H-LENS: EXTRACTING PHASE-SPACE FROM WORLD MODEL REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

World models hold great promise in learning representations for planning, but the dynamical structure hidden in their latent space is opaque. This also bleeds into the problem of evaluating whether the model's predictions have diverged away from reality. We introduce **H-Lens**, a framework for extracting phase-space dynamics from a world model. Using Hamiltonian Neural Networks (HNN), we extract coordinate and momentum-like variables out of latent states predicted by a world model. We use the learned HNN, to then extract a vector field which can be used to interpret learned dynamics. We further define a phase space error measure that predicts the rollout degradation entirely within the obtained vector field without requiring access to any ground truth states. We evaluate H-Lens on Dreamer, LeWorldModel and DINO-WM on the Pendulum, Mountain Car, and Lunar Lander environments trained on pixel observations. Models are trained and evaluated under default and dynamics-shifted environment parameters. The resulting phase space visualizations and rollout error in the vector field provide a way to inspect the dynamics learned by world models and predict their rollout degradation.

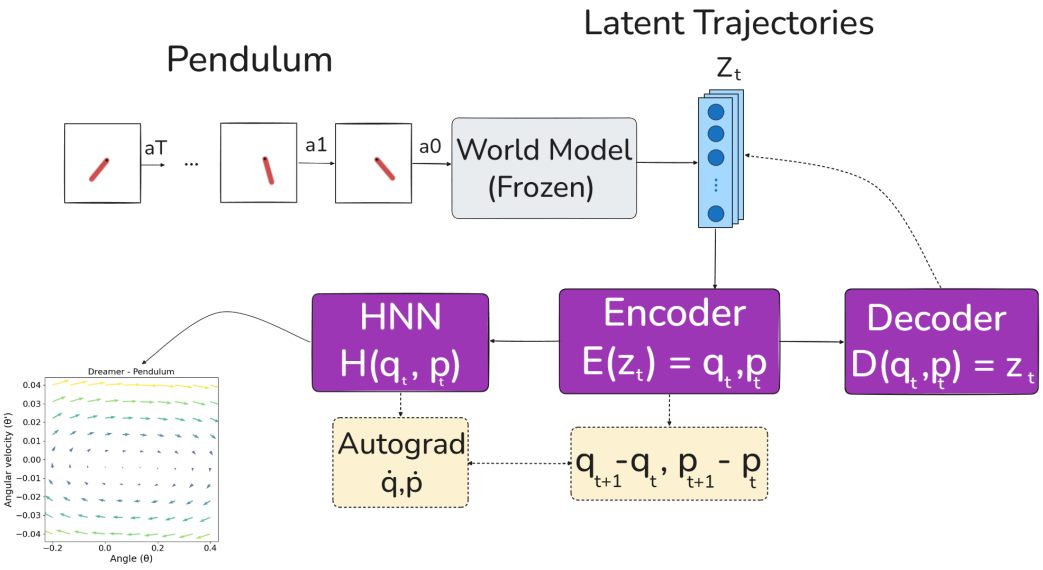


Figure 1: Extracting the phase-space of a world model trained on Pendulum. Latent trajectories (z_t) are generated by supplying context frames and action sequence to a frozen world model. Encoder maps each latent to phase-space coordinates $(\boldsymbol{q}, \boldsymbol{p})$. Hamiltonian Network H_θ, learns an energy-like function. Decoder learns to map $(\boldsymbol{q}, \boldsymbol{p})$ back to world model's latent space. Derivatives of H_θ are used to obtain phase-space.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

World models (Ha & Schmidhuber, 2018) learn a compact predictive representation of the dynamics of the world like mechanics and interactions over time (Maes et al., 2026; Hafner et al., 2025; Lyu et al., 2026). The compact modeling of the present observation and history lends itself to anticipate an observation after a given action or after a rollout of action sequences, and even allows planning directly in latent space. Accuracy of future state predictions and the learned transition dynamics are closely linked, and must be considered jointly when building general agents (Richens et al., 2025). However, the opacity of neural latent spaces makes it difficult to access, interpret, or validate the underlying learned dynamics that drive the model's predictions (Lyu et al., 2026). Particular for critical potential applications of world models (Lyu et al., 2026), a method to extract the dynamical structure to interpret their behavior becomes necessary.

Fortunately, it is hinted at that neural networks do in fact learn the underlying true data-generating variables and, thus, it should be possible to extract human-interpretable state variables as well as their dynamics (Richens et al., 2025; Zhang et al., 2025; Klindt et al., 2026; Ross et al., 2025). Albeit, extraction requires carefully chosen extra assumptions since recovered variables must be both human-interpretable and relevant to the dynamics.

We propose a **H-Lens** for *world model interpretability* to recover an interpretable mechanical dynamical system from the latent trajectories collected from rollouts of a frozen world model. Particularly, we aim to recover the mechanical variables of coordinates and velocity of objects over time. Our key assumption to achieve this is that the underlying dynamics of the environment is Hamiltonian. That is, we assume the learned system can be described by interpretable coordinate variables and their velocities that together approximately follow energy preservation over time. This assumption allows us to train a Hamiltonian Neural Network (HNN) to map latents to a low-dimensional phase-space (Greydanus et al., 2019). Phase-spaces provide a way to jointly represent configuration and motion of a dynamical system. This allows us to analyze the time evolution of a dynamical system. Validation of our method, on altered dynamics (such as modified gravity, pendulum length, modified speed confirmed that H-Lens (i) extracts dynamics that faithfully reflect the learned dynamics and (ii) predicts rollout degradation in a self-supervised explainable fashion. To the best of our knowledge, ours is the first attempt to propose such a framework, opening world models for validation and inference-time self-supervised verification.

Related Work. These aims relate to several lines of existing work. Though they do not recover explicit dynamical models from latent space, there have been first steps towards this (Joseph et al., 2026; Maes et al., 2026), primarily using probing methods to verify some level of physics understanding (properties such as position, velocity) and finetuning on downstream physics tasks (Qu et al., 2026). Recently, there have been attempts to use physics-informed neural networks to evaluate physics understanding in generative models (Tragoudaras et al., 2026), or to use contrastive learning to extract governing equations (Muratore & Mathis, 2026) and build interpretable model based reinforcement learning by construction (Zolman et al., 2025). However, these techniques depend on building surrogate models, verifying the trajectory with expected dynamics using computer vision tools to track objects instead of giving direct access to learned dynamics. Lastly, techniques from concept-based explainability (Poeta et al., 2025) to recover interpretable notions from latent spaces have so far not been applied to world models.

Contributions.

- We introduce a post-hoc pipeline that **recovers interpretable dynamics** from a frozen world model latents using HNNs.
- We introduce a measure to **predict rollout degradation** using the recovered dynamics.

We evaluate our framework on Dreamer (Hafner et al., 2025), LeWorldModel (Maes et al., 2026), and DINO-WM (Zhou et al., 2024) across the Pendulum, LunarLander and MountainCar environments (Brockman et al., 2016).

2 BACKGROUND - HAMILTONIAN NEURAL NETWORKS

Hamiltonian Neural Networks (HNNs) are a type of physics informed network for learning continuous time dynamics (Greydanus et al., 2019). HNNs incorporate Hamiltonian mechanics into the learning process. Rather than predicting the time derivatives of a state, an HNN learns a scalar Hamiltonian function, $H_\theta(\boldsymbol{q}, \boldsymbol{p}) \in \mathbb{R}$, where $\boldsymbol{q}$ captures generalized coordinates (such as position, angle) and $\boldsymbol{p}$ momenta. The Hamiltonian corresponds to the total energy of the system. The dynamics are determined by Hamilton's equations:

$$\dot{\boldsymbol{q}} = \frac{\partial H_\theta}{\partial \boldsymbol{p}}, \qquad \dot{\boldsymbol{p}} = -\frac{\partial H_\theta}{\partial \boldsymbol{q}}. \tag{1}$$

Given trajectories $(\boldsymbol{q}_t, \boldsymbol{p}_t)$ over time t and estimates of $(\dot{\boldsymbol{q}}_t, \dot{\boldsymbol{p}}_t)$, a HNN is trained to minimize

$$\mathcal{L}_{\text{HNN}} = \left| \frac{\partial H_\theta}{\partial \boldsymbol{p}} - \dot{\boldsymbol{q}} \right|^2 + \left| -\frac{\partial H_\theta}{\partial \boldsymbol{q}} - \dot{\boldsymbol{p}} \right|^2. \tag{2}$$

Using automatic differentiation, the derivatives of H_θ are computed. HNNs can also be extended to high-dimensional observations such as images. This is demonstrated by (Greydanus et al., 2019) on pixel observations of a pendulum. When there is no direct access to $(\boldsymbol{q}_t, \boldsymbol{p}_t)$, consecutive image frames $(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$ are passed through an encoder that produces representations, which are interpreted as $(\boldsymbol{q}_t, \boldsymbol{p}_t)$. Using consecutive frames allows the network to infer information about momentum-like variables. This low-dimensional representation is then provided to the HNN and trained end-to-end. With additional reconstruction loss and coordinate loss, the network recovers the dynamics:

$$\mathcal{L}_{\text{rec}} = \left| (\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) - (\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{x}}_{t+1}) \right|^2 \qquad \text{and} \qquad \mathcal{L}_{\text{CC}} = \left| \boldsymbol{p}_t - (\boldsymbol{q}_{t+1} - \boldsymbol{q}_t) \right|^2. \tag{3}$$

This extension shows that it is possible to recover phase space from high-dimensional data. The HNN is used here as a *coordinate-recovery mechanism*. Inductive bias imposed by the HNN encourages the recovered representation to organize latent dynamics into a coherent phase space.

3 METHOD

We use a frozen world model to extract an interpretable dynamical description. Our pipeline contains four stages: (i) Generating latent trajectories with a frozen world model, (ii) Learning an energy function H_θ using HNNs, (iii) Recovering a low-dimensional phase-space representation using the derivatives of the trained H_θ and (iv) Measuring rollout consistency using the obtained dynamics. Figure 1 illustrates the architecture. We collect rollouts (latent trajectories) since that requires making predictions beyond one-step, thus requiring making use of learned dynamics.

3.1 LATENT TRAJECTORIES FROM A WORLD MODEL

Let $\boldsymbol{x}_0$ be an initial observation and let $\boldsymbol{a}_0, ..., \boldsymbol{a}_{T-1}$ be an action sequence. A world model WM_θ produces a latent trajectory of predictions,

$$\boldsymbol{z}_{t+1} = \text{WM}_\theta(\boldsymbol{z}_t, \boldsymbol{a}_t), \qquad t = 0, \ldots, T-1, \tag{4}$$

where $\boldsymbol{z}_t \in \mathbb{R}^{d_z}$ is the latent state at time t. We collect trajectories under multiple initial conditions and action sequences, yielding

$$\mathcal{D}_z = \left\{ \left(\boldsymbol{z}_0^{(i)}, \ldots, \boldsymbol{z}_{T_i}^{(i)} \right) \;\middle|\; i = 1, \ldots, N \right\}. \tag{5}$$

The world models are trained on different action values. However, here action sequence $\boldsymbol{a}_0, ..., \boldsymbol{a}_{T-1}$ is zero since we want the trajectories to follow "natural paths" to extract the underlying learned dynamics. It is also to be noted that T is usually a small number ($T \leq 10$). We assume that during the initial rollout steps, the predictions of the dynamics model (component of the world model that takes in context encoding and action to predict the next latent state) is closer to the representations learned by the encoder (of the world model). Alternatively, one can also compute encodings directly from the world model's encoder by feeding in a sequence of frames.

3.2 USING HNNS TO DISCOVER DYNAMICS

The latent encodings from the world model are generally not aligned with physical variables. They may be nonlinear mixtures of quantities such as position, velocity, or momentum. So, we learn an encoder

$$f_\phi : \mathbb{R}^{d_z} \to \mathbb{R}^{d_q} \times \mathbb{R}^{d_p},$$

which maps each latent independently to phase-space variables: $(\boldsymbol{q}_t, \boldsymbol{p}_t) = f_\phi(\boldsymbol{z}_t)$. The encoder g_ψ receives a *single* latent $\boldsymbol{z}_t$. Since $\boldsymbol{z}_t$ is the latent state predicted by the world model, it is expected to contain relevant information about both configuration and motion. The encoder disentangles this information into coordinate-like variables $\boldsymbol{q}_t$ and momentum-like variables $\boldsymbol{p}_t$ using the HNN losses described in Equations. 2 and 3. That way,

$$g_\psi \colon \mathbb{R}^{d_q} \times \mathbb{R}^{d_p} \to \mathbb{R}^{d_z} \qquad \text{reconstructs the latent state} \qquad \hat{\boldsymbol{z}}_t = g_\psi(\boldsymbol{q}_t, \boldsymbol{p}_t). \tag{6}$$

Using $(\boldsymbol{q}_t, \boldsymbol{p}_t)$ from the encoder f_ϕ, another neural network H_θ learns a scalar Hamiltonian $H_\theta(\boldsymbol{q}_t, \boldsymbol{p}_t) \in \mathbb{R}$, whose gradients define the predicted dynamics described in Eq 1.

