Denoising Hamiltonian Network for Physical Reasoning

Anonymous authors
Paper under double-blind review

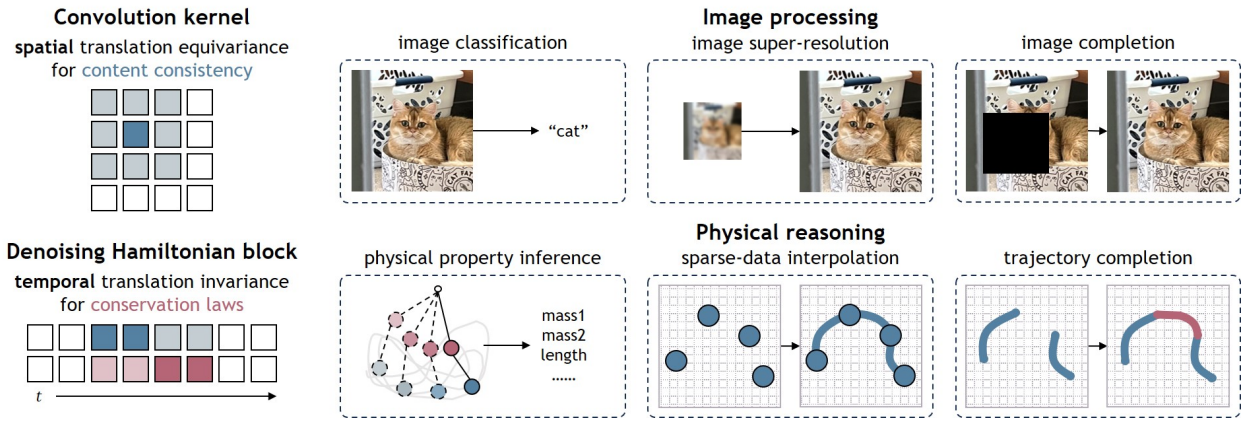

Figure 1: **Denoising Hamiltonian Network (DHN)** generalizes Hamiltonian mechanics into neural operators. It enforces physical constraints while leveraging the flexibility of neural networks, opening pathways for broader applications in physical reasoning.

Abstract

Machine learning frameworks for physical problems must capture and enforce physical constraints that preserve the structure of dynamical systems. Many existing approaches achieve this by integrating physical operators into neural networks. While these methods offer theoretical guarantees, they face two key limitations: (i) they primarily model local relations between adjacent time steps, overlooking longer-range or higher-level physical interactions, and (ii) they focus on forward simulation while neglecting broader physical reasoning tasks. We propose the Denoising Hamiltonian Network (DHN), a novel framework that generalizes Hamiltonian mechanics operators into more flexible neural operators. DHN captures non-local temporal relationships and mitigates numerical integration errors through a denoising mechanism. DHN also supports multi-system modeling with a global conditioning mechanism. We demonstrate its effectiveness and flexibility across three diverse physical reasoning tasks with distinct inputs and outputs.

1 Introduction

Physical reasoning – the ability to infer, predict, and interpret the behavior of dynamic systems – is fundamental to scientific inquiry. Machine learning frameworks designed to address such challenges are often expected to go beyond merely memorizing data distributions, aiming to uphold the laws of physics, account for energy and force relationships, and incorporate structured inductive biases that surpass those of purely data-driven models. Scientific machine learning addresses this challenge by embedding physical constraints directly into neural network architectures, often through explicitly constructed physical operators.

However, these methods face two key limitations. (i) These methods primarily learn local temporal updates, predicting state transitions from one time step to the next, without capturing long-range dependencies or abstract system-level interactions. (ii) They focus predominantly on forward simulation, forecasting a system's

evolution from initial conditions, while largely overlooking complementary tasks such as super-resolution, trajectory inpainting, or parameter estimation from sparse observations. To address these limitations, we aim to design more general neural operators that both follow physical constraints, and unleash the expressivity of neural networks to learn high-level information from data.

To motivate our approach, we draw inspiration from the success of CNNs. Classical image-processing systems utilized handcrafted convolution filters for low-level tasks such as edge detection. The breakthrough came with deep convolutional neural networks (CNNs), which generalized these filters by making them learnable, stacking them with nonlinearities, and expanding them into high-dimensional feature channels. This transition enabled progress from local processing to high-level tasks like object recognition and image generation.

Building on this inspiration, we introduce the **Denoising Hamiltonian Network (DHN)**, a framework that generalizes Hamiltonian mechanics into more flexible neural operators (Fig. 1). Similar to CNNs absorbing parameters from classical image processing (*e.g.*, filter size, stride) into learnable and tunable components, DHN reformulates parameters of numerical integration as architectural hyperparameters, enabling more adaptive and expressive physical modeling. For example, state discretization becomes a learnable block size, and small, fixed time steps are replaced by variable strides, analogous to how CNNs adapt their receptive fields.

More concretely, DHN enforces physical constraints while leveraging the flexibility of neural networks with three key innovations: (i) First, DHN extends Hamiltonian neural operators to capture non-local temporal relationships by treating groups of system states as tokens, allowing it to reason holistically about system dynamics rather than in isolated steps. (ii) Second, DHN introduces a denoising objective analogous to variational integration but learned through data. It mitigates integration errors without relying on additional off-trajectory data for state optimization. Additionally, with different noise patterns, DHN supports flexible training and inference across various task contexts. (iii) Third, we introduce global conditioning to facilitate multi-system modeling, enabling DHN to model heterogeneous physical systems under a unified framework.

To evaluate DHN's versatility, we test it across three distinct reasoning tasks: (i) trajectory prediction and completion, (ii) inferring physical parameters from state observations, and (iii) interpolating sparse trajectories via progressive super-resolution.

In summary, this work moves toward more general network architectures that embed physical constraints beyond local temporal relationships, opening pathways for broader applications in physical reasoning beyond conventional forward simulation and next-state prediction.

2 Related Work

Machine learning approaches for physical modeling span from fundamental equations of motion to high-dimensional operator learning. Our work extends Hamiltonian neural networks (HNNs) into a flexible, sequence-based paradigm that enables multi-task inference and generative conditioning.

Hamiltonian Neural Networks (HNNs) Scientific machine learning aims to embed physical laws into neural architectures. Hamiltonian Neural Networks (HNNs) (Greydanus et al., 2019) enforce symplectic structure and energy conservation in learned dynamics, inspiring various extensions: Lagrangian Neural Networks (LNNs) (Cranmer et al., 2020), Symplectic ODE-Nets (Zhong et al., 2019), and Dissipative SymODEN (Zhong et al., 2020), which introduce damping terms. Constraints have also been incorporated into HNNs (Finzi et al., 2020), and some models infer Hamiltonian dynamics directly from image sequences (Toth et al., 2019). Despite their strengths in forward simulation, standard HNNs typically model one system at a time and rely on uniform-step integration, limiting their use in trajectory completion, sparse-data interpolation, or super-resolution.

Physics-informed and operator-based methods Another approach embeds partial differential equation (PDE) constraints directly into neural models. Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) enforce PDE-based losses for solving forward and inverse problems, while Fourier Neural Operators (FNOs) (Li et al., 2020) learn mappings between function spaces using global Fourier transforms. Neural

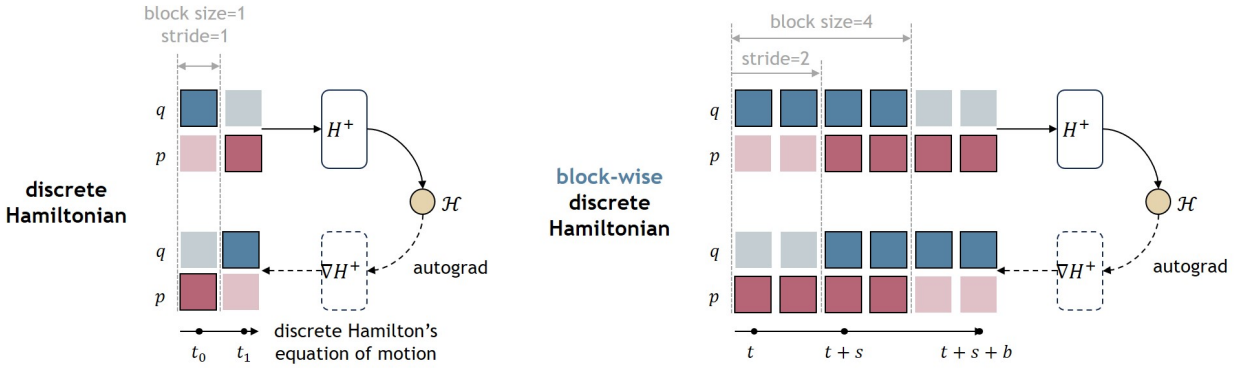

Figure 2: **Block-wise Hamiltonian.** Left: Classical HNN with the discrete Hamiltonian. It can be viewed as a special case with block size $b = 1$ and stride $s = 1$. Right: A discrete (right) Hamiltonian block with $b = 4, s = 2$. Dark blue and dark red indicate network inputs and outputs. Light colors are for placeholder states.

ODEs (Chen et al., 2018; Dupont et al., 2019) parameterize continuous-time dynamics with learnable differential equations. While these methods effectively model spatiotemporal PDEs, they are less suitable for discrete Hamiltonian dynamics with irregular sampling. In contrast, our method directly operates on discrete Hamiltonian structures using block-wise transformations, enhancing flexibility while preserving interpretability and stability.

System identification and multi-system modeling Learning from heterogeneous physical systems requires system identification, traditionally performed via parametric models (Ljung, 1999) or hybrid PDE-constrained approaches (Raissi et al., 2019). While Hamiltonian methods implicitly encode system parameters through energy landscapes, conventional HNNs often require training separate models per system. We introduce a generative conditioning mechanism via a learned latent code, enabling a single model to generalize across multiple systems while preserving inductive biases from Hamiltonian dynamics.

3 Method

We aim to design neural operators that respect physical constraints while leveraging the flexibility and expressivity of neural networks. To this end, we extend Hamiltonian operators to a block-wise form that learns more flexible state relations with generalized conserved quantities (Sec. 3.2). We then introduce a denoising mechanism that replaces the state optimization in variational integration with a learned process; using different masking patterns, it also enables adaptive inference strategies across tasks within a unified framework (Sec. 3.3). Finally, we incorporate global conditioning to extract high-level information from physical trajectories (Sec. 3.4).

