta(\psi) = \mathbb{E}_{q_{\bar{\theta}}(z_{1:T}|\zeta^{real})}\Bigg[\sum_{t=1}^{T} \mathbb{D}[q_{\bar{\theta}}(z_t|x_t)||\\
p_{\bar{\theta},\psi}(\hat{z}_t^{real}|z_{t-1}, \sigma_{t-1}^{real}, a_{t-1})]\Bigg]. \qquad (20)
\end{aligned}$$

Because we consider fully observable environments, the target encoder latent-state distribution $q_\theta(z_t|x_t)$ depends solely on x_t and can be frozen after pretraining in M_{sim} if the collected source-environment data adequately covers the state space.

Finally, given the ReDRAW world model with dynamics adapted to match M_{real}, the actor–critic can learn a high-performing policy for the new environment by training in the world model under the new rectified dynamics, using $\hat{z}_t^{real}$ as input during training. In our experiments, we still alternate agent and world-model training during adaptation, although in principle, the agent could be trained after world-model training is stopped.

Notably, the latent-state representation for ReDRAW in M_{real} is unchanged from DRAW in M_{sim}. As a result, the frozen DRAW M_{sim} reward function $p_\theta(\hat{r}_t|\hat{z}_t)$ can be reused in world-model rollouts to train the ReDRAW agent with $p_\theta(\hat{r}_t|\hat{z}_t^{real})$ in M_{real}, eliminating the need for reward data from M_{real}. This is particularly beneficial since setting up a reward recording system in real-world scenarios, such as robotics, often requires costly and complex setups like additional sensors or feedback mechanisms, which may be infeasible in certain environments.

5. Experiments

We evaluate ReDRAW in two distinct settings: (1) adapting from DeepMind Control (DMC) (Tassa et al., 2018) environments to modified counterparts with changed physics, and (2) transferring from a simulation in Unreal Engine to a real robot visual lane-following task using the Duckietown platform. Our experiments address three main questions:

1. How do latent-space residuals compare to traditional finetuning methods in correcting world-model dynamics under limited target-domain data?
2. How do data quantity and collection policies influence transfer performance?
3. Can ReDRAW effectively close the sim-to-real gap in a physical robotics task with visual inputs?

5.1. DeepMind Control Experiments

5.1.1. DMC Domains

We first consider four pairs of source and target environments from the DMC suite, each pair having the same state and reward structure but mismatched dynamics. We use original environments from DMC as sources, while the target environments introduce physics modifications such as applied wind, external torque, or reversed actions. For a detailed description, refer to Appendix F. These differences in dynamics between source and target environments are substantial enough to require policy adaptation for optimal performance. Although dynamics differ between source and target, the state spaces, reward functions, and termination conditions remain unchanged.

To pretrain on each source environment, we collect 9 million environment steps (4.5e6 decisions steps with an action repeat of 2) using Plan2Explore (Sekar et al., 2020), which

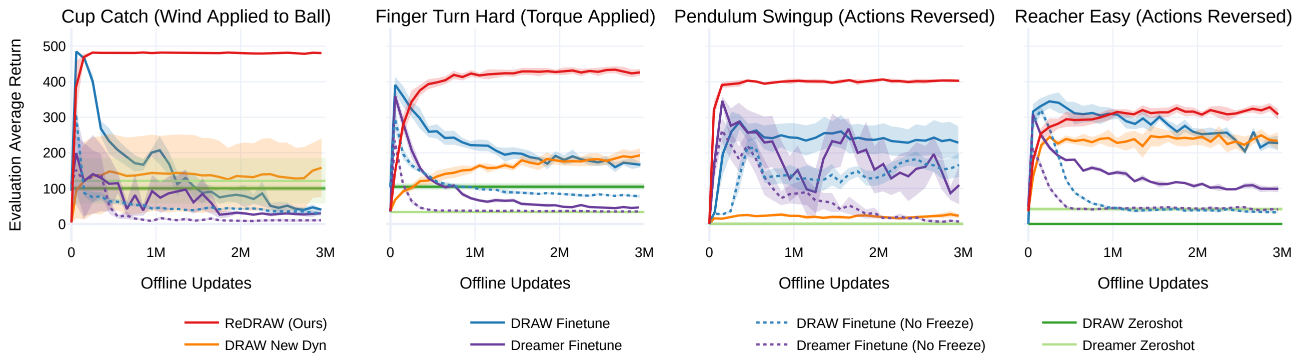


Figure 2. Average evaluation episode return transferring from each DMC environment to a modified variant of it given $4e4$ offline target environment transition samples. Shaded regions indicate the standard error of the mean over 4 seeds for each method. ReDRAW consistently achieves high returns in the target environments and avoids overfitting.

promotes diverse state visitation rather than narrowly exploiting the original environment's reward function. After this phase, we adapt to each target environment using a small offline dataset of 40K decision steps (equivalent to 80 episodes), gathered by an expert policy in the target domain.

5.1.2. Comparison with Finetuning

We compare ReDRAW with several baselines that attempt to adapt a pretrained world model to the new domain. Critically, except where noted with *, the methods we test do not use reward labels or train with a reward-reconstruction objective during the adaptation phase. These baselines include:

DRAW/DreamerV3 Zeroshot: We take the source-trained DRAW or DreamerV3 agent and deploy it in the target environment without any adaptation.

DRAW/DreamerV3 Finetune: The world model and agent are finetuned on the target domain offline dataset. To mitigate overfitting on the small dataset, we freeze the world model encoder and decoder parameters and only retrain the agent and dynamics components. For DRAW this entails optimizing only f_θ with $\mathcal{L}_{\text{dyn}}$. Analogously, for DreamerV3, the RSSM recurrent prior and posterior components are updated while leaving the observation feature embeddings and decoders unchanged.