Consecutive states $(\boldsymbol{q}_t, \boldsymbol{p}_t)$ and $(\boldsymbol{q}_{t+1}, \boldsymbol{p}_{t+1})$ are used with time interval Δt, to estimate

$$\dot{\boldsymbol{q}}_t \approx \frac{\boldsymbol{q}_{t+1} - \boldsymbol{q}_t}{\Delta t}, \qquad \dot{\boldsymbol{p}}_t \approx \frac{\boldsymbol{p}_{t+1} - \boldsymbol{p}_t}{\Delta t}. \tag{7}$$

The full objective for the encoder f_ϕ, HNN H_θ, and decoder g_ψ is

$$\mathcal{L} = \mathcal{L}_{\text{HNN}} + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{cc}}\mathcal{L}_{\text{cc}}, \tag{8}$$

as given in Equation 2, 3, respectively. The architecture is shown in Figure 1.

3.3 VISUALIZING THE PHASE-SPACE VECTOR FIELD

After training, the encoder f_ϕ and H_θ define a dynamical system in the recovered phase space. We visualize the dynamics of this phase space as a *vector field*. It assigns a vector to every point $(\boldsymbol{q}, \boldsymbol{p})$. Each vector $(\dot{\boldsymbol{q}}, \dot{\boldsymbol{p}})$ is the direction at which the system evolves from $(\boldsymbol{q}, \boldsymbol{p})$. We derive the vector field using H_θ. Hamilton's equations specify the vector associated with a point $(\boldsymbol{q}, \boldsymbol{p})$:

$$\boldsymbol{v}(q, p) = [\dot{q}\ , \dot{p}] = \left[\frac{\partial H_\theta(q,p)}{\partial p}\ , -\frac{\partial H_\theta(q,p)}{\partial q}\right]. \tag{9}$$

We obtain $q_{\min}, q_{\max}$ and $p_{\min}, p_{\max}$ for $(\boldsymbol{q}, \boldsymbol{p})$, by passing a collection of predicted latents z, obtained from the world model through the encoder f_ϕ. We construct a regular grid over the resulting rectangular region,

$$\mathcal{G} = \left\{(q^{(j)}, p^{(k)}) \mid q^{(j)} \in [q_{\min}, q_{\max}], ; p^{(k)} \in [p_{\min}, p_{\max}]\right\}. \tag{10}$$

For every grid point $(q^{(j)}, p^{(k)})$, we use automatic differentiation to compute $v(p, q)$ from Eq. 9. Then we draw a vector at each grid point with $\dot{q}$ and $\dot{p}$ as the horizontal and vertical components, respectively. This results in the vector field.

3.4 MEASURING ROLLOUT CONSISTENCY IN THE VECTOR FIELD

We measure rollout consistency using the obtained vector field. We first encode each predicted world model latent into phase space coordinates,

$$\boldsymbol{s}_t = [\boldsymbol{q}_t, \boldsymbol{p}_t] = f_\phi(\boldsymbol{z}_t). \tag{11}$$

At each point $\boldsymbol{s}_t$, we compute a local vector,

$$\hat{\dot{\boldsymbol{s}}}_t = \left[\frac{\partial H_\theta}{\partial \boldsymbol{p}_t}\ , -\frac{\partial H_\theta}{\partial \boldsymbol{q}_t}\right]. \tag{12}$$

This vector indicates the expected direction for the next state according to learned phase space dynamics. The actual direction of the encoded state is given by the world model rollout,

$$\dot{\boldsymbol{s}}_t \approx \frac{\boldsymbol{s}_{t+1} - \boldsymbol{s}_t}{\Delta t}. \tag{13}$$

We define the total rollout error until step t as:

$$\mathcal{E}_{\text{phase}}(t) = \sum_{\tau=1}^{t} \left| \hat{\dot{\boldsymbol{s}}}_t - \dot{\boldsymbol{s}}_t \right|_2 . \tag{14}$$

Equivalently, we compare the point predicted by the local HNN arrow, $\boldsymbol{s}_t + \Delta t \hat{\dot{\boldsymbol{s}}}_t$, with the rollout-predicted next point $\boldsymbol{s}_{t+1}$. As the rollout trajectory begins to disagree with the vector field, the error increases. This is a *label-free* measure in the sense that it does not require any access to ground truth frames or states. It assesses whether the world model's own predicted trajectory is consistent with the recovered dynamical structure.

4 Experiments

In the following experiments, (i) we validate whether our approach faithfully captures latent mechanical dynamics in world models (subsection 4.1, and (ii) in how far disagreement of rollout and extracted vector field dynamics indicate rollout errors, thus serving as label-free inference-time monitoring mechanism (subsection 4.3).

World Model Architectures. We use three world model architectures to evaluate our approach, namely DreamerV3 (Hafner et al., 2025), LeWorldModel (Maes et al., 2026) and DINO-WM (Zhou et al., 2024). These models depend on different approaches to world modelling.

Environments. We use three control environments: Pendulum, MountainCar and LunarLander (Brockman et al., 2016). The dynamics that govern the Pendulum are angle θ and angular velocity $\dot{\theta}$. The state of MountainCar is represented by position x and velocity $\dot{x}$. LunarLander has a two-dimensional state x, y and velocities $\dot{x}, \dot{y}$ where y is affected by gravity. Observations from all environments are grayscale images of size 32×32. We train a second set of world models on the same environments but with dynamics-shifted parameters. This serves as control for testing whether the recovered dynamics reflect the changes in the environment dynamics, see subsection 4.1.

4.1 Dynamics-shifted Evaluation

We evaluate two sets of models under two conditions. The *default* condition uses the default environment parameters during training. The *dynamics-shifted* condition changes parameters that affect the environment dynamics during training. We leave the observation and action spaces unchanged. The parameter changes are as follows [1]:

- **Pendulum:** the pendulum length is changed from $l = 1.0$ to $l = 0.5$.
- **LunarLander:** gravity is changed from $g = -10$ to $g = -1$.
- **MountainCar:** the engine power is changed from 0.0015 to 0.00225, and the maximum speed is changed from 0.07 to 0.105.

We choose values at the extremes of the allowed parameter range. These values help to make changes in dynamics more pronounced. Thus, allowing us to investigate the difference between obtained dynamics. Lowering Pendulum length allows for higher angular velocity ($\dot{\theta}$) to be attained, lowering gravity in Lunar Lander results in less steeper downward pull ($\dot{y}$) and increasing power and maximum speed in MoutainCar results in higher speeds ($\dot{x}$) and more covered distance (x).

4.2 Metrics and Baselines

Latent Probing. We evaluate each frozen world model in both *default* and *dynamics-shifted* conditions using linear and non-linear probes. The probing dataset for all setups contains 32 episodes of 100 steps. The probes receive world model latent states and learn to map them to ground-truth states which are supplied during training.

The linear probe looks for a linear recovery of target variables while non-linear probes look for a non-linear encoding of the same information. Probe implementation details are described in Appendix A.7. We report probing performance for each model across each environment and across

[1] Environment parameter settings: Stable World Model environments and Gymnasium documentation.

each dynamics condition. We use probing as a baseline to assess how explicitly the underlying state is represented in a world model's latent space. Lower probing errors suggest underlying state variables are easily recoverable and high probing errors the opposite.

We evaluate all combinations under three world models, three environments, and two dynamics conditions. We report linear and non-linear probe performance.

World Model Training. Training episodes are collected using a uniform random action policy. For Dreamer, we collect $10{,}000$ training episodes of 100 time steps for each environment. The model is trained for approximately $100{,}000$ optimization steps. For LeWM and DINO-WM, 100 time steps and train each model for 10 epochs. Training continues until convergence. Further details about the training settings are given in Appendix A.7.

We freeze the world model parameters after training. The same frozen models are used for all subsequent analyses involving probing, HNN training, vector field visualization and rollout consistency evaluation.

Rollout Collection Policy. All evaluation trajectories are generated using a zero-action policy (in contrast to world model training, where the actions were random). To understand the true underlying learned dynamics, it is useful to remove sources of variations that are not intrinsic to the system. This isolates intrinsic dynamics represented by the world model. The same zero-action policy is used for all the evaluations described below.

Hamiltonian Neural Networks. We train separate HNNs with the same architecture across world models and environment variations. We train HNNs on the latents predicted by a frozen world model. The dimensionality of the canonical representation is selected according to the number of dynamical variables required to describe each environment as summarized in Table 1.

Table 1: Phase-space dimensions and their physical interpretations.

Environment	Dimension	Intended interpretation
Pendulum	2	$(q, p) \approx (\theta, \dot{\theta})$
MountainCar	2	$(q, p) \approx (x, v_x)$
LunarLander	4	$(q, p) \approx ((x, y), (v_x, v_y))$

For each experimental condition, we collect latent predictions of $5{,}000$ episodes with varying initial conditions. To contain the contamination from rollout degradation, we collect only first five open loop predictions. The initial condition and context length k vary across episodes before the open-loop predictions. HNN implementation details are provided in Appendix A.7. For each combination of experimental setup an HNN is trained.

4.3 Relating Phase Space Error to Rollout Degradation

We investigate whether the local phase-space error described in Eq. 14 can be related to world model rollout degradation. We collect two latent trajectories from the same episode: an open-loop trajectory and an observation-conditioned trajectory (the latent state at each step is inferred from the corresponding ground-truth observation[2]). The observation-conditioned latent is not a ground truth state. But it serves as a reference latent produced by the same world model when ground truth observation was supplied. This assumes the world model has learned meaningful representations.

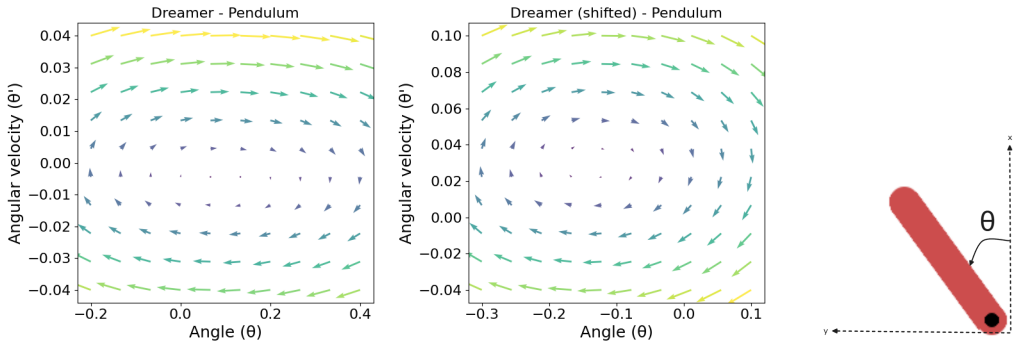
At each rollout step t, we measure the discrepancy between open-loop prediction and observation-conditioned latents using mean squared error. For each episode and rollout horizon, we obtain a paired observation $(\mathcal{E}_{\text{phase}}(t), \mathcal{E}_{\text{latent}}(t))$ where $\mathcal{E}_{\text{latent}}(t) = \left\| \boldsymbol{z}_t^{\text{OL}} - \boldsymbol{z}_t^{\text{GT}} \right\|_2^2$. Here, $\boldsymbol{z}_t^{\text{OL}}$ is the open-loop predicted encoding and $\boldsymbol{z}_t^{\text{GT}}$ is the reference encoding obtained by conditioning the model on ground-truth observations.

[2]The exact mechanism depends on the architecture. For example, in LeWM this means passing observations into its encoder whereas in Dreamer it is the output from its posterior component

Table 2: Probe Results. Linear and nonlinear probe MSE for default (ID) and dynamics-shifted models. Lower is better. Values are reported as mean ± standard deviation.

