Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Structure-Preserving Modeling of Non-Conservative Systems via Random Feature Hamiltonian Networks

Deniz Erdoğan

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Structure-Preserving Modeling of Non-Conservative Systems via Random Feature Hamiltonian Networks

Author:	Deniz Erdoğan
Examiner:	Univ.-Prof. Dr. Felix Dietrich
Assistant advisor:	Atamert Rahma
Submission Date:	February 2nd, 2026

Acknowledgments

I would like to express my sincere gratitude to my advisor, M.Sc. Atamert Rahma, whose insights and expertise were invaluable throughout the course of this research. His guidance and feedback shaped the direction of this work and helped me navigate the challenges along the way. I am also grateful to Prof. Felix Dietrich for his supervision and for providing the opportunity to pursue this research. Finally, I would like to thank my friend M.Sc. Yılkut Murat Aydın for the many productive discussions, thoughtful suggestions, and for generously sharing his office space during the writing of this thesis.

Abstract

Modeling dynamical systems with complex interactions remains a significant challenge in computational science. While data-driven approaches offer flexibility, they often fail to respect fundamental physical laws, leading to poor generalization and energy drift over long time horizons. Although physics-informed learning, specifically Hamiltonian Neural Networks (HNNs), addresses this by enforcing a symplectic structure on the prediction vector field, standard HNNs are ill-equipped to handle real-world dissipative systems characterized by friction and external forces. Furthermore, training these networks using iterative backpropagation is often computationally expensive and sensitive to hyperparameter tuning. To address these limitations, this work introduces the Random Dissipative Hamiltonian Neural Networks (RDHNN), a framework that combines structure-preserving deep learning with the non-iterative Sampling Where It Matters (SWIM) algorithm. By decomposing the vector field into conservative and dissipative components, the RDHNN simultaneously learns the Hamiltonian and a dissipative function. The network constructs hidden layer activations from sampled data points and computes output weights analytically which eliminates the need for iterative optimization. We evaluate the performance of RDHNN and a parameter efficient variant, the SharedRDHNN, on benchmark dynamical systems including the damped single and double pendulums, mass-spring systems, RLC circuits and Duffing and Morse oscillators. The proposed framework demonstrates extreme training efficiency and achieves speedup factors of over five orders of magnitude compared to Adam optimization while maintaining trajectory prediction accuracy comparable to or better than that of standard neural networks and HNNs. Furthermore, since it explicitly decomposes the system into conservative and dissipative terms, it provides interpretable insights into the dynamical contributions of separate components rather than functioning as a black box approximator.

Contents

Part I.

Introduction

1. Introduction

Over the years, various machine learning techniques have been developed and extensively studied across multiple fields such as computer vision [17], healthcare [62], robotics [86] and reinforcement learning [63]. Among these different techniques, neural networks have shown great success in learning complex functions from data and modeling systems across many domains. Although these models are capable of fitting the data remarkably well, their predictions can often violate or be inconsistent with the underlying dynamics of the domain, especially when extrapolated to longer time horizons.

Consider a simple physical system with energy conservation. A neural network model trained on a dataset of trajectories of the system can achieve high accuracy in reproducing the local trajectories but over time may drift away into higher or lower energy states as the simulation progresses [35]. To achieve better long term generalization on the learned system, different biases can be introduced into the model architecture or the training process to guide the model towards solutions that respect the underlying laws. The main idea behind this work is to make the best use of the knowledge and physical laws that have been derived over centuries of research instead of disregarding it completely by training on the data only. These biases can take many forms such as known symmetries, conservation laws or other constraints that are known to be present in the system. Such networks are referred to as physics-informed neural networks (PINNs) [75]. These models are not only useful to predict the dynamics of a system, but also to extract interpretable patterns and information [52]. Across various fields, PINNs have been utilized to discover previously unknown physical properties of systems [83, 82]. In this thesis, we focus on learning dissipative Hamiltonian systems using PINNs by imposing the Hamiltonian structure into the model architecture and the training process. Previous work on learning both conservative and dissipative systems using PINNs has been done [87, 35] using traditional gradient descent based iterative optimization methods. However, these methods are often computationally expensive, require long training times [50], and are sensitive to initialization and hyperparameters [91].

We take a different approach by sampling activations in the hidden layers and computing the output weights analytically and reach the same performance as the iterative optimization methods with a fraction of the training time. We build on the work on sampling Hamiltonian functions by Rahma et al. [73, 74] by extending the model to learn dissipative systems as well. This extension makes the model significantly more flexible and applicable to real world systems as many systems are dissipative in nature. In our proposed framework, the physical priors are utilized in both the model architecture as well as the optimization process through using the Hamiltonian structure and Rayleighian dissipation. The performance and trade-offs of the proposed methods are evaluated through numerical experiments on various dissipative Hamiltonian systems.

We review the state of the art methods on the topic in Part II, where we introduce the preliminary topics needed in Section 2.1, a review of data-driven learning of Hamiltonian systems in Section 2.2 and neural network sampling algorithms in Section 2.3. In Part III, we introduce our framework for learning dissipative Hamiltonian systems by sampling networks and evaluate the performance of the proposed methods through numerical experiments. Finally, in Part IV, we summarize our findings and discuss the potential future work.

Part II.

State of the Art

2. State of the Art

This section covers the preliminary topics needed to understand the state of the art methods for learning Hamiltonian systems and dissipative systems using numerical methods and machine learning techniques. We start by giving brief introductions to dynamical systems (Section 2.1.1), Hamiltonian mechanics (Section 2.1.2), numerical integration methods (Section 2.1.3), least squares approximation (Section 2.1.4), Deep Feedforward Networks (Section 2.1.5), and Helmholtz decomposition (Section 2.1.6). Following this, we cover the state of the art methods on learning Hamiltonian systems from data (Section 2.2) through physics-informed learning (Section 2.2.1), physics-informed learning for Hamiltonian systems (Section 2.2.2), and dissipative Hamiltonian systems (Section 2.2.3). Finally, we introduce and review the literature regarding sampling neural networks (Section 2.3) with a focus on the Sampling Where It Matters (SWIM) algorithm (Section 2.3.1).