3.1 Preliminaries

Learning with Hamiltonian mechanics Let's start with *phase-space coordinates* (q, p), where q is the *generalized coordinates* and p is the *generalized momenta* or *conjugate momenta*. If q represents the particle positions in Euclidean coordinates, then p corresponds to their linear momenta. If q represents angular positions in spherical coordinates, p corresponds to the associated angular momenta. We consider the time-invariant *Hamiltonian*, which is a scalar function $\mathcal{H}(q, p)$ satisfying

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \nabla_p \mathcal{H}, \quad \frac{\mathrm{d}p}{\mathrm{d}t} = -\nabla_q \mathcal{H}. \tag{1}$$

Eq. 1 is known as Hamilton's equations of motion and describes system evolution by defining a trajectory in phase space along the vector field $(\nabla_p \mathcal{H}, -\nabla_q \mathcal{H})$. This field, called the *symplectic gradient*, governs the

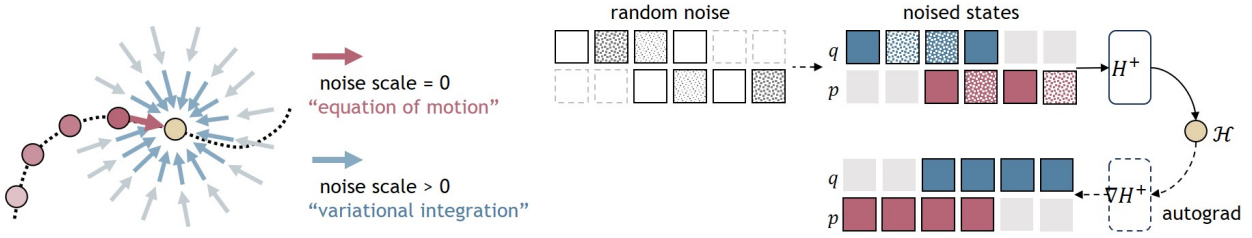

Figure 3: **Denoising Hamiltonian block.** The model integrates the Hamiltonian equation of motion (pink arrow) and state optimization (blue arrow) into a unified framework (left) by learning state denoising conditioned on different noise scales (right).

dynamics such that movement along $\mathcal{H}$ induces the most rapid change in the Hamiltonian, whereas motion in the symplectic direction preserves the system's energy structure.

Hamiltonian Neural Networks (HNN) Greydanus et al. (2019) treat the Hamiltonian as a black-box function $\mathcal{H}(q, p; \theta)$ parameterized by a neural network and optimize the network parameters to minimize the loss function

$$\mathcal{L}_{\text{HNN}}(\theta) = \left\| \nabla_p \mathcal{H} - \frac{\mathrm{d}q}{\mathrm{d}t} \right\| + \left\| \nabla_q \mathcal{H} + \frac{\mathrm{d}p}{\mathrm{d}t} \right\|. \tag{2}$$

Starting with an initial state (q_0, p_0), one can compute the trajectory (q_t, p_t) by integrating the symplectic gradient $(\nabla_p \mathcal{H}(q_t, p_t; \theta), -\nabla_q \mathcal{H}(q_t, p_t; \theta))$ over time t.

Discrete Hamiltonian Aside from the continuous Hamiltonian $\mathcal{H}$ and its discretizations, one can also directly define the discrete Hamiltonian with discrete mechanics and duality theory in convex optimization Gonzalez (1996). The discrete "right" Hamiltonian H^+ gives the equation of motion in the form

$$q_{t+1} = \nabla_p H^+(q_t, p_{t+1}), \quad p_t = \nabla_q H^+(q_t, p_{t+1}). \tag{3}$$

The "right" means that q is forward and p is backward in time. This formulation serves as a first-order discrete approximation of the continuous Hamiltonian $\mathcal{H}$ by

$$q_{t+1} = q_t + \Delta t \nabla_p \mathcal{H}(q_t, p_{t+1}), \quad p_t = p_{t+1} + \Delta t \nabla_q \mathcal{H}(q_t, p_{t+1}). \tag{4}$$

Fig. 2 left illustrates a discrete right Hamiltonian network for computing the state relations between time steps t_0 and t_1. In the following sections, we describe our network design primarily using the right Hamiltonian H^+, but similar equations can also define the left Hamiltonian H^-. Additional details can be found in the Appendix.

Our motivations Exemplified by HNN, physical networks generally learn the state relations between adjacent time steps t and $t+1$ modeled by an update rule $(q_{t+1}, p_{t+1}) = \texttt{update\_rule}(q_t, p_t)$. Compared to forward modeling, the discretization in Eq. 3 is more accurate and better preserves the symplectic structure of the system under temporal integrations. However, the implicit nature of these update rules poses inference-time challenges, as determining new system states requires solving an optimization problem, which is difficult only with data points on the simulation trajectory but without additional reference points outside the trajectory. Our solution is to incorporate the optimization process into the network, leading to the *denoising* Hamiltonian network (Sec. 3.3) that unifies the state optimization at each time step and the Hamiltonian-modeled state relations across time steps.

3.2 Block-wise Discrete Hamiltonian

We define state blocks as a stack of (q, p) states concatenated along the time dimension $Q_t^{t+b} = [q_t, \cdots, q_{t+b}], P_t^{t+b} = [p_t, \cdots, p_{t+b}]$, with b being the block size. We also introduce the stride s as a hyperparameter that can be flexibly defined, replacing the fixed time interval Δt in Eq. 4. This approach enables the

network to capture broader temporal correlations while preserving the underlying Hamiltonian structure. We define our block-wise discrete (right) Hamiltonian by relating two overlapping blocks of system states, each of size b, with a shift stride of s

$$Q_{t+s}^{t+s+b} = \nabla_P H^+(Q_t^{t+b}, P_{t+s}^{t+s+b}), \quad P_t^{t+b} = \nabla_Q H^+(Q_t^{t+b}, P_{t+s}^{t+s+b}). \tag{5}$$

Fig. 2 illustrates a block-wise discrete Hamiltonian of a block size $b = 4$ and a stride $s = 2$. Classical HNNs can be viewed as a special case of block size $b = 1$ and stride $s = 1$. Physical interpretations of the block-wise Hamiltonian with $b > 1, s > 1$ and theoretical derivations of its conserved quantities can be found in the Appendix. Similar to the equation-motion loss for HNN (Eq. 5), a block-wise discrete Hamiltonian network H_θ^+ can be trained with the equation-of-motion loss

$$\mathcal{L}_{\text{eom}}(\theta) = \left\| \nabla_P H_\theta^+(Q_t^{t+b}, P_{t+s}^{t+s+b}) - Q_{t+s}^{t+s+b} \right\| + \left\| \nabla_Q H_\theta^+(Q_t^{t+b}, P_{t+s}^{t+s+b}) - P_t^{t+b} \right\|. \tag{6}$$

3.3 Denoising Hamiltonian Network

Masked modeling and denoising Following our motivations introduced in Sec. 3.1, we want the Hamiltonian blocks to not only model the state relations across time steps, but also learn the state optimization per time step for inference. To achieve that, we adopt a masked modeling strategy He et al. (2022) by training the network with a part of the input states masked out (Fig. 3).

Concretely, taking the blocked input state Q_t^{t+b} as an example, we define known and unknown states in this block with a binary mask $M_t^{t+b} = [m_t, \cdots, m_{t+b}]$, with 0 for unknown states and 1 for known states. Rather than simply masking out the unknown states, we perturb them with noise sampled at varying magnitudes and learn to denoising them with iterative refinements (Fig. 3 left). With a sequence of increasing noise levels $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_N = 1$, we randomly sample Gaussian noises $\mathcal{E}_t^{t+b} = [\varepsilon_t, \cdots, \varepsilon_{t+b}]$ and noise scales $A_t^{t+b} = [\alpha_t, \cdots, \alpha_{t+b}]$. Applying the noise to the unknown states, we get the final perturbed state $\widetilde{Q}_t^{t+b}$ with

$$\tilde{q}_s = \begin{cases} q_s & \text{if } m_s = 1 \\ (1-\alpha_s)q_s + \alpha_s \varepsilon_s & \text{if } m_s = 0 \end{cases} \quad (\text{for } s = t, \cdots, t+b). \tag{7}$$

Similarly, we can get the perturbed state $\widetilde{P}_t^{t+b}$, and together with $\widetilde{Q}_t^{t+b}$ define the denoising loss

$$\mathcal{L}_{\text{denoise}}(\theta) = \left\| \nabla_P H_\theta^+(\widetilde{Q}_t^{t+b}, \widetilde{P}_{t+s}^{t+s+b}) - Q_{t+s}^{t+s+b} \right\| + \left\| \nabla_Q H_\theta^+(\widetilde{Q}_t^{t+b}, \widetilde{P}_{t+s}^{t+s+b}) - P_t^{t+b} \right\|. \tag{8}$$

This encourages the network H_θ^+ to learn a conditioned state optimization, with the known states with mask values $m_s = 1$ as conditions and the unknown ones with $m_s = 0$ as the states to denoise (Fig. 3 right). Combining Eq. 6 and Eq. 8, the final loss function for the network is $\mathcal{L}_{\text{eom}}(\theta) + \mathcal{L}_{\text{denoise}}(\theta)$.

To note, unlike the denoising diffusion model that learns multi-modal distributions, this denoising process has a fixed target and thus doesn't require many denoising steps (we set it to be 10 in our experiments). More details for training and inference are in the Appendix.

Different masking patterns By designing distinct masking patterns during training, we enable flexible inference strategies tailored to different tasks. Fig. 4 shows three types of different masking patterns: *autoregression* by masking out the last few states of a block, which resembles physical simulation in terms of next-state prediction with forward modeling; *super-resolution* by masking out the states in the middle of a block, which can be applied to data interpolation; and more generally, *arbitrary-order* masking including random masking, which can adapt to the different tasks.