Environment	Model	Condition	Linear MSE (↓)	Nonlinear MSE (↓)
Pendulum	Dreamer	ID	0.1838 ± 0.3857	0.0223 ± 0.0616
Pendulum	Dreamer	Shifted	0.4759 ± 0.7544	0.1339 ± 0.2467
Pendulum	LeWM	ID	4.6805 ± 4.9887	4.9109 ± 5.3419
Pendulum	LeWM	Shifted	6.6659 ± 6.7023	6.4232 ± 6.5843
Pendulum	DINO-WM	ID	0.6783 ± 1.8309	0.1564 ± 0.8008
Pendulum	DINO-WM	Shifted	5.0404 ± 6.3738	1.1906 ± 3.1150
MountainCar	Dreamer	ID	$6.1\times10^{-5} \pm 1.0\times10^{-4}$	$1.9\times10^{-5} \pm 3.3\times10^{-5}$
MountainCar	Dreamer	Shifted	204.61 ± 7317.43	22.99 ± 822.24
MountainCar	LeWM	ID	$3.9\times10^{-4} \pm 1.2\times10^{-3}$	$5.1\times10^{-5} \pm 8.3\times10^{-5}$
MountainCar	LeWM	Shifted	$1.5\times10^{-3} \pm 5.4\times10^{-3}$	$3.9\times10^{-4} \pm 3.7\times10^{-3}$
MountainCar	DINO-WM	ID	$2.7\times10^{-5} \pm 1.3\times10^{-4}$	$6.7\times10^{-6} \pm 8.4\times10^{-6}$
MountainCar	DINO-WM	Shifted	$5.0\times10^{-5} \pm 7.8\times10^{-5}$	$1.8\times10^{-5} \pm 3.2\times10^{-5}$
LunarLander	Dreamer	ID	0.0516 ± 0.2439	0.0266 ± 0.2365
LunarLander	Dreamer	Shifted	0.0484 ± 0.0623	0.0043 ± 0.0104
LunarLander	LeWM	ID	0.0529 ± 0.0968	0.0283 ± 0.0727
LunarLander	LeWM	Shifted	0.1618 ± 0.1746	0.0749 ± 0.0884
LunarLander	DINO-WM	ID	0.0519 ± 0.2487	0.0247 ± 0.1699
LunarLander	DINO-WM	Shifted	0.0224 ± 0.0262	0.0023 ± 0.0038

5 RESULTS

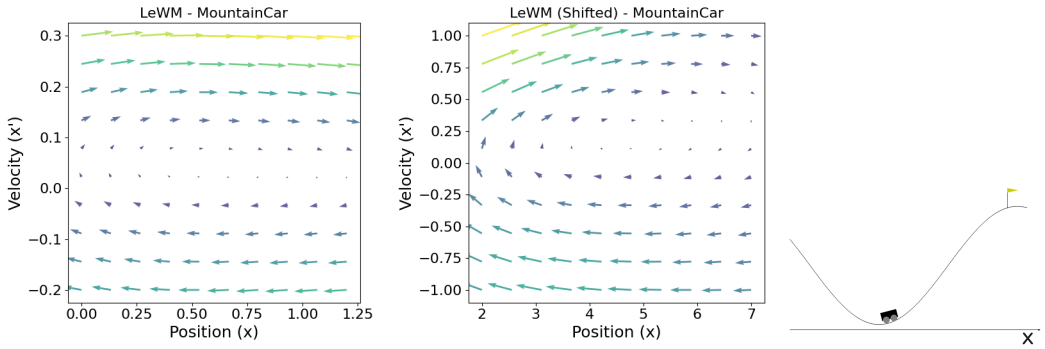


(a) Dreamer – ID (b) Dreamer – Shifted (c) Pendulum Setup

Figure 2: Recovered Pendulum phase-space for Dreamer. Since for Dreamer-Shifted, the pendulum length is shorter, it attains a higher velocity range. See Figure 10 for all obtained vector fields.

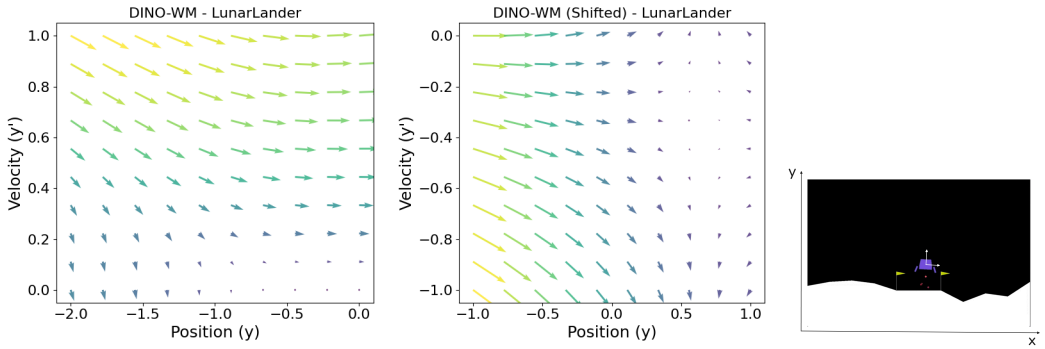
Probe Results Table 2 reports linear-probe error and nonlinear-probe error. Generally for each setting, across models, the probing errors are somewhat similar. Dreamer-shifted for MountainCar reports an unusually high probing error compared to other models in the same setting suggesting that the converged model has not learned readily separable representations.

Does H-Lens faithfully reflect learned dynamics? As a qualitative observation, for Pendulum across ID and dynamics-shifted world models, we see the expected phase-space of concentric circles (Figure 2; see expected phase-spaces in Appendix subsection A.5). The same is repeated in MountainCar and the vertical coordinate (y) of Lunar Lander as well (Figure 3, Figure 4; see Appendix subsubsection A.5.1 for LunarLander x coordinate phase-space). We notice important expected changes in the recovered vector fields. For Pendulum, the velocities in the shifted-dynamics condition are higher since a shorter length allows the pendulum to swing faster. For MountainCar Figure 3, the power and maximum speed are higher, which allows the car to reach higher speeds and cover more distance. We can also see in MountainCar ID that the velocities flatten towards the



(a) LeWM – ID (b) LeWM – Shifted (c) MountainCar Setup

Figure 3: Recovered MountainCar phase-space for LeWM. Since the shifted model is allowed higher power and speed, it attains higher velocity. See Figure 12 for all obtained vector fields.



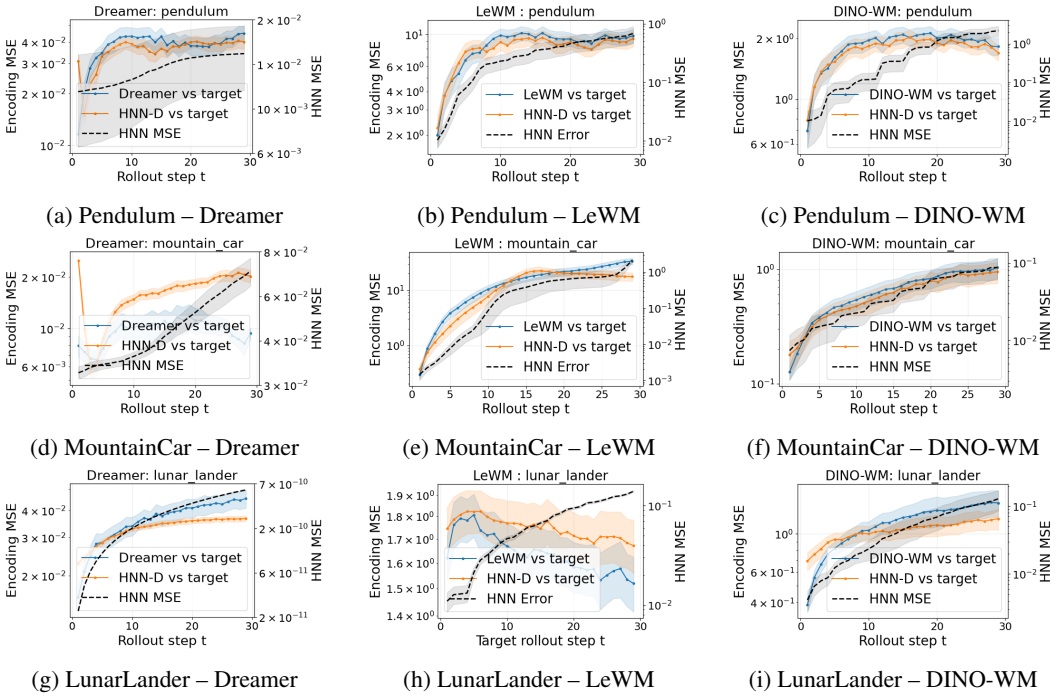
(a) DINO-WM – ID (b) DINO-WM – Shifted (c) LunarLander Setup

Figure 4: Recovered LunarLander phase-space for DINO-WM along the vertical axis. DINO-ID starts off steeper than DINO-WM-Shifted due to stronger gravity. See Figure 14 for all obtained vector fields.

extreme values since the default values allow lower maximum speed. We see similar trends playing out in LunarLander as well (Figure 4). Since the dynamics-shifted setup has the lowest gravity, the vector field contains less steep velocity compared to ID. Dreamer-shifted in MountainCar does not recover any interpretable structure (see Appendix Figure 12) due to the high probing error. It should be noted here that H-Lens is not trying to recover ground truth dynamics. Instead, it recovers structures that are sensitive to changes in learned dynamics. In our settings, the qualitative changes observed align with the expected changes.

Does $\mathcal{E}_{\text{phase}}(t)$ effectively predict rollout error? We see in Figure 5 a general trend of $\mathcal{E}_{\text{phase}}(t)$ aligning with the $\mathcal{E}_{\text{latent}}(t)$ error. Interestingly, this trend is stronger where HNN-D is also aligned with $\mathcal{E}_{\text{latent}}(t)$ (see subsection A.2 for correlations). HNN-D is the error obtained by mapping the world-model rollouts into the recovered phase-space $(\boldsymbol{q}, \boldsymbol{p})$, and decoding the resulting phase-space trajectory back into latent space using the HNN decoder before comparing it with the target encoding [3]. We suspect this is due to the HNN decoder forcing the phase-space to be aligned with the latent space of the world model. LeWM on LunarLander is an exception. It achieves only slightly higher probing error (see Table 2) compared to other models on Lunar Lander, but the two error measures show weak alignment. A reason may be because the latent discrepancy saturates. This should be taken into account along with the fact that ground-truth-conditioned encodings are used as an imperfect reference. In general, these results suggest that $\mathcal{E}_{\text{phase}}(t)$ can serve as a proxy for rollout degradation insofar as the HNN decoder faithfully maps the phase-space dynamics back into a world

[3] In simpler terms, it is the HNN Decoder error



(a) Pendulum – Dreamer (b) Pendulum – LeWM (c) Pendulum – DINO-WM

(d) MountainCar – Dreamer (e) MountainCar – LeWM (f) MountainCar – DINO-WM

(g) LunarLander – Dreamer (h) LunarLander – LeWM (i) LunarLander – DINO-WM

Figure 5: Three rollout-error curves. The HNN MSE is $\mathcal{E}_{\text{phase}}(t)$. The encoding MSE is $\mathcal{E}_{\text{latent}}(t)$. HNN-D is the same predicted rollout in the phase-space, decoded back into latent space using the HNN decoder, and compared with the target encoding.

model's latent space. However, it should be interpreted as an internal consistency signal rather than as ground-truth prediction error.

6 Discussion and Conclusion

We introduce **H-Lens** as a framework for recovering dynamical structuresfrom trained world models. By making a *Hamiltonian* assumption, the dynamics can be extracted from latent states. H-Lens produces a visual and quantitative description of the learned dynamics. With the learned HNN, H_θ can then be used to predict the quality of rollouts without any access to ground-truth frames. H-Lens may also potentially support rollout correction. If the predicted rollout drifts away, it could be projected to a nearby locally consistent state and decoded back to latent space to continue rollout. Such a correction could improve longer horizon predictions.

The learned energy-like function H_θ can support applications beyond what is analyzed in this paper. H-Lens could serve as an interface between learned representations of a world model and dynamical systems. Moreover, being able to move between latent space and phase-space of a world model may allow interventions and counter-factual scenarios to be investigated.

Limitations The recovered phase-space should not be interpreted as uniquely identified physical coordinates. The trained encoders may shift, re-scale, flip the signs, or may apply other transformations while preserving dynamics. Therefore, the focus should be on recovered structure rather than ground truth values of coordinates. This study has investigated the dynamics for single objects and systems that are Hamiltonian. Future work should investigate extending the framework to systems that are not Hamiltonian (Sosanya & Greydanus, 2022), and extend evaluation to multi-object systems. Despite its limitations, H-Lens opens up a direction toward making the dynamics of world-model representations inspectable and their dynamics predictable.