2.1. Preliminaries

2.1.1. Dynamical Systems

Dynamical systems are a mathematical framework for describing the state evolution and long term behavior of evolving systems. This framework provides tools to analyze trajectories, equilibria, and stability of such systems [13]. Different fields like classical mechanics, fluid dynamics, biology, chemistry, linguistics and economics use dynamical systems to make predictions about the behavior of phenomena [88]. At its core, dynamical systems consist of a set of rules that govern the evolution of a system. Over time, denoted by $t \in T$, the state of the system changes according to these rules, where T is the set of numbers that represents time [72].

When T represents discrete time ($T = \mathbb{Z}$), the dynamical system becomes a map that transforms the system's state at time t to the state at time $t + 1$. A map $f : \mathcal{X} \to \mathcal{X}$ is defined as

$$f^{n+1} = f \circ f^n$$

for each $n \in \mathbb{N}$ where $\mathcal{X}$ is the state space of the system. The base condition for the recursive definition is $f^0 = id$, where id is the identity map [4]. These systems are called discrete time dynamical systems. When T represents continuous time ($T = \mathbb{R}$), it is called a continuous time dynamical system. These systems use a map $\varphi^t : \mathcal{X} \to \mathcal{X}$ that transforms an initial state, x_0, to a state at time t, denoted by x_t, by

$$x_t = \varphi^t(x_0).$$

A family of such maps where $\varphi^0 = id$ and $\varphi_{t+s} = \varphi^t \circ \varphi^s$ for every $t, s \in \mathbb{R}$ is called a *flow* [4]. For a flow φ,

$$\varphi^{t+s} = \varphi^t \circ \varphi^s$$

for every $t, s \in T$. This property of the flow signifies the autonomous behavior of the system, where the state of the system over $t + s$ units of time can be obtained by first calculating the state of the system over s units of time and then applying the flow over t units of time. Autonomous systems are obtained for systems where the rules governing their evolution do not depend on time. An autonomous system of first order ordinary differential equations (ODEs) has the form

$$\dot{x} = f(x)$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ is a vector field, $x(t) \in \mathbb{R}^d$ is the state of the system at time t and $\dot{x}$ is the time derivative. In their component form, the state and the system can be defined as $x = (x_1, x_2, \ldots, x_d)$ and $f(x) = (f_1(x_1, x_2, \ldots, x_d), \ldots, f_d(x_1, x_2, \ldots, x_d))$.

2.1.2. Hamiltonian Mechanics

Named after its pioneer, Sir William Rowan Hamilton, Hamiltonian mechanics is a formulation of dynamical systems in which the evolution of a system is described by Hamilton's equations of motion [40, 41]. It is frequently used in various fields like thermodynamics, quantum mechanics and statistical mechanics [31, 68, 3, 78].

Although it is based on Lagrangian mechanics and is equivalent to both Lagrangian and Newtonian mechanics, the Hamiltonian formalism introduces a convenient geometric interpretation of the system. In this formalism, the state evolution of a system is represented as a flow in phase space and based on an energy-like quantity, the Hamiltonian.

The Hamiltonian of the system is taken to be a function of position $\mathbf{q}$ and momentum $\mathbf{p}$, denoted by $\mathcal{H}(\mathbf{q}, \mathbf{p})$. This representation allows us to represent the system in a phase space with coordinates $\mathbf{q}$ and $\mathbf{p}$. When modeling the same system, a second order Lagrangian formulation that has n independent coordinates will require 2n initial conditions. However, when modeled using the Hamiltonian formalism, the same system can be defined with 2n first order equations. In the scope of this thesis, reducing the order of the equations for the price of doubling the number of equations will be beneficial since first order methods are more convenient to work with [12].

Using this formulation, the evolution of a system can be defined using ODEs in terms of the rate of change, $\dot{x}_i = \frac{dx_i(t)}{dt}$ as functions of the coordinates in a Euclidean state space $x = (x_1, x_2, \ldots, x_d) \in \mathcal{X} = \mathbb{R}^d$ in continuous time dynamical systems. A Hamiltonian system is defined on a phase space, $\mathcal{X} \subseteq \mathbb{R}^d \times \mathbb{R}^d$, with coordinates $(q_1, q_2, \ldots, q_n, p_1, p_2, \ldots, p_n)$ or $(\mathbf{q}, \mathbf{p})$ in vector form. Within the system, the Hamiltonian is defined as $\mathcal{H} : \mathcal{X} \to \mathbb{R}$. The laws of motion in Hamiltonian systems are defined to be

$$\dot{q} = \frac{\partial \mathcal{H}(q,p)}{\partial p} = \nabla_p \mathcal{H}(q,p), \tag{2.1}$$

$$\dot{p} = -\frac{\partial \mathcal{H}(q,p)}{\partial q} = -\nabla_q \mathcal{H}(q,p), \tag{2.2}$$

where q, p, $\dot{q}$ and $\dot{p}$ are the position, momentum and their time derivatives respectively in vector form. Hamilton's equations (Equation (2.1) and Equation (2.2)) hold for all $x \in \mathcal{X}$ and $t \in \mathbb{T}$. For an autonomous Hamiltonian system, the Hamiltonian does not depend on time and its value is constant along any Hamiltonian flow curve in the phase space. This property represents the conservation of energy as the system evolves in time. This property can be demonstrated with

$$\begin{aligned}\frac{d}{dt}\mathcal{H}\big(q(t),p(t)\big) &= \frac{\partial \mathcal{H}}{\partial q}\cdot\frac{dq}{dt} + \frac{\partial \mathcal{H}}{\partial p}\cdot\frac{dp}{dt}\\ &= \frac{\partial \mathcal{H}}{\partial q}\cdot\frac{\partial \mathcal{H}}{\partial p} - \frac{\partial \mathcal{H}}{\partial p}\cdot\frac{\partial \mathcal{H}}{\partial q}\\ &= 0\end{aligned}$$

for all $x \in \mathcal{X}$ where we write $\mathcal{H}$ instead of $\mathcal{H}(q,p)$ and q and p instead of $q(t)$ and $p(t)$ for brevity. This property can also be interpreted as the conservation of energy in mechanical systems, where the Hamiltonian is equivalent to the total energy of the system. In many such cases, the Hamiltonian can be defined as $\mathcal{H} = T + V$, where T is the kinetic energy and V is the potential energy of the system.