3.4 Network Architecture

Decoder-only transformer For each Hamiltonian block, the network inputs are a stack of Q_t^{t+b} of different time steps, a stack of $P_{t'}^{t'+b}$, and we also introduce a global latent code z for the entire trajectory as conditioning. We employ a decoder-only transformer Radford et al. (2019); Jin et al. (2024), which resembles

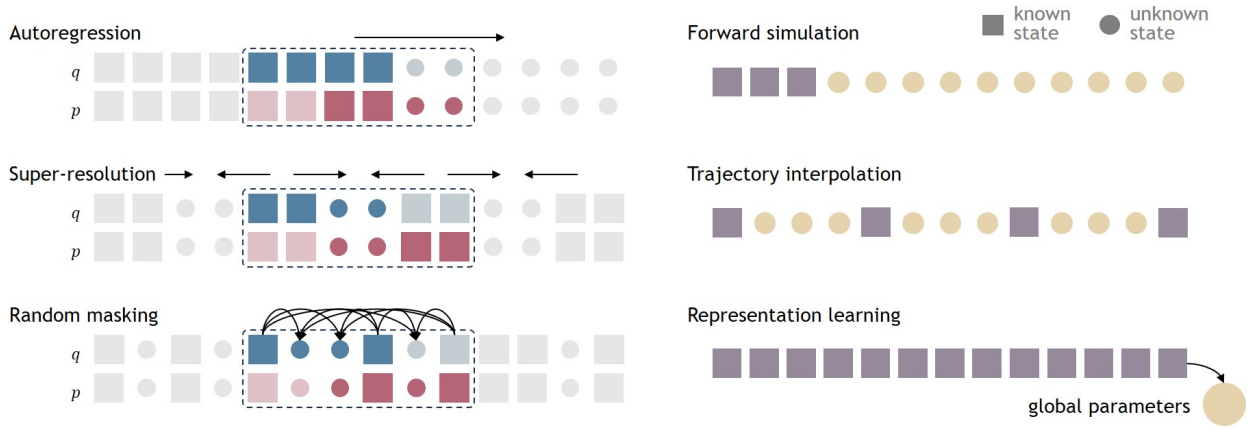

Figure 4: **Different masking patterns (left) adapt to different reasoning tasks (right).** Blocks with pinks and blues surrounded by dotted lines are the denoising Hamiltonian blocks.

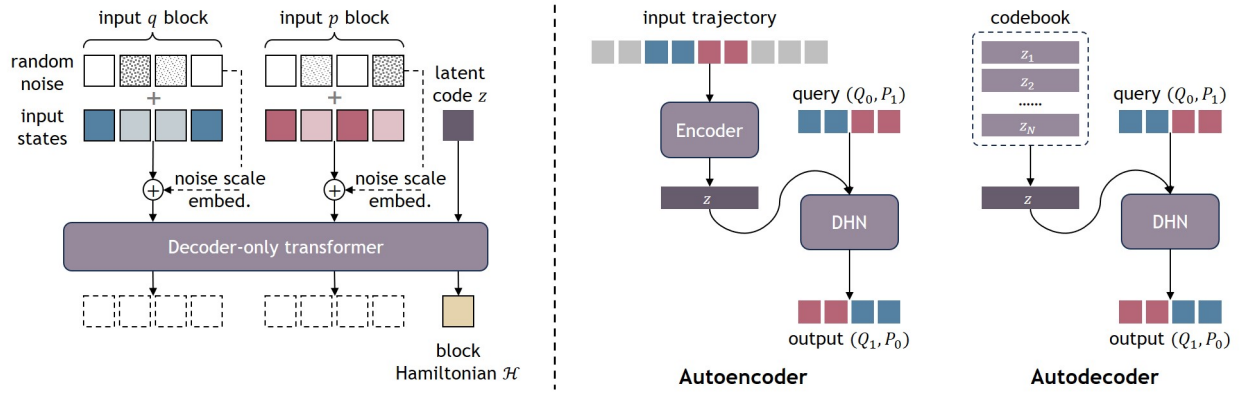

Figure 5: **Network architecture. Left: Decoder-only transformer.** We use a latent code z for each trajectory to serve as the query token for the Hamiltonian value output. Per-state noise scales are encoded and added to the positional embeddings. Dark purples (in all shades) indicate trainable modules or variables. **Right: Autodecoder.** Instead of encoding the input trajectory with an encoder, we maintain a codebook for the entire dataset with a learnable latent code for each trajectory. Dark purples (in all shades) indicate trainable modules or variables.

a GPT-like decoder-only architecture but without a causal attention mask, as shown in Fig. 5 left. We apply self-attention to all input tokens $[Q_t^{t+b}, P_{t'}^{t'+b}, z]$ as a sequence of length $2b+1$. The global latent code z serves as a query token for outputing the Hamiltonian value $\mathcal{H}$. We also encode the per-state noise scales into the network by adding their embeddings to the positional embedding.

Autodecoding Rather than relying on an encoder network to infer the global latent code from the trajectory data, we adopt an autodecoder framework Park et al. (2019), maintaining a learnable latent code z for each trajectory (Fig. 5 right). This approach allows the model to store and refine system-specific embeddings efficiently without requiring a separate encoding process. During training, we jointly optimize the network weights and the codebook. After training, given a novel trajectory, we freeze the network weights and only optimize the latent code for the new trajectory.

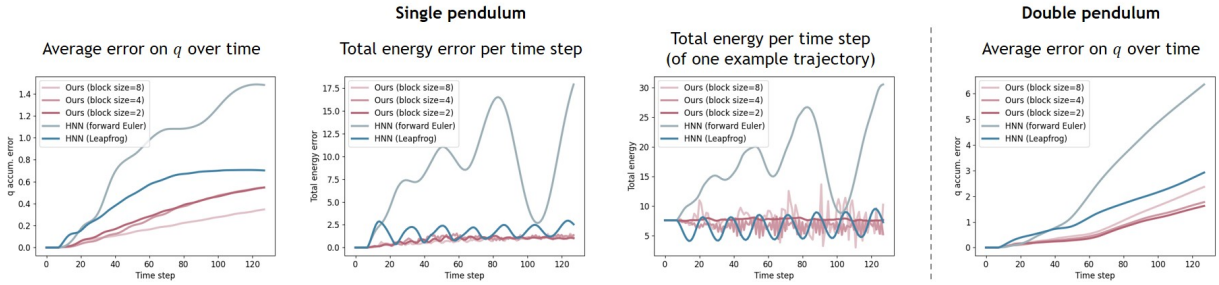

Figure 6: **Forward modeling: fitting known trajectories.** The results of our method are shown in pink, and the results of HNN with different numerical integrators are shown in different shades of blue. The 2nd column shows the error of total energy for the single pendulum system, calculated with state (q_t, p_t) at each time step analytically. The 3rd column shows the total energy predicted by the network over time steps (which wasn't supervised during training) on one example trajectory.

4 Experiments

We evaluate our model with two settings: the *single pendulum* and the *double pendulum*. Both settings comprise a dataset of simulated trajectories. The single pendulum is a periodic system where the total energy at each state can be directly computed from (q, p), and thus we use it to evaluate the models' energy conservation ability. The double pendulum is a chaotic system where small perturbations can lead to diverged future states. More detailed experimental setups are in the Appendix.

Unlike prior works Toth et al. (2019) that generate data using a fixed set of system parameters while varying initial conditions, we introduce variation by altering the string lengths of the pendulums while keeping initial states fixed. This modification evaluates whether models can generalize to a broader class of parameterized dynamical systems rather than fit into a single-instance system. For both settings, we split the dataset into 1000 training trajectories and 200 testing trajectories. Each trajectory is discretized into 128 time steps. More details can be found in the Appendix.

We test our model with three different tasks corresponding to the three different masking patterns, as illustrated in Fig. 4. They are (i) next-state prediction (autoregression) for forward simulation, (ii) representation learning with random masking for physical parameter inference, and (iii) progressive super-resolution for trajectory interpolation. These tasks highlight DHN's adaptability to diverse physical reasoning challenges, testing its ability to generate, infer, and interpolate system dynamics under varying observational constraints.

4.1 Forward Simulation

We start with the forward simulation task, where the model predicts the future states of a physical system step-by-step given the initial conditions. We implement this by applying a masking strategy within each DHN block, where the last few tokens are masked during training, requiring the model to iteratively refine and denoise them (Fig. 4 top). For one DHN block of block size b and stride s, the mask is applied to the last $b - s$ tokens. At inference time, given the known states at time steps $[0, \cdots, t]$, we apply the DHN block to the time steps $[t - b + 1, \cdots, t + s]$, where we use the known states $[t - b + 1, \cdots, t]$ to predict the unknown states $[t + 1, \cdots, t + s]$. We experiment with block sizes $b = 2, 4, 8$ with strides $s = b/2$.

Fitting known trajectories We first evaluate the model's capability to represent known physical trajectories with forward modeling. In this experiment, we train the model to fit 1000 training trajectories, and we test by giving the first 8 time steps of each trajectory and using the model to predict the future 120 steps. As all models are only trained with states of nearby time steps (pairs of adjacent time steps for the baselines, and blocks of $b + s$ states for DHN), small fitting errors can accumulate over time in forward modeling. Beyond accumulated prediction errors inherent to the network, inaccuracies also arise from numerical integration approximations, which can amplify deviations over time.

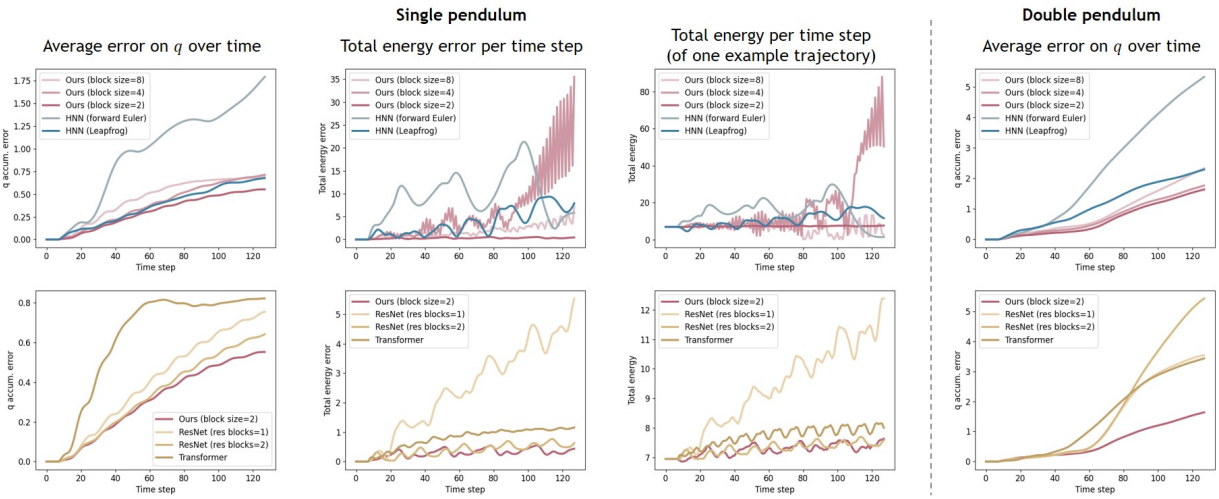

Figure 7: **Forward modeling: completion on novel trajectories.** Top row: Comparison between our method (shown in pink) and HNN with different numerical integrators (shown in blue). Bottom row: Comparison between our method (shown in pink) and vanilla networks with different architectures (shown in yellow). The vanilla networks directly predict the next state (q_{t+1}, p_{t+1}) from the current state (q_t, p_t) with one feedforward step. Note that the y-axis scales between the two rows are different.