ACKNOWLEDGMENTS

removed to maintain anonymity.

AI USE STATEMENT

In this work, we used generative AI tools for grammar and spelling checking, language polishing, identifying potentially relevant papers, and writing or debugging auxiliary code for routine tasks such as plotting and data visualization. We have not used generative AI tools to design or implement the core learning methods, generate experimental data, conduct the experiments, or interpret the results. All other required disclosure tasks are not applicable to this work.

All AI-assisted code was reviewed, tested, and checked by the authors, and all suggested references were independently verified against the original publications. We take responsibility for the final content of this work, including all text, claims, code, plots, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We describe the complete experimental pipeline in the main text, including the world-model architectures, latent-space probing procedure, H-Lens architecture, HNN training objective, vector-field construction, and rollout-error measurement. The experimental setup specifies the three world models, control environments, image resolution, trajectory-collection policies, default and dynamics-shifted parameters, and evaluation policy. Additional implementation details, including model configurations, optimization settings, training schedules, probe architectures, HNN dimensions, data-collection procedures, and environment parameters, are provided in the appendix. All trajectories are generated procedurally from the specified environments, so no proprietary dataset is required. The code, configuration files, and scripts used to train the models, probes and compute error measures, and generate vector fields will be released upon acceptance.

REFERENCES

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks, 2019. URL https://arxiv.org/abs/1906.01563.

David Ha and Jürgen Schmidhuber. World Models. March 2018. doi: 10.5281/zenodo.1207631.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pp. 1–7, 2025.

Sonia Joseph, Quentin Garrido, Randall Balestriero, Matthew Kowal, Thomas Fel, Shahab Bakhtiari, Blake Richards, and Mike Rabbat. Interpreting physics in video world models, 2026. URL https://arxiv.org/abs/2602.07050.

David Klindt, Yann LeCun, and Randall Balestriero. When does lejepa learn a world model? *arXiv preprint arXiv:2605.26379*, 2026.

Qi Lyu, Jiahua Dong, Baichen Liu, Xudong Wang, Wenqi Liang, Duzhen Zhang, Jiahang Tu, Hongliu Li, Hanbin Zhao, Henghui Ding, Yulun Zhang, Zhi Han, Nicu Sebe, Fahad Shahbaz Khan, Salman Khan, Mubarak Shah, Philip Torr, Ming-Hsuan Yang, and Dacheng Tao. Learning to model the world: A survey of world models in artificial intelligence. *TechRxiv*, 2026(0305), 2026. doi: 10.36227/techrxiv.177274570.09578608/v1. URL https://www.techrxiv.org/doi/abs/10.36227/techrxiv.177274570.09578608/v1.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworld-model: Stable end-to-end joint-embedding predictive architecture from pixels. 2026.

Paolo Muratore and Mackenzie Weygandt Mathis. Extracting governing equations from latent dynamics via multi-view contrastive learning, 2026. URL https://arxiv.org/abs/2606.13260.

Eleonora Poeta, Gabriele Ciravegna, Eliana Pastor, Tania Cerquitelli, and Elena Baralis. Concept-based Explainable Artificial Intelligence: A Survey. *ACM Comput. Surv.*, November 2025. ISSN 0360-0300. doi: 10.1145/3774643.

Helen Qu, Rudy Morel, Michael McCabe, Alberto Bietti, François Lanusse, Shirley Ho, and Yann LeCun. Representation learning for spatiotemporal physical systems, 2026. URL https://arxiv.org/abs/2603.13227.

Jonathan Richens, David Abel, Alexis Bellot, and Tom Everitt. General agents contain world models. *arXiv preprint arXiv:2506.01622*, 2025.

Edmund Ross, Claudia Drygala, Leonhard Schwarz, Samir Kaiser, Francesca di Mare, Tobias Breiten, and Hanno Gottschalk. When do world models successfully learn dynamical systems?, 2025. URL https://arxiv.org/abs/2507.04898.

Andrew Sosanya and Sam Greydanus. Dissipative hamiltonian neural networks: Learning dissipative and conservative dynamics separately, 2022. URL https://arxiv.org/abs/2201.10085.

Antonios Tragoudaras, Chenyu Zhang, Daniil Cherniavskii, Antonios Vozikis, Thijmen Nijdam, Derck W. E. Prinzhorn, Mark Bodracska, Nicu Sebe, Andrii Zadaianchuk, and Efstratios Gavves. Evaluating newtonian mechanics in video generative models with real physical systems, 2026. URL https://arxiv.org/abs/2504.02918.

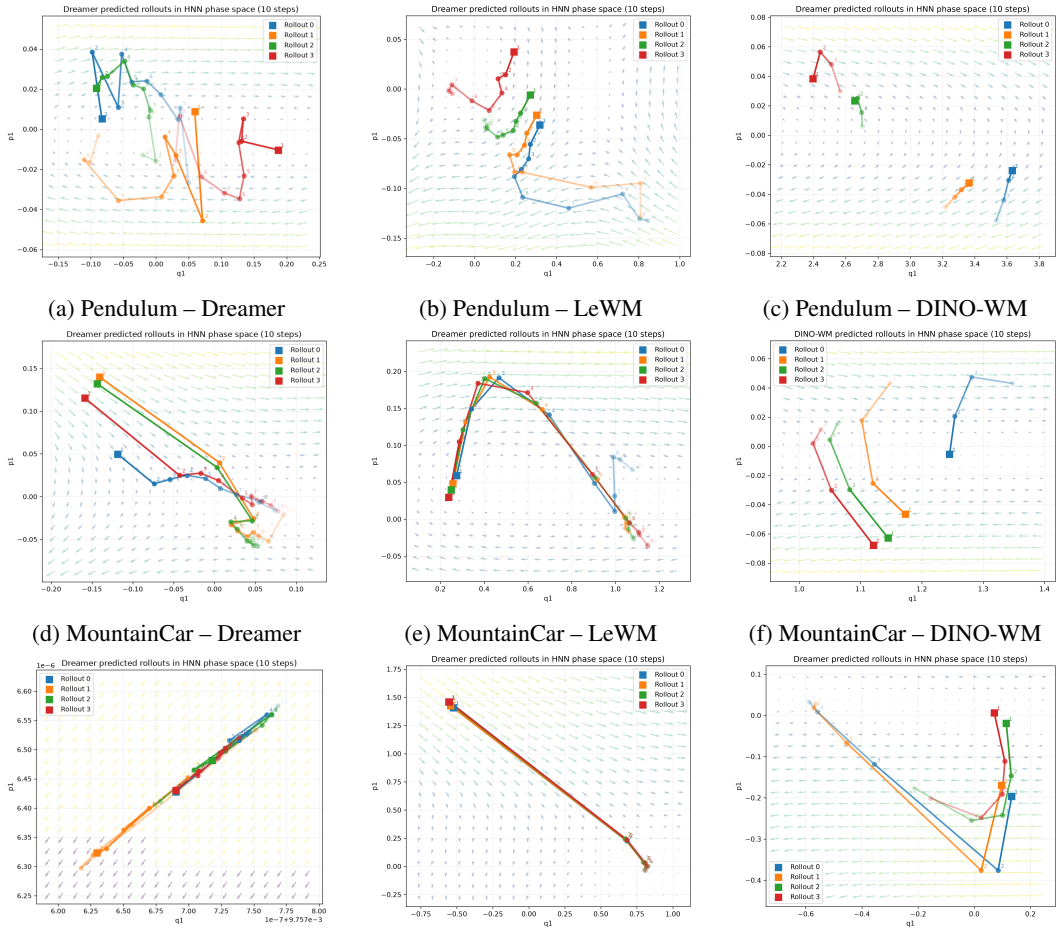
Tianren Zhang, Guanyu Chen, and Feng Chen. When do neural networks learn world models? In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 74639–74670. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhang25j.html.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning, 2024. URL https://arxiv.org/abs/2411.04983.

Nicholas Zolman, Christian Lagemann, Urban Fasel, J Nathan Kutz, and Steven L Brunton. Sindy-rl for interpretable and efficient model-based reinforcement learning. *Nature Communications*, 16 (1):10714, 2025.

A Appendix

A.1 Rollouts in Phase-Space



(a) Pendulum – Dreamer (b) Pendulum – LeWM (c) Pendulum – DINO-WM

(d) MountainCar – Dreamer (e) MountainCar – LeWM (f) MountainCar – DINO-WM

(g) LunarLander – Dreamer (h) LunarLander – LeWM (i) LunarLander – DINO-WM

Figure 6: World-model trajectories overlaid on the recovered phase-space vector fields under ID evaluation.

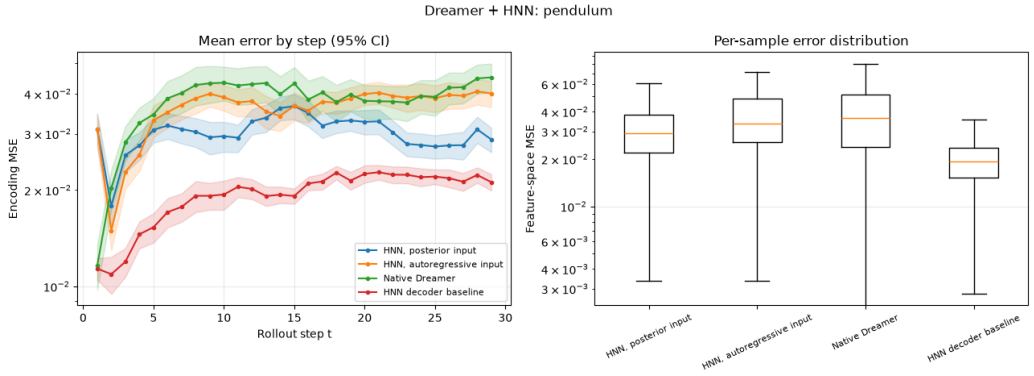
A.2 HNN PHASE SPACE ERROR CORRELATIONS

Table 3: Correlation between H-Lens error measures and world-model rollout error. ρ_{phase} denotes the correlation between $\mathcal{E}_{\text{phase}}$ and $\mathcal{E}_{\text{latent}}$. $\rho_{\text{HNN-D}}$ denotes correlation between HNN-D and $\mathcal{E}_{\text{latent}}$.

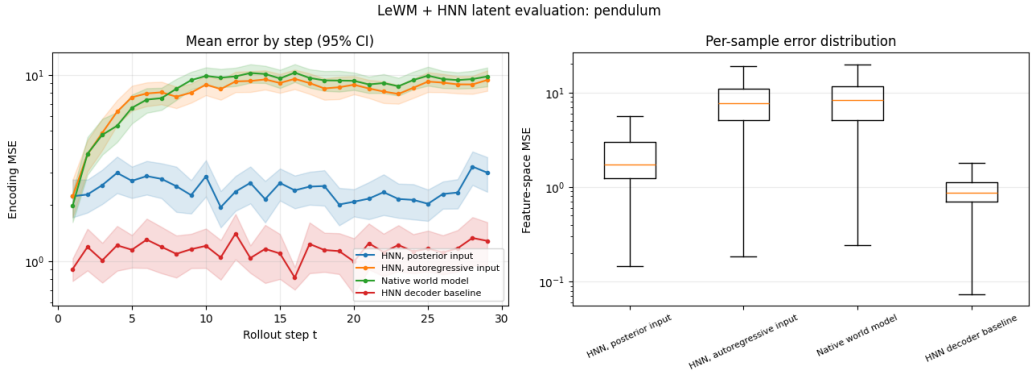
World Model	**Environment**	ρ_{phase}	$\rho_{\text{HNN-D}}$
DINO-WM	MountainCar	0.9532	0.9962
	Pendulum	0.3970	0.9878
	LunarLander	0.8895	0.9924
LeWM	MountainCar	0.8900	0.8676
	Pendulum	0.7022	0.9669
	LunarLander	-0.9016	0.9399
Dreamer	MountainCar	0.2228	0.4853
	Pendulum	0.4928	0.7329
	LunarLander	0.9423	0.9850