DRAW/DreamerV3 Finetune (No Freeze)*: Every component, including the encoder and decoder, is finetuned using all original world model loss terms. These are the only two baselines requiring access to reward data during adaptation.

DRAW New Dyn: The entire world model is frozen after pretraining, but instead of learning a residual addition to f_θ, we train a new dynamics function $\hat{\sigma}_t^{real} = g_\psi(\hat{\sigma}_t, z_{t-1}, a_{t-1})$ conditioned on the distribution predicted by the frozen source dynamics (Eq. 3). This ablation demonstrates an alternate way to leverage frozen dynamics predictions learned from the source environment. Other variations of this baseline are compared in Appendix E.

Figure 2 shows the returns in each target domain as a function of offline updates on each target dataset. Direct deployment without adaptation fares poorly in these altered dynamics. Finetuning approaches initially improve in some cases but all eventually overfit to the small dataset. The *No Freeze** variations are quicker to overfit than their partially frozen counterparts. In contrast, ReDRAW's latent-space residual method attains a sustained level of high-performance and avoids overfitting during the 3 million updates (1-3 days of training) we test on. This highlights a critical benefit of the ReDRAW transfer method: once ReDRAW reaches high performance in the target domain, it demonstrates a remarkable resistance to performance degradation. ReDRAW's ability to avoid overfitting for long periods of time makes it highly applicable to sim to real scenarios where validation testing on a real robot often cannot practically be done repeatedly and educated guesses need to be reliably made regarding stopping conditions.

ReDRAW excels at maintaining a high degree of validation performance by preserving existing dynamics predictions learned in simulation where data is abundant and using the limited target data to learn a low-complexity adjustment to those predictions. Comparing ReDRAW with *DRAW New Dyn*, we see that while both approaches utilize both the previous state and the frozen simulation dynamics predictions, the residual operation appears to play a key role in limiting the complexity of the changes made to the original world model dynamics, allowing ReDRAW to avoid overfitting.

5.1.3. Data Policies and Quantity

Figure 3 examines how pretraining and target-domain data collection methods influence ReDRAW's performance. In our default scenario, we pretrain in the source domain with Plan2Explore, and then adapt to 40K steps of expert demon-

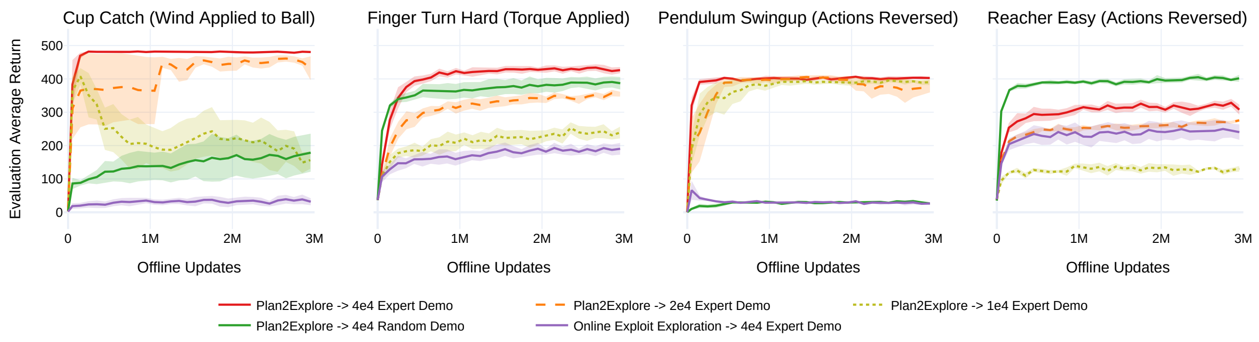


Figure 3. Impact of offline adaptation dataset size and source/target domain data collection strategies on ReDRAW. Expert demonstrations consistently provide useful target domain data for adaptation. Collecting diverse simulation experience with a method like Plan2Explore is essential for good transfer performance.

strations in the target environment. Alternatively, we test gathering source data with the training exploit policy (rather than Plan2Explore) and replacing the expert dataset with random actions in the target domain. While expert trajectories reliably lead to effective adaptation, random actions alone only occasionally suffice, suggesting that a mixture of expert data and high entropy actions may help the agent learn counterfactuals and avoid suboptimal outcomes.

We further see that reducing the number of expert demonstrations from 40K to 10K can cause ReDRAW to overfit, particularly in the Cup Catch environment. On the pretraining side, collecting a broad range of trajectories with Plan2Explore proves beneficial. When we collect data with a policy optimized only for source environment, the necessary transitions for the target domain's altered dynamics may be missed. Because it is difficult to predict which states and actions will matter under new physics, gathering diverse pretraining data is likely essential for reliable transfer.

Table 1. Mean and SEM performance on the real Duckiebots lane-following task aggregated over 5 evaluation episodes each for 4 training seeds. Agents are given 300 steps to complete a lap from a fixed starting position. *Center Offset* denotes distance from the lane center. Absence of a *Lap Time* indicates all runs either failing to complete a lap or terminating early by driving off the track.

