WESTWORLD: A KNOWLEDGE-ENCODED SCALABLE TRAJECTORY WORLD MODEL FOR DIVERSE ROBOTICS

Yuchen Wang, Jiangtao Kong, Xiaochang Li & Huajie Shao
William & Mary
{ywang142, jkong01,xli59, hshao}@wm.edu

Sizhe Wei & Lu Gan
Georgia Institute of Technology
{swei,lgan}@gatech.edu

Haohong Lin
Carnegie Mellon University
haohongl@cmu.edu

Hongjue Zhao
UIUC
hongjue2@illinois.edu

Tianyi Zhou
MBZUAI
Tianyi.Zhou@mbzuai.ac.ae

ABSTRACT

Trajectory world models have emerged as a cornerstone of robotic dynamics learning, enabling more effective planning and control in complex environments. Recent studies have explored pre-training such models across diverse robotic systems, but they still face two major challenges: 1) scaling to a large number of heterogeneous robotic systems, and 2) failing to incorporate domain knowledge of robot morphology, which limits zero-shot generalization to previously unseen systems. To address these challenges, we introduce `WestWorld`, a kno**W**ledge-**E**ncoded **S**calable **T**rajectory **World** model for diverse robotics. To address the challenge of scalability, `WestWorld` uses a system-aware Mixture-of-Experts (Sys-MoE) that routes inputs to specialized experts via a learnable system embedding. To enhance zero-shot generalization, we incorporate domain knowledge of robot physical structure through a structural embedding that aligns trajectory representations with morphological information. After pretraining on 89 environments spanning diverse morphologies in both simulation and real-world settings, `WestWorld` significantly outperforms state-of-the-art baselines in zero-shot trajectory prediction. Notably, it demonstrates strong scalability as the number of robotic environments increases.

1 INTRODUCTION

Trajectory world models Chua et al. (2018); Schubert et al. (2023); Yin et al. (2025); Xie et al. (2025) are predictive models that learn to represent and forecast how a robotic system evolves over time by modeling entire state–action trajectories. They play a crucial role in robotic dynamics learning, planning, and control. However, building a general-purpose trajectory world model that generalizes across diverse robotic systems poses two key challenges: *(i) sensor and actuator heterogeneity*, where differences in sensing and actuation modalities (e.g., channel semantics, dimensionality, and sampling rates) hinder learning a shared representation; and *(ii) system dynamics gaps*, where variations in kinematic structures and morphology lead to distinct transition dynamics across systems.

To address these challenges, a few recent studies Schubert et al. (2023); Yin et al. (2025) discretize continuous states and actions across diverse systems into tokens via quantization and leverage flexible Transformer architectures for joint training. Although these approaches enable multi-system pretraining within a shared parameter set, they still face *scalability and generalization limitations* across diverse robotic dynamics for two main reasons. *First*, all robots share a common set of model parameters, so jointly training dissimilar dynamics can lead to gradient conflicts and negative transfer, resulting in poor scalability as the number and diversity of robots grow. *Second*, these methods treat trajectories as sequences of tokens without encoding robot morphology or physical structure,

which leads to weak zero-shot generalization even to structurally similar robots outside the training distribution.

To overcome these limitations, we develop `WestWorld`, a knowledge-encoded scalable trajectory world model that injects robot morphology priors to learn the dynamics of diverse robotics (Fig. 1). Developing such a model poses two *key challenges*: *(i)* scaling to many distinct system dynamics while reducing cross-robot interference during joint training, and *(ii)* incorporating physical structure as an inductive bias to improve zero-shot generalization. To address the first challenge of scalability, we propose a system-aware mixture-of-experts (Sys-MoE) that implicitly learns distinct system dynamics through expert learning. Unlike existing trajectory world models, which learn multiple system dynamics using a single large dense model, the proposed Sys-MoE dynamically combines and routes specialized experts for different robotic systems via a learnable system embedding. This design mitigates task interference across robots, thus significantly improving scalability. For the second challenge of generalization, we introduce a structure-based channel embedding that aligns low-level state trajectories with morphology information, thereby improving the model's ability to generalize to unseen robotic systems.

We pretrain the proposed `WestWorld` on 89 complex environments using a combination of simulated and real-world data. Extensive experiments show that `WestWorld` substantially outperforms strong baselines in zero-shot trajectory prediction on unseen environments. Moreover, it scales to diverse robotic systems learning without an apparent loss in predictive performance.

Our contributions include: 1) We propose `WestWorld`, a system-aware mixture-of-experts (MoE) architecture that scales trajectory world model pretraining across diverse robotic systems; 2) We introduce a knowledge-encoded structural embedding that injects morphology priors as an explicit inductive bias, improving zero-shot generalization to unseen environments; 3) We conduct extensive experiments to validate scalability and generalization, demonstrating consistent gains over strong baselines.

2 Related Work

World Models for Single Robots. World models Ha & Schmidhuber (2018) serve as a foundational tool for sequential decision-making in embodied agents Wei et al. (2025); Long et al. (2025), enabling them to model, understand, and predict environmental dynamics. In robotics, one line of work Agarwal et al. (2025); Chi et al. (2025) leverages video generation models as world models to synthesize temporally coherent observations, thereby implicitly capturing underlying physical dynamics. Another line of research develops action-conditioned *dynamics* world models that explicitly predict future states for planning and control Guo et al. (2025); Ebert et al. (2018); Zhu et al. (2025); Hansen et al. (2022); Chua et al. (2018). These dynamics world models can be broadly categorized into video world models and trajectory world models, offering complementary perspectives for learning physical dynamics.

In this work, we focus on *studying low-level trajectory world models*. Generalizing such models across diverse robotic systems is non-trivial, since trajectories are derived from heterogeneous sensors and actuators with mismatched channel semantics across systems. Thus, most existing trajectory world models Wu et al. (2023); Hansen et al. (2022); Chua et al. (2018) are tailored to a single robot, and transferring to new platforms typically requires retraining or substantial adaptation. In contrast, our work aims to learn a unified trajectory world model that scales across diverse robotic systems.

Trajectory World Model for Diverse Robotics. Recent works Hansen et al. (2024); Yin et al. (2025); Schubert et al. (2023) have explored trajectory world models for diverse robotic systems. A key challenge is handling varying sensor and actuator dimensionalities across different robots. A common strategy is to zero-pad inputs to a shared maximum dimension Hansen et al. (2024). However, padding-based approaches suffer from dimensionality limits and often degrade generalization across environments Yin et al. (2025). To address this, a few recent studies Schubert et al. (2023); Yin et al. (2025) treat states and actions as token sequences and jointly train flexible Transformer architectures across multiple robots. Despite enabling joint pretraining across diverse robots, their scalability and generalization remain limited. First, most existing methods learn heterogeneous robotic systems using a single shared set of model parameters, which induces cross-system interference as robot diversity grows and makes scaling difficult. Second, these methods treat trajectories purely as token sequences

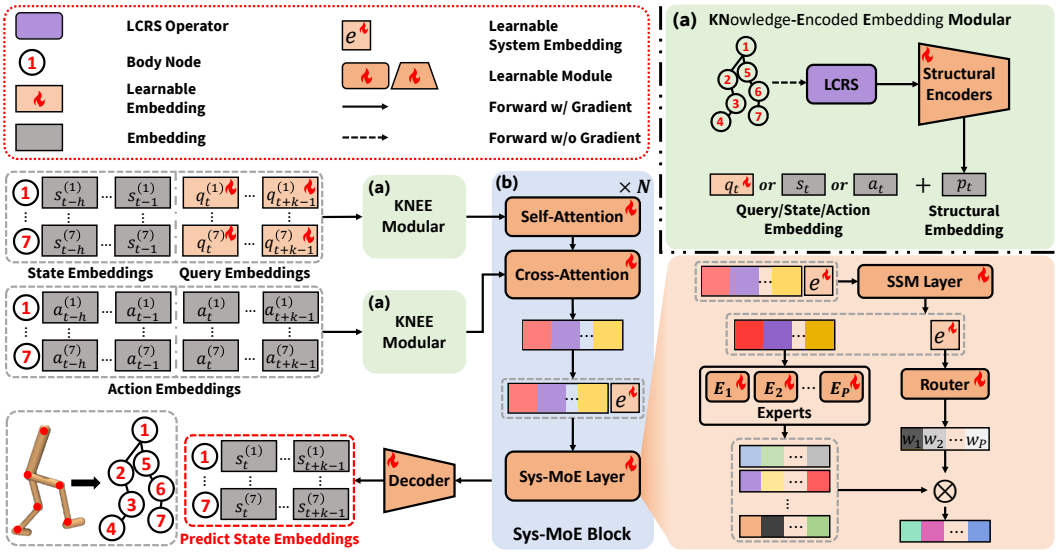

Figure 1: The overall architecture of our proposed `WestWorld`, consisting of two core components: (a) a Knowledge-Encoded Embedding Modular that injects structural embeddings as an inductive bias into trajectory representations, and (b) a System-aware MoE block that models diverse system dynamics via system-aware expert routing.

while ignoring robot morphology information, which results in poor zero-shot generalization to unseen robotics.

Unlike prior works, we propose a system-aware MoE model that incorporates robot structural information to enable both scalable pretraining and strong zero-shot performance.

3 PRELIMINARIES

Notations. The detailed descriptions of important notations are presented in Table 3 in Appendix A.

Problem Statement. A robotic system can be viewed as a controlled dynamical process with state space $\mathcal{S}$, action space $\mathcal{A}$, and transition dynamics $\boldsymbol{f}$. In practice, the state $\boldsymbol{s}_t \in \mathcal{S}$ consists of physical sensor readings such as joint positions and joint velocities, while the action $\boldsymbol{a}_t \in \mathcal{A}$ represents control commands such as joint torques.