A.3 Additional Rollout Error Plots

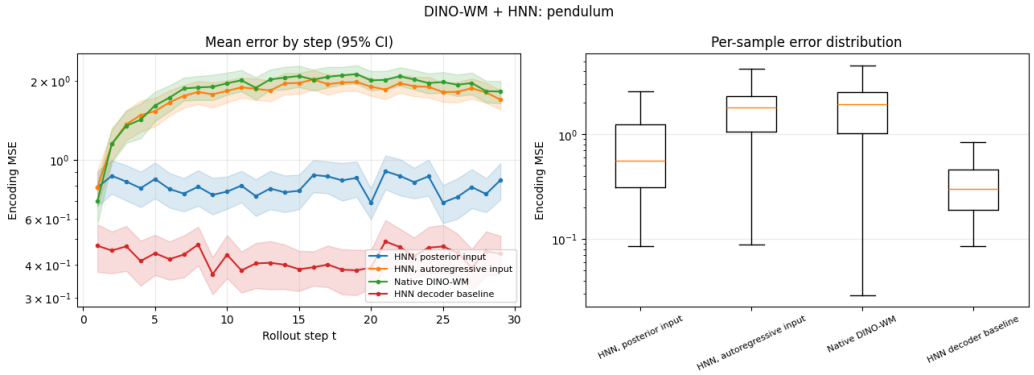
A.3.1 Pendulum



(a) Dreamer



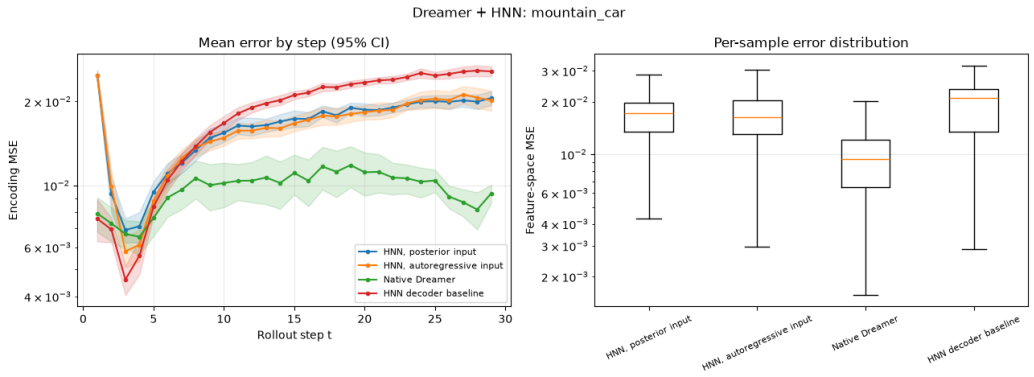
(b) LeWM



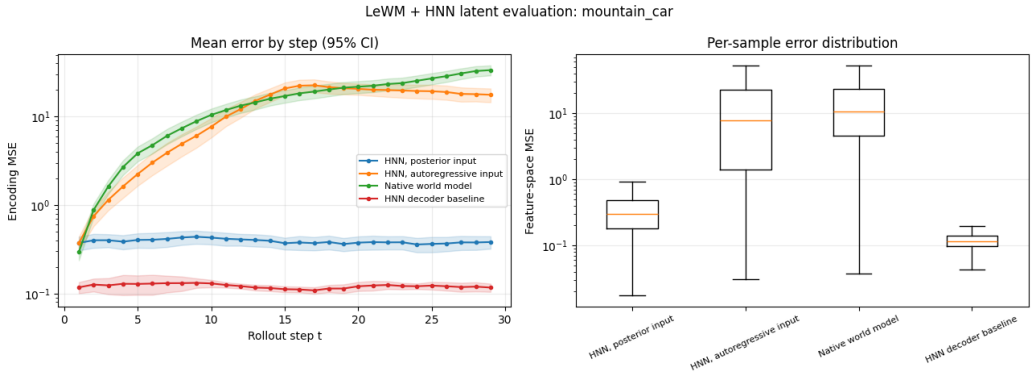
(c) DINO-WM

Figure 7: Rollout encoding errors for Pendulum across the three world-model architectures.

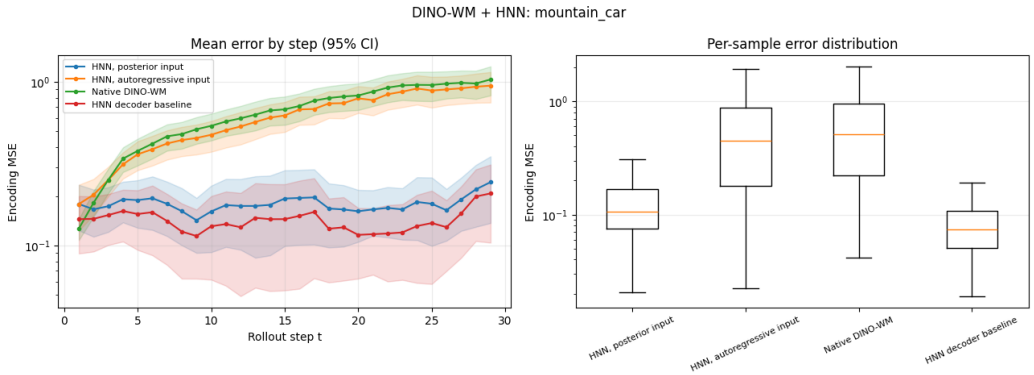
A.3.2 MountainCar



(a) Dreamer



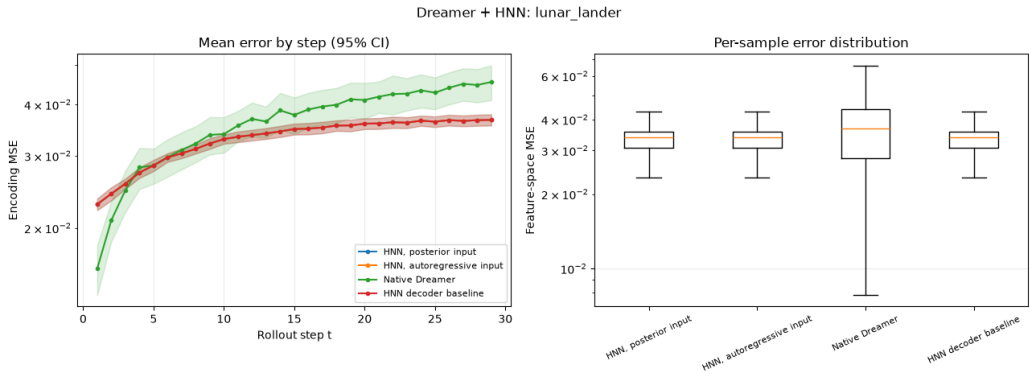
(b) LeWM



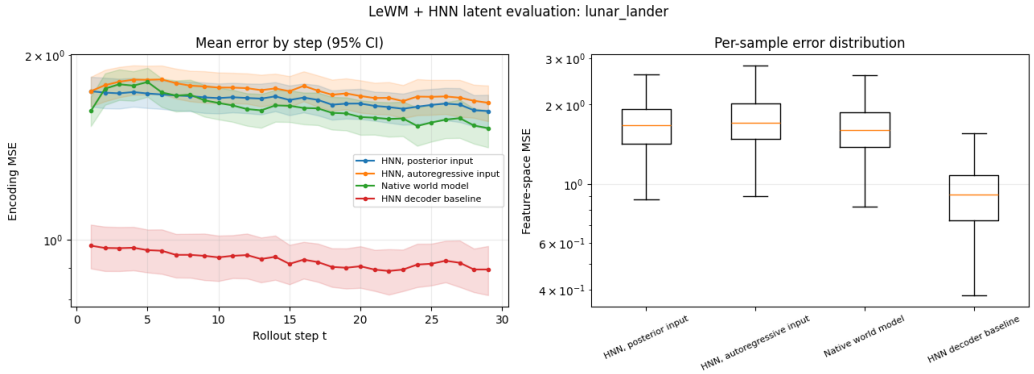
(c) DINO-WM

Figure 8: Rollout encoding errors for MountainCar across the three world-model architectures.

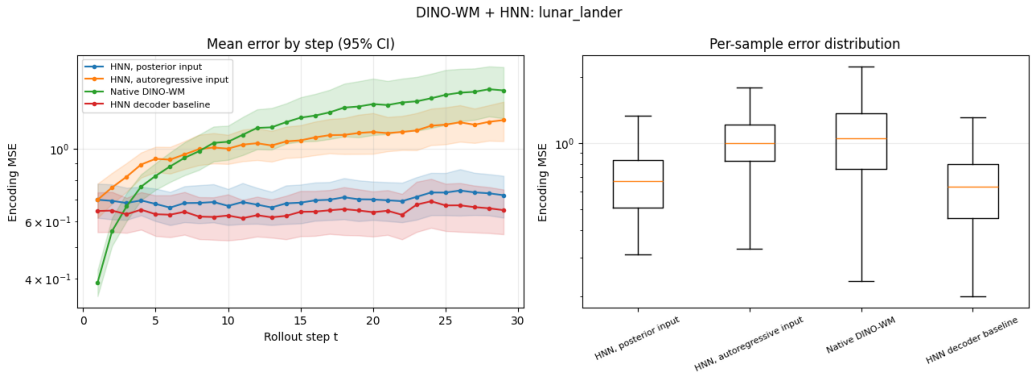
A.3.3 LUNARLANDER



(a) Dreamer



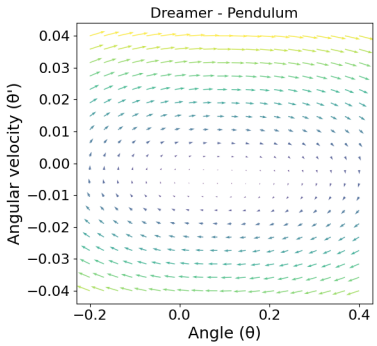
(b) LeWM



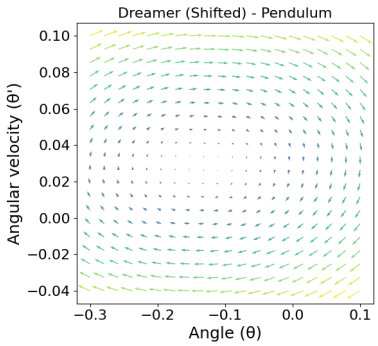
(c) DINO-WM

Figure 9: Rollout encoding errors for LunarLander across the three world-model architectures.

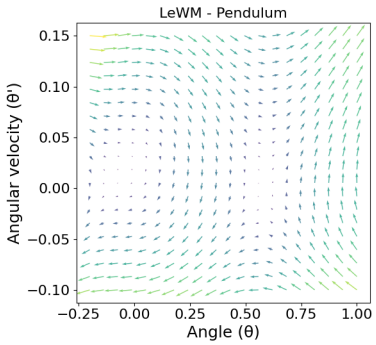
A.4 All Obtained Phase-Spaces



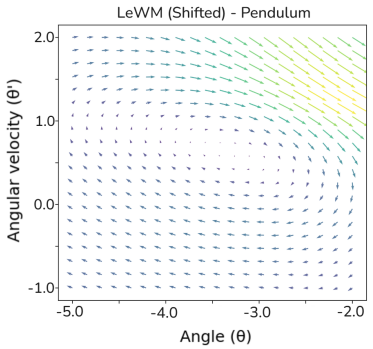
(a) Dreamer – ID



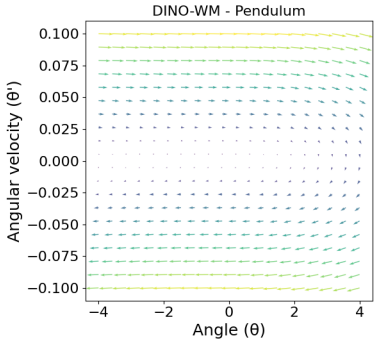
(b) Dreamer – Shifted



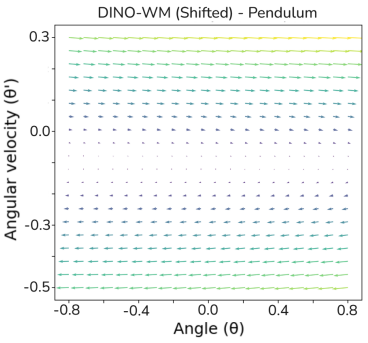
(c) LeWM – ID



(d) LeWM – Shifted

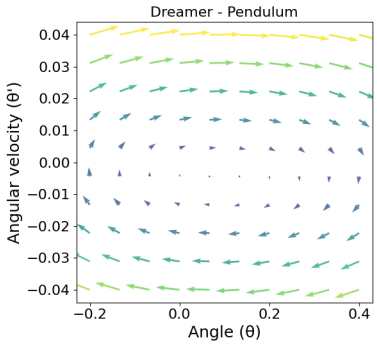


(e) DINO-WM – ID

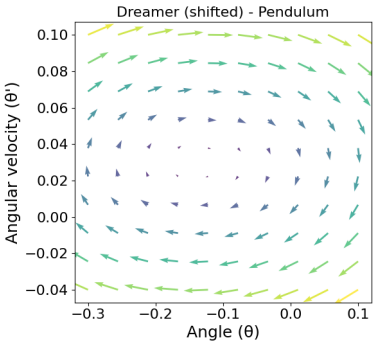


(f) DINO-WM – Shifted

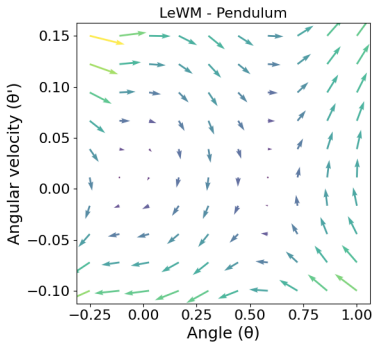
Figure 10: Recovered Pendulum phase-space vector fields.