Method	**Avg Dense Reward (↑)**	**Avg Lap Time (↓)**	**Avg Center Offset (↓)**
Transfer Sim to Unmodified Real			
ReDRAW (Ours)	**0.38** ± 0.02	**22.75** ± 0.75	**2.39** ± 0.15
DRAW Zeroshot	0.12 ± 0.03	**22.41** ± 0.73	4.79 ± 0.20
Dreamer Finetune	-0.87 ± 0.31	–	5.45 ± 1.10
Dreamer Zeroshot	-1.18 ± 0.13	–	6.86 ± 0.45
Transfer Sim to Actions Reversed Real			
ReDRAW (Ours)	**0.40** ± 0.01	**24.21** ± 0.72	**2.07** ± 0.15
DRAW Zeroshot	-2.72 ± 0.17	–	9.39 ± 0.68
Dreamer Finetune	-1.61 ± 0.27	–	7.75 ± 0.98
Dreamer Zeroshot	-2.35 ± 0.27	–	13.36 ± 0.82

5.2. Duckiebot Sim-to-Real Transfer

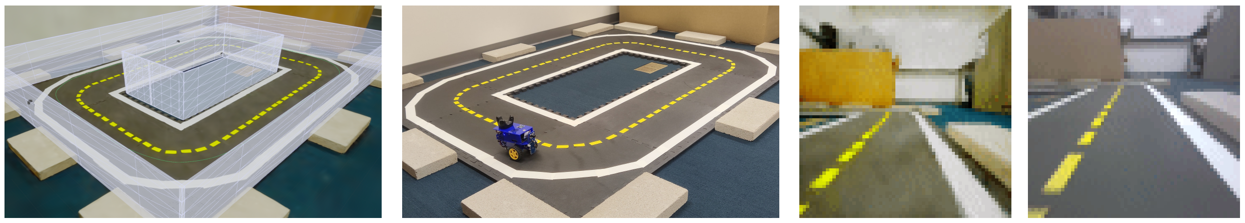
5.2.1. Task and Setup

Finally, we evaluate ReDRAW in a sim-to-real robotic lane-following task using the Duckietown platform (Paull et al., 2017). Here, the agent controls a wheeled robot to navigate around a track while remaining centered in its lane. The state space includes a forward-facing camera image plus egocentric forward and yaw velocity values, and actions defined as continuous forward and yaw target velocities in [-1, 1].

To provide a simulation to transfer from, we construct an environment in Unreal Engine using a Gaussian splat (Kerbl et al., 2023) reconstruction of the robot's environment to mimic the robot's state space. Figure 4a and 4b show the digital twin and real environment, respectively. We also implement the simulation with a rough approximation of real dynamics, although details like precise handling while driving and control rate (6Hz sim vs 10Hz real) still differ from the real robot.

The Duckiebot receives rewards proportional to its projected velocity along the lane-center path but instead incurs penalties when it deviates too far from this path. When moving forward, we also penalize the agent proportionally its yaw velocity to encourage smooth driving. Episodes terminate either when the robot leaves the track, with a large penalty applied, or after 200 steps. Exact experiment details are presented in Appendix B.

We provide the agent with reward data during simulation pretraining, and we do not provide reward labels in training data collected from the real environment. In order to measure



(a) Digital Twin Simulation (b) Real Robot Environment (c) Sim State Image (d) Real State Image

Figure 4. (a) Digital-twin simulation constructed using Gaussian splatting (Kerbl et al., 2023). (b) Real-world robot lane-following environment. (c) Simulation state image component. (d) Real-world state image component. The agent is tasked to drive quickly around the track while staying near the lane center using an egocentric camera and velocity sensor. We train our DRAW world model in simulation and calibrate its dynamics with ReDRAW on offline real trajectories, producing a successful agent in the real environment.

test-time deployment performance, during real evaluation only, we record the robot's location with an HTC Vive motion tracker to measure equivalent simulation rewards, lap times, and the robot's distance from the lane center.

5.2.2. Bridging the Sim-to-Real Vision Gap

Despite efforts to recreate the real environment, visual disparities between the simulation and real environment still exist (Figure 4c vs. 4d). Although our main focus in this paper is adapting dynamics, we employ mild per-episode domain randomization (Tobin et al., 2017) along with image augmentation (Buslaev et al., 2020) to bridge the sim-to-real vision gap. We domain-randomize the simulation camera's mounted location on the robot, camera tilt, and its field of view. We also apply image augmentations at train time to both sim and real image inputs to learn world model image encoders robust to task-irrelevant features like lighting, color hue, and lab furniture placement. Although we train with augmented inputs, our decoder reconstructs the original images as targets in $\mathcal{L}_{\text{pred}}$, thus focusing the latent-space features on task-relevant elements rather than the irrelevant augmentations we apply. We apply this asymmetric decoding objective to both DRAW and DreamerV3. As described above, ReDRAW trains its residual with augmented inputs but has no decoding objective during transfer learning.

5.2.3. Transferring to the Real Robot

We pretrain DRAW and DreamerV3 in simulation using 600K random actions followed by 1.4 million online steps with Plan2Explore. On the real robot, we collect a small offline adaptation dataset of 1e4 timesteps ($\sim$17 minutes) using human demonstrations employing a mixture of proficient driving and random safe actions. Table 1 compares performance using this offline dataset to adapt to two variations of the real environment, *unmodified real* where minor physics disparities between sim and real are the natural result of inaccurate dynamics modeling, and *actions-reversed real*, where actions (in adaptation data and deployment) are inverted, requiring large but regular adaptation to drive successfully. We adapt ReDRAW and DreamerV3 *Finetune* for 2e5 offline updates. We evaluate 5 episodes attempting a lap on the robot each for 4 training seeds.

