Neural Motion Simulator
Pushing the Limit of World Models in Reinforcement Learning

Chenjie Hao[1*], Weyl Lu[1*], Yifan Xu[2], Yubei Chen[1,2†]

[1]UC Davis [2]Open Path AI Foundation

{cjhao, adslu, ybchen}@ucdavis.edu, yifan.xu@openpathai.com

Figure 1. This figures demonstrate the long-horizon precise prediction by the Neural Motion Simulators. In each of the three agents, the first row shows the groud-truth states and the second row shows the predicted states with the same initial condition and actions sequence. Humanoid predicts for 30 steps with rendering every 3 steps; Panda predicts for 200 steps with rendering every 20 steps; myohand predicts for 400 steps with rendering every 40 steps.

Abstract

An embodied system must not only model the patterns of the external world but also understand its own motion dynamics. A motion dynamic model is essential for efficient skill acquisition and effective planning. In this work, we introduce the neural motion simulator (MoSim), a world model that predicts the future physical state of an embodied system based on current observations and actions. MoSim achieves state-of-the-art performance in physical state prediction and provides competitive performance across a range of downstream tasks. This works shows that when a world model is accurate enough and performs precise long-horizon predictions, it can facilitate efficient skill acquisition in imagined worlds and even enable zero-shot reinforcement learning. Furthermore, MoSim can transform any model-free reinforcement learning (RL) algorithm into a model-based approach, effectively decoupling physical environment modeling from RL algorithm development. This separation allows for independent advancements in RL algorithms and world modeling, significantly improving sample efficiency and enhancing generalization capabilities. Our findings highlight that world models for motion dynamics is a promising direction for developing more versatile and capable embodied systems.

1. Introduction

Human and other animals build world models of the external world and themselves to efficiently learn skills, reason, and plan [30]. By using such a world model, natural intelligence can learn tasks with a small exposure or general-

*Equal Contribution. †Corresponding Author

ize to the situations they have never encountered. A world model [11, 23, 38] is an inner model that simulates how the world evolves and in essence it does one thing: given the current state and action, it predicts the future state [38]. In this work, we focus on building the world model, for motion dynamics. Given an arbitrary robot, a world model must learn the dynamics of the physical body and predict the future physical state given its current state and actions. While there were many pioneering works proposed in the past [14, 17–20] for solving different reinforcement learning tasks, almost all of these works evaluate the world models indirectly by using the downstream tasks as surrogates. This work aims to fill this neglected gap. How good are the world models at doing what they do – to predict the future given the current state and action? If the world models are truly generalizable, can we possibly use them to generalize to the never encountered situations or even learn the new tasks in a zero-shot manner?

In this work, we push the limit of world models in reinforcement learning and build the state-of-the-art world model, *Neural Motion Simulator* (MoSim), that significantly surpasses the previous ones in direct evaluations, predicting the future raw and latent states. We show, for the first time, that when a world model's prediction horizon and accuracy are sufficient, it is possible to train or search for a new policy completely in the predicted space in a zero-shot manner. In order to model the motion precisely, we introduce a world model architecture with rigid-body dynamics and Neural Ordinary Differential Equations (Neural ODE) [3], which enables accurate long-horizon predictions of future physical states. Since MoSim predicts in the raw state space, it can be combined with essentially any model-free reinforcement learning (RL) algorithm and turn into a model-based approach. This effectively decouples the physical environment modeling from the development of the RL algorithm. Such a separation allows MoSim benefit from the independent advancements from both world models and RL algorithms, significantly improving the sample efficiency and enhancing generalization capabilities. Our findings highlight that modeling world models for motion dynamics is a promising direction for developing more versatile and capable embodied systems.

Our main contributions are summarized as follows:

- We propose **MoSim**, a neural simulator with **state-of-the-art** long-horizon prediction accuracy.
- Leveraging MoSim's predictive capability, we show that it is possible to achieve **zero-shot model-based RL**, which can be integrated with any model-free algorithm.
- We take initial steps toward addressing the key challenges of zero-shot RL, demonstrating the potential of this direction.

We created a website https://oamics.github.io/mosim_page/ to provide additional details.

2. Method

In this section, we introduce the model architecture, benchmark, and training strategy.

2.1. Architecture

Dynamics Model. MoSim performs predictions within the continuous physical state space $\mathcal{S}$ of the robot. The physical state $\boldsymbol{s}(t) := (q_1, q_2, \ldots, q_n, \dot{q}_1, \dot{q}_2, \cdots, \dot{q}_n)^T = (\boldsymbol{q}^T, \dot{\boldsymbol{q}}^T)^T$ is a vector that changes continuously over time t, including the coordinates $\boldsymbol{q}$ and their corresponding velocities $\dot{\boldsymbol{q}}$[2]. $\boldsymbol{s}$ includes variables such as joint motor angles, angular velocities, spatial positions and velocities, etc. In practice, these state variables can be obtained through sensors mounted on the robot. As shown in Figure 2(b), We model the system dynamics by two terms: 1) a *Predictor* term f, which models the deterministic rigid body dynamics and 2) a *Corrector* term, which handles noise or other unmodeled factors that deviate from the rigid body motion:

$$\dot{\boldsymbol{s}}(t) = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t)) + \boldsymbol{\epsilon}(\boldsymbol{s}(t), \boldsymbol{a}(t)), \tag{1}$$

where $\boldsymbol{a}(t)$ denotes a time-varying action vector, which can be joint torque inputs or other control signal inputs, $\dot{\boldsymbol{s}}(t)$ represents the time derivative of $\boldsymbol{s}(t)$, $\boldsymbol{f}$ denotes the rigid body dynamic function, and ϵ represents the residual unmodeled factors. In the absence of stochasticity, the initial state $\boldsymbol{s}(t_0)$ and a given action $\boldsymbol{a}(t)$ uniquely determine the state $\boldsymbol{s}(t_0 + T)$ for any T.

Ideal Rigid Body Dynamics. For ideal rigid body motion, the dynamic equations have an explicit form [9, 41]:

$$\dot{\boldsymbol{s}}_{ideal} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix}\boldsymbol{q}\\ \dot{\boldsymbol{q}}\end{pmatrix} = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t)) \tag{2}$$

$$= \begin{pmatrix}\dot{\boldsymbol{q}}\\ M(\boldsymbol{q})[\mathbf{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\tau}(\boldsymbol{a}) + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a})]\end{pmatrix}. \tag{3}$$

where $M(\boldsymbol{q})$ is the inverse of inertia matrix that depends only on the position and it is symmetric positive definite; $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a vector-valued function of the state, describing the effects of conservative forces, such as gravity; $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a})$, as a residual term, accounts for the effects of equality constraints and contact forces (e.g., collision and friction forces) which are difficult to be explicitly modeled. If we let $\boldsymbol{\epsilon}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a})$ in Equation 1 absorb $M(\boldsymbol{q})\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{a})$ in Equation 3, then we can reach the following formulation:

$$\dot{\boldsymbol{s}}(t) = \begin{pmatrix}\dot{\boldsymbol{q}}\\ M(\boldsymbol{s})[\boldsymbol{b}(\boldsymbol{s}) + \boldsymbol{\tau}(\boldsymbol{a})]\end{pmatrix} + \boldsymbol{\epsilon}(\boldsymbol{s}(t), \boldsymbol{a}(t)) \tag{4}$$

In MoSim, we parameterize each of $M(\boldsymbol{s})$ (Position Encoder), $\boldsymbol{b}$ (State Encoder), $\boldsymbol{\tau}$ (Action Encoder), and $\boldsymbol{\epsilon}$ (Corrector) using a different neural network, as shown in Figure 2. We use standard ResNet [21] to implement M, $\boldsymbol{b}$,

[2] In this paper, we use dots to denote derivatives with respect to time. For example, $\ddot{\boldsymbol{q}}$ represents the second-order derivative of q with respect to time, i.e., acceleration.

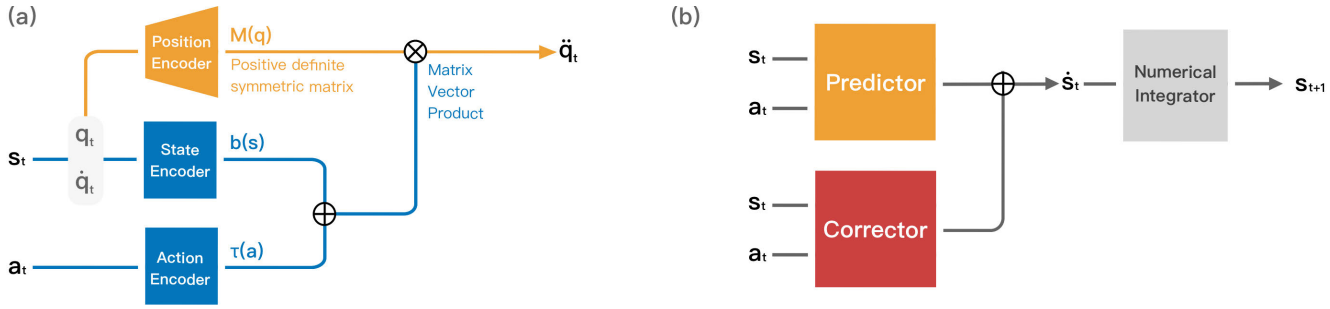


Figure 2. (a) Structure of predictor for a sub-step. (b) Structure of MoSim using Neural ODE to integrate many sub-steps to make a prediction for the next state.