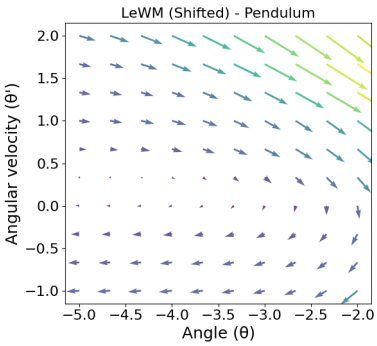
(a) Dreamer – ID



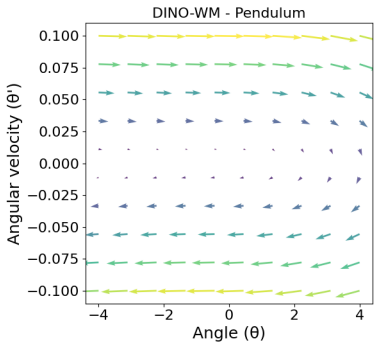
(b) Dreamer – Shifted



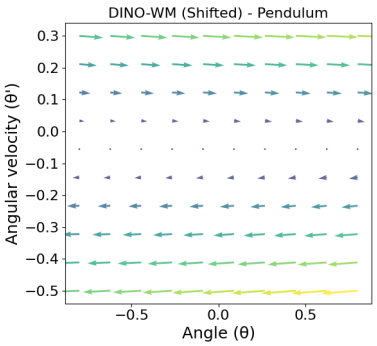
(c) LeWM – ID



(d) LeWM – Shifted

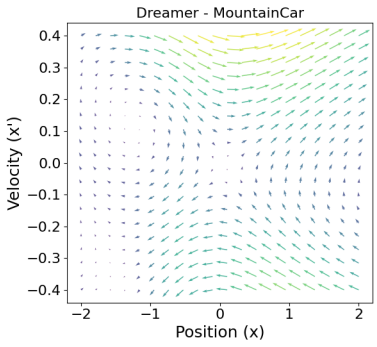


(e) DINO-WM – ID

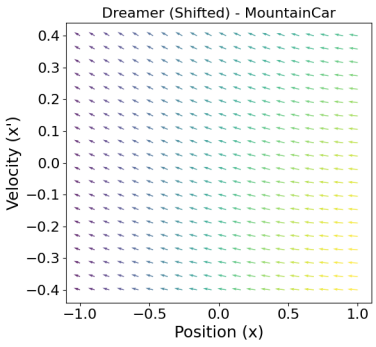


(f) DINO-WM – Shifted

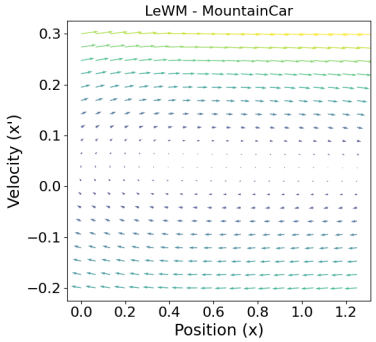
Figure 11: Recovered Pendulum phase-space vector fields. Lower resolution vector fields for more visual clarity.



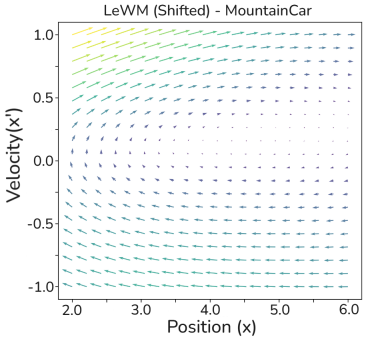
(a) Dreamer – ID



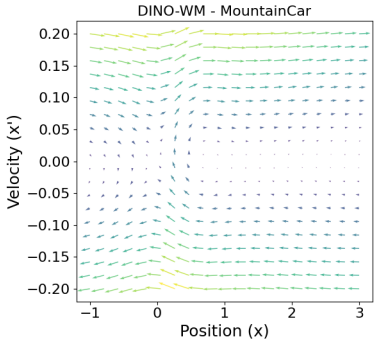
(b) Dreamer – Shifted



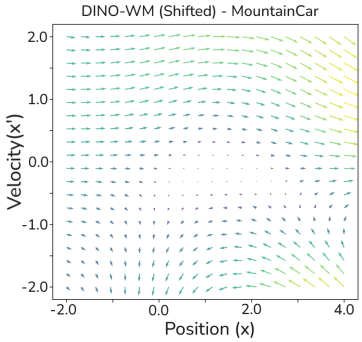
(c) LeWM – ID



(d) LeWM – Shifted

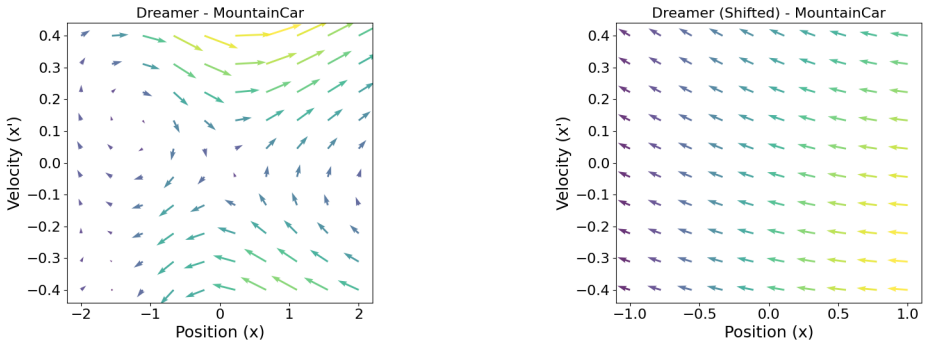


(e) DINO-WM – ID



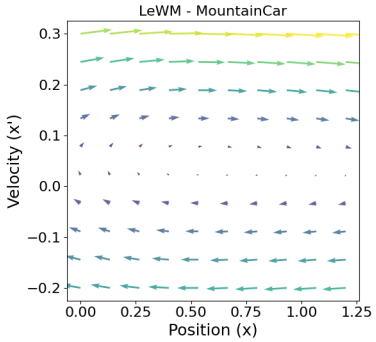
(f) DINO-WM – Shifted

Figure 12: Recovered MountainCar phase-space vector fields (Dreamer-shifted doesn't show the expected vector field. This is due to the probing error being unusually high for Dreamer-shifted model)

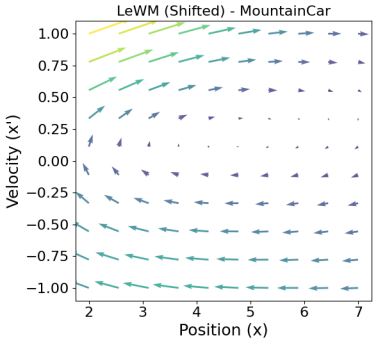


(a) Dreamer – ID

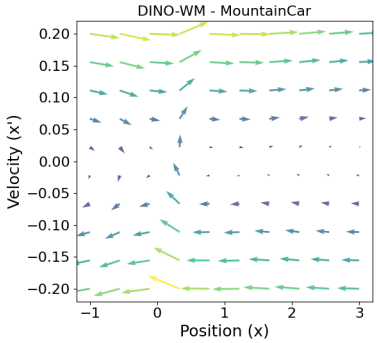
(b) Dreamer – Shifted



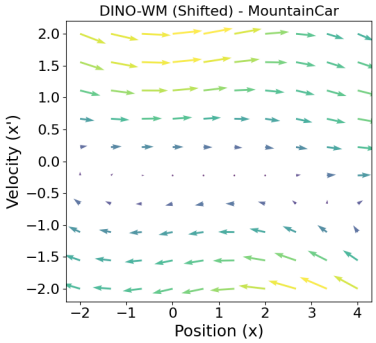
(c) LeWM – ID



(d) LeWM – Shifted

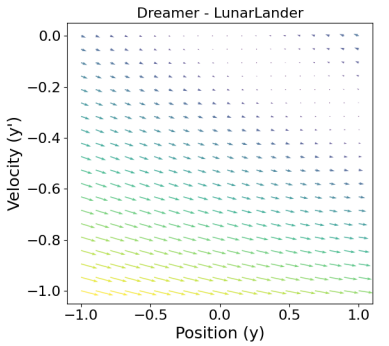


(e) DINO-WM – ID

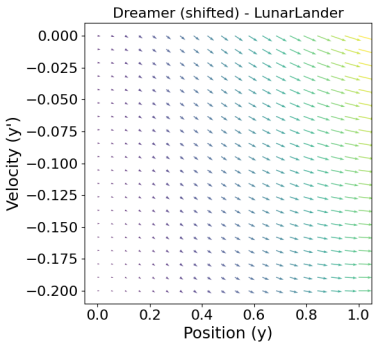


(f) DINO-WM – Shifted

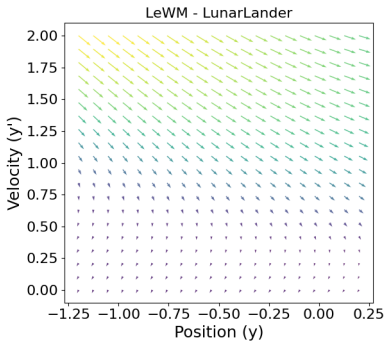
Figure 13: Recovered MountainCar phase-space vector fields (Dreamer-shifted doesn't show the expected vector field. This is due to the probing error being unusually high for Dreamer-shifted model). Low resolution vector field for visual clarity.