In *unmodified real*, DRAW *zeroshot* is able to successfully drive despite never seeing real data but incurs low rewards by veering far from the lane center. DreamerV3 *zeroshot* fails, driving off the track in all lap attempts. We speculate that DreamerV3 *zeroshot* fails while DRAW *zeroshot* succeeds because DreamerV3's recurrent model observes OOD sequences under changed dynamics, resulting in inaccurate latent-state predictions. DRAW and ReDRAW are non-recurrent in deployment time and cannot suffer from this same issue. Similar to DMC experiments, DreamerV3 *Finetune* fails to adapt, possibly due to overfitting, and ReDRAW achieves significantly higher average dense rewards than DRAW *zeroshot* by training with corrected dynamics and staying close to the lane center.

In the more extreme *actions-reversed real* transfer task, ReDRAW is the only method that successfully adapts and completes laps on the real robot due to the incompatibility of zero-shot policies to this environment and the limited real data in the case of DreamerV3 *Finetune*. These results demonstrate that ReDRAW can be effectively used to adapt dynamics from simulation to reality using a limited offline real dataset without rewards, and that ReDRAW can be combined with visual adaptation methods to do so.

6. Limitations and Future Work

A potential limitation with ReDRAW is that it excels at maintaining high target-environment performance over many updates because the residual avoids overfitting due to its low complexity. This suggests that only conceptually simple changes to dynamics may effectively be modeled with low amounts of data, warranting future investigation. We additionally want to explore if residual adaptation methods can be meaningfully applied to foundation world models, efficiently converting them from generators of plausible dynamics to generators of specific dynamics.

References

Allevato, A., Short, E. S., Pryor, M., and Thomaz, A. Tunenet: One-shot residual tuning for system identification and sim-to-real robot task transfer. In *Conference on Robot Learning*, pp. 445–455. PMLR, 2020.

Arcari, E., Minniti, M. V., Scampicchio, A., Carron, A., Farshidian, F., Hutter, M., and Zeilinger, M. N. Bayesian multi-task learning mpc for robotic mobile manipulation. *IEEE Robotics and Automation Letters*, 2023.

Ball, P. J., Lu, C., Parker-Holder, J., and Roberts, S. Augmented world models facilitate zero-shot dynamics generalization from a single offline environment. In *International Conference on Machine Learning*, pp. 619–629. PMLR, 2021.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. Albumentations: Fast and flexible image augmentations. *Information*, 11 (2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL https://www.mdpi.com/2078-2489/11/2/125.

Cassandra, A. R. *Exact and approximate algorithms for partially observable Markov decision processes*. Brown University, 1998.

Epic Games. Unreal engine, 2024. URL https://www.unrealengine.com.

Golemo, F., Taiga, A. A., Courville, A., and Oudeyer, P.-Y. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pp. 817–828. PMLR, 2018.

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

Hafner, D. Dreamerv3: Mastering diverse domains through world modeling. https://github.com/danijar/dreamerv3, 2023. GitHub repository.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2020.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Hansen, N., Su, H., and Wang, X. Td-mpc2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2023.

Hansen, N. A., Su, H., and Wang, X. Temporal difference learning for model predictive control. In *International Conference on Machine Learning*, pp. 8387–8406. PMLR, 2022.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12627–12637, 2019.

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., and Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature*, 620 (7976):982–987, 2023.

Kerbl, B., Kopanas, G., Leimkühler, T., and Drettakis, G. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

Kim, K., Lanier, J., Baldi, P., Fowlkes, C., and Fox, R. Make the pertinent salient: Task-relevant reconstruction for visual control with distractions. *arXiv preprint arXiv:2410.09972*, 2024.

Mallasto, A., Arndt, K., Heinonen, M., Kaski, S., and Kyrki, V. Affine transport for sim-to-real domain adaptation. *arXiv preprint arXiv:2105.11739*, 2021.

Mehta, B., Diaz, M., Golemo, F., Pal, C. J., and Paull, L. Active domain randomization. In *Conference on Robot Learning*, pp. 1162–1176. PMLR, 2020.

Paull, L., Tani, J., Ahn, H., Alonso-Mora, J., Carlone, L., Cap, M., Chen, Y. F., Choi, C., Dusek, J., Fang, Y., et al. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1497–1504. IEEE, 2017.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.

Schperberg, A., Tanaka, Y., Xu, F., Menner, M., and Hong, D. Real-to-sim: Predicting residual errors of robotic systems with sparse data using a learning-based unscented kalman filter. In *2023 20th International Conference on Ubiquitous Robots (UR)*, pp. 27–34. IEEE, 2023.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pp. 8583–8592. PMLR, 2020.

Si, Z., Zhu, Z., Agarwal, A., Anderson, S., and Yuan, W. Grasp stability prediction with sim-to-real transfer from tactile sensing. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7809–7816. IEEE, 2022.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Wu, P., Escontrela, A., Hafner, D., Goldberg, K., and Abbeel, P. Daydreamer: World models for physical robot learning. *Conference on Robot Learning*, 2022.