Model-based control relies on an internal dynamics model to support planning by rolling out candidate action sequences in imagination Long et al. (2025).

In this work, we use a *trajectory world model* purely as a dynamics model Sekar et al. (2020); Long et al. (2025); Ha & Schmidhuber that predicts future states $\boldsymbol{s}$ conditioned on action interactions $\boldsymbol{a}$. Formally, a world model parameterizes a transition distribution

$$p_\theta(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}), \qquad \boldsymbol{s}_t \in \mathcal{S},\ \boldsymbol{a}_t \in \mathcal{A}, \tag{1}$$

which can be used to unroll state trajectories under a proposed action sequence. A rollout induces a trajectory

$$\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_t, \boldsymbol{a}_t), \tag{2}$$

and the model is trained on the recorded trajectories $\mathcal{D} = \{\tau_i\}$ by maximizing predictive accuracy of future states.

The goal of this work is to develop a pretrained trajectory world model to learn the system dynamics across varying robotic systems and environments. Given a trajectory dataset from n distinct robotic systems, $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$, our objective is to learn a single model θ that captures the dynamics of all n systems. Specifically, given a history of h past states and actions, the model tries to predict the next k future states, conditioned on a sequence of k future actions.

4 PROPOSED METHOD

4.1 OVERVIEW OF WESTWORLD

To enable scalable pretraining and zero-shot generalization across diverse robotic systems, we propose WestWorld, a knowledge-encoded scalable trajectory world model with a system-aware Mixture-of-Experts (MoE) design. As shown in Fig. 1, the proposed model consists of two core components: (1) *Knowledge-Encoded Embedding Modular* and (2) *System-Aware MoE*.

The *core idea* is to first perform channel-wise normalization and discretize each scalar variable for tokenization. The resulting representations are then processed by *Knowledge-Encoded Embedding Modular*, which extracts the robot's morphological connectivity and injects structural embeddings as an inductive bias into trajectory representations. These structure-aware embeddings are subsequently fed into multiple *System-Aware MoE* blocks for dynamics modeling. Finally, a linear decoder maps the hidden states to future trajectory predictions. We detail these two core components in the following.

4.2 KNOWLEDGE-ENCODED EMBEDDING MODULAR

Motivation. Existing trajectory world models are predominantly data-driven, relying solely on state-action observations and largely ignoring domain knowledge that different robotic morphologies should obey distinct physical constraints. The lack of encoding explicit structural information makes it difficult for these models to capture the underlying system dynamics and limits their ability to generalize across environments. We hypothesize that robots with similar connectivity patterns often exhibit shared high-level dynamical behaviors (e.g., SLIP-like locomotion Schwind (1998)). This insight motivates us to incorporate morphological connectivity into model design as an inductive bias. Below, we first introduce the trajectory data tokenization before diving into proposed knowledge-encoded structural embedding.

Trajectory Tokenization. Given a trajectory, we treat each state or action dimension at time step t as a *scalar channel*. Let $x_t^{(m)} \in \mathbb{R}$ denote the value of channel m at time t, where m represents the index of state channels or action channels. We apply channel-wise min–max normalization, and discretize it into a K-bin categorical vector through $\boldsymbol{\phi} : \mathbb{R} \to \mathbb{R}^K$ following Yin et al. (2025). We then map $\boldsymbol{\phi}(x_t^{(m)})$ to a d-dimensional embedding via a learned projection. After that, we incorporate timestep embeddings, channel order index embeddings, and modality indicator (state or action) embeddings, yielding $\boldsymbol{z}_t^{(m)} \in \mathbb{R}^d$. For convenience, we stack the per-channel embeddings from states and actions at time step t into $\boldsymbol{S}_t \triangleq [\boldsymbol{s}_t^{(1)}; \ldots; \boldsymbol{s}_t^{(M_s)}]^\top \in \mathbb{R}^{M_s \times d}$ and $\boldsymbol{A}_t \triangleq [\boldsymbol{a}_t^{(1)}; \ldots; \boldsymbol{a}_t^{(M_a)}]^\top \in \mathbb{R}^{M_a \times d}$, where M_s and M_a are the numbers of state and action channels.

Knowledge-Encoded Structural Embedding. To incorporate morphology structure priors into latent representations, we introduce a knowledge-encoded structural embedding, as shown in Fig. 1 (a). Specifically, we first model each articulated object as a rooted kinematic tree and convert it to a binary tree using the left-child-right-sibling (LCRS) transformation Hong et al. (2021). Each body node is assigned three traversal indices from pre-/in-/post-order walks. For object i and its body node j, let $\left(\pi_{\text{pre}}^{i,j}, \pi_{\text{in}}^{i,j}, \pi_{\text{post}}^{i,j}\right)$ denote these indices. In scenes with multiple articulated objects, we additionally assign an object identifier π_{obj}^i: the robot is indexed as $\pi_{\text{obj}}^i = 0$, and other objects are ordered by increasing Euclidean distance to the robot (see Appendix B for an example). With this tuple indices, we can uniquely identify each robot body node in the LCRS-converted binary tree derived from the robot's structure. Then, we embed these indices to obtain a structure embedding:

$$\begin{aligned} \boldsymbol{p}^{(i,j)} = \text{Concat}\Big(& \boldsymbol{e}_{\text{obj}}\big(\pi_{\text{obj}}^i\big),\ \boldsymbol{e}_{\text{pre}}\big(\pi_{\text{pre}}^{i,j}\big), \\ & \boldsymbol{e}_{\text{in}}\big(\pi_{\text{in}}^{i,j}\big),\ \boldsymbol{e}_{\text{post}}\big(\pi_{\text{post}}^{i,j}\big)\Big). \end{aligned} \tag{3}$$

where each $\boldsymbol{e}_{\{\text{obj,pre,in,post}\}}(\cdot)$ denotes a structural encoder that maps a discrete index to a $d/4$-dimensional vector, and their concatenation forms $\boldsymbol{p}^{(i,j)} \in \mathbb{R}^d$. Finally, we inject morphology knowledge by adding $\boldsymbol{p}^{(i,j)}$ to the corresponding state/action embeddings, yielding structure-aware trajectory embeddings that are used as inputs to our model.

4.3 SYSTEM-AWARE MOE BLOCK

Motivation. Robotic systems with diverse morphologies often exhibit markedly different dynamics, making it difficult to develop a single unified model that accurately captures their underlying dynamics. When such dissimilar dynamics are trained simultaneously using shared parameters, optimization is prone to gradient conflicts and task interference, leading to poor scalability. To address this challenge, we introduce a novel system-aware Mixture-of-Experts (Sys-MoE) block for learning distinct system dynamics, in which each expert specializes in a basis dynamics module. Our *key insight* is that complex system dynamics can be effectively approximated by composing a set of basis dynamics with system-dependent coefficients.

Block design. To scale joint training across diverse robotic systems while mitigating interference, we parameterize the transition model in Eq. (1) with a stack of *Sys-MoE Blocks*. Each block contains two parts: *i)* an attention-based aggregation module that fuses state–action information, and *ii)* a system-aware MoE layer that captures diverse system dynamics. We detail the two parts below.

***i)* Attention-based aggregation.** As shown in Fig. 1(b), we use attention to aggregate information across state channels and to inject action-dependent control signals, while naturally supporting variable state/action dimensionalities across systems. Concretely, we apply: 1) self-attention to capture correlations among state variables, and then use 2) cross-attention to condition state features on the action embeddings. To enable k-step prediction in a single forward pass, we concatenate the history state embeddings with k learnable query embeddings $\{\boldsymbol{q}_t, \ldots, \boldsymbol{q}_{t+k-1}\}$, which serve as latent queries for future states.

At each time step, self-attention is computed as

$$\tilde{\boldsymbol{S}}_t = \mathrm{LN}\Big(\boldsymbol{S}_t + \text{Self-Atten}(\boldsymbol{S}_t)\Big), \tag{4}$$

where self-attention is applied along the state channel, and $\mathrm{LN}(\cdot)$ is layer normalization. We then condition $\tilde{\boldsymbol{S}}_t$ on the action embeddings $\boldsymbol{A}_t$ via multi-head cross-attention:

$$\hat{\boldsymbol{S}}_t = \mathrm{LN}\Big(\tilde{\boldsymbol{S}}_t + \text{Cross-Atten}(\tilde{\boldsymbol{S}}_t, \boldsymbol{A}_t)\Big), \tag{5}$$

This operation injects action-dependent signals while remaining compatible with variable action dimensionalities.

ii) System-aware MoE layer. After obtaining the action-conditioned latent states above, we model continuous-time system dynamics using a system-aware Mixture-of-Experts (Sys-MoE) layer, as shown in Fig. 1(b). Unlike the MoE design commonly used in large language models, where routing selects experts to directly produce token embeddings from the input, our routing is *system-aware*. Specifically, we introduce a learnable system embedding that propagates through the SSM to extract system-level properties of the underlying dynamics. The resulting system embeddings are then used to compute mixture weights over experts, so that the model forms a system-conditioned combination of different experts.

Let $L = h + k$ be the number of state embeddings after concatenating history states with k queries. For each state channel m, we denote the attention outputs as $\hat{\boldsymbol{S}}^{(m)}_{1:L} = \{\hat{\boldsymbol{s}}^{(m)}_{t-h:t-1}, \hat{\boldsymbol{s}}^{(m)}_{t:t+k-1}\}$, where the last k tokens correspond to the query positions. We append a learnable system embedding $\boldsymbol{e} \in \mathbb{R}^d$ to attention outputs, yielding

$$\overline{\boldsymbol{S}}^{(m)} = \mathrm{Concat}\Big(\hat{\boldsymbol{S}}^{(m)}_{1:L},\, \boldsymbol{e}\Big). \tag{6}$$

We apply an SSM layer to obtain outputs $\boldsymbol{U}_\ell$:

$$\boldsymbol{U}^{(m)}_{1:L+1} = \mathrm{SSM}(\overline{\boldsymbol{S}}^{(m)}), \tag{7}$$

where $\boldsymbol{U}^{(m)}_{1:L+1}$ denotes the SSM outputs for all embeddings. In our implementation, $\mathrm{SSM}(\cdot)$ follows a Mamba-style selective SSM Gu & Dao (2024), enabling causal computation and efficient long-range dependency modeling.