From Hamilton's equations, we can see that the time evolution of the system is in the direction of $\mathbf{S}_{\mathcal{H}} = \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\right)$ where $\mathbf{S}$ is a vector field over the input space $\mathcal{X}$. This vector field is called the *symplectic gradient*. While the gradient of a function is a vector field that points in the direction of the steepest ascent, the symplectic gradient conserves the Hamiltonian constant along the flow. That is, the system's trajectory in its phase space draws a contour line where $\mathcal{H}$ is constant. Integrating the system using the dynamics given by the symplectic gradient conserves the total energy of the system. Equation (2.1) and Equation (2.2) can be reformulated as a linear system of equations as

$$\mathcal{J} \cdot \nabla \mathcal{H}(x) - v(x) = \vec{0}, \tag{2.3}$$

where

$$\mathcal{J} = \begin{bmatrix} 0_d & I_d \\ -I_d & 0_d \end{bmatrix} \in \{0,1\}^{2d \times 2d}, \quad I_d \in \{0,1\}^{d \times d}$$

is the identity matrix, and 0_d is a $d \times d$ square matrix of zeros. In this thesis, we will assume that the flow map of a Hamiltonian system is symplectic, as Poincaré showed in [38, Chapter VI.2], given that no dissipation or other external forces are acting on the system. Definition 2.1 defines a symplectic map.

Definition 2.1 *A differentiable map $\varphi : \mathcal{X} \to \mathcal{X}$ is symplectic if*

$$\varphi'(x)^T \, \mathcal{J} \, \varphi(x) = \mathcal{J},$$

where $\varphi'(x)$ is the Jacobian of $\varphi(x)$.

Later on in this work where other methods are introduced for systems in which the Hamiltonian is not conserved, this assumption will still stand. However, the Hamiltonian approximations will be broken down into conservative and non-conservative parts.

2.1.3. Numerical Integration of Ordinary Differential Equations

As shown in the previous sections, the equations that predict the evolution of Hamiltonian systems in this thesis are frequently given in the form of ODEs. Once the governing equations are obtained, we need tools and methods to use those equations to approximate the evolution of the system over time, starting from a state x. This is called an Initial Value Problem (IVP), where the system state is approximated at different time steps, $t_0, t_1, \ldots, t_n$ starting from an initial state x_0 with a time step of $\Delta t = t_n - t_{n-1} = h$.

The core idea behind many numerical integration methods is to use discretization to approximate the derivatives, which are otherwise continuous. The Taylor series expansion of a function $y(t)$ is given by

$$\begin{aligned} y(t + \Delta t) &= y(t) + \Delta t \, y'(t) + \frac{1}{2}\Delta t^2 y''(t) + \frac{1}{3!}\Delta t^3 y'''(t) + \ldots \\ &= \sum_{k=0}^{\infty} \frac{(\Delta t)^k}{k!} y^{(k)}(t), \end{aligned}$$

where $y^{(k)}(t)$ denotes the k-th derivative of $y(t)$. If we only use the first term of this series, it becomes

$$y(t + \Delta t) = y(t) + \Delta t \, y'(t) + \frac{1}{2}\Delta t^2 y''(\tau)$$

where τ is some value between t and $t + \Delta t$. From this, we can approximate the derivative as

$$y'(t) = \frac{y(t + \Delta t) - y(t)}{\Delta t} + O(\Delta t).$$

Plugging this expression into the truncated Taylor series, we get

$$y(t + \Delta t) \approx y(t) + \Delta t \, y'(t).$$

This is called the Explicit Euler method. When applied to our Hamiltonian system, our update equations become

$$\begin{aligned} q_{n+1} &= q_n + \Delta t \, \nabla_p \mathcal{H}(q_n, p_n) \\ p_{n+1} &= p_n - \Delta t \, \nabla_q \mathcal{H}(q_n, p_n) \end{aligned} \tag{2.4}$$

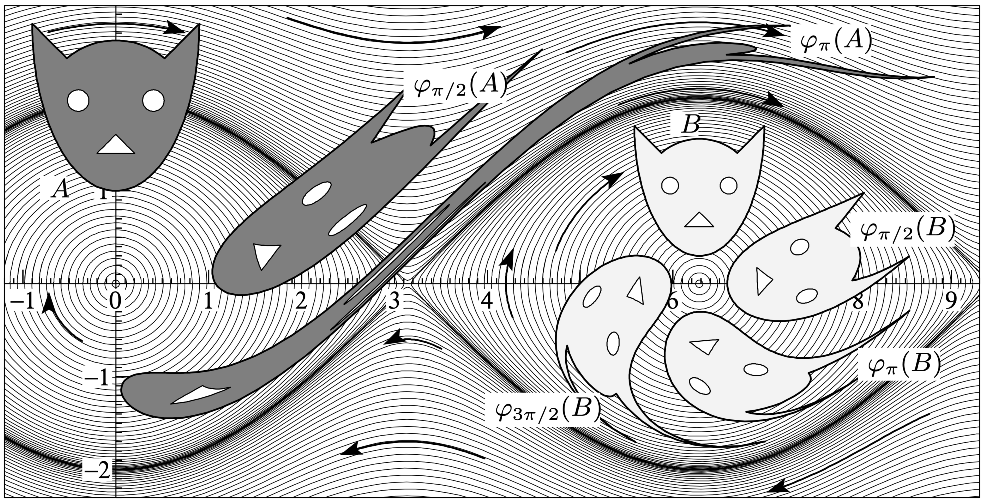


Figure 2.1.: Area preservation in phase space for the normalized pendulum problem through symplectic integration, taken from [39].

where q_n and p_n are the position and momentum at time t_n. This method is first order accurate, meaning that the error is proportional to Δt. Using the first n terms of the Taylor series rather than just the first term, we can get a more accurate approximation. However, this requires the function to be differentiable n times and increases the computational cost.