A. Actor-Critic Training and Deployment

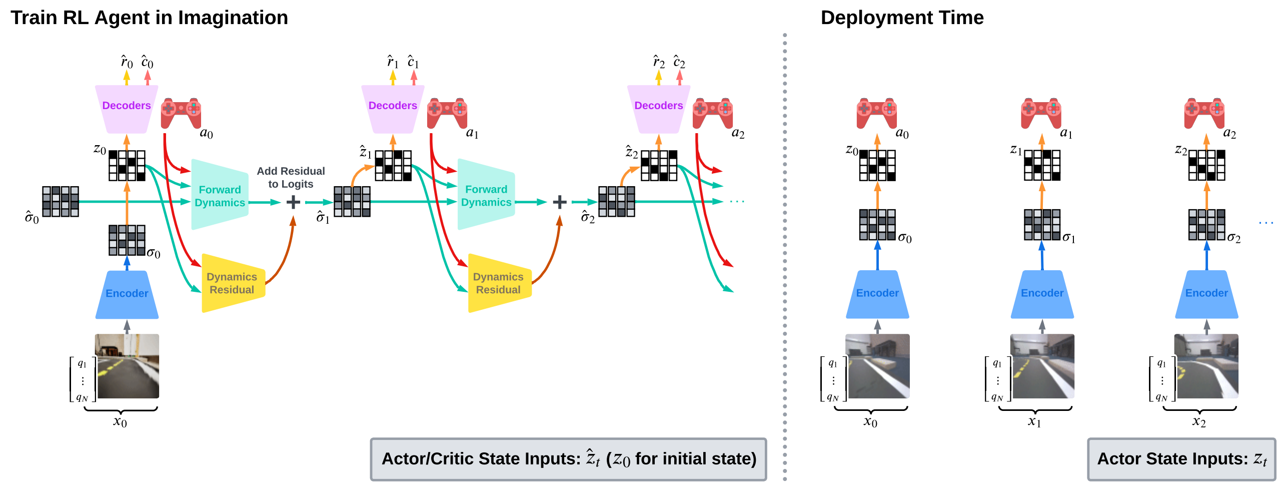
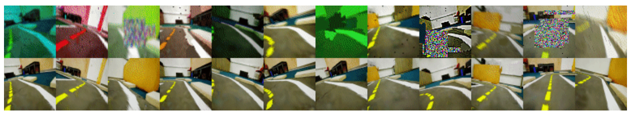

Train RL Agent in Imagination
Deployment Time
Decoders
Forward Dynamics
Dynamics Residual
Encoder
Add Residual to Logits
Actor/Critic State Inputs: $\hat{z}_t$ (z_0 for initial state)
Actor State Inputs: z_t

Figure 5. (**Left**) The actor and critic are trained by interacting with the world model. Starting from an environment state sampled from the replay buffer, the world model generates imagined rollouts using actions provided by the actor. The residual component is omitted during DRAW pretraining. (**Right**) At deployment, only the encoder and actor modules are utilized. The immediate environment state is processed by the encoder, and the actor generates an action based on z_t sampled from σ_t.

B. Duckiebots Experiment Details

Simulation Reward Details In simulation, the agent is densely rewarded at each timestep with a value in $[0, 1]$ proportional to its projected velocity along the lane center unless its location is more than 5cm from the lane center, in which case it incurs a penalty of -1. When moving forward, we additionally provide a dense penalty proportional to egocentric yaw velocity to encourage turning while at speed. The simulation episode horizon is 200 steps, slightly more than enough time to complete a lap. We do not provide a termination signal when the horizon is reached. We terminate early with a done signal and a penalty of -100 if the agent drives off the track.

Image Augmentations During simulation pretraining and offline adaptation to real data, we apply image augmentations to world model encoder inputs, but we still train decoder objectives on the original non-augmented images. Figure 6b shows original images (bottom) and their augmented counterparts (top) for both simulation and the real environment offline human demonstration dataset. In world model training for DRAW/ReDRAW and Dreamer, we apply new image augmentations to each mini-batch after it is sampled from the experience buffer.


(a) Simulation Images with Augmentations

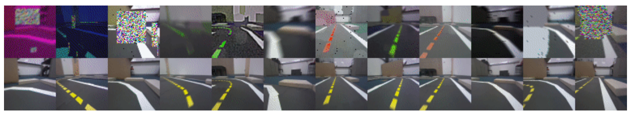

(b) Real-world Images with Augmentations

Figure 6. Comparison of image observations in simulation and the real world. (**Top**): Augmented images. (**Bottom**): Original images.

C. Learning Curves During Pretraining

Figure 7 and Figure 8 show training curves in the source environments in the DMC and Duckiebot domains, respectively. Both DRAW and DreamerV3 converge to similar performance in the source environments.

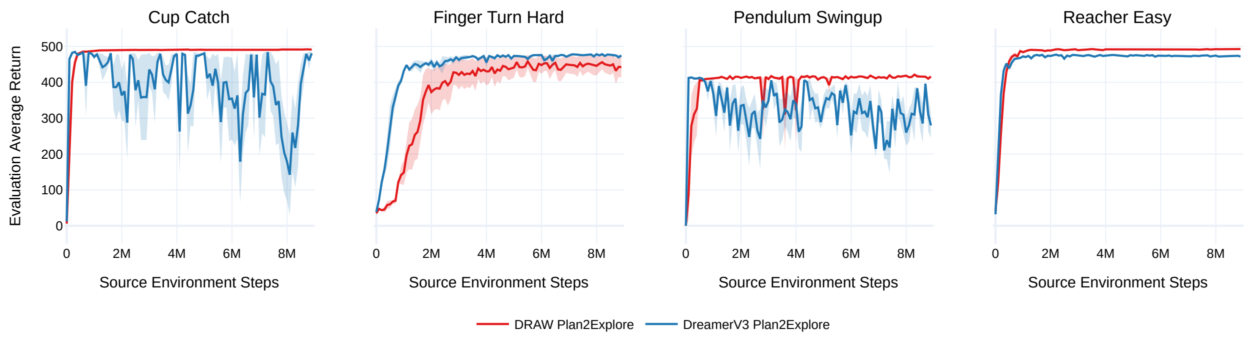


Figure 7. Training curves during pretraining for DRAW and DreamerV3 across four environments from DMC. Plan2Explore is used for data collection during pretraining. The mean and standard error are shown over 4 seeds.