ϵ and a simple MLP to implement τ. Detailed configuration of these networks are left to the Appendix. We refer to $f(\boldsymbol{s}, \boldsymbol{a}) = M(\boldsymbol{s})[\boldsymbol{b}(\boldsymbol{s}) + \boldsymbol{\tau}(\boldsymbol{a})]$ as the *predictor* and $\epsilon(\boldsymbol{s}, \boldsymbol{a})$ as the *corrector*. **Predictor**. In the position encoder, the ResNet output is rearranged into a lower triangular matrix L, from which M is calculated as $M = LL^T$, based on the property of Cholesky decomposition [10] that any symmetric positive definite matrix can always be expressed in this form. We assembled M, b and τ into $\ddot{\boldsymbol{q}}$, as illustrated in the Figure 2. Subsequently, $\ddot{\boldsymbol{q}}$ is combined with $\dot{\boldsymbol{q}}$ to form the output $\dot{\boldsymbol{s}}$. We embed strong inductive biases into the predictor by leveraging the general structure of rigid body dynamics. As demonstrated in the ablation study in Section 3, these inductive biases significantly improve the prediction accuracy. It is important to emphasize that we do not incorporate any additional prior knowledge of physics beyond the compositional structure, and, in practice, our network does not rely on the physical interpretation of these intermediate variables. **Correctors.** On the other hand, the correctors are made up entirely of one or more parallelly connected standard residual networks. Unlike the predictor, we do not introduce as much prior knowledge into the corrector, as its role is to handle the remaining complexities, such as friction, collisions, and unmodeled factors.

Neural ODE. The general form of Neural ODE is a continuous extension of recurrent neural networks:

$$\frac{\mathrm{d}z(t)}{\mathrm{d}t} = g_\theta(z(t), t) \tag{5}$$

Here, $z(t)$ represents the hidden variables over continuous t, and the dynamical function g_θ is parameterized by a neural network. Its solution takes an integral form:

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} g_\theta(z(t), t)\,\mathrm{d}t \tag{6}$$

The integration in practice is computed iteratively through numerical integration methods such as the Euler method[7] and the Runge-Kutta method[2, 28]. Our work employs the DOPRI5 integrator[6], which is an adaptive-order method based on the Runge-Kutta framework.

Neural ODE addresses the challenge of backpropagation in the continuous setting[3], which is inherently more difficult than in the discrete case, by cleverly reformulating it as another numerical integration:

$$\frac{\mathrm{d}L}{\mathrm{d}\theta} = -\int_{t_1}^{t_0} \alpha(t)\frac{\partial g_\theta(z(t), t)}{\partial \theta}\,\mathrm{d}t, \tag{7}$$

where L represents the loss function, and $\alpha(t) := \frac{\partial L}{\partial z(t)}$ is deined as the adjoint state of $z(t)$. The value of $\alpha(t)$ can also be obtained through an integral process.

$$\alpha(t_0) = \alpha(t_1) - \int_{t_1}^{t_0} \alpha(t)\frac{\partial g_\theta(z(t), t)}{\partial z(t)}\,\mathrm{d}t \tag{8}$$

The integrands in Equations 7 and 8 can both be computed using automatic differentiation. In MoSim, we treat $s(t)$ as the latent variable, and the dynamics network $g_\theta(s(t), t) = f_\theta(s(t), a(t)) + \sum_i \epsilon_\theta^i(s(t), a(t))$ is precisely composed of the predictor and corrector we mentioned earlier.

2.2. Benchmark

While previous world models have typically been evaluated in reinforcement learning tasks, we propose a benchmark specifically designed to evaluate the predictive capability of the world model itself: continuous multi-step prediction of robotic motion. We use DeepMind Control [39] as the virtual environment for robotics, which is based on the physical dynamics simulation provided by MuJoCo [41].

We construct the test set in two ways: by generating random action sequences using Poisson sampling and by generating action sequences based on policies from pre-trained reinforcement learning agents. The former provides untrained, dispersed data across the action and state space, while the latter offers task-specific, meaningful data. We generate sufficiently long data sequences to ensure that both the random and policy-based data reach a steady state, and then randomly sample fragments of the desired horizon length for evaluation.

Our benchmark includes prediction horizons critical for existing models: 3 steps (for TD-MPC2[20]), 16 steps (for

DreamerV3[18]), and 100 steps (for long-term prediction). The world model is required to predict continuously for the specified horizon, given the initial state and the action sequence. For models that require a warm-up period to initialize latent variables and achieve better performance, our benchmark provides a certain number of ground truth steps as a condition before evaluation.

Finally, we compute the mean squared error loss between the model's predictions and the ground truth over the entire segment as a measure of prediction accuracy.

2.3. Multi-Stage Training.

To fully utilize the model architecture proposed in Section 2.1, we introduce a training strategy called multi-stage training. We first train the predictor without incorporating the corrector, assuming that it should capture the smooth, non-abrupt components of robotic dynamics, such as inertia, effects of gravity, and elastic forces. Once the predictor converges, we freeze it and train the corrector on top of it. At this stage, the corrector focuses on modeling the abrupt and unmodeled dynamics, attempting to bridge the gap between the actual dynamics and the smooth dynamics captured by the predictor. For more complex robots, we employ multiple correctors to perform stepwise refinement, with each one addressing the non-smooth dynamics that the previous level struggles to handle.