We use the output of the system embedding, $\boldsymbol{U}_{L+1}$, to extract system-aware properties for routing. A router produces mixture weights over P experts via a softmax gate:

$$\boldsymbol{w} = \mathrm{Softmax}(\mathrm{Router}(\boldsymbol{U}_{L+1})) \in \mathbb{R}^P. \tag{8}$$

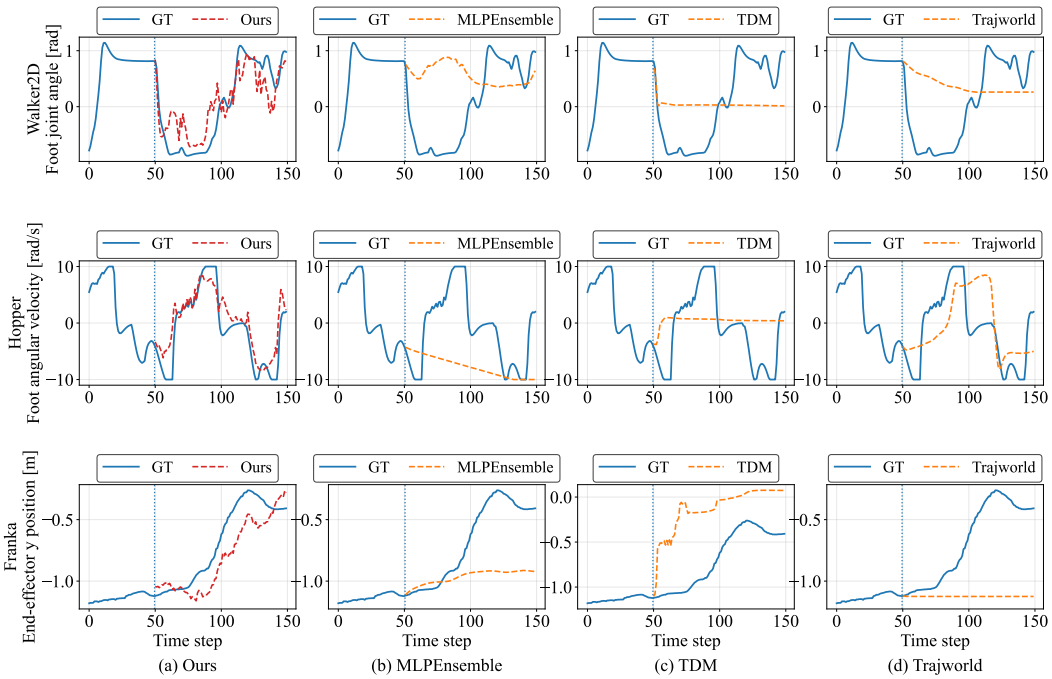

Figure 2: Trajectory plot comparison of our method and three baselines for 100-step rollout prediction on three robots: *Walker2D* foot joint angle, *Hopper* foot angular velocity, and *Franka* end-effector y position, given a 50-step history window as input. We can observe that our method tracks the ground-truth dynamics substantially more closely than the baselines over the 100-step horizon.

Given the output $\boldsymbol{U}_{1:L}^{(m)}$, the Sys-MoE block outputs $\boldsymbol{Y}_{1:L}^{(m)}$ is computed as a weighted combination of expert predictions:

$$\boldsymbol{Y}_{1:L}^{(m)} = \sum_{p=1}^{P} w_p \, E_p(\boldsymbol{U}_{1:L}^{(m)}), \tag{9}$$

where w_p is the p-th entry of $\boldsymbol{w}$, and each expert $E_p(\cdot)$ is implemented as an MLP. Finally, we stack multiple Sys-MoE blocks to increase expressivity for complex system dynamics.

4.4 Objective Function.

After stacking *Sys-MoE Blocks*, we obtain the per-channel output sequence $\boldsymbol{Y}_{1:L}^{(m)}$ We apply a linear decoder head to produce logits over K uniform bins. Concretely, for each channel m outputs, we compute

$$\boldsymbol{P}_{L}^{(m)} = \mathrm{Softmax}(\boldsymbol{W}_{\mathrm{dec}} \, \boldsymbol{Y}_{1:L}^{(m)}) \in [0,1]^K. \tag{10}$$

We train the model with a next-token cross-entropy loss on the state channels to match the categorical representation of the inputs:

$$\mathcal{L}_{\mathrm{CE}} = -\sum_{\ell=1}^{L-1} \sum_{m=1}^{M_s} \boldsymbol{\phi}(x_{\ell+1}^{(m)})^{\top} \log \boldsymbol{P}_{\ell}^{(m)}. \tag{11}$$

During inference time, we run the model in a *sequence-to-sequence manner*, enabling multi-step prediction in a single forward pass.

5 Experiments

In this section, we first pretrain `WestWorld` on large-scale, diverse robotic datasets and then conduct extensive experiments to answer the following research questions (RQ).

RQ1: Can `WestWorld` achieve strong zero-shot performance on unseen environments involving robots similar to those used during pretraining?

RQ2: Can it effectively learn diverse system dynamics simultaneously as the number of pretraining environments scales?

RQ3: Can it improve downstream tasks (e.g., model-based control) across a wide range of robotic systems?

Diverse Pretraining Datasets. For pretraining the proposed trajectory world model, we collect a large amount of simulated and real-world data: i) UniTraj dataset Yin et al. (2025), which contains 80 simulated robotic environments; and ii) 9 real-world robot-arm datasets from the Open X-Embodiment Vuong et al. (2023). A detailed list of the pretraining and evaluation environments used in our experiments is provided in Appendix C.

Baseline Methods. We compare our method against several state-of-the-art trajectory world models. (1) *MLP Ensemble* Chua et al. (2018): a widely used baseline in model-based RL for learning probabilistic dynamics through an ensemble of multilayer perceptrons. (2) *TDM* Schubert et al. (2023): a Transformer-based model built upon the Gato architecture, which flattens spatial and temporal features into a single sequence and applies one-dimensional attention for autoregressive prediction. (3) *TrajWorld* Yin et al. (2025): a Transformer-based trajectory model that employs temporal-variate attention for autoregressive rollout.

Implementation Details. We follow each baseline's original pretraining configuration. Detailed training and implementation settings for `WestWorld` and all baselines are provided in Appendix D. For fairness, all baseline models are pretrained from scratch on the above same dataset as the proposed `WestWorld`.

Table 1: Zero-shot generalization performance of different models on three dynamical systems. Errors are computed in the normalized space and reported as MAE and MSE ($\times 10^{-2}$); lower is better.

Method	Walker2d ($\times 10^{-2}$)		Hopper ($\times 10^{-2}$)		Franka ($\times 10^{-2}$)	
	MAE ↓	MSE ↓	MAE ↓	MSE ↓	MAE ↓	MSE ↓
MLPEnsemble	26.006	12.028	19.987	7.216	12.164	4.271
TDM	20.122	6.428	17.634	5.076	23.686	8.435
Trajworld	22.261	8.623	17.388	5.441	13.102	5.127
Ours	**16.350**	**5.064**	**13.731**	**3.368**	**7.737**	**2.539**

5.1 Main Results

Evaluation on Zero-shot Performance (RQ1). We first evaluate the zero-shot performance of our model on three unseen robotic environments that share similar structural morphology with those in the pretraining data. Specifically, we use datasets from three environments as our testbeds. These include *Hopper* and *Walker2D* from D4RL Fu et al. (2020), as well as a *real-world* dataset of a mobile *Franka* manipulator interacting with articulated objects Schiavi et al. (2023). We evaluate 100-step consecutive predictions using a 50-step history window as input. Mean Absolute Error (MAE) and Mean Squared Error (MSE) are used to assess the accuracy of long-horizon prediction.

As shown in Table 1, our method achieves the best performance across all three unseen environments in long-horizon prediction. This improvement is attributed to the combination of our system-aware MoE architecture and the structural inductive bias introduced through morphology-aware design. The MoE design enables the model to learn distinct dynamics for different morphologies while mitigating task interference during pretraining. In addition, we visualize trajectory plots for all three robots in Fig. 2. We can see that `WestWorld` tracks the ground-truth dynamics substantially more closely than the baselines over the 100-step horizon. The reason is that, in a zero-shot setting on unseen but structurally similar systems, our model selects appropriate experts to produce accurate dynamics predictions, whereas baseline methods lack morphology-aware representations and fail to generalize.

Evaluation on Scalability (RQ2). We further verify the scalability of our method by varying the number of robotic environments while keeping the data budget per environment fixed, as shown in Fig. 4. Specifically, we pretrain the model on $N \in \{1, 2, 5, 10, 20, 30\}$ environments. The detailed task-level environment information is presented in Appendix C.3. For each environment, we use 1000 episodes for training, 500 episodes for validation, and 500 episodes for testing.