Although these Taylor Expansion based methods are simple to implement and can be applied to a wide range of systems, they are not suitable for Hamiltonian systems. This is because they are not symplectic and do not preserve the Hamiltonian when integrated. Symplectic integrators [80, 39, 27, 15], are volume preserving in the state space and preserve the Hamiltonian.

Figure 2.1 shows the level curves of the normalized pendulum problem in phase space and illustrates the symplectic integration of areas A and B on flow φ^t. Although the shapes look vastly different, the areas of A and $\varphi(A)$ and B and $\varphi(B)$ stay constant. The symplectic Euler method for Hamiltonian systems admits two semi-implicit integration schemes, depending on the order in which q and p are updated. The first variant, known as the q-first semi-implicit scheme is given by

$$\begin{aligned} q_{n+1} &= q_n + h \cdot \nabla_p \mathcal{H}(q_{n+1}, p_n), \\ p_{n+1} &= p_n - h \cdot \nabla_q \mathcal{H}(q_{n+1}, p_n). \end{aligned} \tag{2.5}$$

Alternatively, the p-first semi-implicit (symplectic Euler) scheme updates p before q as

$$\begin{aligned} p_{n+1} &= p_n - h \cdot \nabla_q \mathcal{H}(q_n, p_{n+1}), \\ q_{n+1} &= q_n + h \cdot \nabla_p \mathcal{H}(q_n, p_{n+1}). \end{aligned} \tag{2.6}$$

Both these methods are semi-implicit. However, for separable Hamiltonians, these equations can be solved explicitly, without the need for any implicit terms or fixed point iterations. In mechanical systems, the Hamiltonian is often separable since the potential energy is a function of the position and the kinetic energy is a function of the momentum. Equation (2.5) can be formulated as

$$\begin{aligned} q_{n+1} &= q_n + h \cdot \nabla_p \mathcal{H}(q_{n+1}, p_n) = q_n + h \cdot \nabla_p T(p_n), \\ p_{n+1} &= p_n - h \cdot \nabla_q \mathcal{H}(q_{n+1}, p_n) = p_n - h \cdot \nabla_q U(q_{n+1}). \end{aligned} \tag{2.7}$$

where T and U are the kinetic and potential energies respectively. In this case, $\nabla_p \mathcal{H}(q_{n+1}, p_n)$ and $\nabla_p \mathcal{H}(q_n, p_n)$ are equal if the Hamiltonian is separable, since the partial derivative with respect to p is independent of q.

In this thesis, other integration methods -both symplectic and non-symplectic- are tested and utilized. Some of these methods are higher order methods that use function evaluations at multiple locations to approximate the derivative, like symplectic Runge-Kutta methods [105], Störmer-Verlet method [26] and Implicit Midpoint method [85]. These symplectic methods have been studied for Hamiltonian systems [102, 38, 81, 92] and large scale problems [71, 104]. An important point to consider when choosing an integration method in our case is the fact that when training neural networks that use these integrators, the backpropagation will often need to flow through the integrator as well. For this reason, integrators that can be reformulated explicitly are preferred over implicit ones in certain cases.

2.1.4. Least Squares Method

In many practical problems, we are faced with situations where the number of measurements exceeds the number of unknowns, and an exact solution to the resulting system of equations does not exist. For example, when fitting a model to experimental data, we often have multiple noisy observations that cannot be perfectly described by any single parameter set. In such cases, rather than seeking an exact fit, the goal is to find the model parameters that approximate the data as closely as possible. In this thesis, we use the least squares method to solve for weights and biases of the last layer of the network, as will be explained in Section 3.1.

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, where $m > n$. Such systems are called overdetermined since there are more equations than unknowns. A vector $x^* \in \mathbb{R}^n$ that minimizes the residual error is defined as

$$||\vec{r}||^2 = ||A\vec{x} - \vec{b}||^2$$

where $||\cdot||$ is the Euclidean norm, is said to be the least squares solution to the system and can be defined as

$$x^* = \arg\min_{\vec{x}} ||\vec{r}||^2 = \arg\min_{\vec{x}} ||A\vec{x} - \vec{b}||^2.$$

This minimization problem can be solved using normal equations as

$$\begin{aligned}\|A\vec{x} - \vec{b}\|^2 &= (A\vec{x} - \vec{b})^T (A\vec{x} - \vec{b}) \\ &= \vec{x}^T A^T A\vec{x} - 2\vec{x}^T A^T \vec{b} + \vec{b}^T \vec{b}.\end{aligned}$$

If we take the gradient of the residual error with respect to x and set it to 0, we get the normal equations to be

$$\begin{aligned} &\vec{0} = \nabla \|A\vec{x} - \vec{b}\|^2 \big|_{\vec{x}=\vec{x}^*}, \\ &\vec{0} = 2A^T A\vec{x}^* - 2A^T \vec{b}, \\ \Rightarrow \quad &A^T \vec{b} = A^T A\vec{x}^*.\end{aligned}$$

Hence, a well-defined linear system is obtained. Although this method is simple to implement, the $A^T A$ terms can be ill-conditioned since

$$\begin{aligned}\kappa_{\text{rel}}(A^T A) &= \|A^T A\| \cdot \|(A^T A)^{-1}\| \\ &= \|A^T A\| \cdot \|A^{-1}(A^T)^{-1}\| \\ &\leq \|A^T\| \cdot \|A\| \cdot \|A^{-1}\| \cdot \|(A^T)^{-1}\| \\ &= \big(\kappa_{\text{rel}}(A)\big)^2\end{aligned}$$

where $\kappa_{rel}(A)$ denotes the relative condition number of A. In this thesis, in order to get around this condition number issue, least squares methods which use singular value decomposition (SVD) or QR decomposition are utilized. Specifically, the `lstsq` method from `numpy.linalg` [42] and the `scipy.linalg` [97] library are used. These two libraries both use LAPACK [2] routines that take advantage of the said SVD and QR methods.

2.1.5. Deep Feedforward Networks

Deep Feedforward Networks, also known as Multilayer Perceptrons (MLPs), are a class of neural networks that are composed of multiple layers of artificial neurons. They are called feedforward because the information flows in one direction, from the input layer to the output layer, without any feedback loops [33]. At its core, MLPs are used to approximate functions by learning the parameters of the network. For instance, in a regression problem similar to the problems seen in this thesis, the MLP maps a vector of inputs to a scalar output or a vector of outputs. We utilize these networks to parameterize the Hamiltonian calculation and to predict the time evolution of the system.