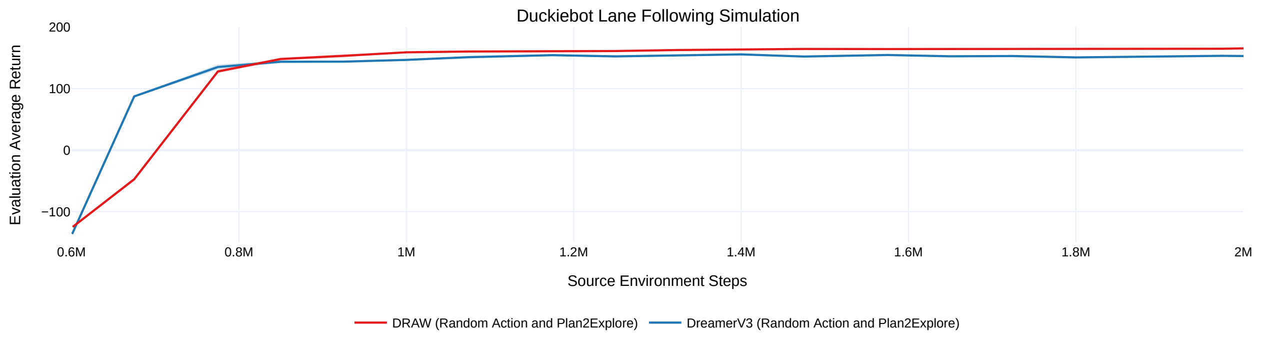


Figure 8. Training curves during pretraining for DRAW and DreamerV3 in the Duckiebot lane following simulation environment. Data collection is performed using random actions for the first 0.6M steps, followed by Plan2Explore for 1.4M steps. Each episode starts from a valid random position. The mean and standard error are shown over 4 seeds.

D. Architectural Ablations

In this section, we examine how different choices for the inputs of the DRAW forward dynamics function f_θ and the ReDRAW residual function δ_ψ affect transfer performance.

D.1. Forward Dynamics Inputs

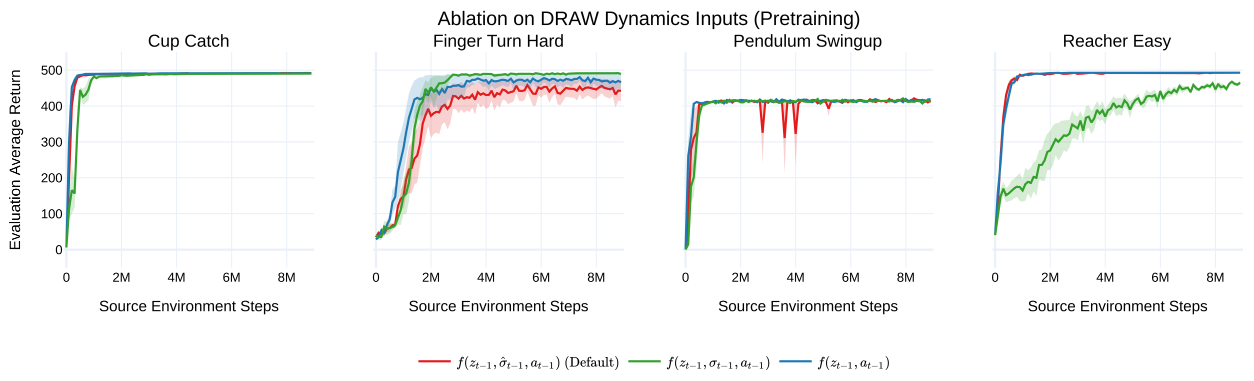
In the default DRAW architecture, f_θ is conditioned on the previous latent state z_{t-1}, the previous action a_{t-1}, and the additional input of the previous latent-state dynamic distribution $\hat{\sigma}_{t-1}$ (or $\hat{\sigma}^{real}_{t-1}$ for ReDRAW). In DMC environments, we compare this choice of inputs against two alternatives: (1) the minimal sufficient set (z_{t-1}, a_{t-1}), and (2) conditioning on the encoder latent-state distribution $\sigma_t = q_\theta(z_t|x_t)$. Figure 9a presents the performance of these different dynamics functions on source environments during DRAW Plan2Explore pretraining, while Figure 9b shows their transfer performance on target environments during offline ReDRAW adaptation.

During pretraining, most input choices yield similar source-task performance. However, during adaptation, the default configuration, $f_\theta(z_{t-1}, \hat{\sigma}_{t-1}, a_{t-1})$, consistently outperforms the alternatives, achieving and maintaining higher performance in the target environments. We hypothesize that because including $\hat{\sigma}_{t-1}$ during world model pretraining facilitates gradient propagation over multiple timesteps, this inclusion enables the learning of features that improve long-term predictions.