Compared to training the entire large network jointly from the start, multistage training process allows the model to first capitalize on the advantages of inductive biases to quickly learn the simpler and more fundamental parts of the task. Subsequently, the corrector, which lacks strong inductive biases, handles the more complex and task-specific components. Ablation studies in Section 3 demonstrate that models trained with the multistage training strategy achieve faster training speeds and improved final performance.

3. Results

We evaluated MoSim on five robotic environments from the DM Control suite[42], as well as two additional environments: Panda[45], a robotic arm operating in a three-dimensional space, and Go2[33, 45], a quadruped robot. We designed experiments to compare its predictive performance against current state-of-the-art world models, both in state space and latent space. The results demonstrate that MoSim achieved significant improvements in predictive capability compared to previous world models. Additionally, we integrated MoSim into model-free algorithms, transforming them into model-based approaches, demonstrating that MoSim can be easily adapted to various model-free algorithms. Furthermore, we implemented zero-shot learning in specific tasks using MoSim-based RL algorithms, enabling reinforcement learning training without any real-world data, and explored the potential of few-shot learning in more complex tasks.

To provide a clear understanding of the model naming conventions used in our experiments: The suffix '-r' indicates models trained on random datasets generated by sampling action sequences from a Poisson distribution with a parameter of 1 at each step in the action space, '-e' denotes models trained on experience datasets from Dreamer's replay buffer, and '-rm' refers to MoSim trained using a multistage approach on random data.

3.1. Raw State Space Evaluation

We first compare the predictive capabilities of DreamerV3, whose backbone network is the Recurrent State-Space Model (RSSM)[16], a variant of recurrent neural networks, with those of MoSim.

RSSM Baseline. We trained RSSM in two ways: one following the original DreamerV3 training method, and the other for an ablation experiment, following the same training approach as MoSim. We provided RSSM with 1-step and 5-step ground truth as initial conditions, based on our observation that the RSSM requires several initialization steps to prepare its latent variables, for better predictions.

Generalization Ability. As mentioned in the Introduction, we found that models trained on data collected in a random manner exhibit higher prediction accuracy and better generalization ability. We trained and compared both models(-r and -e) on datasets collected using both methods, and the results in table 2 of the ablation experiments highlighted the superiority of the random data collection approach.

Training Method Comparison. We compared the impact of two different training methods on the results, demonstrating the multistage training approach's effectiveness in terms of both time efficiency and overall performance.

In Table 1, We categorize the tasks into easy and hard, where easy tasks generally have smoother dynamics, making them more suited for predictor-based solutions. In these cases, we did not use the multistage training approach. On the other hand, hard tasks exhibit non-smooth dynamics, where multistage training significantly outperforms standard end-to-end training in terms of final performance. As illustrated in Figure 4, the multistage approach also improves training efficiency and stability. Overall results indicate that MoSim consistently outperforms DreamerV3 across both easy and hard tasks, demonstrating its superior capability in handling diverse dynamics.

3.2. Latent Space Evaluation

MoSim also exhibits strong prediction accuracy in the latent space of TD-MPC2. We utilized the signal task pre-trained TD-MPC2 model and leveraged its encoder to encode our predicted states into their latent space. We trained MoSim using tdmpc policy data and compared it against the model that generated the training and test data (TD-MPC2) in table

Environment	Horizon	DreamerV3-e		DreamerV3-r		MoSim-e	MoSim-r	MoSim-rm
		1-step	5-step	1-step	5-step			
Cheetah	16	8.2538	6.5270	0.8747	0.1925	0.5342	**0.1206**	/
	100	6.5507	5.8468	0.4048	0.2297	1.1876	**0.2185**	/
Reacher	16	0.7370	0.1081	0.7451	0.0972	0.0037	**0.0005**	/
	100	0.7405	0.1743	0.7532	0.0988	0.0053	**0.0009**	/
Acrobot	16	3.6390	0.9356	3.7423	0.1015	/	**0.0001**	/
	100	9.7392	4.8957	9.7547	4.9276	/	**0.1043**	/
Panda	16	/	/	0.2396	0.0434	/	**0.0010**	/
	100	/	/	0.3367	0.0971	/	**0.0043**	/
Hopper	16	/	/	0.6406	0.1114	/	0.0743	**0.0375**
	100	/	/	0.9239	0.3199	/	0.4049	**0.2507**
Go2	16	/	/	1.0097	0.3685	/	0.0430	**0.0410**
	100	/	/	0.9243	0.4165	/	**0.1282**	0.1401
Humanoid	5	/	/	5.2171	1.3947	/	0.9382	**0.6535**
	10	/	/	6.0500	1.8263	/	1.5344	**1.0063**
	16	/	/	6.5078	2.1291	/	1.9735	**1.2737**

Table 1. MoSim versus DreamerV3. Evaluated on random policy datasets. Easy tasks. The suffix '-e' denotes models trained on experience data from Dreamer's replay buffer, '-r' indicates models trained on random policy data, and '-rm' refers to MoSim trained on random policy data with the multistage approach.