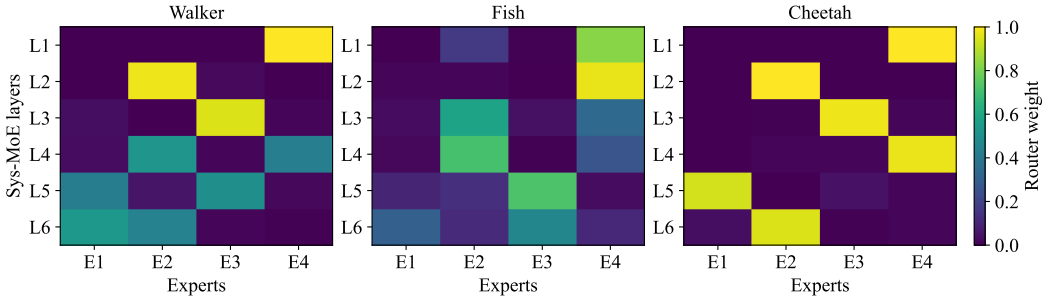

Figure 3: Sys-MoE routing weights across six layers (L1–L6), each containing four experts (E1–E4), for three robotic systems. Color indicates the router weight, where brighter values correspond to higher expert activation. The router exhibits near-sparse, system-dependent expert specialization, suggesting that different systems are modeled by different combinations of experts to capture their distinct dynamics.

We compare the performance of our method with the state-of-the-art TrajWorld using test data. All models take a 50-step history window as input and produce 100-step consecutive predictions.

Fig. 4 reports the long-horizon prediction errors at each N environments. We observe that the accuracy of our method remains low and does not vary significantly with increasing N. The results show that our method can simultaneously learn distinct system dynamics across diverse environments. Conversely, TrajWorld's performance degrades significantly as the number of environments increases. A plausible explanation is that optimizing a single shared model across multiple dissimilar dynamics exacerbates gradient interference and negative transfer, thereby limiting scalability Chen et al. (2018).

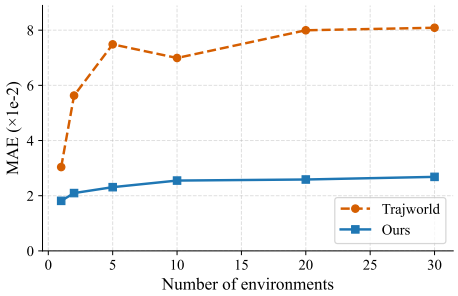

Figure 4: Comparison between our method against the best performing SOTA by scaling the number of environments.

To further analyze scalability, we visualize Sys-MoE routing weights for three distinct systems in Fig. 3. Across these systems, the router exhibits sparse, system-dependent expert selection. These results support our key insight: complex dynamics can be effectively approximated by composing a set of basis dynamics modules with system-dependent coefficients. Such system-aware design mitigates interference in multi-system joint learning and enables scalable pretraining across a wide range of robotic systems.

Evaluation on Downstream Control Task (RQ3). In addition, we evaluate whether pretraining improves downstream model-based control across diverse robotic systems, which aims to isolate the effect of pretraining on dynamics modeling. Jointly optimizing the controller or policy is a separate question and is not considered here. We consider two robotic systems with distinct dynamics: Walker2D and Hopper from OpenAI Gym Towers et al. (2024). For each system, we collect an offline trajectory dataset from the environment. We then compare two training regimes for each world model: *(i)* fine-tuning from a pretrained checkpoint and *(ii)* training from scratch under the same dataset. After training, we deploy the learned dynamics model within MPPI Williams et al. (2015), a commonly used sampling-based MPC controller. Additional details of the MPPI implementation are provided in Appendix G.

We set the MPPI planning horizon to 100 for Walker2D and Hopper. The setting is challenging: MPPI relies on long-horizon rollouts, so small model errors can compound and degrade control. In addition, the planner may explore actions that push the system outside the offline training distribution, further causing compounding errors and suboptimal control Parthasarathy et al. (2025). Table 2 reports the accumulated episode reward. We draw two key observations. First, for nearly all methods and systems, pretraining consistently improves control performance compared with training from scratch. This suggests that pretraining yields dynamics representations that generalize better under

Table 2: Downstream model-based control performance using MPPI on Walker2D and Hopper. We report accumulated episode reward (higher is better) averaged over the evaluation episodes. All methods are evaluated with fixed random seeds and identical MPPI hyperparameters for fair comparison.

Method	Pretrain	Walker2d	Hopper
		Accumulate Reward ↑	Accumulate Reward ↑
MLPEnsemble	✗	119.18	147.37
	✓	190.51	200.56
TDM	✗	122.65	242.34
	✓	207.61	154.64
Trajworld	✗	395.23	366.26
	✓	1933.52	534.32
Ours	✗	707.61	554.92
	✓	**2134.60**	**2253.51**

distribution shift between offline training and online MPC rollouts. Second, `WestWorld` achieves the best performance across all systems under both training regimes, with particularly large gains after pretraining. This indicates that our scalable model design and morphology-informed inductive bias significantly improve downstream control performance.

Evaluation on Inference Latency Additionally, we provide a detailed comparison of inference-time latency against autoregressive Transformer baselines. More details are provided in Appendix E.

5.2 Ablation Studies

In this section, we further analyze the impact of two core components on model performance: (1) the knowledge-encoded embedding (KNEE) module and (2) the system-aware Mixture-of-Experts (Sys-MoE) layer. Detailed results are deferred to Appendix F.

6 Conclusion and Limitation

In this work, we introduced `WestWorld`, a knowledge-encoded scalable trajectory world model for learning dynamics across diverse robotic systems. `WestWorld` scales multi-system pretraining via a Sys-MoE block and improves generalization by injecting morphology priors through knowledge-encoded structural embeddings. Extensive experiments demonstrated that `WestWorld` significantly improves zero-shot trajectory prediction on unseen environments and maintains strong scalability as the number of training systems increases.

`WestWorld` currently models state–action trajectories and does not explicitly encode visual observations. A promising direction is to extend `WestWorld` to a multimodal world model that fuses vision with trajectory signals.

Acknowledgments

Research reported in this paper was sponsored in part by NSF CPS 2311086, NSF CIRC 716152, NSF RITEL 2506890, NAIRR 250288, and Faculty Research Grant at William & Mary 141446.

REFERENCES

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025.

Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. Hydra: Hybrid robot actions for imitation learning. In *Conference on Robot Learning*, pp. 2113–2133. PMLR, 2023.

Lili Chen, Shikhar Bahl, and Deepak Pathak. Playfusion: Skill acquisition via diffusion from language-annotated play. In *Conference on Robot Learning*, pp. 2012–2029. PMLR, 2023.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pp. 794–803. PMLR, 2018.

Xiaowei Chi, Peidong Jia, Chun-Kai Fan, Xiaozhu Ju, Weishi Mi, Kevin Zhang, Zhiyuan Qin, Wanxin Tian, Kuangzhi Ge, Hao Li, et al. Wow: Towards a world omniscient world model through embodied interaction. *arXiv preprint arXiv:2509.22642*, 2025.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Quentin Gallouédec, Edward Beeching, Clément Romac, and Emmanuel Dellandréa. Jack of all trades, master of some, a multi-purpose transformer agent. *arXiv preprint arXiv:2402.09844*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A suite of benchmarks for offline reinforcement learning. *Advances in neural information processing systems*, 33:7248–7259, 2020.

Yanjiang Guo, Lucy Xiaoyang Shi, Jianyu Chen, and Chelsea Finn. Ctrl-world: A controllable generative world model for robot manipulation. *arXiv preprint arXiv:2510.10125*, 2025.

Agrim Gupta, Stephen Tian, Yunzhi Zhang, Jiajun Wu, Roberto Martín-Martín, and Li Fei-Fei. Maskvit: Masked visual pre-training for video prediction. *arXiv preprint arXiv:2206.11894*, 2022.

David Ha and Jürgen Schmidhuber. World models.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

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

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *The Twelfth International Conference on Learning Representations*, 2024.

Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. *The International Journal of Robotics Research*, 44(10-11):1863–1891, 2025.

Sunghoon Hong, Deunsol Yoon, and Kee-Eung Kim. Structure-aware transformer policy for inhomogeneous multi-task reinforcement learning. In *International Conference on Learning Representations*, 2021.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pp. 4455–4464. PMLR, 2020.

Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International conference on robotics and automation (ICRA)*, pp. 8943–8950. IEEE, 2019.

Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. Robot learning on the job: Human-in-the-loop autonomy and learning during deployment. *The International Journal of Robotics Research*, 44(10-11):1727–1742, 2025.

Xiaoxiao Long, Qingrui Zhao, Kaiwen Zhang, Zihao Zhang, Dingrui Wang, Yumeng Liu, Zhengjie Shu, Yi Lu, Shouzheng Wang, Xinzhe Wei, et al. A survey: Learning embodied intelligence from physical simulators and world models. *arXiv preprint arXiv:2507.00917*, 2025.

Arjun Parthasarathy, Nimit Kalra, Rohun Agrawal, Yann LeCun, Oumayma Bounou, Pavel Izmailov, and Micah Goldblum. Closing the train-test gap in world models for gradient-based planning. *arXiv preprint arXiv:2512.09929*, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Amrita Sawhney, Steven Lee, Kevin Zhang, Manuela Veloso, and Oliver Kroemer. Playing with food: Learning food item representations through interactive exploration. In *International Symposium on Experimental Robotics*, pp. 309–322. Springer, 2020.

Saumya Saxena, Mohit Sharma, and Oliver Kroemer. Multi-resolution sensing for real-time control with vision-language models. In *Conference on Robot Learning*, pp. 2210–2228. PMLR, 2023.

Giulio Schiavi, Paula Wulkop, Giuseppe Rizzi, Lionel Ott, Roland Siegwart, and Jen Jen Chung. Learning agent-aware affordances for closed-loop interaction with articulated objects. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5916–5922. IEEE, 2023.