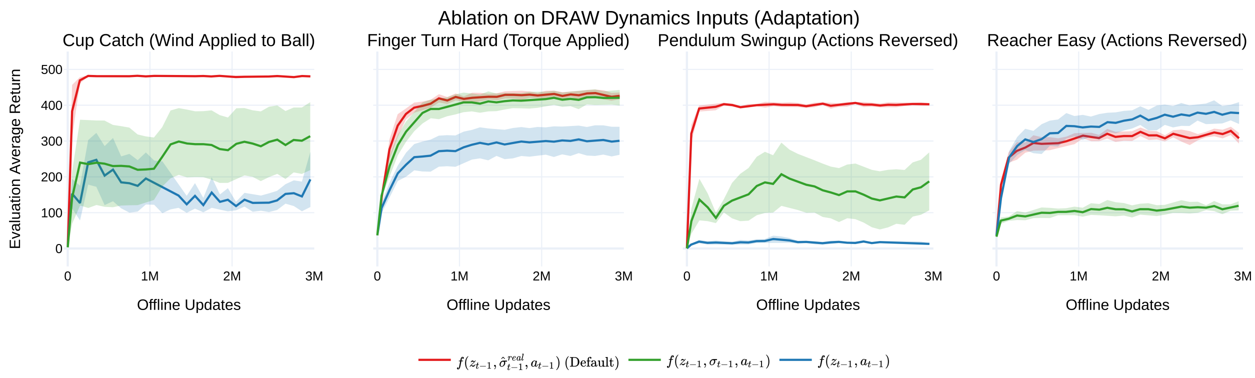
This advantage is achieved without increasing the residual's complexity, which could have otherwise negatively impacted transfer performance. During ReDRAW adaptation, $\hat{\sigma}_{t-1}^{real}$ serves as an input to $f_\theta(z_{t-1}, \hat{\sigma}_{t-1}^{real}, a_{t-1})$. While conditioning on $\hat{\sigma}_{t-1}^{real}$ increases the dimensionality of f_θ's input space, it has minimal impact on the complexity of the residual prediction δ_ψ. Since $\hat{\sigma}_t^{real}$ is already an output of the calibrated dynamics,

$$\hat{\sigma}_t^{real} = \text{softmax}(f_\theta(z_{t-1}, \hat{\sigma}_{t-1}^{real}, a_{t-1}) + \delta_\psi(z_{t-1}, a_{t-1})),$$

it can be included as an input to f_θ without increasing the dimensionality of the input or output spaces of δ_ψ. This helps maintain the residual function's simplicity, reducing the risk of overfitting.



(a) DMC source environment average return during DRAW pretraining with alternate dynamics function inputs.



(b) DMC target environment average return during ReDRAW residual adaptation with alternate dynamics function inputs.

Figure 9. Comparison of different dynamics function architectures of DRAW during pretraining (a) and adaptation (b).

D.2. Residual Inputs

Next, in Figure 10, we compare the target environment transfer performance of our default residual function, $\delta_\psi(z_{t-1}, a_{t-1})$, against two alternative input configurations. The first, $\delta_\psi(z_{t-1}, \hat{\sigma}_{t-1}^{real}, a_{t-1})$, conditions on the same inputs as f_θ, while the second, $\delta_\psi(\hat{\sigma}_t, z_{t-1}, a_{t-1})$, additionally incorporates the original source environment dynamics predictions made by the frozen forward belief, $p_\theta(\hat{z}_t | z_{t-1}, \hat{\sigma}_{t-1}^{real}, a_{t-1})$.

Although the additional inputs, $\hat{\sigma}_{t-1}^{real}$ and $\hat{\sigma}_t$, could theoretically provide useful information for the residual prediction task, we observe that their inclusion leads to a decrease in target-environment performance. We hypothesize that conditioning the residual function on an added real-valued vector, alongside the discrete latent-state z_{t-1}, significantly expands the space of representable residual functions. Given the limited dataset, this increased complexity likely impairs generalization to the target domain.

This result underscores the importance of bottlenecking state information through the compressed discrete representation z_t

for effective low-data adaptation.

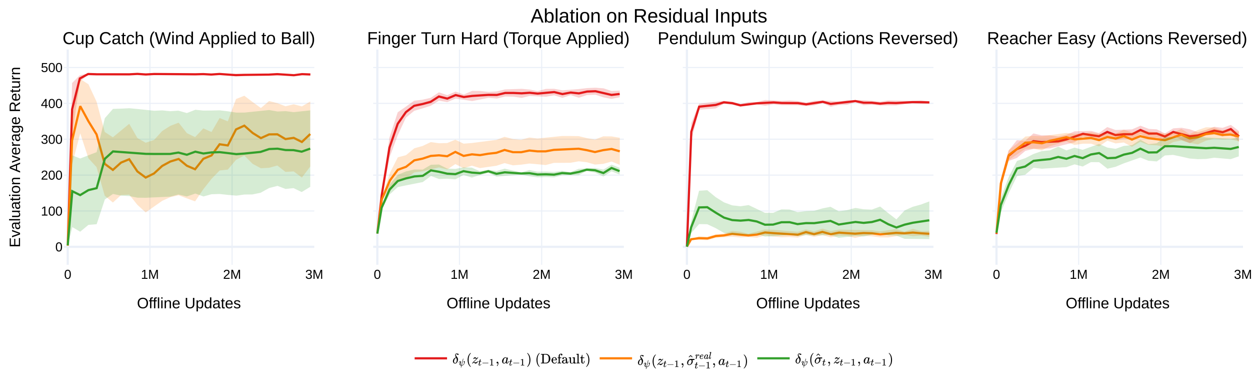


Figure 10. Comparison of different residual inputs for ReDRAW.