Fig. 6 shows the results of our model with different block sizes, compared to HNN Toth et al. (2019) with different numerical integrators. Left and right are the mean squared error (MSE) on the q predictions at each time step for the single and double pendulum systems, respectively. The middle plots show the averaged total energy error and the evolution of total energy on one example trajectory. Although HNN is a symplectic network with guaranteed energy conservation, the numerical integrator can still induce uncontrollable energy drifts. This additional numerical error is inevitable with forward methods. As mentioned in Sec. 3.1, while this can be addressed by variational integration methods, such implicit state optimizations require off-trajectory data, whereas DHN achieves similar effects through a denoising mechanism without this overhead. With block size $b = 2$, our model conserves the total energy stably. Increased block sizes can cause energy fluctuations at long time ranges, but this fluctuation doesn't show an obvious inclination of energy drift.

Completion on novel trajectories We then evaluate our models on novel trajectories with partial observations. Concretely, we give the first 16 time steps in each testing trajectory and use them to optimize for the per-trajectory global latent codes with the network weights frozen, as described in Sec. 3.4. After optimizing these latent codes, we use them to predict the next 112 time steps. This task evaluates DHN's ability to infer system dynamics from sparse initial observations and accurately forecast future states.

Fig. 7 shows our results compared to HNN (top row) and various baseline models without physical constraints (bottom row). Our DHN with small block sizes shows more accurate state prediction with better energy conservation compared to both baselines. Large block sizes can cause error explosion at long time ranges as it is hard for our simple 2-layer network to fit very complex multi-state relations.

4.2 Representation Learning

Next, we test the model's ability to effectively encode and distinguish the parameters of different physical systems. Denoising and random masking are well-established techniques in self-supervised learning, producing state-of-the-art representations in language modeling Devlin (2018) and vision Vincent et al. (2008); He et al. (2022). Here, we apply the random masking pattern (Fig. 4 bottom) and study whether similar paradigms can enhance representation learning in dynamic physical systems.

To quantify the quality of the learned representations, we follow the widely adopted self-supervised representation learning paradigm in computer vision Chen et al. (2020); Oord et al. (2018); He et al. (2020); Kolesnikov

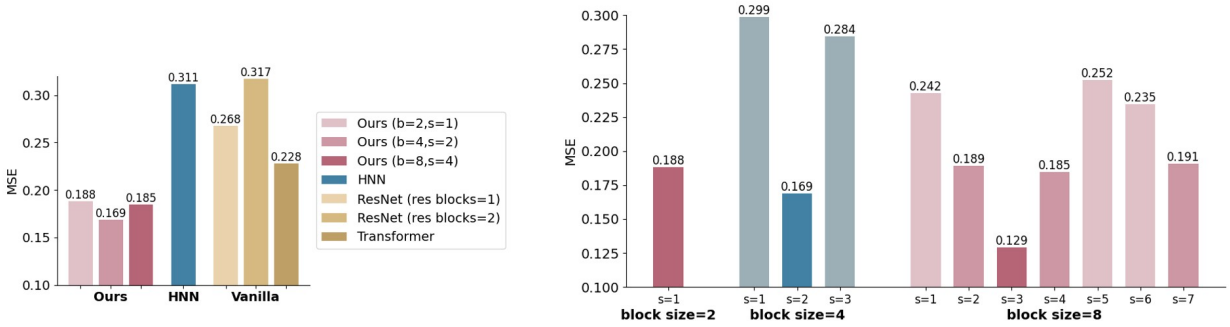

Figure 8: **Linear probing on latent codes (MSE ↓).** We predict l_2/l_1 by applying a linear regression layer to the global latent code. **Left:** Comparison with baselines. **Right:** DHN results with different block sizes and strides. Better results are achieved with block size b and stride s around $s \approx b/2$.

et al. (2019) with feature pre-training and linear probing. Specifically, we pre-train the autodecoder alongside the codebook using the training set, then freeze the learned representations and train a simple linear regression layer on top to predict system parameters. This approach assesses whether DHN's latent codes capture meaningful physical properties. We experiment with the double pendulum and predict the length ratio l_2/l_1, because this physical quantity is dimensionless and therefore invariant under normalizations in data preprocessing.

Fig. 8 left shows the linear probing results of our DHN with different block sizes (with $s = b/2$), compared to the HNN and vanilla networks. Our model achieves a much lower MSE compared to the baseline networks. As illustrated in Fig. 2, HNN can be viewed as a special case of our Hamiltonian block with kernel size and stride being 1, which is of the most locality. The block sizes and strides we introduce allow the model to observe the system at different scales. In this double pendulum system, a block size of 4 is the best temporal scale for inferring its parameters.

Fig. 8 right shows DHN results with varying block sizes and strides. An explanation is in the "overlap" between the input and output states of $b - s$ time steps. The generalized energy conservation in DHN relies on overlapping regions having identical inputs and outputs, enforced during training via the state prediction loss. Larger overlaps strengthen self-coherence but reduce focus on inter-state relations, while smaller overlaps (with larger strides) encourage learning from distant states but weaken self-coherence, affecting stability. At the extreme, full overlap with zero stride reduces training to enforcing self-coherence alone with identical inputs and outputs. HNN is the special case at another extreme with zero overlap. The results suggest that the optimal block sizes and strides are around $s \approx b/2$, leading to moderate overlaps.

4.3 Trajectory Interpolation

To demonstrate the flexibility of the DHN block, we show trajectory interpolation (super-resolution) with the masking pattern in Fig. 4 middle. We conduct $4\times$ super-resolution by repeatedly applying $2\times$ super-resolutions, as shown in Fig. 9 left. Each trajectory is associated with a global latent code, shared across all three stages. During training, both the network weights and these latent codes are optimized jointly with all stages (0, 1, 2). At inference time, given a novel trajectory with known states only at the sparsest level (stage 0), we freeze all network weights in the DHN blocks and optimize for the global latent code with stage 0. After this test-time optimization (autodecoding), we apply the DHN blocks of stages (1, 2) to predict the unknown states in between the known states.

We evaluate the models with two test settings: (i) trajectories with the same initial states as the training ones, and (ii) trajectories of unseen initial states. To set this up, we crop all training trajectories to time steps $[0, \cdots, 64]$. For each trajectory in the test set, we divide it into two segments: time steps $[0, \cdots, 64]$ and $[65, \cdots, 128]$, the former having the same initial state as the training set and the latter having different initial states.

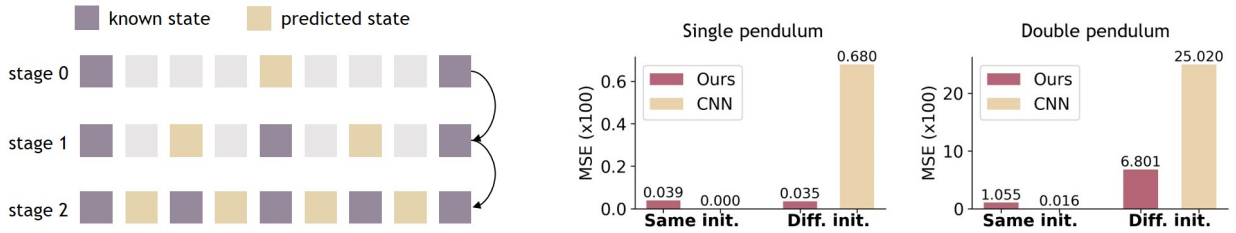

Figure 9: **Interpolation via progressive super-resolution (MSE↓).** Left: The three stages for 2× super-resolution repeated twice. Right: Result comparison between DHN (*Ours*) and a CNN-based super-resolution network (*CNN*). Results are MSE multiplied by 100.

We compare our model to a Convolutional Neural Network (CNN) for super-resolution. Fig. 9 shows our results. For the trajectories with the same initial state as the training data, both models show good interpolation results with lower MSEs. The baseline CNN shows slightly better results, as it has no regularization in itself and can easily overfit the training trajectories. For testing trajectories with unseen initial states, the CNN struggles to generalize, as its interpolations rely heavily on the training distribution. In contrast, DHN demonstrates strong generalization, as its physically constrained representations enable it to infer plausible intermediate states even under distribution shifts.

5 Discussions and Conclusions

Balancing flexibility with physical constraints is key to advancing physics-based learning. Inspired by unified architectures in vision and NLP (*e.g.*, CNNs, transformers), we ask: can a single model handle tasks ranging from global parameter inference to local state prediction, while preserving physical consistency? This motivates three broader questions beyond DHN:

- *What defines physical reasoning in deep learning?* Beyond next-state prediction, it encompasses parameter estimation, system identification, and discovering high-level relationships in dynamical systems. We aim for unified physical learning frameworks that adapt to diverse tasks while enforcing fundamental physical principles.
- *What is physical simulation?* Simulation doesn't always require generating trajectories step by step from an initial state. Even if used sequentially, a full trajectory can be generated in any order. For example, video generation with denoising models produces all frames at once, and Diffusion Policy Chi et al. (2023) predicts chunks of future actions ahead of time, even though they're executed step by step.
- *What physical attributes should a neural network possess?* While PDEs describe systems through local temporal relations, directly using them in networks imposes the same locality. But networks aren't limited to this. Instead of strictly following physical operators, we can extend them, preserving core properties while relaxing others, to design more flexible architectures.

We believe that physics-based learning is on the verge of a major transformation, similar to the rise of self-supervised learning in vision and NLP. By reframing physical reasoning as a reconstruction problem, predicting system states from partial or corrupted inputs, we move toward a unified modeling paradigm that blends deep learning flexibility with the rigor of physical laws. We also provide more discussions in the Appendix.

Limitations and future work While our current work provides increased flexibility in Hamiltonian-based network designs, it incurs a higher computational cost due to intensive gradient evaluations inherited from HNN. In addition, current experiments focus on small-scale systems with simple temporal dynamics. Scaling to complex spatial-temporal systems may benefit from hierarchical or attention-based architectures inspired by modern vision models.

Broader Impact Statement

This work aims to advance scientific studies by developing AI tools for physics-based reasoning. By incorporating physical constraints into neural networks, we seek to improve the interpretability and reliability of learning-based models in scientific contexts. However, as with other machine learning approaches, applying neural networks to scientific problems requires caution, as they can produce hallucinations that potentially lead to incorrect conclusions if not properly validated. While incorporating physical constraints can mitigate such risks, it does not eliminate the need for rigorous verification. It remains essential that results from AI-assisted analyses are evaluated critically and supported by both physical principles and empirical evidence.

References

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Kristjanson Duvenaud. Neural ordinary differential equations. In *Neural Information Processing Systems*, 2018. URL `https://api.semanticscholar.org/CorpusID:49310446`.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, pp. 02783649241273668, 2023.