Ingmar Schubert, Jingwei Zhang, Jake Bruce, Sarah Bechtle, Emilio Parisotto, Martin Riedmiller, Jost Tobias Springenberg, Arunkumar Byravan, Leonard Hasenclever, and Nicolas Heess. A generalist dynamics model for control. *arXiv preprint arXiv:2305.10912*, 2023.

William John Schwind. *Spring loaded inverted pendulum running: A plant model*. University of Michigan, 1998.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International conference on machine learning*, pp. 8583–8592. PMLR, 2020.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Quan Vuong, Sergey Levine, Homer Rich Walke, Karl Pertsch, Anikait Singh, Ria Doshi, Charles Xu, Jianlan Luo, Liam Tan, Dhruv Shah, et al. Open x-embodiment: Robotic learning datasets and rt-x models. In *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*, 2023.

Sizhe Wei, Xulin Chen, Fengze Xie, Garrett Ethan Katz, Zhenyu Gan, and Lu Gan. Ms-ppo: Morphological-symmetry-equivariant policy for legged robot locomotion. *arXiv preprint arXiv:2512.00727*, 2025.

Ying Wen, Ziyu Wan, Ming Zhou, Shufang Hou, Zhe Cao, Chenyang Le, Jingxiao Chen, Zheng Tian, Weinan Zhang, and Jun Wang. On realization of intelligent decision-making in the real world: A foundation decision model perspective. *arXiv preprint arXiv:2212.12669*, 2022.

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

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on robot learning*, pp. 2226–2240. PMLR, 2023.

Fengze Xie, Sizhe Wei, Yue Song, Yisong Yue, and Lu Gan. Morphological-symmetry-equivariant heterogeneous graph neural network for robotic dynamics learning. In *7th Annual Learning for Dynamics\& Control Conference*, pp. 1392–1405. PMLR, 2025.

Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

Shaofeng Yin, Jialong Wu, Siqiao Huang, Xingjian Su, Xu He, Jianye HAO, and Mingsheng Long. Trajectory world models for heterogeneous environments. In *Forty-second International Conference on Machine Learning*, 2025.

Fangqi Zhu, Hongtao Wu, Song Guo, Yuxiao Liu, Chilam Cheang, and Tao Kong. Irasim: A fine-grained world model for robot manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9834–9844, 2025.

Xinghao Zhu, Ran Tian, Chenfeng Xu, Mingxiao Huo, Wei Zhan, Masayoshi Tomizuka, and Mingyu Ding. Fanuc manipulation: A dataset for learning-based manipulation with fanuc mate 200id robot, 2023.

A NOTATIONS

The table below summarizes the notation used in this paper. Lowercase letters (e.g., x) denote scalars, bold lowercase letters (e.g., $\boldsymbol{x}$) represent vectors, and bold uppercase letters (e.g., $\boldsymbol{A}, \boldsymbol{B}$) denote matrices.

Table 3: Summary of notations

Notation	Definition
$\mathcal{S}$	state space
$\mathcal{A}$	action space
$\boldsymbol{s}$	state
$\boldsymbol{a}$	action
$n \in \mathbb{N}^+$	number of robotics systems
N	number of task-level environments in scalability evaluation
$\mathcal{D}_n$	trajectory data for n-th robotics system
$h \in \mathbb{N}^+$	history state time steps
$k \in \mathbb{N}^+$	predicted future time steps
$m \in \mathbb{N}^+$	channel index
$x_t^{(m)} \in \mathbb{R}$	m-th channel states or actions at time t
$\boldsymbol{\phi} : \mathbb{R} \to \mathbb{R}^K$	discrete function
$\boldsymbol{z}_t^{(m)} \in \mathbb{R}^d$	token embedding
$\boldsymbol{e}_{\text{time}} \in \mathbb{R}^d$	timestep embedding
$\boldsymbol{e}_{\text{ch}} \in \mathbb{R}^d$	channel order index embedding
$\boldsymbol{e}_{\text{type}} \in \mathbb{R}^d$	modality embedding
$\boldsymbol{S}_t \in \mathbb{R}^{M_s \times d}$	state embedding at time t, where M_s denote the number of state channel
$\boldsymbol{A}_t \in \mathbb{R}^{M_a \times d}$	action embedding at time t, , where M_a denote the number of action channel
$\pi_{\text{obj}}^i \in \mathbb{N}^+$	i-th object index
$\pi_{\text{pre}}^{i,j} \in \mathbb{N}^+$	pre-order index for object i and its body node j
$\pi_{\text{in}}^{i,j} \in \mathbb{N}^+$	in-order index for object i and its body node j
$\pi_{\text{post}}^{i,j} \in \mathbb{N}^+$	post-order index for object i and its body node j
$\boldsymbol{p} \in \mathbb{R}^d$	structural embedding
$\boldsymbol{q}_t \in \mathbb{R}^d$	query embedding at time t
$\tilde{\boldsymbol{S}}_t \in \mathbb{R}^{M_s \times d}$	state hidden states after state self-attention at time t
$\hat{\boldsymbol{S}}_t \in \mathbb{R}^{M_s \times d}$	state hidden states after action-state cross-attention at time t
$L \in \mathbb{N}^+$	the length after concatenating history states with queries
$\overline{\boldsymbol{S}}^{(m)} = [\hat{\boldsymbol{S}}_{1:L}^{(m)}; \boldsymbol{e}]$	state embedding concate with system embedding
$\boldsymbol{e} \in \mathbb{R}^d$	the learnable system embedding
$\boldsymbol{U}_\ell$	l-th SSM output
$P \in \mathbb{N}^+$	the number of expert
$\boldsymbol{Y}_{1:L}^{(m)} \in \mathbb{R}^{L \times d}$	m-th channel System-Aware MoE Block output
$\boldsymbol{P}_\ell^{(m)} \in \mathbb{R}^K$	categorical representation output
M_s	the number of state channels
M_a	the number of action channels
K	the number of categorical bins

B A WALK-THROUGH EXAMPLE FOR STRUCTURAL EMBEDDING

This section provides a concrete example of how we construct the knowledge-encoded structural embedding in Eq. (3). We use the Walker robot as an illustrative example.

Step 1: Extract the robot kinematic tree. Given the Walker structural file (e.g., an MJCF or URDF specification), we extract the articulated-body hierarchy and treat it as a rooted kinematic tree. Walker contains a single articulated object (the robot itself), so we set $\pi^i_{\mathrm{obj}} = 0$. The robot has seven body nodes: torso, right thigh, right leg, right foot, left thigh, left leg, and left foot.

Step 2: LCRS conversion and traversal ranks. We convert the rooted kinematic tree into a binary tree via the left-child-right-sibling (LCRS) transformation Hong et al. (2021). We then compute traversal ranks on the LCRS-converted binary tree, including pre-order, in-order, and post-order indices. For each body node j, we obtain a tuple of traversal ranks $\big(\pi^{0,j}_{\mathrm{pre}}, \pi^{0,j}_{\mathrm{in}}, \pi^{0,j}_{\mathrm{post}}\big)$. For Walker, the indices for all seven body nodes are listed in Table 4.

Table 4: Walker example of LCRS traversal ranks used to construct structural embeddings. Here $\pi_{\mathrm{obj}} = 0$ since the scene contains only the robot.

Body	Global idx	π_{pre}	π_{in}	π_{post}
torso	0	0	6	6
right_thigh	1	1	3	5
right_leg	2	2	5	4
right_foot	3	3	4	3
left_thigh	4	4	2	2
left_leg	5	5	1	1
left_foot	6	6	0	0

Step 3: Construct per-body structural embeddings. For each body node (i, j), we form the discrete index tuple

$$\big(\pi^i_{\mathrm{obj}}, \pi^{i,j}_{\mathrm{pre}}, \pi^{i,j}_{\mathrm{in}}, \pi^{i,j}_{\mathrm{post}}\big),$$

which uniquely identifies the node in the LCRS-converted binary tree (and also disambiguates multiple objects when present). We embed each index using a lookup table and concatenate the results as in Eq. (3):

$$\boldsymbol{p}^{(i,j)} = \mathrm{Concat}\Big(\boldsymbol{e}_{\mathrm{obj}}(\pi^i_{\mathrm{obj}}), \boldsymbol{e}_{\mathrm{pre}}(\pi^{i,j}_{\mathrm{pre}}), \boldsymbol{e}_{\mathrm{in}}(\pi^{i,j}_{\mathrm{in}}), \boldsymbol{e}_{\mathrm{post}}(\pi^{i,j}_{\mathrm{post}})\Big) \in \mathbb{R}^d.$$

Finally, we inject the structural prior by adding $\boldsymbol{p}^{(i,j)}$ to the corresponding per-body token embedding (e.g., state/action/query tokens associated with body j). This yields structure-aware trajectory embeddings that are consistent across morphologies.

C Detailed Dataset Settings

In this appendix, we provide the detailed dataset settings in the experiments. This includes the datasets for pretraining, zero-shot evaluation, and scalability evaluation.

C.1 Pretraining Datasets

For pretraining the trajectory world model, we use both large-scale simulated and real-world data. Specifically, we use: (i) the UniTraj dataset Yin et al. (2025), which aggregates trajectories from ExORL Yarats et al. (2022), RL Unplugged Gulcehre et al. (2020), JAT Gallouédec et al. (2024), DB-1 Wen et al. (2022), TD-MPC2 Hansen et al. (2024), and Modular RL Huang et al. (2020), covering in total 80 task-level simulated environments; and (ii) 9 real-world robot-arm datasets from Open X-Embodiment Vuong et al. (2023). We first summarize the robot morphology categories in Table 5, and then list all 89 task-level environments used for pretraining in Table 6.