In this thesis, we follow the notation of Bolager et al. [10] and Rahma et al. [74]. Let $\Phi : \mathcal{X} \to \mathcal{Y}$ be a feedforward network with L hidden layers and N_L neurons per layer. The network is defined to be

$$\Phi^{(l)}(x) = \begin{cases} x, & \text{for } l = 0, \\ \sigma(W_l \Phi^{(l-1)}(x) - b_l), & \text{for } 0 < l \leq L, \\ W_l \Phi^{(l-1)}(x) - b_l, & \text{for } l = L+1. \end{cases}$$

where W_l and b_l are the weights and biases of the l-th layer with dimensions $N_l \times N_{l-1}$ and N_l respectively, and σ is the activation function. When denoted without the superscript, $\Phi(x)$ is the output of the network for input x, i.e. $\Phi(x) = \Phi^{(L+1)}(x)$.

The choice of activation function σ (e.g., ReLU, tanh, or GELU) [65] crucially influences the expressivity and optimization landscape of the network. Nonlinear activations enable MLPs to represent complex mappings, while the number of layers and neurons controls the model's capacity and generalization behavior.

The parameters W_l, b_l are optimized by minimizing a loss function $\mathcal{L}$ that measures the discrepancy between predicted and target outputs, typically using stochastic gradient descent (SGD) or one of its variants such as Adam [54]. Backpropagation is employed to compute the gradients of $\mathcal{L}$ with respect to each parameter efficiently through recursive application of the chain rule.

2.1.6. Helmholtz Decomposition

In a Hamiltonian system, the vector field $\mathbf{S}$ is the symplectic gradient of the Hamiltonian $\mathcal{H}$. This forces the vector field to be conservative and the Hamiltonian to stay constant. However, purely conservative systems are rare in practice as dissipative forces or external driving forces are often present on real world systems. In order to model such cases, a mathematical tool called the Helmholtz decomposition is used to split the vector field into separate conservative and dissipative components [9, 101, 87].

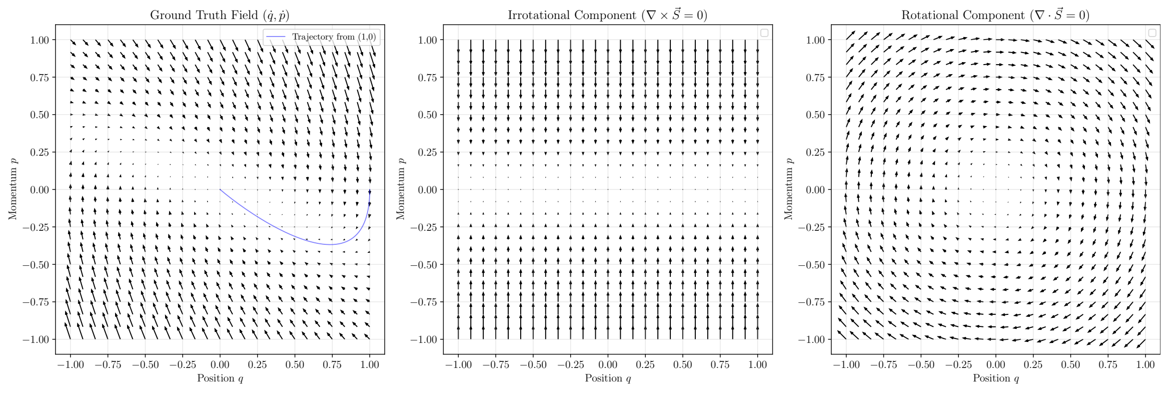


Figure 2.2.: Helmholtz decomposition of a 2D vector field. The original vector field is shown in the left panel. The irrotational component is shown in the middle panel. The rotational component is shown in the right panel.

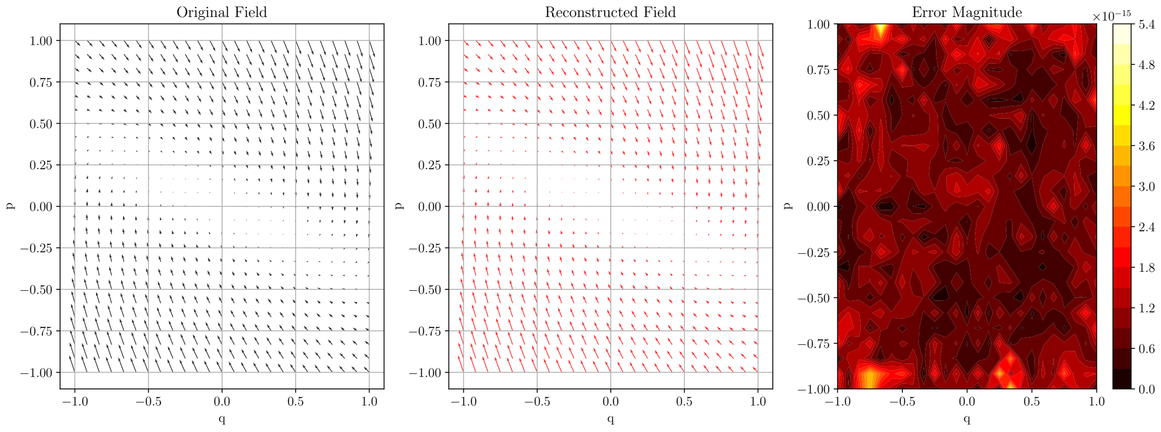


Figure 2.3.: Reconstruction from the Helmholtz decomposition for a 2D vector field. The original vector field is shown in the left panel. The reconstructed vector field is shown in the middle panel. The error is shown in the right panel.