M. Cranmer, Sam Greydanus, Stephan Hoyer, Peter W. Battaglia, David N. Spergel, and Shirley Ho. Lagrangian neural networks. *ArXiv*, abs/2003.04630, 2020. URL `https://api.semanticscholar.org/CorpusID:212644628`.

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Emilien Dupont, A. Doucet, and Yee Whye Teh. Augmented neural odes. *ArXiv*, abs/1904.01681, 2019. URL `https://api.semanticscholar.org/CorpusID:102487914`.

Marc Finzi, Ke Alexander Wang, and Andrew Gordon Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *ArXiv*, abs/2010.13581, 2020. URL `https://api.semanticscholar.org/CorpusID:225067856`.

Oscar Gonzalez. Time integration and discrete hamiltonian systems. *Journal of Nonlinear Science*, 6:449–467, 1996.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Neural Information Processing Systems*, 2019. URL `https://api.semanticscholar.org/CorpusID:174797937`.

Ernst Hairer, Marlis Hochbruck, Arieh Iserles, and Christian Lubich. Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882, 2006.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9729–9738, 2020.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16000–16009, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. Lvsm: A large view synthesis model with minimal 3d inductive bias. *arXiv preprint arXiv:2410.17242*, 2024.

Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1920–1929, 2019.

S Lall and Matthew West. Discrete variational hamiltonian mechanics. *Journal of Physics A: Mathematical and general*, 39(19):5509, 2006.

Zong-Yi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *ArXiv*, abs/2010.08895, 2020. URL `https://api.semanticscholar.org/CorpusID:224705257`.

Lennart Ljung. System identification: theory for the user. 1999. URL `https://api.semanticscholar.org/CorpusID:53821855`.

Jerrold E Marsden and Matthew West. Discrete mechanics and variational integrators. *Acta numerica*, 10: 357–514, 2001.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 165–174, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019. URL `https://api.semanticscholar.org/CorpusID:57379996`.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. *ArXiv*, abs/1909.13789, 2019. URL `https://api.semanticscholar.org/CorpusID:203593936`.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. *ArXiv*, abs/1909.12077, 2019. URL `https://api.semanticscholar.org/CorpusID:202889233`.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. *ArXiv*, abs/2002.08860, 2020. URL `https://api.semanticscholar.org/CorpusID:211205165`.

A Additional Discussions and Motivations

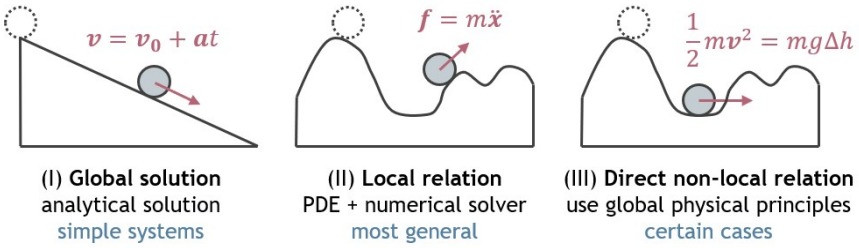

Figure 10: **How can we solve for a physical state?** (I) Closed-form analytical solutions for simple systems. (II) For more complex physical systems, most physical PDEs only model local relations of close-by time steps. (III) For certain physical systems, states can be directly related even if they are not close in time.

Our goal is to design more general neural operators that both follow physical constraints and unleash the flexibility and expressivity of neural networks as optimizable black-box functions. We start by asking the question: *What "physical relations" can we model beyond next-state prediction?*

Fig. 10 compares three classical approaches to modeling physical systems without machine learning:

- Case (I): Global analytical solution. For simple systems with regular structures, one can derive a closed-form solution directly.
- Case (II): PDE + numerical integration. In more complex settings where no closed-form solution exists, the standard practice is to formulate the system's dynamics as a PDE and solve it step-by-step over time via numerical methods. This local integration approach underlies most physics-constrained neural network designs that incorporate the PDE operators into the network.
- Case (III): Direct global relation. In some complex systems, states that are temporally far apart can be related directly using global conservation laws (*e.g.*, energy conservation). This is akin to high-school physics problems: one can compute an object's velocity at a certain position from initial conditions alone, without solving for a full trajectory. While this is less general than PDE-based approaches, it suggests a promising avenue: leveraging global physical principles within a black-box neural network could extend this technique to more complex, real-world dynamical systems beyond simple textbook problems.

Prior works such as HNN Greydanus et al. (2019) draw inspiration from the intuition of Case (III) to leverage global conservation laws, but are still realized in the form of local, step-by-step constraints between adjacent time steps, as in Case (II). To move a step forward, we aim to design network architectures that enforce global conservation principles in a truly global manner, allowing black-box neural networks to capture complex, long-range, and indirect physical relationships. This approach seeks to fully exploit the expressive power of neural networks while remaining faithful to fundamental physical laws.

B Discrete Left Hamiltonian H^-

The discrete right Hamiltonian H^- gives the equation of motion in the form

$$q_t = -\nabla_p H^-(q_{t+1}, p_t), \quad p_{t+1} = -\nabla_q H^-(q_{t+1}, p_t). \tag{9}$$

It can be a first-order approximation of the continuous Hamiltonian $\mathcal{H}$ by

$$q_t = q_{t+1} - \Delta t \nabla_p \mathcal{H}(q_t, p_{t+1}), \quad p_{t+1} = p_t - \Delta t \nabla_q \mathcal{H}(q_t, p_{t+1}). \tag{10}$$

When the states are extended to blocks, the block-wise discrete left Hamiltonian is defined as

$$Q_t^{t+b} = -\nabla_P H^-(Q_{t+s}^{t+s+b}, P_t^{t+b}), \quad P_{t+s}^{t+s+b} = -\nabla_Q H^-(Q_{t+s}^{t+s+b}, P_t^{t+b}). \tag{11}$$

Fig. 11 below illustrates the relation between discrete left and right Hamiltonians in both classical forms and our block-wise extensions. Both the left and right Hamiltonians take each other's outputs as inputs.

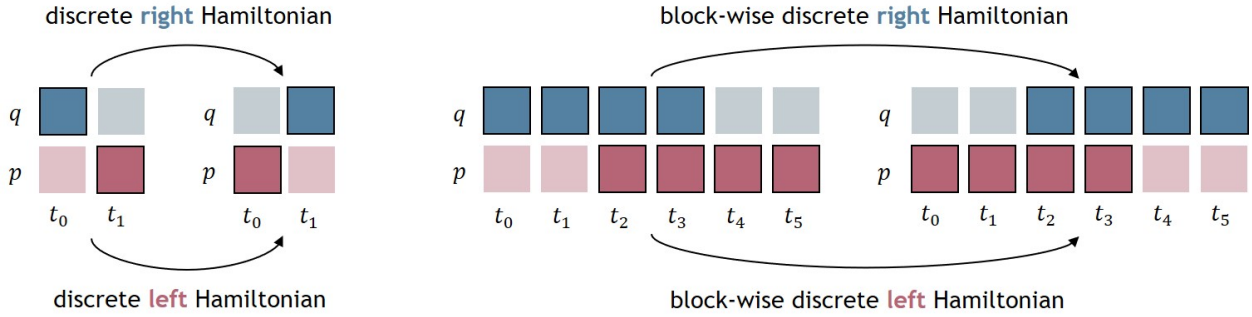

Figure 11: **Discrete left and right Hamiltonian blocks.** They describe the same formulation of a physical system with different discretizations (that both preserve the system's symplectic structures). They take each other's outputs as inputs and vice versa.

C Physical Interpretations for DHN

In this section, we discuss whether extending the discrete Hamiltonian to block sizes and strides greater than 1 still allows for explicit physical interpretations. Specifically, we address the following two questions:

(i) *What is the conserved quantity with the block-wise Hamiltonian?* For a discrete Hamiltonian block of size b, the conserved quantity is the sum of the total energy of b independent states. More specifically, the states within a discrete Hamiltonian block can be interpreted as those of identical physical systems, each starting at a different time. Fig. 12 provides an illustration of this concept.

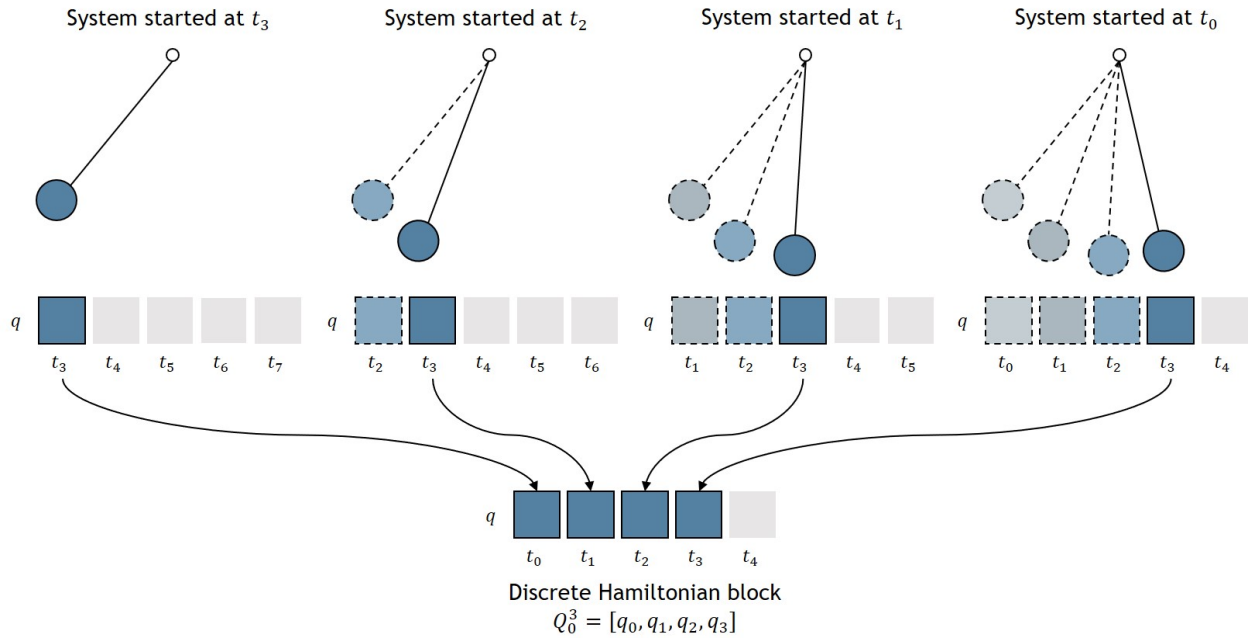

Figure 12: **Physical interpretations of block-wise discrete Hamiltonian.** The figure shows an example of block size $b = 4$. The states within a discrete Hamiltonian block can be interpreted as those of identical physical systems, each starting at a different time step.