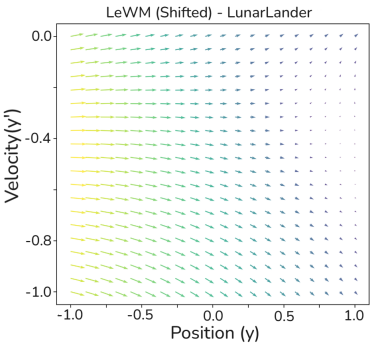
(a) Dreamer – ID



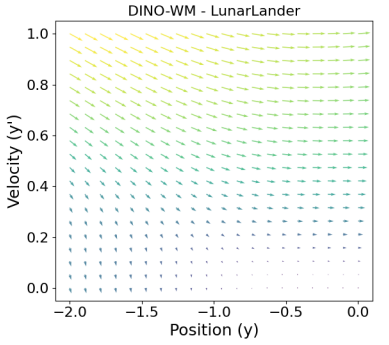
(b) Dreamer – Shifted



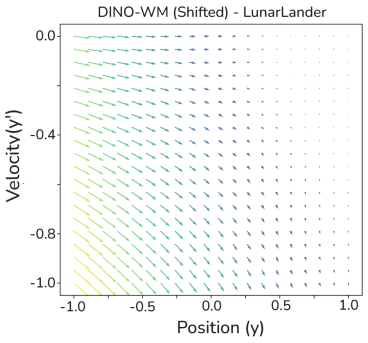
(c) LeWM – ID



(d) LeWM – Shifted

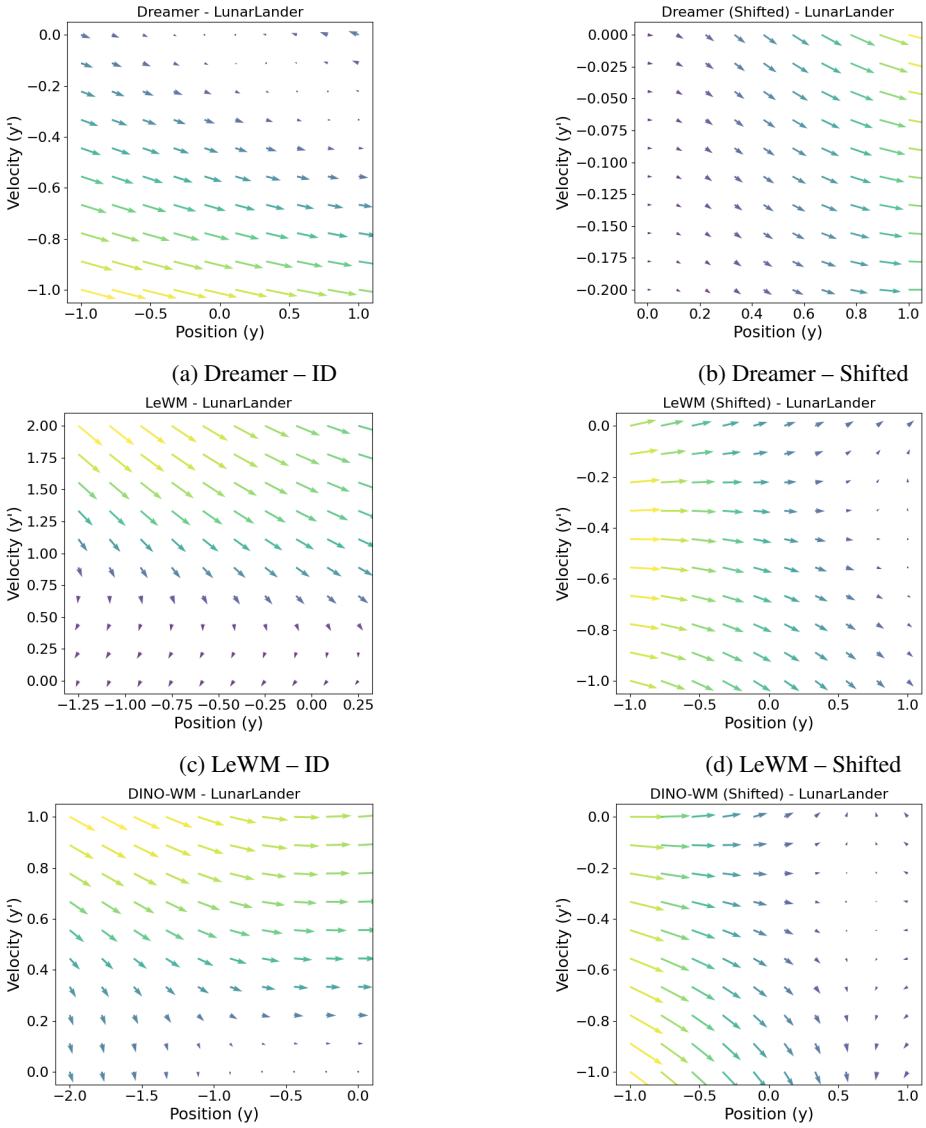


(e) DINO-WM – ID



(f) DINO-WM – Shifted

Figure 14: Recovered LunarLander phase-space vector fields.



(a) Dreamer – ID

(b) Dreamer – Shifted

(c) LeWM – ID

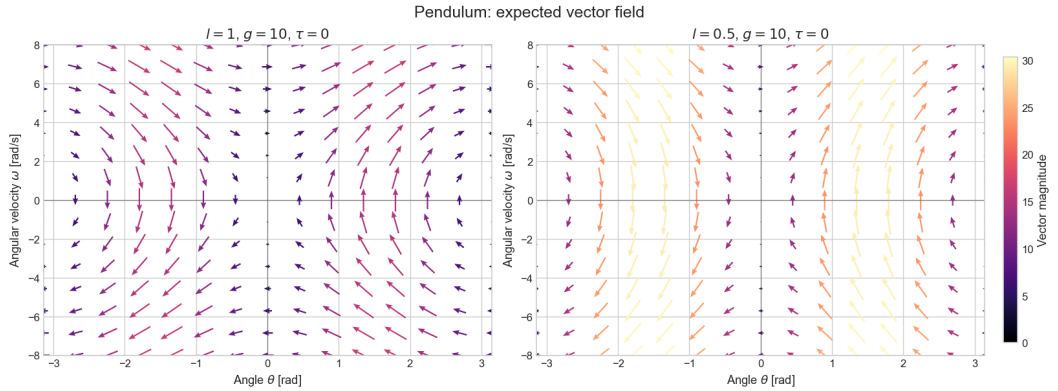
(d) LeWM – Shifted

(e) DINO-WM – ID

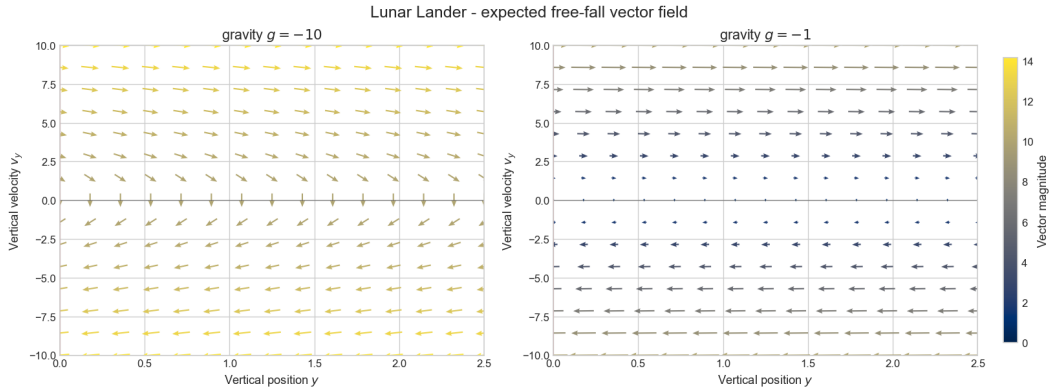
(f) DINO-WM – Shifted

Figure 15: Recovered LunarLander phase-space vector fields. Lower resolution vector fields for visual clarity.

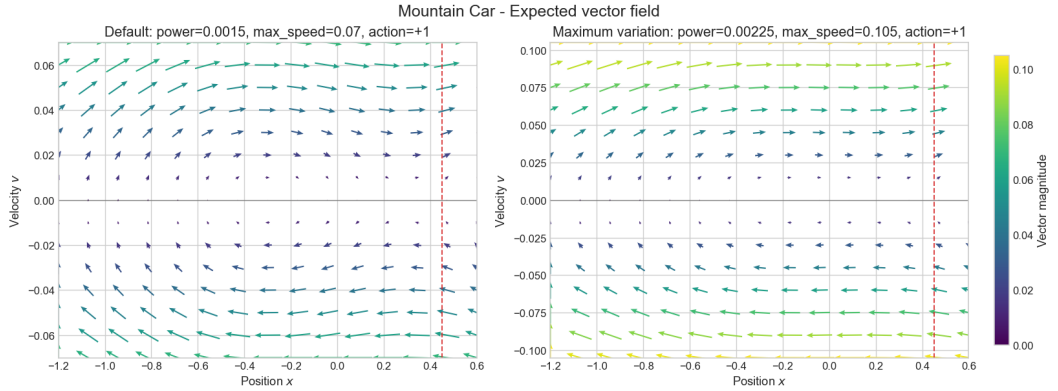
A.5 GROUND-TRUTH (EXPECTED) PHASE-SPACES



(a) Pendulum



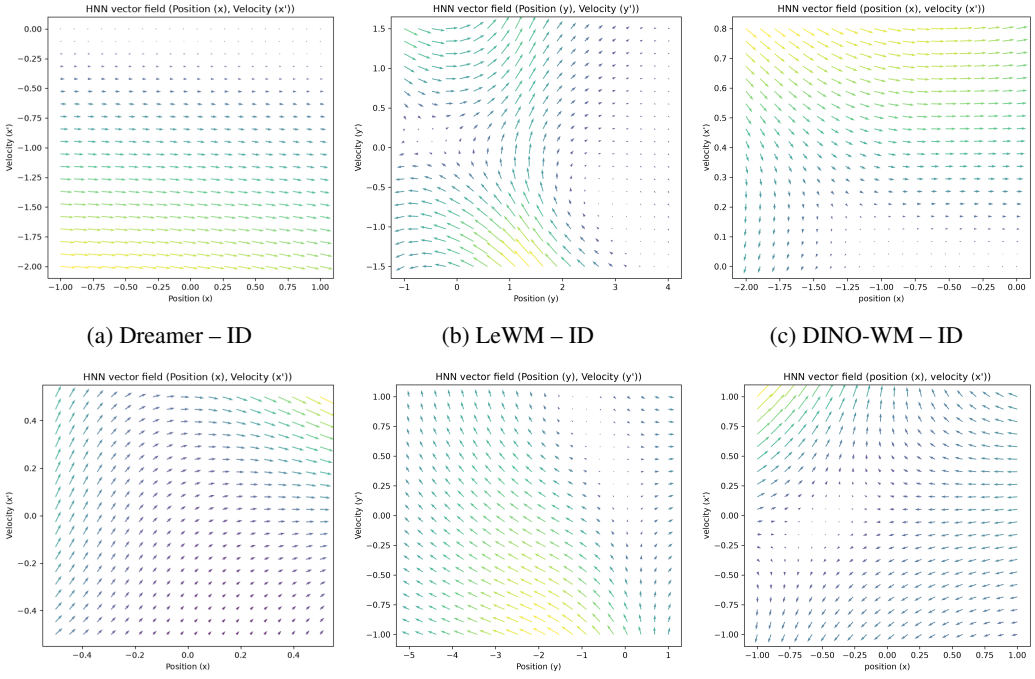
(b) LunarLander free fall



(c) MountainCar

Figure 16: Expected phase-space vector fields for the three control environments under default physics values. The LunarLander field represents the passive free-fall dynamics with no thrusters active. For Lunar Lander, the free-fall vector field is analyzed from 0 and lower values in the y-axis, since the object is falling *downwards*. Full free-fall vector field is given here for completeness.

A.5.1 OBTAINED X-COORDINATE PHASE-SPACE FOR LUNARLANDER

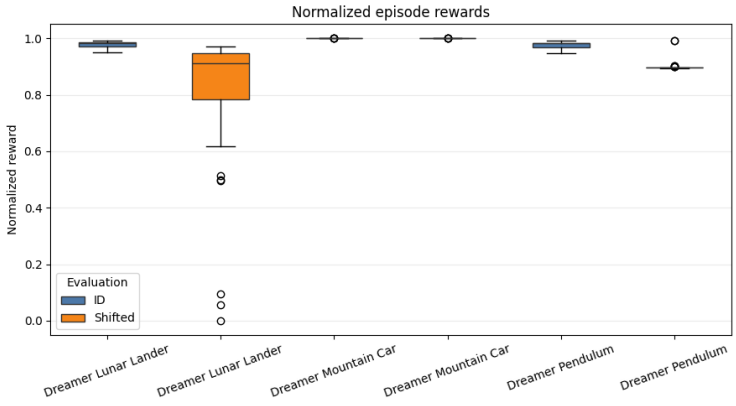


(a) Dreamer – ID (b) LeWM – ID (c) DINO-WM – ID

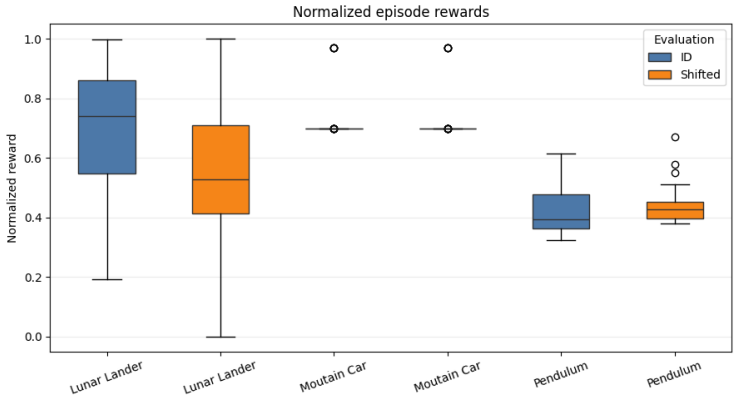
(d) Dreamer – Shifted (e) LeWM – Shifted (f) DINO-WM – Shifted

Figure 17: Recovered LunarLander phase-space vector fields for horizontal position and horizontal velocity, (x, v_x), across all three world models under default (ID) and dynamics-shifted environment parameters. Ideally, the (x, v_x) field remains constant under free-fall. Dreamer-ID matches the ideal vector field for x-axis. However, since it is a constant, the world models may ignore that structure. (Typo in LeWM-ID and shifted title. Should be read as position (x) and velocity $(\dot{x})$).