Environment	Horizon	DreamerV3-e		DreamerV3-r		MoSim-e	MoSim-r
		1-step	5-step	1-step	5-step		
Cheetah	16	13.1264	4.8908	18.4404	11.1325	**3.8841**	5.6052
	100	13.8990	9.4270	18.6197	4.5623	13.0104	**3.8434**
Reacher	16	0.3430	0.0514	0.4670	0.04875	0.0027	**0.0005**
	100	0.3562	0.0803	0.4364	0.05712	0.0030	**0.0008**
Acrobot	16	5.0681	1.3900	5.0681	1.3900	/	**0.0121**
	100	16.8498	11.1576	16.8498	11.1576	/	**1.2864**

Table 2. MoSim versus DreamerV3. Evaluated on the TD-MPC2 policy dataset, which means the evaluation data is out-of-distribution (OOD) for all models. The results show that models trained on random datasets (-r) generally achieve lower prediction errors than model trained on experience datasets (-e), indicating a stronger generalization capability.

Environment	TD-MPC2-e	MoSim-e
Humanoid	0.00011	**0.00009**
Cheetah	0.0009	**0.0007**
Reacher	4.8101e-5	**2.9256e-7**

Table 3. MoSim versus TD-MPC2. Evaluated on TD-MPC policy datasets.

Environment	TD-MPC2-e	MoSim-rm
Humanoid	0.0050	**0.0020**
Cheetah	0.0011	**0.0003**
Reacher	9.9229e-5	**7.0000e-7**

Table 4. MoSim versus TD-MPC2. Evaluated on random datasets.

3. In all environments, we performed three-step predictions, following the default prediction steps of TD-MPC2. We also do the comparison for random data in table 4.

3.3. Stochasticity

While MoSim assumes deterministic dynamics, we examine its robustness to observation noise and its capability to reliably sample multiple future trajectories.

Table 5 shows that MoSim remains robust to noise in

training data while maintaining consistent or even achieving better performance.

Prediction MSE Loss	DreamerV3	MoSim	MoSim w/ noise $\mathcal{N} \sim (0, 0.01^2)$
Reacher(100 steps)	0.0988	0.0009	0.0008
Panda(100 steps)	0.0971	0.0043	0.0042
Go2(100 steps)	0.4165	0.1282	0.2111

Table 5. Noise training.

For a self-consistent chaotic system such as Acrobot, Multiple future trajectories sampling can be reached through ensemble sampling. Table 6 shows that MoSim's Lyapunov characteristic exponent aligns with MuJoCo, confirming the reliability of ensemble sampling.

3.4. Zero- and Few-Shot Reinforcement Learning

Incorporating MoSim enables us to seamlessly transform any model-free reinforcement learning algorithm into a model-based approach, leveraging the predictive capabilities of MoSim's dynamics model. In this section, we integrate MoSim with TQC [29] and SAC [13], introducing three distinct integration strategies, each offering varying levels of data efficiency.

Zero-Shot Learning. By capitalizing on the robust generalization capabilities and predictive performance of MoSim, we employed a MoSim model trained on randomized data as a surrogate for the dm_control environment. This enabled the agent to interact directly with the world model for training, achieving scores on three dm_control benchmark tasks that closely approximate those from the real environment.

When attempting the cheetah task, two significant challenges have hindered progress. First, the model's absolute upper limit in predictive capability is insufficient. In the cheetah task, the MoSim model is able to maintain stable predictions only up to approximately 100 steps. However, it was observed that the existing model-free algorithms fail to successfully learn the cheetah-run task within the original environment when the time limit is set to 100. Additionally, as training progresses, a severe distribution shift in the training data becomes apparent, eventually exceeding MoSim's generalization capacity. At this stage, due to increasing prediction inaccuracies, the RL training plateaus (around a score of 100 for the cheetah task) and may even begin to degrade. These two issues have impeded further progress in MoSim's zero-shot learning for the cheetah task.

Few-Shot Learning. To address the two challenges encountered in zero-shot learning, we propose a few-shot learning strategy. Instead of solely relying on MoSim to replace the virtual environment, we periodically collect real environment data during training. Specifically, every 5,000 virtual environment steps, we gather 1,000 steps of real environment data. This data is then used to train MoSim by sampling from the real dataset every 100 virtual steps, helping to mitigate the distribution shift that occurs during training. Additionally, to address the issue of limited prediction horizon, we initialize the virtual environment with states sampled from the real environment replay buffer at each reset, rather than starting from the initial state every time. This aims to mimic the effect of having a time limit of 1,000 steps in the real environment. The results indicate that, unlike in zero-shot learning where performance plateaus and then deteriorates, the RL training stabilizes and fluctuates around a score of 100. However, the curve still fails to improve and remains at a relatively low score.