Consider the case where the block size is $b = 4$. Suppose we have four identical physical systems, each initialized at different times: t_0, t_1, t_2, t_3. By time t_3, these systems will have evolved for 0, 1, 2, and 3 time steps, respectively. If we take their states at t_3 and stack them together, we obtain a state block that effectively represents four consecutive states spanning four time steps within a single system. Importantly, the four states at t_3 remain independent, as the four duplicated systems do not interact with one another. Thus, the conserved quantity in this framework is the total energy summed across all these identical, non-interacting systems. Fig. 13 provides a detailed illustration of this.

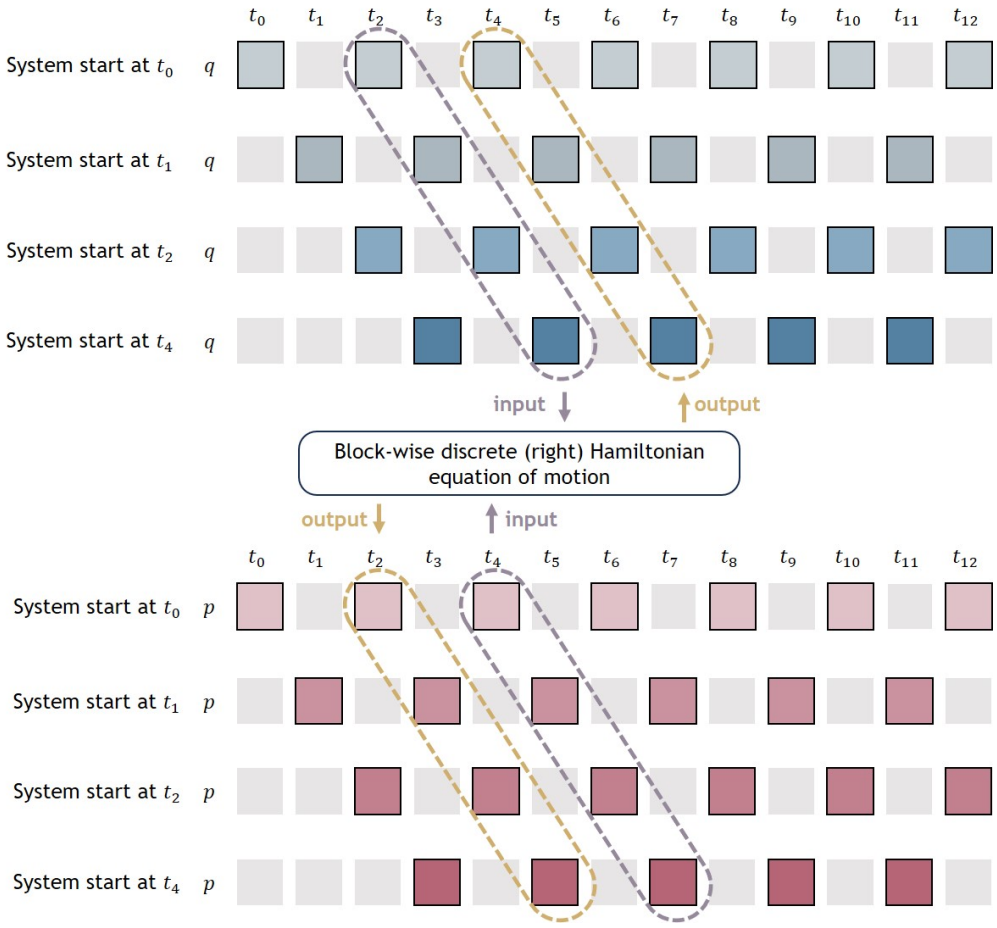

Figure 13: **Detailed explanations of the block-wise discrete Hamiltonian.** The figure shows an example of block size $b = 4$ and stride $s = 2$.

(ii) *What are the relaxations compared to the classical discrete Hamiltonian?* When extending the classical discrete Hamiltonian to a block-wise formulation, certain physical constraints are relaxed. The two main relaxations are as follows:

First, instead of conserving the energy of each individual state, the block-wise Hamiltonian conserves the total energy summed over b states. This allows for different energy distributions across the b states, making the constraint weaker than enforcing per-state energy conservation.

Second, as discussed in Sec. 4.2, when the stride s is smaller than the block size b, there is an overlap of $b - s$ between network inputs and outputs. In theory, exact energy conservation (in the generalized form) requires that the overlapping states remain identical. However, in practice, this self-consistency loss is rarely minimized to exactly zero. The extent to which it is minimized depends on factors such as network expressivity, architecture, and hyperparameters b and s, which in turn affect how well energy conservation is maintained.

Despite these relaxations, the model still enforces a form of physical consistency across the trajectory. Rather than strictly conserving per-state energy, it shifts toward preserving higher-level conserved quantities. This relaxation also opens the door to developing more abstract notions of physical consistency on latent embeddings instead of the raw observed states.

C.1 Formal Derivations of the Conserved Quantity

Here we provide formal derivations of the (generalized) energy conservation in our block-wise Hamiltonian. We show that after extending the discrete Hamiltonian into our block-wise discrete Hamiltonian with block size b and stride s, it still has a conserved quantity that relates to the system energy.

The discrete Hamiltonian can be derived in two ways: on the one hand, it can be directly derived from discrete systems, and the energy conservation (time invariance) is expressed as its symplectic structure (intuitively, it means that across a long time, the energy fluctuate around a constant value, instead of staying constant precisely); on the other hand, it can be derived as the first-order approximation of the continuous Hamiltonian, and the energy conservation also refers to approximately conserving the energy of continuous system. Here, for our block-wise Hamiltonian, we analyze its conserved quantity with the second interpretation, because the symplectic structure is highly relevant to numerical integrations for state updates from t to $t+1$, while in our case with stride $s > 1$ and different masking patterns, our state update is much more flexible.

First-order approximation of the continuous Hamiltonian Given time steps $t = 0, \cdots, T$, suppose we have a physical trajectory with states $(q_0, p_0), (q_1, p_1), \cdots, (q_T, p_T)$. The total action with the discrete right Hamiltonian is

$$S^+ = \sum_{t=0}^{T-1} (p_{t+1} \cdot q_{t+1} - H^+(q_t, p_{t+1})). \tag{12}$$

On the other hand, the action of the continuous Hamiltonian is

$$\mathcal{S} = \int_0^1 (p(\tau) \cdot q(\tau) - \mathcal{H}(q(\tau), p(\tau))) \mathrm{d}\tau. \tag{13}$$

Here we assume that the discrete time steps are uniformly sampled between $[0, 1]$ with interval $\Delta t = 1/T$. To derive numerical integrators using discrete Hamiltonians, it is important to recognize that the discrete action sum must approximate the continuous action Lall & West (2006); Marsden & West (2001); Hairer et al. (2006), that is, $S^+ \approx \mathcal{S}$. This means that the right Hamiltonian should satisfy

$$H^+(q_t, p_{t+1}) \approx p_{t+1} \cdot q_{t+1} - \int_{t\Delta t}^{(t+1)\Delta t} (p(\tau) \cdot \dot{q}(\tau) - \mathcal{H}(q(\tau), p(\tau))) \, \mathrm{d}\tau. \tag{14}$$

With first-order approximation, we have

$$\begin{aligned} &H^+(q_t, p_{t+1}) \\ &= p_{t+1} \cdot (q_t + \Delta t \nabla_p \mathcal{H}(q_t, p_{t+1})) - \Delta t (p_{t+1} \nabla_p \mathcal{H}(q_t, p_{t+1}) - \mathcal{H}(q_t, p_{t+1}))) + \mathcal{O}(\Delta t^2) \\ &= p_{t+1} \cdot q_t + \Delta t \mathcal{H}(q_t, p_{t+1}) + \mathcal{O}(\Delta t^2). \end{aligned} \tag{15}$$

With time invariance, the continuous Hamiltonian $\mathcal{H}$ conserves the total energy of the system, and the discrete right Hamiltonian H^+ is an approximation of that.

Block-wise approximation Consider a discrete right Hamiltonian block of block size b and stride s, with inputs are $Q_t^{t+b} = [q_t, \cdots, q_{t+b}]$ and $P_{t+s}^{t+s+b} = [p_{t+s}, \cdots, p_{t+s+b}]$ and satisfying the equation of motion as in Eq. 5 for any t. The total action approximation generalizes to

$$H^+(Q_t^{t+b}, P_{t+s}^{t+s+b}) \approx \sum_{k=t}^{t+b} \left(p_{k+s} \cdot q_{k+s} - \int_{k\Delta t}^{(k+s)\Delta t} (p(\tau) \cdot \dot{q}(\tau) - \mathcal{H}(q(\tau), p(\tau))) \, \mathrm{d}\tau \right). \tag{16}$$

And similar to Eq. 15, we have

$$H^+(Q_t^{t+b}, P_{t+s}^{t+s+b}) = \sum_{k=t}^{t+b} (p_{k+s} \cdot q_k + s\Delta t \mathcal{H}(q_k, p_{k+s})) + \mathcal{O}(\Delta t^2). \tag{17}$$

Omitting the higher-order terms $\mathcal{O}(\Delta t^2)$, this can be viewed as a summation of b identical systems starting at time steps $t = 0, \cdots, b$, and their continuous Hamiltonians are discretized with time intervals $s\Delta t$. In this analogy, the total conserved quantity is an approximation of

$$\sum_{k=0}^{b} \mathcal{H}(q(\tau + k\Delta t), p(\tau + k\Delta t)). \tag{18}$$

D Denoising Inference

As mentioned in Sec. 3.3, at training time, we apply noises with randomly sampled scales to different unknown states. However, at inference time, we progressively denoise the unknown states with a sequence of decreasing noise scales that are synchronized on all unknown states. Fig. 14 illustrates the iterative denoising process at inference time with a pair of DHN blocks H^+ and H^-.

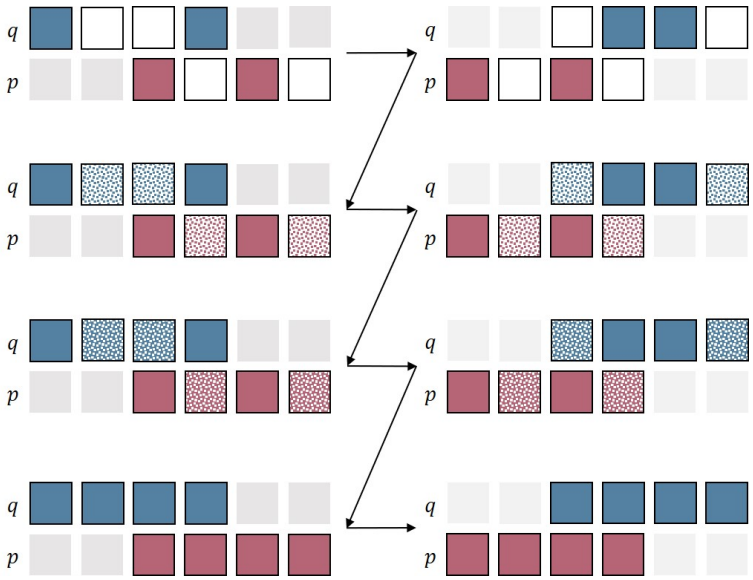

Figure 14: **Iterative denoising at inference time.** A pair of DHN blocks, H^+, H^-, with block size b and stride s, are jointly applied to a stack of $b + s$ states to denoise the unknown blocks. Progressive denoising is conducted by iteratively applying block-wise H^+ and H^- and gradually decreasing the noise scales.