Data preprocessing. To stabilize training across heterogeneous robotic systems, we follow UniTraj Yin et al. (2025) and apply channel-wise min–max normalization. For each robot and each feature dimension (e.g., joint position, joint velocity (angular or linear), or torque), we compute the empirical minimum and maximum values on the training split and rescale inputs to $[0, 1]$. Each trajectory segment is clipped or zero-padded to a maximum length of 150 time steps. Across all pretraining datasets, the maximum state dimension is 78 and the maximum action dimension is 21.

Table 5: A detailed list of diverse robotics used in the pretraining dataset. For robotics sharing the same name, we mark those from OpenAI Gym with an asterisk (*) and those from DeepMind Control Suite with a dagger (†).

Dataset	**Robot morphology categories**
ExORL	Cartpole, Jaco, Quadruped, Walker†
RL Unplugged	Cartpole, Fish, Humanoid, Manipulator, Walker†
JAT	Double Pendulum*, Pendulum*, Pusher*, Reacher*, Swimmer*
DB-1	Acrobat, Ball In Cup, Cartpole, Cheetah-2-back, Cheetah-2-front, Cheetah-3-back, Cheetah-3-balanced, Cheetah-3-front, Cheetah-4-allback, Cheetah-4-allfront, Cheetah-4-back, Cheetah-4-front, Cheetah-5-back, Cheetah-5-balanced, Cheetah-5-front, Cheetah-6-back, Cheetah-6-front, Finger, Fish, Hopper†, Hopper-3, Hopper-5, Humanoid, Humanoid-2d-7-left-arm, Humanoid-2d-7-left-leg, Humanoid-2d-7-lower-arms, Humanoid-2d-7-right-arm, Humanoid-2d-7-right-leg, Humanoid-2d-8-left-knee, Humanoid-2d-8-right-knee, Humanoid-2d-9-full, Manipulator, Reacher, Swimmer6, Swimmer15, Walker†, Walker-2-flipped, Walker-2-main, Walker-3-flipped, Walker-3-main, Walker-4-flipped, Walker-4-main, Walker-5-flipped, Walker-5-main, Walker-6-flipped, Walker-6-main
TD-MPC2	Acrobot, Ball In Cup, Cartpole, Cheetah†, Finger, Fish, Hopper†, Pendulum†, Reacher†, Walker†
Modular RL	Cheetah-2-back, Cheetah-2-front, Cheetah-3-back, Cheetah-3-balanced, Cheetah-4-allback, Cheetah-4-back, Cheetah-4-front, Cheetah-5-back, Cheetah-5-balanced, Cheetah-5-front, Cheetah-6-back, Cheetah-6-front, Hopper-3, Hopper-5, Walker-2-flipped, Walker-3-flipped, Walker-4-flipped, Walker-5-flipped, Walker-6-flipped, Walker-7-flipped
Open X-Embodiment	Fanuc Mate 6-DoF industrial arm Zhu et al. (2023), KUKA iiwa 7-DoF arm Lee et al. (2019), Franka Emika Panda arm Heo et al. (2025), Franka arm Belkhale et al. (2023); Liu et al. (2025); Saxena et al. (2023); Sawhney et al. (2020); Chen et al. (2023), Sawyer arm Gupta et al. (2022).

Table 6: Task-level environments used in the pretraining dataset.

Component	Task-level environments	#
TD-MPC2	walker-stand, walker-walk, walker-run, cheetah-run, reacher-easy, reacher-hard, acrobot-swingup, pendulum-swingup, cartpole-balance, cartpole-balance-sparse, cartpole-swingup, cartpole-swingup-sparse, cup-catch, finger-spin, finger-turn-easy, finger-turn-hard, fish-swim, hopper-stand, hopper-hop, walker-walk-backwards, walker-run-backwards, cheetah-run-backwards, cheetah-run-front, cheetah-run-back, cheetah-jump, hopper-hop-backwards, reacher-three-easy, reacher-three-hard, cup-spin, pendulum-spin	30
ExORL	jaco-reach, quadruped-locomotion	2
RL-Unplugged	humanoid_run, manipulator-insert-ball, manipulator-insert-peg	3
JAT	inverted_double_pendulum, inverted_pendulum, pusher, reacher, swimmer	5
DB-1	acrobot, cheetah_2_back, cheetah_2_front, cheetah_3_back, cheetah_3_balanced, cheetah_3_front, cheetah_4_allback, cheetah_4_allfront, cheetah_4_back, cheetah_4_front, cheetah_5_back, cheetah_5_balanced, cheetah_5_front, cheetah_6_back, cheetah_6_front, hopper_3, hopper_5, humanoid, humanoid_2d_7_left_arm, humanoid_2d_7_left_leg, humanoid_2d_7_lower_arms, humanoid_2d_7_right_arm, humanoid_2d_7_right_leg, humanoid_2d_8_left_knee, humanoid_2d_8_right_knee, humanoid_2d_9_full, manipulator-bring_ball, swimmer6, swimmer15, walker_2_flipped, walker_2_main, walker_3_flipped, walker_3_main, walker_4_flipped, walker_4_main, walker_5_flipped, walker_5_main, walker_6_flipped, walker_6_main	39
Modular-RL	walker_7_flipped	1
Open X-Embodiment	Fanuc-Mate-6DoF-tabletop-manipulation, KUKA-iiwa-7DoF-insertion, Franka-Emika-Panda-assemble-furniture, Franka-kitchen-manipulation, Franka-insertion, Franka-tabletop-manipulation, Franka-food-manipulation, Franka-play-with-object, Sawyer-pick-and-place	9
Total		**89**

C.2 Zero-shot experiment datasets.

To evaluate zero-shot generalization, we consider three unseen environments: *Walker2D* and *Hopper* from OpenAI Gym Towers et al. (2024), and a real-world dataset of a mobile *Franka* manipulator interacting with articulated objects Schiavi et al. (2023). None of these systems appears during pretraining, but they share similar structural morphology with robots in the pretraining data, making them suitable for testing generalization to unseen environments.

For *Walker2D* and *Hopper*, we use expert demonstrations from D4RL Fu et al. (2020) for evaluation. The *Walker2D* dataset contains 1,267 episodes, and the *Hopper* dataset contains 1,029 episodes. The mobile *Franka* dataset contains 118 episodes.

C.3 SCALING OF ENVIRONMENTAL DATASETS

In the environment-scaling study, we vary the number of pretraining environments and consider $N \in \{1, 2, 5, 10, 20, 30\}$. For each value of N, we train on the task-level environment subset listed in Table 7. Within each selected environment, we use the same fixed episode split: 1000 episodes for training, 500 episodes for validation, and 500 episodes for testing. As a result, the total numbers of training, validation, and test episodes increase proportionally with N.

Table 7: Detailed task-level environment subsets for the environment-scaling study. For each N, each environment uses a fixed split of 1000/500/500 episodes for train/val/test.

N	Selected task-level environments
1	Cartpole Swingup
2	Cartpole Swingup; Acrobot Swingup
5	Walker Walk Backwards; Finger Turn Easy; Cartpole Swingup; Reacher Three Easy; Cheetah Run Front
10	Walker Walk Backwards; Finger Turn Easy; Cartpole Swingup; Reacher Three Easy; Cheetah Run Front; Acrobot Swingup; Reacher Hard; Cup Catch; Cartpole Swingup Sparse; Finger Turn Hard
20	Walker Walk Backwards; Finger Turn Easy; Cartpole Swingup; Reacher Three Easy; Cheetah Run Front; Acrobot Swingup; Reacher Hard; Cup Catch; Cartpole Swingup Sparse; Finger Turn Hard; Cartpole Balance Sparse; Hopper Hop Backwards; Pendulum Spin; Cheetah Run Backwards; Fish Swim; Reacher Three Hard; Walker Walk; Finger Spin; Hopper Hop; Walker Run
30	Walker Walk Backwards; Finger Turn Easy; Cartpole Swingup; Reacher Three Easy; Cheetah Run Front; Acrobot Swingup; Reacher Hard; Cup Catch; Cartpole Swingup Sparse; Finger Turn Hard; Cartpole Balance Sparse; Hopper Hop Backwards; Pendulum Spin; Cheetah Run Backwards; Fish Swim; Reacher Three Hard; Walker Walk; Finger Spin; Hopper Hop; Walker Run; Hopper Stand; Cup Spin; Reacher Easy; Cheetah Jump; Pendulum Swingup; Cartpole Balance; Cheetah Run Back; Walker Stand; Cheetah Run; Walker Run Backwards

C.4 DOWNSTREAM CONTROL TASK DATASETS.

In addition to trajectory prediction, we evaluate downstream model-based control using the learned `WestWorld`. We consider two robotic systems: Walker2D and Hopper from OpenAI Gymnasium Towers et al. (2024). For each system, we construct an offline dataset for learning the dynamics model from MPPI rollouts and then evaluate control performance by deploying MPPI online.

MPPI rollout collection. We run MPPI for 100 episodes per system. For Walker2D and Hopper, each episode has 1000 environment steps, with planning horizon $H = 100$ and $N = 256$ sampled action sequences per step. The MPPI cost definition for each task is provided in Appendix G.

Training/validation data construction. At each time step, we rank the N sampled rollouts by MPPI cost and keep only the lowest-cost trajectories. Specifically, for training we retain the lowest 10% rollouts from the 100 MPPI episodes and use them as supervised targets for dynamics learning. To improve data diversity, we additionally include all sampled rollouts from 3 extra episodes as augmentation. For validation, we use 5 episodes and retain the lowest 30% rollouts at each time step for early stopping and model selection. For testing, we report test performance by running MPPI online in the corresponding environment using the learned world model for rollouts.

D Detailed Experimental Settings

We present the detailed experimental settings in this Section. All pre-training experiments are conducted on a server equipped with 4 NVIDIA H200 GPUs, utilizing the PyTorch framework Paszke et al. (2019).