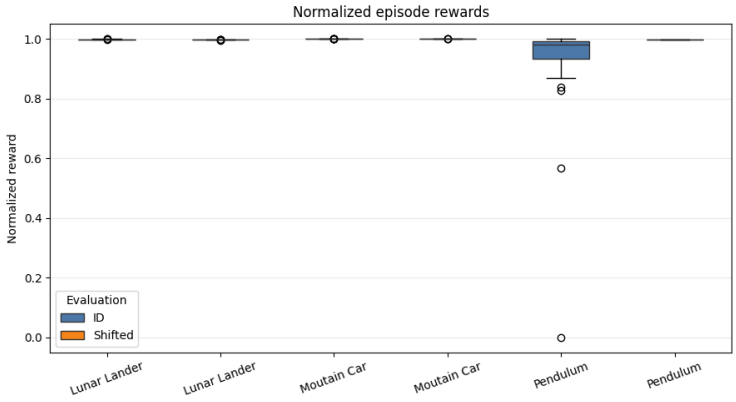
A.6 Mean Rewards



(a) Dreamer



(b) LeWM



(c) DINO-WM

Figure 18: Normalized episode-return distributions for Dreamer, LeWM, and DINO-WM under in-distribution and dynamics-shifted evaluation. Higher is better.

A.7 HYPER-PARAMETERS

$$\mathcal{L}_{\text{LeWM}} = \mathcal{L}_{\text{pred}} + 0.09\,\mathcal{L}_{\text{SIGReg}} + \mathcal{L}_{\text{reward}}.$$

$$\mathcal{L}_{\text{DINO-WM}} = \text{MSE}\left(\hat{\mathbf{z}}_{t+1}^{\setminus a}, \text{sg}(\mathbf{z}_{t+1}^{\setminus a})\right) + \mathcal{L}_{\text{reward}}.$$

Table 4: LeWorldModel training hyperparameters. P, MC, and LL denote Pendulum, Mountain Car, and Lunar Lander, respectively.

Hyperparameter	P	MC	LL
Training sequences	`pendulum_train`	`mountaincar_train`	`ll_train`
Sequence length	100	100	100
History length	99	99	99
Prediction horizon	1	1	1
Batch size	32	32	64
Epochs	10	10	10
Train/validation split	0.9/0.1	0.9/0.1	0.9/0.1
Optimizer	Adam	Adam	Adam
Learning rate	10^{-4}	10^{-4}	10^{-4}
Weight decay	10^{-5}	10^{-5}	10^{-5}
Gradient clipping	None	None	None
Learning-rate scheduler	None	None	None
Random seed	0	0	0
Data-loader workers	0	0	0
SIGReg weight	0.09	0.09	0.09
Reward-loss weight	1.0	1.0	1.0
Checkpoint interval (epochs)	1	1	1
Device	CUDA	CUDA	CUDA

$\mathcal{L}_{\text{reward}}$ is the MSE of an auxiliary reward predictor MLP head that takes in the predicted latent state and predicts the reward. This is used to evaluate planning.

HNN Losses:

$$\mathcal{L}_{\text{AE}} = \|\mathbf{x}_t - D(E(\mathbf{x}_t))\|_2^2, \tag{15}$$

$$\mathcal{L}_{\text{CC}} = \left\| \mathbf{p}_t - \frac{\mathbf{q}_{t+1} - \mathbf{q}_t}{\Delta t} \right\|_2^2, \tag{16}$$

$$\mathcal{L}_{\text{HNN}} = \left\| \mathbf{z}_{t+1} - \left(\mathbf{z}_t + \Delta t\, J^{\mathsf{T}} \nabla H(\mathbf{z}_t)\right) \right\|_2^2, \tag{17}$$

$$\mathcal{L} = \mathcal{L}_{\text{AE}} + \mathcal{L}_{\text{CC}} + 0.1\,\mathcal{L}_{\text{HNN}}. \tag{18}$$

Table 5: Dreamer training hyperparameters.

Hyperparameter	Value
Environment steps	100,000
Pretraining steps	2,048
Parallel environments	4
Batch size	32
Sequence length	100
Replay ratio	0.5
Replay-buffer capacity	10^6
Imagination horizon	1
Random seed	42
Evaluation interval	10,000 steps
Checkpoint interval	10,000 steps
Optimizer	Adam
World-model learning rate	10^{-4}
Actor learning rate	8×10^{-5}
Critic learning rate	8×10^{-5}
World/actor/critic Adam ϵ	$10^{-8}/10^{-5}/10^{-5}$
World-model gradient clip	1,000
Actor/critic gradient clip	100/100
Free nats	1.0
Dynamics KL weight	0.5
Representation KL weight	0.1
KL regularization weight	1.0
Continuation-loss weight	1.0
Discount γ	0.997
GAE parameter λ	0.95
Target-update coefficient τ	0.02
Critic target-update interval	1
Image observation	$1 \times 32 \times 32$ grayscale
Action space	P/MC: continuous; LL: discrete, 4 actions

Table 6: DINO-WM training hyperparameters from the training package and pipeline notebook.

Hyperparameter	Value
History length	99
Prediction horizon	1
Sequence length	100
Batch size	128
Epochs	10
Data-loader workers	0
Training loader	Shuffled, drop last
Validation loader	Not shuffled, retain last batch
Optimizer	AdamW
Learning rate	5×10^{-4}
Weight decay	0
Gradient clipping norm	1.0
Learning-rate scheduler	None
Random seed	42
Frozen visual backbone	Yes
Reward-loss weight	1.0
Image normalization	ImageNet mean and standard deviation
Periodic checkpoint interval	Disabled by default
Action dimensions	P/MC: 1; LL: 2
Action-embedding dimension	10

Table 7: Architecture hyperparameters for the three world models.

Model	Component	Architecture
LeWorldModel	Visual encoder	ViT-HF "tiny", trained from scratch; 32×32 input; configured patch size 8; no mask token. Embedding dimensions are P/MC/LL = 1024/1024/128.
	Action encoder	Linear embedder: action dimension $\rightarrow D$; action dimensions P/MC/LL = 1/1/2.
	Predictor	Causal transformer with 6 blocks, 16 attention heads, head dimension 64, feed-forward dimension 1024, dropout 0.1, embedding dropout 0.
	Projection heads	Projector and prediction projector: $D \rightarrow 1024 \rightarrow D$, with BatchNorm.
	Reward head	MLP $D \rightarrow D \rightarrow 1$, with BatchNorm.
Dreamer	Image encoder	Four convolution blocks with channels $2, 4, 8, 16$; 4×4 kernels, stride 2, padding 1, ChannelNorm ($\epsilon = 10^{-3}$), and SiLU. For a 32×32 grayscale image, the encoder output dimension is 64.
	Image decoder	Linear projection from the 1280-dimensional RSSM feature, followed by four transposed-convolution blocks with channels $16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$.
	RSSM state	32 categorical stochastic variables with 32 categories each (1024 flattened dimensions), plus a 256-dimensional deterministic state; combined Dreamer feature dimension = 1280.
	RSSM transition	Dense width 256, 256-dimensional deterministic state, and 8 parallel recurrent blocks.
	Posterior/prior networks	Posterior: $(64 + 256) \rightarrow 256 \rightarrow 1024$; prior: $256 \rightarrow 256 \rightarrow 1024$.
	Reward/continue heads	Reward: $1280 \rightarrow 256 \rightarrow 255$ bins; continue: $1280 \rightarrow 256 \rightarrow 1$.
	Actor/critic	Actor: three 256-unit hidden layers followed by environment-specific action heads. Critic: $1280 \rightarrow 256 \rightarrow 256 \rightarrow 256 \rightarrow 255$.
DINO-WM	Visual encoder	Frozen pretrained `facebook/dinov2-small`; hidden dimension 384; normalized 32×32 RGB input.
	Extra/action encoder	Temporal Conv1d with kernel and stride 1: action dimension $\rightarrow 10$. The per-token joint embedding dimension is $384 + 10 = 394$.
	Predictor	Causal transformer with 6 blocks, 16 heads, head dimension 64, attention inner dimension 1024, feed-forward dimension 1024, dropout 0.1, embedding dropout 0, and final LayerNorm.
	Positional embedding	Learned positional embeddings allocated using 50×16 nominal frame–patch positions.
	Reward head	Token-mean pooling followed by $394 \rightarrow 394 \rightarrow 1$, with BatchNorm and ReLU.

Table 8: Environment-specific HNN configurations. F is the input world-model feature dimension, d is the canonical latent dimension, H_{AE} is the autoencoder width parameter, and H_{HNN} is the Hamiltonian MLP width.

Upstream model	Environment	F	d	H_{AE}	H_{HNN}	Rollouts $\times$ steps	Batch
LeWM	Pendulum	1024	2	512	128	5000×5	256
LeWM	Lunar Lander	128	4	1024	256	5000×5	256
LeWM	Mountain Car	256	2	512	512	5000×5	256
Dreamer	Pendulum	1280	2	1024	256	5000×5	256
Dreamer	Lunar Lander	1280	4	1024	256	5000×5	256
Dreamer	Mountain Car	1280	2	1024	128	5000×5	128
DINO-WM	Pendulum	1576	2	512	128	5000×5	256
DINO-WM	Lunar Lander	1576	4	1024	512	5000×5	128
DINO-WM	Mountain Car	1576	2	1024	256	5000×5	256

Table 9: Training hyperparameters shared by the HNN experiments.

Hyperparameter	Value
Optimizer	Adam
Optimization steps	5,000
Learning rate	10^{-4}
Weight decay	10^{-5}
Train/validation/test split	0.8/0.1/0.1
Training-pair mode	Single successive latents (x_t, x_{t+1})
Latent integration step Δt	1.0
Input-noise standard deviation	0
Gradient clipping norm	10
Validation/logging interval	250 steps
Autoencoder-loss weight	1.0
Canonical-coordinate-loss weight	1.0
Hamiltonian-dynamics-loss weight	0.1
Autoencoder activation	$\tanh$
Hamiltonian MLP activation	$\tanh$
Hamiltonian field type	Solenoidal
Numerical update	Forward Euler
Random seed	42
Data-loader workers	0

Table 10: Shared HNN architecture. F, d, H_{AE}, and H_{HNN} take the values in Table 8.

Component	Architecture
Autoencoder encoder	$F \to H_{\mathrm{AE}}/2 \to H_{\mathrm{AE}}/4 \to H_{\mathrm{AE}}/8 \to H_{\mathrm{AE}}/16 \to H_{\mathrm{AE}}/32 \to d$, with $\tanh$ after every hidden layer.
Autoencoder decoder	$d \to H_{\mathrm{AE}}/32 \to H_{\mathrm{AE}}/16 \to H_{\mathrm{AE}}/8 \to H_{\mathrm{AE}}/4 \to F$, with $\tanh$ hidden activations and a linear output.
Hamiltonian network	$d \to H_{\mathrm{HNN}} \to H_{\mathrm{HNN}} \to 1$; two $\tanh$ hidden layers and a bias-free output layer.
Vector field	Canonical solenoidal field $\dot{\mathbf{z}} = J^{\top}\nabla_{\mathbf{z}} H(\mathbf{z})$.
State update	$\widehat{\mathbf{z}}_{t+1} = \mathbf{z}_t + \Delta t\, \dot{\mathbf{z}}_t$.

Table 11: Training hyperparameters for the state probes.

Hyperparameter	Value
World-model parameters	Frozen
World-model mode	Evaluation
Probe objective	State regression
Target key	Environment observation
Train/validation split	0.8/0.2
Split granularity	Individual timestep samples
Split seed	0
Input standardization	Per feature, training statistics
Target standardization	Per state dimension, training statistics
Standard-deviation floor	10^{-6}
Optimizer	AdamW
Learning rate	10^{-3}
Weight decay	10^{-4}
Batch size	256
Linear-probe epochs	100
Nonlinear-probe epochs	150
Loss	Mean-squared error
Learning-rate scheduler	None
Gradient clipping	None
Early stopping	None