Consider a block of states $(Q_t^{t+s+b}, P_t^{t+s+b})$ with some unknown slots indicated by binary masks $(M_t^{t+s+b}, N_t^{t+s+b})$. We iterative apply $H^+(Q_t^{t+b}, P_{t+s}^{t+s+b})$ and $H^-(Q_{t+s}^{t+s+b}, P_t^{t+b})$ to infer the unknown states conditioned on the known states. We sample the initial state predictions from a Gaussian distribution and then apply iterative denoising with decreasing noise levels $1 = \alpha_N > \alpha_{N-1} > \cdots > \alpha_1 > \alpha_0 = 0$, as in Algorithm 1. This is similar to the diffusion models Ho et al. (2020); Song et al. (2020).

E Network Architecture Details

As mentioned in Sec. 3.4, we employ a decoder-only transformer to learn the block-wise Hamiltonian. Its detailed architecture is shown in Fig. 15. The network is very lightweight, with only two transformer blocks, plus input embeddings and the (linear) output projection. The two transformer blocks are identical, with input/output feature dimension 128, FFN hidden feature dimension 120×4, and 4 heads in their multi-head attentions.

Algorithm 1: Iterative denoising with the Hamiltonian blocks

Data: Known states $(Q_{\text{known}})_t^{t+s+b}, (P_{\text{known}})_t^{t+s+b}$, binary state mask $(M_t^{t+s+b}, N_t^{t+s+b})$.
Result: The complete block Q_t^{t+s+b}, P_t^{t+s+b} with the masked states predicted.
Sample initial state predictions $Q_t^{t+s+b}, P_t^{t+s+b} \sim \mathcal{N}(0, \mathbf{I})$.
for $n = N-1, \cdots, 0$ **do**

Sample noise $\mathcal{E}, \mathcal{E}' \sim \mathcal{N}(0, \mathbf{I})$, and apply H^+ denoising:

$$(Q_{\text{pred}})_{t+s}^{t+s+b} \leftarrow (1-\alpha_n)\nabla_P H^+(Q_t^{t+b}, P_{t+s}^{t+s+b}) + \alpha_n \mathcal{E},$$
$$(P_{\text{pred}})_t^{t+b} \leftarrow (1-\alpha_n)\nabla_Q H^+(Q_t^{t+b}, P_{t+s}^{t+s+b}) + \alpha_n \mathcal{E}'.$$

Combine the predicted states with the known states:

$$Q_{t+s}^{t+s+b} \leftarrow (1 - M_{t+s}^{t+s+b})(Q_{\text{pred}})_{t+s}^{t+s+b} + M_{t+s}^{t+s+b}(Q_{\text{known}})_{t+s}^{t+s+b},$$
$$P_t^{t+b} \leftarrow (1 - M_t^{t+b})(P_{\text{pred}})_t^{t+b} + M_t^{t+b}(P_{\text{known}})_t^{t+b}.$$

Sample noise $\mathcal{E}, \mathcal{E}' \sim \mathcal{N}(0, \mathbf{I})$, and apply H^- denoising:

$$(Q_{\text{pred}})_t^{t+b} \leftarrow -(1-\alpha_n)\nabla_P H^-(Q_{t+s}^{t+s+b}, P_t^{t+b}) + \alpha_n \mathcal{E},$$
$$(P_{\text{pred}})_{t+s}^{t+s+b} \leftarrow -(1-\alpha_n)\nabla_Q H^-(Q_{t+s}^{t+s+b}, P_t^{t+b}) + \alpha_n \mathcal{E}'.$$

Combine the predicted states with the known states:

$$Q_t^{t+b} \leftarrow (1 - M_t^{t+b})(Q_{\text{pred}})_t^{t+b} + M_t^{t+b}(Q_{\text{known}})_t^{t+b},$$
$$P_{t+s}^{t+s+b} \leftarrow (1 - M_{t+s}^{t+s+b})(P_{\text{pred}})_{t+s}^{t+s+b} + M_{t+s}^{t+s+b}(P_{\text{known}})_{t+s}^{t+s+b}.$$

end

F Experimental Setups

F.1 Physical Systems and Parameter Randomization

Here we elaborate on the details of the two settings we experiment with: the *single pendulum* and the *double pendulum*, as illustrated in Fig. 16. In both settings, we first define the generalized coordinate q and the system's Lagrangian $\mathcal{L}(q, \dot{q})$. The generalized momenta is then defined by $p = \nabla_{\dot{q}}\mathcal{L}$. We set the gravitational acceleration $g = 0.981$.

Single pendulum In this system, the varied parameter is the string length l, randomly sampled between [0.5, 1.0] for each trajectory. The mass of the ball is set to be $m = 1$. The generalized coordinate is defined as $q = \theta$, with initial value $\theta = \pi/2$ for all trajectories. The Lagrangian of the system is

$$\mathcal{L} = \frac{1}{2}ml^2\dot{q}^2 - mgl(1 - \cos q). \tag{19}$$

Here (q, p) are the standard angular position and angular momentum in spherical coordinates.

Double pendulum In this system, the varied parameter is the string length l_2, randomly sampled between [0.5, 1.5] for each trajectory. The remaining fixed parameters are $l_1 = 1, m_1 = m_2 = 1$. The generalized coordinate is defined as $q = (\theta_1, \theta_2)$, with initial values $\theta_1 = \theta_2 = \pi/2$ for all trajectories. The Lagrangian of the system is

$$\mathcal{L} = \frac{1}{2}(m_1 + m_2)l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2 l_2^2\dot{\theta}_2^2 + m_2 l_1 l_2 \dot{\theta}_1\dot{\theta}_2\cos(\theta_1 - \theta_2) + (m_1 + m_2)gl_1\cos\theta_1 + m_2 g l_2 \cos\theta_2. \tag{20}$$

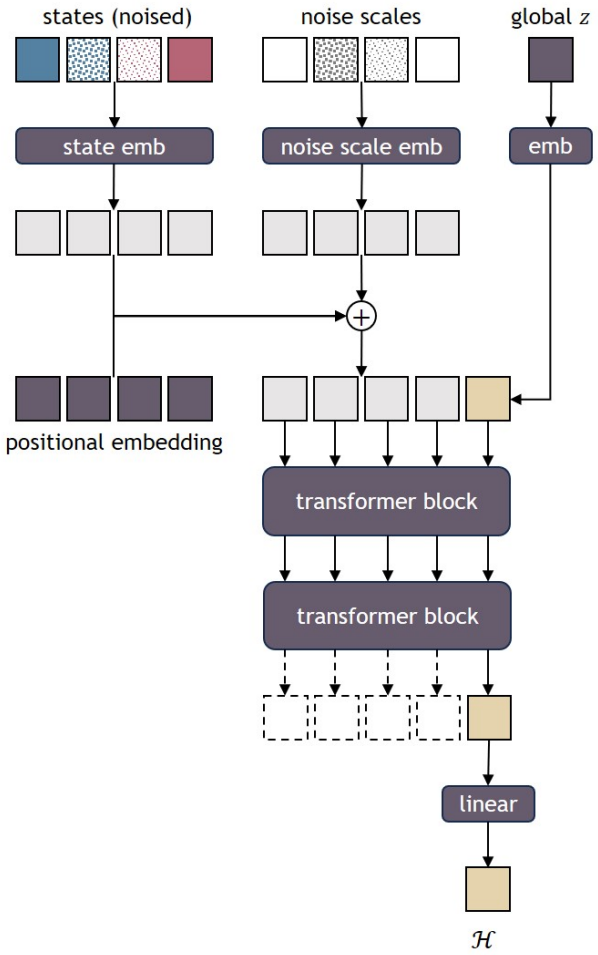

Figure 15: **Detailed architecture of the decoder-only transformer.** Learnable modules or variables are shown in dark purple. The final output of the network is a scalar Hamiltonian value, decoded from the position of the global latent code in the transformer (shown in yellow).

F.2 Network Training and Computational Resources

At the training stage, we optimize for both the network weights and the latent codes. We use an Adam optimizer with a learning rate 10^{-4} and weight decay 10^{-4} and train for 200 epochs with a batch size of 32.

At the test-time optimization stage, we freeze the network weights and only optimize for the latent codes. e use an Adam optimizer with a learning rate 10^{-2} and weight decay 10^{-4}. As all latent codes are optimized independently, we optimize for 1000 epochs with a batch size of 100.

All networks are trained on a single Titan RTX 24GB GPU.

G Error Intervals of Experimental Results

Here we show the error intervals of the experimental results in Sec. 4.1. We report the mean and (1-sigma) standard deviation of the data points, averaged across the dataset (for overfitting experiments, it is computed on the training set; for completion experiments, it is computed on the test set). As showing the error intervals of all the 128 time steps of the trajectories will make the tables extremely large, we only present the results every 16 steps.

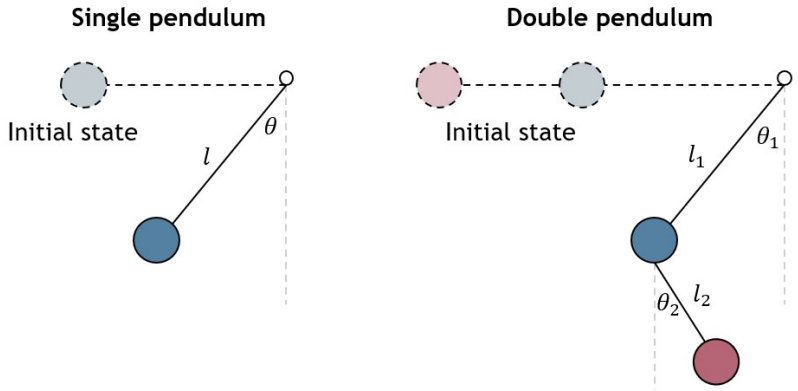

Figure 16: **Physical systems for the experiments.** Circles with dotted lines and swallower colors are the initial states, which are identical to all training and testing trajectories. Circles with solid lines and darker colors illustrate the intermediate states along the simulated trajectory.