D.1 Hyperparameters for Models

This subsection summarizes the model hyperparameters used in our experiments. For all baseline methods, we follow the official implementations and the hyperparameter choices reported in prior work Yin et al. (2025); Schubert et al. (2023).

TrajWorld. We use a TrajWorld architecture with discretized prediction using 256 uniform bins. The model has 6 Transformer blocks and 4 attention heads, with dropout rate 0.1. The hidden dimension is set to 256. Each block uses an MLP with hidden sizes $[1024,\ 256]$ and GeLU activation.

TDM. We use the TDM architecture with discretized prediction using 256 uniform bins. The model has hidden dimension 384, 6 Transformer blocks, and 4 attention heads, with dropout rate 0.1. Each block uses an MLP with hidden sizes $[1536,\ 384]$ and GeLU activation.

MLPEnsemble. For MLPEnsemble, we train an ensemble of transition models parameterized as a diagonal Gaussian over the next state, implemented with MLPs and optimized by negative log-likelihood using bootstrapped training samples. We use 7 MLPs, each with four hidden layers of width 640 (i.e., $[640,\ 640,\ 640,\ 640]$), and select the top 5 *elite* models by validation loss. To handle heterogeneous systems, we pad state and action vectors to fixed sizes before concatenation, producing a fixed-dimensional input to the MLP.

WestWorld. Our model follows the same discretized prediction setup as TrajWorld, using 256 uniform bins. The backbone consists of 6 Sys-MoE blocks with 4 attention heads, hidden dimension 256, and dropout rate 0.1. Each block uses an MLP with hidden size 512 and GeLU activation. For the SSM module in our Sys-MoE layer, implemented with a Mamba-style state-space model using state size 64, convolution width 4, expansion factor 2. Unless otherwise noted, each Sys-MoE layer contains $P = 4$ experts, which we select via validation as described below.

Determining the number of experts. Our Sys-MoE uses P experts per Sys-MoE layer, where P is a tunable hyperparameter. We select P using a held-out validation split from the pretraining data. Specifically, we randomly sample 1% of trajectories from the 89 pretraining environments as a validation set and exclude them from optimization. This split is used only for model selection and does not overlap with any test environments. We select $P \in \{2, 4, 6, 8\}$ and report (i) the best validation MAE and (ii) the corresponding training step at which the best MAE is achieved (Table 8). While $P = 8$ achieves the lowest validation MAE, it requires substantially more training (about $1.6\times$ the steps of $P = 4$), leading to higher computational cost. The improvement from $P = 4$ to $P = 8$ is relatively modest on this validation split. Therefore, we use $P = 4$ as the default setting to balance accuracy and pretraining efficiency.

Table 8: Best validation MAE and the optimizer update step $s^\star$ at which the best MAE is achieved during pretraining. We report $s^\star$ in thousands of updates (k). MAE is computed in the normalized space. Lower values indicate better performance. † denotes the setting used in all experiments.

#Experts P	Val. MAE ↓	Best step $s^\star$ (k updates)
2	0.0409	108.6
4†	0.0379	137.2
6	0.0390	176.7
8	0.0340	214.0

D.2 Training and Evaluation Settings

Below, we summarize the training and evaluation details for all experiments.

Pretraining settings: For all baseline methods, we follow the original pretraining protocols reported in prior work Yin et al. (2025); Schubert et al. (2023). Specifically, for *TDM* and *TrajWorld*, we pretrain for a total of *up to* 1M gradient steps using the `Adam` optimizer with batch size 64 and learning rate 1×10^{-4}. We apply dropout with rate 0.05, weight decay 1×10^{-5}, and gradient clipping with norm 0.25. We use a warmup cosine decay learning-rate schedule with $10{,}000$ warmup steps. For *MLPEnsemble*, we pretrain for *up to* 1M gradient steps with `Adam`, batch size 256, and learning rate 1×10^{-4}.

For our method, we pretrain for *up to* 1M gradient steps with `AdamW`, batch size 48, and learning rate 2×10^{-4}, while keeping dropout 0.05, weight decay 1×10^{-5}, gradient clipping norm 0.25, and the warmup cosine decay schedule with $10{,}000$ warmup steps. Each training trajectory segment is clipped or zero-padded to a maximum length of 150 time steps. We use the first one-third of the segment (up to 50 steps) as the history window and predict the remaining future steps (up to 100 steps).

Zero-shot evaluation settings: We evaluate zero-shot long-horizon prediction on three *unseen* environments: *Walker2D* and *Hopper* from OpenAI Gym Towers et al. (2024), and a real-world dataset of a mobile *Franka* manipulator interacting with articulated objects Schiavi et al. (2023). For each dataset, we split each episode into trajectory segments of length 150. For all methods, we use the first 50 time steps as the history input and autoregressively roll out the next 100 time steps. We report MAE and MSE by comparing the 100-step predictions against the ground-truth future trajectory.

Downstream control training settings: For all methods trained on the control datasets, we use the `AdamW` optimizer with learning rate 1×10^{-5} for up to $50{,}000$ steps, with early stopping based on validation performance, and a maximum batch size of 48. Each training trajectory segment is clipped or zero-padded to a maximum length of 100 time steps.

Downstream control evaluation settings: For control evaluation, we deploy each learned world model as the dynamics predictor within an MPPI controller. We set the prediction horizon to $H = 100$ for *Walker2D* and *Hopper*. At each control step, MPPI samples 256 candidate action sequences for *Walker2D* and */Hopper*, and uses the world model to roll out the corresponding trajectories for cost evaluation. All methods use identical MPPI hyperparameters for fair comparison, as detailed in Appendix G.

E Inference-time Latency Comparison

Table 9: Wall-clock latency (ms, mean±std) for multi-step prediction on a single NVIDIA RTX A6000. All methods take $T_{\text{hist}} = 50$ history steps as input ($B = 4$, $D_o = 78$, $D_a = 21$). Results are averaged over 10 repeated runs after warm-up. Lower is better.

Method	Latency (ms) ↓				
	$H = 10$	$H = 30$	$H = 50$	$H = 70$	$H = 100$
TDM	388.85±0.41	697.89±0.64	1099.63±1.41	1602.34±1.33	2558.94±1.64
TrajWorld	**94.62±0.30**	236.63±0.33	379.43±0.59	522.37±0.43	736.67±0.91
Ours	113.61±0.71	**138.03±1.45**	**159.90±0.97**	**183.96±1.01**	**238.52±0.47**

We further compare the inference-time latency of our method with SOTA autoregressive transformer baselines, namely *TDM* Schubert et al. (2023) and *TrajWorld* Yin et al. (2025). All methods use a fixed-length history window of $T_{\text{hist}} = 50$ steps as input, with batch size $B = 4$, observation dimension $D_o = 78$, and action dimension $D_a = 21$. We evaluate multi-step prediction horizons $H \in \{10, 30, 50, 70, 100\}$ on a single NVIDIA RTX A6000 GPU. We report the mean and standard deviation of wall-clock latency (in milliseconds) over 10 repeated runs after warm-up, with CUDA synchronization enabled. For autoregressive baselines, we follow the setup in Yin et al. (2025) and use KV-cache decoding to reduce per-step computational overhead. Table 9 summarizes the inference latency for different methods. Our approach consistently achieves fastest inference for long-horizon predictions except for H=10. This advantage arises from our use of learnable query embeddings, which enable the model to predict all H future steps in a single forward pass. In contrast, the autoregressive baselines generate predictions step by step, and their latency increases approximately

linearly with H. As a result, our method is approximately $3.09\times$ faster than *TrajWorld* and $10.73\times$ faster than *TDM* when $H = 100$.

F Ablation Studies

In this section, we explore the impact of two core components on model performance: 1) knowledge-encoded embedding (KNEE) modular and 2) system-aware Mixture-of-Experts (Sys-MoE) layer. Both ablation studies are performed during pretraining and subsequently evaluated under the same zero-shot experimental setting using three robotic systems: *Hopper*, *Walker2D*, and *Franka*. We report long-horizon dynamics prediction errors using MAE and MSE.

Effect of the KNEE Modular. To isolate the role of structural inductive bias, we remove the knowledge-encoded structural embedding during pretraining (denoted as "w/o structural embedding"). As shown in Table 10, removing structural embedding leads to a clear degradation on *Hopper* and *Walker2D*, which have more complex morphologies, while the drop on *Franka* is smaller. This results show that structural embedding is particularly beneficial for unseen complex robotic systems, where explicitly modeling physical connectivity helps align trajectory representations with system structure and improves zero-shot generalization.

Effect of the Sys-MoE Layer. To assess the importance of the Sys-MoE layer, we replace it with a dense SSM layer. For a fair comparison, we increase the depth of the dense-SSM variant so that its total number of parameters is comparable to that of our model. As shown in Table 10, this replacement degrades performance across tasks despite comparable model capacity. These results indicate that the Sys-MoE design is critical for jointly modeling diverse robotic dynamics, as it mitigates inter-task interference during multi-system training.

Based on the ablation study, we conclude that both KNEE and Sys-MoE are essential for model generalization and scalable pretraining.

Table 10: Ablation results under the zero-shot setting. We ablate the Sys-MoE layer by replacing it with a dense SSM, and ablate the structural encoding by removing it during pretraining. Results are computed in the normalized space and reported as MAE and MSE ($\times 10^{-2}$); lower is better.