The Helmholtz decomposition, named after Hermann von Helmholtz, states that a smooth vector field $\mathbf{S}$ can be decomposed into a fully rotational component $\mathbf{S}_{\text{rot}}$ and a fully irrotational component $\mathbf{S}_{\text{irrot}}$ [98]. This expression is given by

$$\mathbf{S} = \mathbf{S}_{\text{irrot}} + \mathbf{S}_{\text{rot}} = \nabla\phi + \nabla \times \mathbf{A}, \tag{2.8}$$

where ϕ is the scalar potential and $\mathbf{A}$ is the vector potential. In this thesis, the decomposition is calculated using the two-potential method described in Algorithm 1. First, the divergence and curl of the input vector field are computed using finite differences across the grid. Two Poisson equations are then solved: $\nabla^2\phi = \text{div}(\mathbf{S})$ for the scalar potential ϕ, and $\nabla^2\psi = \omega$ for the stream function ψ, where $\omega = \nabla \times \mathbf{S}$ is the vorticity. Both equations use a discrete Laplacian matrix with specified boundary conditions (typically periodic for smooth fields). Finally, the irrotational component is computed from the gradient of the scalar potential $\mathbf{S}_{\text{irrot}} = \nabla\phi$, and the rotational component is computed from the perpendicular gradient of the stream function $\mathbf{S}_{\text{rot}} = \nabla^{\perp}\psi = (-\partial\psi/\partial y, \partial\psi/\partial x)$.

This approach ensures both components are mathematically consistent and numerically exact within discretization errors. Algorithm 1 gives the steps to perform the Helmholtz decomposition. Figure 2.2 and Figure 2.3 show the results of the Helmholtz decomposition and the reconstruction from the decomposition, respectively.

Algorithm 1 Helmholtz Decomposition of 2D Vector Field

```
1: **Input:** Vector field $\mathbf{F} = \{(F_x(i,j), F_y(i,j)) \mid i \in [1,H], j \in [1,W]\}$, grid spacing
   $(\Delta x, \Delta y)$, boundary conditions BC
2: **Output:** Irrotational field $\mathbf{F}_{\text{irrot}}$, Rotational field $\mathbf{F}_{\text{rot}}$
3: **Step 1:** Compute divergence and curl fields
4: for $i = 1, 2, \ldots, H$ do
5:     for $j = 1, 2, \ldots, W$ do
6:         $\text{div}_{\mathbf{F}}(i,j) \leftarrow \frac{\partial F_x}{\partial x}\big|_{(i,j)} + \frac{\partial F_y}{\partial y}\big|_{(i,j)}$
7:         $\omega(i,j) \leftarrow \frac{\partial F_y}{\partial x}\big|_{(i,j)} - \frac{\partial F_x}{\partial y}\big|_{(i,j)}$
8:     end for
9: end for
10: **Step 2:** Solve Poisson equations for scalar potential and stream function
11: Construct discrete Laplacian matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ with BC
12: Solve linear system: $\mathbf{A}\boldsymbol{\phi} = \text{div}_{\mathbf{F}}$ for scalar potential $\boldsymbol{\phi}$
13: Solve linear system: $\mathbf{A}\boldsymbol{\psi} = \boldsymbol{\omega}$ for stream function $\boldsymbol{\psi}$
14: **Step 3:** Compute irrotational component from gradient of scalar potential
15: for $i = 1, 2, \ldots, H$ do
16:     for $j = 1, 2, \ldots, W$ do
17:         $\mathbf{F}_{\text{irrot}}(i,j) \leftarrow \nabla\phi(i,j) = \left(\frac{\partial\phi}{\partial x}\big|_{(i,j)}, \frac{\partial\phi}{\partial y}\big|_{(i,j)}\right)$
18:     end for
19: end for
20: **Step 4:** Compute rotational component from perpendicular gradient of stream func-
    tion
21: for $i = 1, 2, \ldots, H$ do
22:     for $j = 1, 2, \ldots, W$ do
23:         $\mathbf{F}_{\text{rot}}(i,j) \leftarrow \nabla^{\perp}\psi(i,j) = \left(-\frac{\partial\psi}{\partial y}\big|_{(i,j)}, \frac{\partial\psi}{\partial x}\big|_{(i,j)}\right)$
24:     end for
25: end for
26: return $\mathbf{F}_{\text{irrot}}, \mathbf{F}_{\text{rot}}$
```

2.2. Data Driven Learning of Hamiltonian Systems

2.2.1. Physics Informed Learning

Classical and numerical models when modeling dynamical systems have been the backbone of modern scientific and engineering simulations. They are researched extensively and are mostly well understood with uses in many fields from solid mechanics and structural analysis [110], computational fluid dynamics [29, 96] to quantum mechanics and more [16]. Partial differential equations (PDEs) are often solved using numerical methods like finite difference, finite volume or finite element methods. However, these methods usually have underlying assumptions about the physics and the geometry of the system and require explicit knowledge about the governing PDEs. For instance, Turbulence models in Reynolds-Averaged Navier–Stokes (RANS) or Large Eddy Simulation (LES) rely on closure relations (e.g., eddy viscosity models) that are hand-crafted from empirical or semi-theoretical formulations [70]. Another example is the Lattice Boltzmann method which relies on a fixed lattice topology and an assumed equilibrium distribution function derived from the Boltzmann equation, which must be specified beforehand [89].

Despite significant advances, modeling and predicting the dynamics of nonlinear multiscale systems characterized by complex interactions remain extremely challenging using traditional analytical or computational methods. These approaches often incur high computational costs and are susceptible to various sources of uncertainty, making them impractical for many real-world scenarios [52].

To address this limitation, data-driven learning approaches have been developed to model the behavior of dynamical systems directly from observational data through Gaussian Processes [66, 8, 6] and other machine learning architectures [49, 58, 76, 48]. These methods leverage machine learning architectures to approximate complex system dynamics without requiring explicit knowledge of the governing equations. However, black-box models like these often perform well only over short time horizons where they accurately reproduce local trajectories but tend to drift over long rollouts. This phenomenon leads to violations of energy conservation, symmetries or other physical invariants.