Time step	15	31	47	63	79	95	111	127
ResNet (res blocks=1)	0.01± 0.01	0.06± 0.04	0.11± 0.07	0.17± 0.11	0.22± 0.14	0.27± 0.17	0.32± 0.20	0.36± 0.22
ResNet (res blocks=2)	0.01± 0.01	0.06± 0.04	0.11± 0.06	0.16± 0.10	0.21± 0.13	0.26± 0.16	0.31± 0.19	0.35± 0.21
Transformer	0.01± 0.00	0.03± 0.02	0.05± 0.04	0.07± 0.06	0.09± 0.07	0.11± 0.09	0.14± 0.11	0.16± 0.13
HNN (forward Euler)	0.07± 0.02	0.34± 0.12	0.80± 0.06	1.02± 0.05	1.08± 0.07	1.19± 0.16	1.43± 0.14	1.48± 0.12
HNN (Leapfrog)	0.13± 0.03	0.27± 0.04	0.45± 0.08	0.59± 0.08	0.67± 0.04	0.69± 0.04	0.71± 0.04	0.70± 0.04
Ours (block size = 8)	0.02± 0.01	0.09± 0.04	0.13± 0.07	0.17± 0.10	0.22± 0.13	0.26± 0.15	0.31± 0.18	0.35± 0.19
Ours (block size = 4)	0.01± 0.01	0.08± 0.03	0.17± 0.09	0.28± 0.14	0.37± 0.18	0.45± 0.20	0.51± 0.22	0.55± 0.22
Ours (block size = 2)	0.02± 0.01	0.12± 0.08	0.21± 0.14	0.30± 0.18	0.38± 0.23	0.45± 0.25	0.50± 0.26	0.55± 0.26

Table 1: Fitting known trajectories: single pendulum, average error on q over time.

Time step	15	31	47	63	79	95	111	127
ResNet (res blocks=1)	0.24± 0.19	0.36± 0.30	0.65± 0.50	0.82± 0.63	1.02± 0.81	1.18± 0.89	1.40± 1.05	1.57± 1.17
ResNet (res blocks=2)	0.23± 0.18	0.18± 0.15	0.38± 0.30	0.38± 0.33	0.46± 0.41	0.55± 0.43	0.55± 0.48	0.62± 0.53
Transformer	0.12± 0.09	0.12± 0.09	0.21± 0.17	0.20± 0.17	0.25± 0.22	0.26± 0.23	0.26± 0.23	0.28± 0.25
HNN (forward Euler)	2.61± 0.35	7.35± 1.62	10.39± 0.75	8.55± 2.17	15.58± 1.49	8.96± 7.05	4.16± 3.54	17.90± 3.20
HNN (Leapfrog)	2.74± 1.44	2.16± 1.03	1.90± 1.01	1.41± 0.81	1.26± 0.76	1.06± 0.75	1.37± 0.96	2.55± 1.54
Ours (block size = 8)	1.26± 0.79	0.69± 0.42	0.55± 0.55	0.73± 0.54	0.59± 0.54	1.10± 0.98	1.11± 0.93	1.48± 1.73
Ours (block size = 4)	0.50± 0.28	1.58± 0.68	0.56± 0.51	2.24± 1.52	2.34± 1.23	1.07± 1.87	2.29± 5.26	2.65± 7.11
Ours (block size = 2)	0.14± 0.10	0.14± 0.10	0.34± 0.20	0.30± 0.22	0.46± 0.26	0.33± 0.20	0.36± 0.29	0.42± 0.22

Table 2: Fitting known trajectories: single pendulum, total energy error per time step.

Time step	15	31	47	63	79	95	111	127
ResNet (res blocks=1)	0.08± 0.04	0.20± 0.17	0.32± 0.31	0.46± 0.46	0.85± 0.89	1.26± 1.41	1.65± 1.90	2.04± 2.36
ResNet (res blocks=2)	0.06± 0.03	0.13± 0.07	0.19± 0.09	0.30± 0.22	0.67± 0.77	1.04± 1.30	1.44± 1.77	1.88± 2.21
Transformer	0.04± 0.01	0.19± 0.11	0.34± 0.27	0.69± 0.80	1.21± 1.51	1.66± 2.03	2.05± 2.33	2.44± 2.59
HNN (forward Euler)	0.08± 0.03	0.49± 0.15	1.02± 0.43	2.31± 1.00	3.52± 1.92	4.57± 2.80	5.49± 3.47	6.35± 4.02
HNN (Leapfrog)	0.26± 0.01	0.57± 0.05	0.78± 0.12	1.28± 0.30	1.69± 0.71	2.04± 1.15	2.43± 1.55	2.92± 1.93
Ours (block size = 8)	0.11± 0.05	0.26± 0.08	0.41± 0.14	0.58± 0.28	1.02± 0.74	1.49± 1.12	1.92± 1.35	2.36± 1.60
Ours (block size = 4)	0.08± 0.03	0.23± 0.12	0.32± 0.17	0.48± 0.33	0.84± 0.81	1.19± 1.24	1.46± 1.52	1.77± 1.83
Ours (block size = 2)	0.09± 0.04	0.20± 0.07	0.26± 0.11	0.40± 0.25	0.77± 0.78	1.09± 1.20	1.34± 1.51	1.62± 1.84

Table 3: Fitting known trajectories: double pendulum, average error on q over time.

Time step	15	31	47	63	79	95	111	127
ResNet (res blocks=1)	0.03± 0.01	0.16± 0.08	0.30± 0.16	0.43± 0.22	0.53± 0.25	0.61± 0.27	0.70± 0.28	0.75± 0.28
ResNet (res blocks=2)	0.03± 0.01	0.13± 0.07	0.24± 0.11	0.35± 0.17	0.45± 0.22	0.52± 0.24	0.58± 0.25	0.64± 0.26
Transformer	0.08± 0.00	0.47± 0.08	0.71± 0.08	0.81± 0.05	0.80± 0.08	0.79± 0.09	0.81± 0.07	0.82± 0.06
HNN (forward Euler)	0.13± 0.01	0.39± 0.14	0.97± 0.04	1.08± 0.03	1.30± 0.05	1.30± 0.07	1.49± 0.03	1.79± 0.06
HNN (Leapfrog)	0.09± 0.01	0.16± 0.06	0.27± 0.16	0.38± 0.22	0.47± 0.21	0.54± 0.17	0.63± 0.14	0.68± 0.10
Ours (block size = 8)	0.04± 0.02	0.25± 0.14	0.42± 0.24	0.53± 0.27	0.62± 0.26	0.65± 0.23	0.67± 0.20	0.69± 0.18
Ours (block size = 4)	0.03± 0.01	0.15± 0.07	0.28± 0.14	0.40± 0.19	0.51± 0.19	0.60± 0.18	0.66± 0.17	0.71± 0.17
Ours (block size = 2)	0.02± 0.02	0.13± 0.10	0.23± 0.15	0.32± 0.20	0.41± 0.25	0.48± 0.27	0.52± 0.28	0.55± 0.27

Table 4: Completion on novel trajectories: single pendulum, average error on q over time.

Time step	15	31	47	63	79	95	111	127
ResNet (res blocks=1)	0.33± 0.14	1.18± 0.24	2.04± 0.34	2.33± 0.36	3.55± 0.70	3.76± 0.72	3.70± 0.51	5.54± 0.94
ResNet (res blocks=2)	0.15± 0.11	0.14± 0.04	0.18± 0.06	0.60± 0.15	0.53± 0.17	0.42± 0.11	0.65± 0.09	0.63± 0.08
Transformer	0.19± 0.05	0.48± 0.08	0.81± 0.09	0.94± 0.21	1.00± 0.16	1.05± 0.14	1.07± 0.20	1.16± 0.15
HNN (forward Euler)	3.78± 0.48	9.48± 3.10	11.16± 2.17	11.79± 5.26	8.50± 1.07	19.76± 4.08	5.55± 3.61	5.82± 1.25
HNN (Leapfrog)	1.65± 1.08	1.34± 0.64	2.56± 1.00	3.44± 1.27	1.72± 1.05	5.15± 1.26	9.30± 0.70	7.88± 2.50
Ours (block size = 8)	0.76± 0.44	1.13± 0.80	0.81± 0.80	1.37± 0.95	2.82± 3.75	2.75± 3.58	3.87± 6.06	7.12± 11.29
Ours (block size = 4)	1.02± 0.55	0.98± 0.52	1.41± 0.87	3.25± 1.82	6.69± 6.91	9.21± 8.73	21.96± 37.86	35.52± 67.18
Ours (block size = 2)	0.15± 0.08	0.11± 0.07	0.37± 0.14	0.25± 0.13	0.54± 0.20	0.31± 0.12	0.23± 0.13	0.44± 0.16

Table 5: Completion on novel trajectories: single pendulum, total energy error per time step.

Time step	15	31	47	63	79	95	111	127
ResNet (res blocks=1)	0.09± 0.02	0.15± 0.02	0.30± 0.07	0.57± 0.16	1.75± 0.48	2.76± 0.73	3.23± 0.92	3.54± 1.14
ResNet (res blocks=2)	0.09± 0.04	0.16± 0.05	0.31± 0.04	0.56± 0.14	1.78± 0.61	3.27± 1.22	4.49± 1.74	5.43± 2.10
Transformer	0.04± 0.02	0.21± 0.13	0.46± 0.39	1.16± 1.23	2.01± 2.32	2.73± 3.18	3.14± 3.58	3.43± 3.82
HNN (forward Euler)	0.13± 0.02	0.40± 0.12	0.92± 0.21	1.93± 0.96	2.94± 1.53	3.84± 2.00	4.65± 2.43	5.32± 2.68
HNN (Leapfrog)	0.19± 0.02	0.41± 0.08	0.64± 0.12	1.05± 0.31	1.45± 0.65	1.81± 0.97	2.02± 1.25	2.29± 1.59
Ours (block size = 8)	0.14± 0.04	0.27± 0.08	0.41± 0.15	0.56± 0.29	1.01± 0.80	1.47± 1.19	1.89± 1.39	2.33± 1.63
Ours (block size = 4)	0.08± 0.04	0.24± 0.12	0.32± 0.17	0.50± 0.31	0.88± 0.83	1.23± 1.26	1.48± 1.50	1.76± 1.82
Ours (block size = 2)	0.09± 0.04	0.17± 0.05	0.24± 0.12	0.38± 0.27	0.79± 0.83	1.12± 1.24	1.37± 1.53	1.64± 1.86

Table 6: Completion on novel trajectories: double pendulum, average error on q over time.