Method	Sys-MoE	Structural embedding	Walker2D ($\times 10^{-2}$)		Hopper ($\times 10^{-2}$)		Franka ($\times 10^{-2}$)	
			MAE ↓	MSE ↓	MAE ↓	MSE ↓	MAE ↓	MSE ↓
Ours w/o Sys-MoE	✗	✓	18.707	6.797	15.978	4.630	9.392	2.873
Ours w/o Structural embedding	✓	✗	21.156	7.872	16.227	4.990	7.897	2.707
Ours	✓	✓	**16.350**	**5.064**	**13.731**	**3.368**	**7.737**	**2.539**

G Description of Planning Algorithm

This section summarizes the trajectory optimization procedure used in our downstream control evaluations. Across Walker2D and Hopper, we use a sampling-based MPC method, *Model Predictive Path Integral* (MPPI) Williams et al. (2015), to optimize an open-loop action sequence over a finite horizon using the learned world model. At each control cycle, MPPI samples candidate action sequences, rolls them out through the world model, evaluates their trajectory costs, and updates the nominal action sequence via an exponentially weighted average. The first action is then executed in a receding-horizon manner.

MPPI with a learned world model. Let $\boldsymbol{f}_\theta$ denote the learned dynamics model. Given the current state $\boldsymbol{s}_0$, MPPI optimizes an action sequence $\boldsymbol{a}_{0:H-1} = (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{H-1})$ over a horizon H by rolling out

$$\boldsymbol{s}_{t+1} = \boldsymbol{f}_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t), \qquad t = 0, \ldots, H-1. \tag{12}$$

We maintain a nominal sequence $\boldsymbol{\mu}_{0:H-1}$ and sample N perturbed sequences:

$$\boldsymbol{a}_t^{(n)} = \boldsymbol{\mu}_t + \boldsymbol{\epsilon}_t^{(n)}, \qquad \boldsymbol{\epsilon}_t^{(n)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \qquad n = 1, \ldots, N,\ t = 0, \ldots, H-1. \tag{13}$$

For each sample, we compute the trajectory cost

$$L^{(n)} = \sum_{t=0}^{H-1} \ell\Big(\boldsymbol{s}_t^{(n)}, \boldsymbol{a}_t^{(n)};\, \boldsymbol{s}_t^{\mathrm{ref}}\Big)\,, \tag{14}$$

and assign importance weights using a temperature parameter λ:

$$w^{(n)} \propto \exp\Big(-\tfrac{1}{\lambda}\big(L^{(n)} - L_{\min}\big)\Big), \qquad L_{\min} = \min_n L^{(n)}, \qquad \sum_{n=1}^{N} w^{(n)} = 1. \tag{15}$$

The nominal sequence is updated by the weighted average

$$\boldsymbol{\mu}_t \leftarrow \sum_{n=1}^{N} w^{(n)} \boldsymbol{a}_t^{(n)}, \qquad t = 0, \ldots, H-1. \tag{16}$$

We execute the first action $\boldsymbol{\mu}_0$ and repeat this procedure at the next control cycle.

For consistent reporting across tasks, we evaluate control performance using *accumulated reward* (higher is better), which is specified for each task below.

G.1 WALKER2D

Task. We consider forward walking in the Walker2D environment Towers et al. (2024). Let the Walker2D state at time t be $\boldsymbol{s}_t$ and the control input be $\boldsymbol{a}_t$.

Cost for forward walking. We define the per-step cost as the negative of the standard Walker2D reward (i.e., lower cost corresponds to higher reward), following the Gymnasium formulation Towers et al. (2024):

$$\ell(\boldsymbol{s}_t, \boldsymbol{a}_t) = -\Big(r_{\mathrm{healthy}}(t) + r_{\mathrm{fwd}}(t) - c_{\mathrm{ctrl}}(t)\Big), \tag{17}$$

where $r_{\mathrm{healthy}}(t)$ is a survival bonus when the walker remains healthy, $r_{\mathrm{fwd}}(t)$ measures forward progress, and $c_{\mathrm{ctrl}}(t)$ penalizes large control inputs.

Specifically, let x_t denote the walker horizontal position (root x coordinate) at time t, and let Δt be the simulation time step. The forward progress reward is computed from the root velocity:

$$r_{\mathrm{fwd}}(t) = w_{\mathrm{fwd}}\, \frac{x_{t+1} - x_t}{\Delta t}, \tag{18}$$

with $w_{\mathrm{fwd}} = 0.5$, and the control cost is

$$c_{\mathrm{ctrl}}(t) = w_{\mathrm{ctrl}} \|\boldsymbol{a}_t\|_2^2, \tag{19}$$

with $w_{\mathrm{ctrl}} = 10^{-3}$. The survival bonus is

$$r_{\mathrm{healthy}}(t) = \begin{cases} 1, & \text{if the walker is healthy,} \\ 0, & \text{otherwise,} \end{cases} \tag{20}$$

where the health condition follows Gymnasium Walker2D and requires the state to remain within predefined ranges, including torso height $z \in [0.8, 2.0]$ and torso angle $\theta \in [-1.0, 1.0]$.

To improve stability of MPPI planning, we additionally include an expert-reference tracking term defined in the state space using an expert trajectory from D4RL Fu et al. (2020):

$$\ell_{\mathrm{ref}}(t) = \big(\boldsymbol{s}_t - \boldsymbol{s}_t^{\mathrm{ref}}\big)^{\top} \boldsymbol{Q}\, \big(\boldsymbol{s}_t - \boldsymbol{s}_t^{\mathrm{ref}}\big), \tag{21}$$

where $\boldsymbol{s}_t^{\mathrm{ref}}$ is the reference state at time t and $\boldsymbol{Q}$ is a diagonal weight vector. In our implementation, the reference is defined over the concatenated generalized positions and velocities $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, and we use per-dimension weights

$$\begin{aligned} \boldsymbol{Q}_q &= [0.0,\ 10.0,\ 10.0,\ 10.0,\ 1.0,\ 1.0,\ 10.0,\ 1.0,\ 1.0], \\ \boldsymbol{Q}_{\dot{q}} &= [0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.01]. \end{aligned}$$

where the first position weight is set to 0 to ignore the root x coordinate. We set $\boldsymbol{Q} = [\boldsymbol{Q}_q; \boldsymbol{Q}_{\dot{q}}]$.

Finally, the MPPI objective over horizon H is

$$L_{\text{walk}} = \sum_{t=0}^{H-1} \Big(\ell(\boldsymbol{s}_t, \boldsymbol{a}_t) + \ell_{\text{ref}}(t) \Big). \tag{22}$$

Accumulated reward. We report the episode-level accumulated reward using the standard Walker2D reward:

$$R_{\text{ep}} = \sum_{t=0}^{T-1} \Big(r_{\text{healthy}}(t) + r_{\text{fwd}}(t) - c_{\text{ctrl}}(t) \Big). \tag{23}$$

MPPI parameters. We use temperature $\lambda = 0.25$, horizon $H = 100$, and $N = 256$ sampled trajectories per control cycle.

G.2 HOPPER

Task. We consider forward hopping in the Hopper environment Towers et al. (2024). Let the Hopper state at time t be $\boldsymbol{s}_t$ and the control input be $\boldsymbol{a}_t$.

Cost for forward hopping. We define the per-step cost as the negative of the standard Hopper reward (i.e., lower cost corresponds to higher reward), following the Gymnasium formulation Towers et al. (2024):

$$\ell(\boldsymbol{s}_t, \boldsymbol{a}_t) = -\Big(r_{\text{healthy}}(t) + r_{\text{fwd}}(t) - c_{\text{ctrl}}(t) \Big), \tag{24}$$

where $r_{\text{healthy}}(t)$ is a survival bonus when the hopper remains healthy, $r_{\text{fwd}}(t)$ measures forward progress, and $c_{\text{ctrl}}(t)$ penalizes large control inputs.

Specifically, let x_t denote the hopper's horizontal position (root x coordinate) at time t, and let Δt be the simulation time step. The forward progress reward is

$$r_{\text{fwd}}(t) = \frac{x_{t+1} - x_t}{\Delta t}, \tag{25}$$

and the control cost is

$$c_{\text{ctrl}}(t) = w_{\text{ctrl}} \|\boldsymbol{a}_t\|_2^2, \tag{26}$$

with $w_{\text{ctrl}} = 10^{-3}$. The survival bonus is

$$r_{\text{healthy}}(t) = \begin{cases} 1, & \text{if the hopper is healthy,} \\ 0, & \text{otherwise,} \end{cases} \tag{27}$$

where the health condition follows Gymnasium Hopper and requires the state to remain within predefined ranges (e.g., torso height and angle).

To improve stability of MPPI planning, we additionally include an expert-reference tracking term defined in the position and velocity space using a expert trajectory from D4RL Fu et al. (2020):

$$\ell_{\text{ref}}(t) = \big(\boldsymbol{s}_t - \boldsymbol{s}_t^{\text{ref}}\big)^\top \boldsymbol{Q} \, \big(\boldsymbol{s}_t - \boldsymbol{s}_t^{\text{ref}}\big), \tag{28}$$

where $\boldsymbol{s}_t^{\text{ref}}$ is the reference state at time t, and $\boldsymbol{Q}$ is a diagonal weight matrix with position and velocity weights. In our implementation, we set the position and velocity weights to 0.15 and ignore the root x position in the tracking loss.

Finally, the MPPI objective over horizon H is

$$L_{\text{hop}} = \sum_{t=0}^{H-1} \Big(\ell(\boldsymbol{s}_t, \boldsymbol{a}_t) + \ell_{\text{ref}}(t) \Big). \tag{29}$$

Accumulated reward. We report the episode-level accumulated reward using the standard Hopper reward:

$$R_{\text{ep}} = \sum_{t=0}^{T-1} \Big(r_{\text{healthy}}(t) + r_{\text{fwd}}(t) - c_{\text{ctrl}}(t) \Big). \tag{30}$$

MPPI parameters. We use temperature $\lambda = 0.5$, horizon $H = 100$, and $N = 256$ sampled trajectories per control cycle.