Lyapunov Characteristic Exponent	MuJoCo	MoSim
$\delta = 10^{-5}, T = 1000,$ avg on 2000 trajectories	1.1738	1.1728

Table 6. LCE

Task Name	Cheetah	Reacher-Easy	Reacher-Hard	Acrobot
Horizon	500	30	30	800
Achieved	60	30	30	60

Table 7. Zero-Shot Horizon Requirements. Zero-Shot Horizon Requirements. The 'horizon' indicates the minimum number of prediction steps required for complete zero-shot performance, while achieved indicates the current number of steps MoSim has successfully reached.

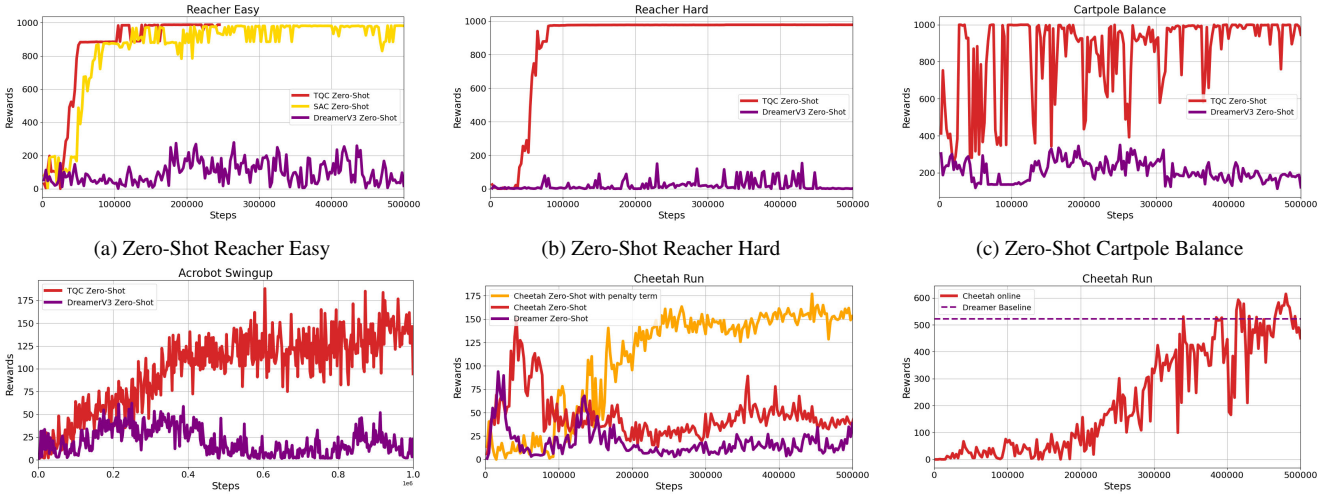
3.5. How Much Better Do We Need It To Be?

We investigated the minimum environment time limit required for the original TQC algorithm to achieve its baseline performance under precise predictions. We varied the time limit in the dm_control environment and evaluated its impact on the algorithm's performance. Table 7 implies that for the TQC algorithm to achieve offline RL without performance degradation, the World Model must possess the capability to continuously predict over the minimum step limit length across any data distribution. This remains a critical goal we aim to achieve in the long term.

It is worth noting that the 'zero-shot horizon requirement' in Table 7 was actually an upper bound, as it was calculated with only one model-free RL algorithm. There exist algorithms that can make this bound significantly tighter. For example, MPC requires only a 50-step horizon for the Acrobot swing-up task. We provide a more detailed discussion of this in the appendix.

3.6. On-Policy World Model for RL

In previous successful world models, the model was updated in an on-policy and online fashion, where the world model was trained using samples from a collected replay buffer at every real environment step. The trained data was then immediately used as the starting point for the model's predictions, creating a continuous loop. However, this ap-



(a) Zero-Shot Reacher Easy

(b) Zero-Shot Reacher Hard

(c) Zero-Shot Cartpole Balance

(d) Zero-Shot Acrobot Swingup

(e) Zero-Shot Cheetah Run

(f) online Cheetah Run

proach inherently involves frequent interactions with the real environment, making it challenging to reduce the number of real-world interactions. MoSim adopted a similar training strategy: every 100 real environment steps, a batch of data was sampled from the replay buffer to train the model. For the subsequent 100 steps, the model used sampled states from the replay buffer as initial states to predict one step ahead and train the policy. With this training strategy, MoSim outperformed DreamerV3[3] within the 500k time step limit on the cheetah task 3f.To provide a contrast, we used a pre-trained MoSim model as the world model, which was pre-trained on random data. In this case, the model predicted one step ahead at each step but was never updated with new data. As shown in Figure 8, while the model initially showed improvement, the prediction quality eventually declined because the world model failed to generalize to the current data distribution, ultimately leading to a collapse in training performance.