Rather than disregarding the rich mathematical structure that has been revealed through centuries of theoretical and experimental research, it is essential to incorporate this intrinsic knowledge into our models. Physics-informed learning addresses this by injecting inductive biases that mirror the governing laws either as soft constraints that penalize PDE/ODE residuals or hard constraints baked into architectures that preserve structure by construction. Such networks are called physics-informed neural networks (PINNs) [75]. By embedding physical constraints as inductive biases, these methods guide models toward solutions that respect known governing principles while learning from data. This is particularly useful in data-scarce problems typical of scientific applications. Karniadakis et al. [52] categorize the biases into the following three categories:

- **Observational bias**: Incorporating physical knowledge through training data that inherently captures the underlying physics or through strategically designed data augmentation techniques. When models are trained on these physically meaningful datasets, they naturally learn to approximate functions, vector fields and operators that preserve and reflect the physical structure present in the observations [57, 59].

- **Inductive bias**: Embedding physical principles directly into the neural network architecture through structural constraints that enforce compliance with fundamental laws. This approach modifies the model's parameterization to guarantee outputs that inherently satisfy physical constraints like energy conservation, symplecticity or certain boundary conditions. This deep integration of the physical laws into the architecture makes the predictions strictly satisfy them instead of softly guiding the solution. Examples include geometric deep learning approaches [14] and kernel flow methods [67].

- **Learning bias**: This approach tackles the problem from a different angle. Rather than changing or manipulating the model's architecture, the physical laws are enforced softly by penalizing the violations in the loss term. For instance, in an incompressible fluid flow problem, the divergence of the velocity field is ideally zero. In order to enforce this constraint, the divergence of the predicted velocity field can be used in the loss term [51]. A similar logic can be used to enforce monotonicity, conservation of mass, or momentum. Researchers have also used this approach in auto-regressive models, deep Galerkin methods, general adversarial networks and more [30, 109, 99].

2.2.2. Physics-informed Learning for Hamiltonian Systems

Two main directions are taken when applying these techniques to Hamiltonian systems. Researchers have either focused on learning the Hamiltonian directly or predicting the time evolution by learning the flow map of the system. Bertalan et al. [8] use a loss function that consists of weighted sums of different losses, $\mathcal{L} = \sum_{k=1}^{4} c_k f_k^2$ when modeling time dependent Hamiltonians where

$$f_1 = \hat{\dot{q}} - \dot{q} = \nabla_p \hat{\mathcal{H}} - \dot{q}, \qquad f_2 = \hat{\dot{p}} - \dot{p} = -\nabla_q \hat{\mathcal{H}} - \dot{p},$$

$$f_3 = \hat{\mathcal{H}}(x_0) - \mathcal{H}_0, \qquad f_4 = \frac{d\mathcal{H}}{dt} = \nabla_q \hat{\mathcal{H}} \dot{q} + \nabla_p \hat{\mathcal{H}} \dot{p},$$

where the values with $\widehat{(\cdot)}$ represent the predictions. Here, f_1 and f_2 represent the loss in predicted gradients using Hamilton's equations, f_3 is used to fix the integration constant for the Hamiltonian value and f_4 represents the time dependence of the system. It is worth noting that except for f_3, all loss components operate solely on the gradients of the output using automatic differentiation [5] and the network is evaluated based on the Hamiltonian outputs.

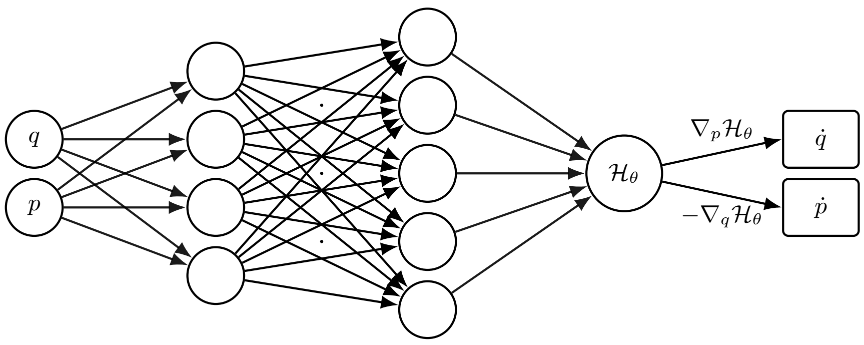


Figure 2.4.: An illustration of the HNN architecture that models the Hamiltonian and outputs time-derivatives according to Hamilton's equations.

Similarly, Greydanus et al. [35] have introduced Hamiltonian Neural Networks (HNN). In their approach, instead of predicting the $\mathbf{S}$ vector field, they learn a parametrized function for the Hamiltonian. During the forward pass, the network uses q and p coordinates to predict the scalar Hamiltonian (or a Hamiltonian-like) value. Following this, in-graph gradients of the scalar with respect to q and p are computed using automatic differentiation. Using Hamilton's equations as in Equation (2.1) and Equation (2.2), the time derivatives of q and p are approximated. Finally, an L^2 loss is computed between the predicted and true time derivatives as

$$\mathcal{L}_{HNN} = \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} - \frac{\partial \mathbf{q}}{\partial t} \right\|_2 + \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} + \frac{\partial \mathbf{p}}{\partial t} \right\|_2$$

where $\mathcal{H}_\theta$ represents the parametrized Hamiltonian. Figure 2.4 shows a schematic illustration of this architecture. An important property of this approach is that different from the previous example by Bertalan et al., this model learns an "energy-like" quantity rather than the Hamiltonian itself. This is a consequence of learning the gradients of the function only without approximating an integration constant.

The fact that the loss is computed using the gradients of the network enforces the satisfaction of the Hamiltonian equations. As a result, Greydanus et al. demonstrate that the HNN is able to conserve an energy-like quantity throughout its predictions whereas the baseline model drifts away in its predictions with time. In their analysis, they use the RK4 integrator to compute the dynamics. This architecture also allows one to "add or remove energy" by taking a step in the direction of the Riemann gradient $\mathbf{R}_\mathcal{H} = (\frac{\partial \mathcal{H}}{\partial \mathbf{q}}, \frac{\partial \mathcal{H}}{\partial \mathbf{p}})$, where this corresponds to increasing or decreasing the energy-like quantity of the system.