E. Latent Residual vs. New Dynamics Function

In this section, we compare the ReDRAW latent-state dynamics residual with an alternative adaptation method that also leverages frozen dynamics predictions learned from the source environment. Specifically, we contrast using a residual with learning a new replacement dynamics function, g_ψ, which optionally conditions on the outputs of the original source environment dynamics f_θ. We evaluate three possible definitions for g_ψ:

1. $\hat{\sigma}^{real}_t = g_\psi(z_{t-1}, a_{t-1})$, where g_ψ conditions on the same inputs as the ReDRAW residual.
2. $\hat{\sigma}^{real}_t = g_\psi(\hat{\sigma}_t, z_{t-1}, a_{t-1})$, where g_ψ additionally conditions on the frozen DRAW predicted source dynamics distribution, $\hat{\sigma}_t = p_\theta(\hat{z}_t|z_{t-1}, \hat{\sigma}^{real}_{t-1}, a_{t-1})$.
3. $\hat{\sigma}^{real}_t = g_\psi(\hat{z}_t, z_{t-1}, a_{t-1})$, where g_ψ additionally conditions on a discrete latent-state sample from the frozen DRAW source dynamics predictions, $\hat{z}_t \sim \mathrm{MultiCategorical}(\hat{\sigma}_t)$ as in (5).

To train the replacement dynamics function on the offline M_{real} dataset, we employ a dynamics loss term equivalent to (20) used by ReDRAW:

$$\mathcal{L}_g(\psi) = \mathbb{E}_{q_{\bar{\theta}}(z_{1:T}|\zeta^{real})}\left[\sum_{t=1}^{T} \mathbb{D}[q_{\bar{\theta}}(z_t|x_t)||g_\psi(\hat{z}^{real}_t|\bullet)]\right] \tag{21}$$

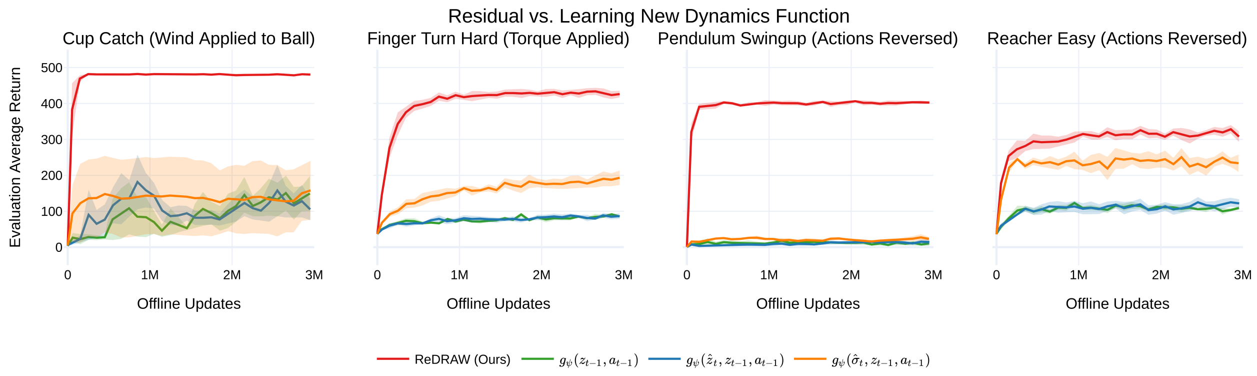


Figure 11. Comparison with a replacement dynamics function g_ψ with the same small capacity as the residual network.

Figure 11 presents the average target environment return during adaptation for ReDRAW and all considered replacement dynamics functions. The results show that ReDRAW outperforms all variations of the replacement function baseline, including those that incorporate predictions from the frozen DRAW source dynamics function.

From this experiment, we conclude that the residual operation, which modifies DRAW dynamics predictions without conditioning on them, is a key factor in achieving effective generalization to the target environment.

F. DMC Experiment Details

To ensure full observability from a single image frame, we expand the observation space by incorporating a vector of joint velocities provided by the simulator. We use an action repeat of two, meaning each episode consists of 500 decision steps, equivalent to 1000 environment steps. Additionally, to preserve state-based rewards, we do not sum rewards over the environment steps skipped due to action repeat.

Below, we describe each pair of source and target environments used in our DMC experiments. The source environment corresponds to the original DMC environment, while each target environment has modified dynamics:

- **Cup Catch**: The agent controls a cup to catch a ball tethered by a string. In the target environment, a constant horizontal wind alters the ball's trajectory, requiring the agent to adapt by compensating for this external force.
- **Finger Turn Hard**: The agent rotates a hinged spinner to a specified goal orientation. In the target environment, an external torque continuously drives the spinner, forcing the agent to counteract this disturbance to maintain control.
- **Pendulum Swingup**: The agent swings a pendulum to an upright position. In the target domain, action effects are reversed, requiring the agent to invert its control policy.
- **Reacher Easy**: The agent maneuvers a two-link arm to reach a target position. As in Pendulum Swingup, actions are inverted in the target environment, posing a challenge for direct policy transfer.

G. Hyperparameters

We implement DRAW and ReDRAW code as a modification to the official DreamerV3 implementation (Hafner, 2023). Except where otherwise stated, we use DreamerV3 default hyperparameters for all methods, including a batch size of 16, batch length of 64, and learning rates of 1×10^{-4} for the world model and 3×10^{-5} for the actor and critic. Additional parameters specific to our method or experiments are listed below.

	Hyperparameter	**Value**
all methods	pretraining replay buffer size	1e7
	online train ratio	512
	Encoder/Decoder CNN Depth	32
	Encoder/Decoder MLP hidden layers	2
	MLP hidden units	512
DRAW/ReDRAW	K (number of categorical distributions)	256
	N (number of categorical classes)	4
	imagination horizon for actor-critic training	40
	β_{pred}	1.0
	β_{dyn}	1.5
	β_{rep}	0.5
	residual learning rate	1e-2
	forward dynamics MLP hidden layers	1
	residual MLP hidden layers	1
	residual MLP hidden units	256

Table 2. Modified or newly introduced hyperparameters used in experiments.

H. Open Source Code

Links to open source code will be made available at `https://redraw.jblanier.net`.