Although MoSim demonstrated strong performance on the cheetah task with an on-policy training strategy, we believe that few-shot learning remains a promising and worthwhile direction to pursue. Achieving purely zero-shot performance on locomotion tasks is nearly impossible, as the data distribution can vary significantly across different policies and training stages. Additionally, the model's predictive capabilities are inherently limited in such tasks. Thus, leveraging real data to assist training can be essential. Few-shot learning provides a feasible and meaningful solution, as it significantly reduces the need for interactions with the real environment, thereby improving data efficiency and making the approach more practical for complex tasks.

[3]All DreamerV3 RL results used in this paper are directly sourced from the open-source TD-MPC2 DreamerV3 baseline.

3.7. Let Model Know Khen It Doesn't Know

To address the issue mentioned in Section 3.4 where performance degrades during later stages of training due to distribution shift—We aim to enable the model to recognize when it is uncertain, allowing exploration to avoid areas where world model tends to generalize poorly and produce inaccurate predictions. Specifically, we use a residual flow with a Gaussian base distribution to fit the distribution of the MoSim training set. We then incorporate the probability density of each data point as a penalty term in the reinforcement learning reward. This way, when a data point has a low probability density under the residual flow (indicating that the world model is unfamiliar with that region of data, i.e., it has encountered a distribution shift), then the policy is discouraged from taking that action. As shown in the figure3e , after adding the penalty term to the reward, the zero-shot cheetah's reward (without the penalty term) no longer declines after a period of training and even achieves a higher score compared to the original maximum. This result provides an initial indication and potential future direction for addressing distribution shift.(A detailed explanation of this method is provided in the Appendix F.)

3.8. Effect of Prediction Horizon on Policy Learning

To explore the impact of the world model's predictive capability on policy learning, we compare the final performance of policies trained with different prediction horizons: 10, 50, and 100. As shown in Figure 6, the results indicate that as the prediction horizon decreases, the final performance of the policy also drops. This highlights the importance of the world model's quality for rl training.

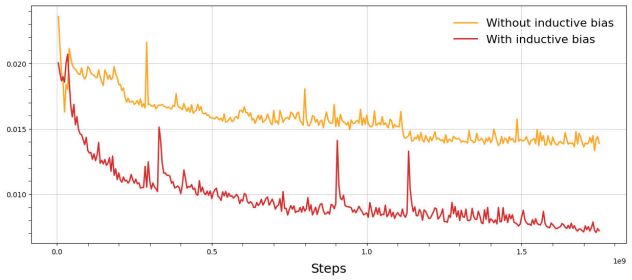


Figure 4. Ablation study of inductive bias on Hopper-hop.

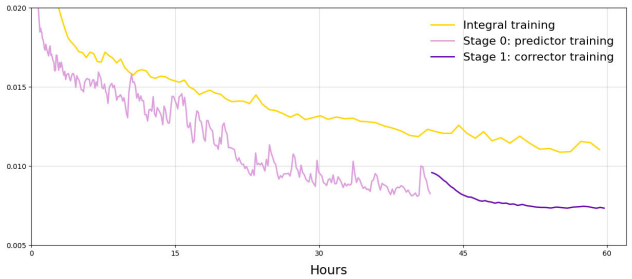


Figure 5. Ablation study of training method on Hopper-hop.

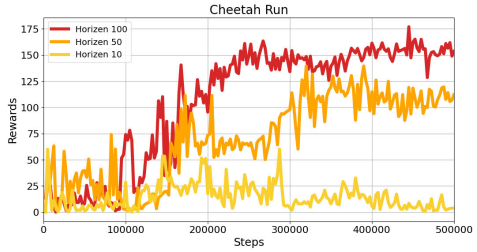


Figure 6. Policy learning with different prediction horizen

3.9. Ablation Study

We conducted two ablation studies to evaluate the effectiveness of the network architecture and training strategies discussed in Section 2.

Inductive Bias of Rigid Body Dynamics: We compared the predictor model with a standard ResNet of nearly identical parameter count to demonstrate the necessity of introducing inductive bias. The results, shown in Figure 4, illustrate the clear benefits of inductive bias in terms of training speed and overall performance.

Multistage Training: We conducted an ablation study using a predictor model and a corrector model with identical parameter counts. Both models were evaluated under full training and multi-stage training regimes. The results, presented in Figure 5, demonstrate the effectiveness of the multi-stage training approach.

4. Additional Related Works

Differentiable Simulation. Recent differentiable physics engines [22, 24] offer end-to-end gradients via hand-coded rigid/soft-body models. In contrast, our data-driven approach minimizes manual force/contact design, using a structural predictor with trainable correctors.

Neural ODE Usage. Neural ODEs have been applied to time-series modeling [34], fluid simulation [31], system ID [27], and generative modeling [44], and have inspired physics-based networks [8]. Building on this, we use a neural ODE backbone with rigid-body insight and residual correctors for robust long-horizon prediction in robotic control.

Parameter Identification. Classical optimal control refines simulations via identifying parameters like mass or friction [32]. In contrast, our method learns a scalable dynamics model from data, capturing smooth and non-smooth effects without manual engineering.

World Models and Model-Based RL. One major application of Existing world models are mainly used in reinforcement learning, where they predict future states in explicit or latent space as action-conditioned models [4, 5, 14, 17, 18, 20, 26, 35, 40]. Such inner-model-based methods are called model-based RL and offer key advantages over model-free approaches by leveraging model predictions.

1. **Pixel-Based Video Generation World Models.** Another line of work predicts future frames from actions for pixel-based policy learning [12, 15]. Models like DreamerV2/V3 [18] learn latent visual representations but struggle with fine-grained physical interactions like contact.
2. **Data Efficient Learning**: RL algorithms can learn from the data generated by a trained model's predictions (or "dreams"), effectively treating the model as a form of data augmenter.[1, 5, 25, 43]
3. **Planning Capability**: Many real-world tasks demand long-term planning, where models predict outcomes over action sequences and search for optimal ones [35, 37, 40]. Planning has shown great success, e.g., AlphaGo [36], but real-world dynamics are far harder to predict than a Go board. Enabling real-world planning first requires a predictive world model.

5. Conclusion

In this work, we focus on evaluating and improving a world model's direct predictive capability. The resulted model, MoSim, significantly outperforms the world models in direct state prediction. We show, for the first time, that when a world model's prediction horizon and accuracy are sufficient, it is possible to train or search for a new policy purely in the predicted space in a zero-shot manner. However, a notable gap remains in achieving few-shot learning or even zero-shot learning in many cases. Our findings highlight that world models for motion dynamics is a promising direction for developing more versatile and capable embodied systems.

References

[1] Michael J. Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Model-based reinforcement learning for atari. In *International Conference on Learning Representations (ICLR)*, 2018. 8

[2] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Chichester, UK, 2003. 3

[3] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 2, 3

[4] 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 (NeurIPS)*, pages 4754–4765, 2018. 8

[5] Marc P. Deisenroth and Carl E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 465–472, 2011. 8

[6] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6:19–26, 1980. 3

[7] Leonhard Euler. *Institutiones Calculi Integralis*. Bousquet, 1768. 3

[8] Marc Finzi, Ke Alexander Wang, Andrew Xu, and Andrew Gordon Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:13880–13890, 2020. 8

[9] Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics*. Pearson, 3rd edition, 2001. 2

[10] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 4th edition, 2013. Cholesky decomposition is covered in Chapter 4. 3

[11] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. 2

[12] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. 8

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. 6

[14] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019. 2, 8

[15] 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 (ICML)*, pages 2555–2565, 2019. 8

[16] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1912.01603*, 2019. 4

[17] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020. 2, 8

[18] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. 4, 8

[19] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control, 2022.

[20] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024. 2, 3, 8

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[22] Enric Heiden, Daniel Millard, Erwin Coumans, Gaurav S. Sukhatme, and Konstantinos Tsotsos. Neuralsim: Augmenting differentiable simulators with neural networks. *International Conference on Learning Representations (ICLR)*, 2021. 8

[23] Mikael Henaff, William F Whitney, and Yann LeCun. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*, 2017. 2

[24] Yifei Hu, Dylan Jacobs, Boyuan Chen Huang, et al. Chainqueen: A real-time differentiable physical simulator for soft robotics. *Robotics: Science and Systems (RSS)*, 2019. 8

[25] Pu Hua, Yubei Chen, and Huazhe Xu. Simple emergent action representations from multi-task policy training, 2023. 8

[26] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 12519–12530, 2019. 8

[27] Roohollah Khodayi-Mehr, Andreas Walch, and P-S Koutsourelakis. Deep learning of mechanical systems: A neural-ode approach. *Computers & Structures*, 235: 106–150, 2020. 8

[28] W. Kutta. Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 46:435–453, 1901. 3

[29] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics, 2020. 6

[30] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022. 1

[31] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Nakamura. Dissecting neural odes. *International Conference on Machine Learning (ICML)*, pages 1–11, 2020. 8

[32] Karline Morris and Anil V Rao. Parameter identification for optimal control problems with applications in aerospace engineering. *Optimization and Engineering*, 12(1):123–144, 2011. 8

[33] Unitree Robotics. Unitree Mujoco, 2021. 4

[34] Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. 8

[35] Julian Schrittwieser and et al. Mastering atari, chess, go, and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. 8

[36] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 8

[37] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 8

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

[39] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018. 3

[40] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Model predictive path integral control: From theory to parallel computation. *The International Journal of Robotics Research*, 31(2):160–187, 2010. 8

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012. 2, 3

[42] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. 4

[43] Yifan Xu, Nicklas Hansen, Zirui Wang, Yung-Chieh Chan, Hao Su, and Zhuowen Tu. On the feasibility of cross-task transfer with model-based reinforcement learning, 2023. 8

[44] Caglar G. Yildiz, Markus Heinonen, and Harri Lähdesmäki. Ode2vae: Deep generative second order odes with bayesian neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019. 8

[45] Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo, 2022. 4