Similar to this formulation, researchers have also developed other architectures that use the Lagrangian of a system. Lagrangian Neural Networks (LNN) learn a system through its Lagrangian and this eliminates the need for the canonical momentum coordinates [23, 100]. Furthermore, in another work, researchers have applied a similar method on various graph neural network (GNN) architectures and integrators and found that constraining models to respect Hamiltonian structure yields improved generalization and accuracy [79]. In contrast, non-Hamiltonian models may perform well under specific training conditions but fail to generalize across integrators. Chen et al. [19] have included Hamiltonian preservation as a constraint in their Symplectic Recurrent Neural Networks (SRNNs) and reported that it improved the long term stability of the system. Separable HNN architecture by Khoo et al. [53] forces the separability of the Hamiltonian as an inductive bias into the network architecture.

Outside the physics domain, Hamiltonian formalism has also been applied to generative models. In this direction, Toth et al. [94] have introduced the Hamiltonian Generative Networks (HGN) where they learn the Hamiltonian dynamics from the pixel space of the images. They hypothesize that the images that depict the dynamics of a Hamiltonian system also lie in a manifold in the pixel space and the movement within this manifold follows the Hamiltonian dynamics. Combining this idea with General Adversarial Networks (GANs) [34], Hamiltonian GANs combine an HNN module with the GAN architecture to produce conservative dynamics [1].

2.2.3. Dissipative Hamiltonian Systems

Although the research discussed in Section 2.2.2 is highly effective for learning Hamiltonian systems, these methods are not directly suitable for dissipative systems. Real world systems are often of dissipative nature. The data is often subject to friction and external forces. These conditions violate the conservation of the Hamiltonian.

Desai et al. [25] demonstrate why damping in the system cannot be directly included in the Hamiltonian using a simple damped mass-spring system. This system has 2 force components, the spring force $F_{\text{spring}} = -kq$ and the damping force $F_{\text{damp}} = -\delta\dot{q}$ where k is the spring constant and δ is the damping coefficient. Using Newton's second law the equation of motion is given by

$$m\ddot{\mathbf{q}} = F = -k\mathbf{q} - \delta\dot{\mathbf{q}}$$

if we set $m = k = 1$, the equation becomes

$$\ddot{\mathbf{q}} = -\mathbf{q} - \delta\dot{\mathbf{q}}.$$

If we then integrate the right hand side of the equation with respect to $\mathbf{q}$ under the assumption that the potential energy is a function of $\mathbf{q}$, we get

$$\int -\mathbf{q} - \delta\dot{\mathbf{q}}\, d\mathbf{q} = -\frac{1}{2}\mathbf{q}^T\mathbf{q} - \delta\,\mathbf{q}^T\dot{\mathbf{q}} + c.$$

Using this as the potential energy, the Hamiltonian can be written when we sum with the kinetic energy as

$$\mathcal{H} = \frac{1}{2m}\mathbf{p}^T\mathbf{p} + \frac{1}{2}\mathbf{q}^T\mathbf{q} + \delta\,\mathbf{q}^T\dot{\mathbf{q}} + c.$$

When we apply Hamilton's equations to this Hamiltonian, we get

$$\frac{\partial \mathcal{H}}{\partial \mathbf{p}} = \dot{\mathbf{q}} + \delta\dot{\mathbf{q}} \neq \dot{\mathbf{q}},$$
$$-\frac{\partial \mathcal{H}}{\partial \mathbf{q}} = -\mathbf{q} - \delta\dot{\mathbf{q}} = \ddot{\mathbf{q}} = \dot{\mathbf{p}}.$$

for when $\dot{q} = p$ for $m = 1$. In this formulation, although $-\frac{\partial \mathcal{H}}{\partial q}$ recovers the correct term, $\frac{\partial \mathcal{H}}{\partial p}$ only holds when the damping coefficient is zero.

To address this issue, Sosanya et al. [87] have proposed the Dissipative Hamiltonian Neural Network (DHNN) structure where they extend the capabilities of the vanilla HNN. DHNN consists of two sub-networks, one yields the Hamiltonian (conservative) function as before and the other yields a Rayleighian dissipative function $\mathcal{D}$ as two scalar outputs. The Rayleighian dissipative function is frequently used to model velocity dependent forces in Hamiltonian mechanics [22, 61]. A metriplectic system is a mathematically consistent combination of Hamiltonian (symplectic) and dissipative (metric) dynamics.

In the DHNN, the gradients of the scalar outputs are combined to yield the time derivatives of the system as

$$\dot{\mathbf{q}} = \frac{\partial \mathcal{D}}{\partial \mathbf{q}} + \frac{\partial \mathcal{H}}{\partial \mathbf{p}},$$
$$\dot{\mathbf{p}} = \frac{\partial \mathcal{D}}{\partial \mathbf{p}} - \frac{\partial \mathcal{H}}{\partial \mathbf{q}}.$$

This approach makes use of the Helmhotz decomposition (Section 2.1.6) to separate the conservative and dissipative components. The Hamiltonian vector field is rotational w.r.t the input space and the dissipative vector field is irrotational. The rotational part conserves the Hamiltonian while the irrotational part models energy dissipation (or addition). Figure 2.5 shows a schematic illustration of the DHNN architecture. During training, the aim is to learn this decomposition implicitly through the loss function

$$L_{DHNN} = \left\| \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}} + \frac{\partial \mathcal{D}}{\partial \mathbf{q}} \right) - \frac{\partial \mathbf{q}}{\partial t} \right\|_2 + \left\| \left(-\frac{\partial \mathcal{H}}{\partial \mathbf{q}} + \frac{\partial \mathcal{D}}{\partial \mathbf{p}} \right) - \frac{\partial \mathbf{p}}{\partial t} \right\|_2. \tag{2.9}$$

In the original HNN paper, the authors used a forward Euler method to integrate the Hamiltonian, as discussed in Section 2.1.3. Further research by David et al. [24] has shown that this method introduced an artificial lower bound to the loss they experimented with using symplectic Euler and midpoint integration. They introduced Symplectic Hamiltonian Neural Networks (SHNNs). The midpoint rule is given to be

$$x_{n+1} = x_n + hJ^{-1}\nabla\mathcal{H}\left(\frac{x_n + x_{n+1}}{2}\right). \tag{2.10}$$

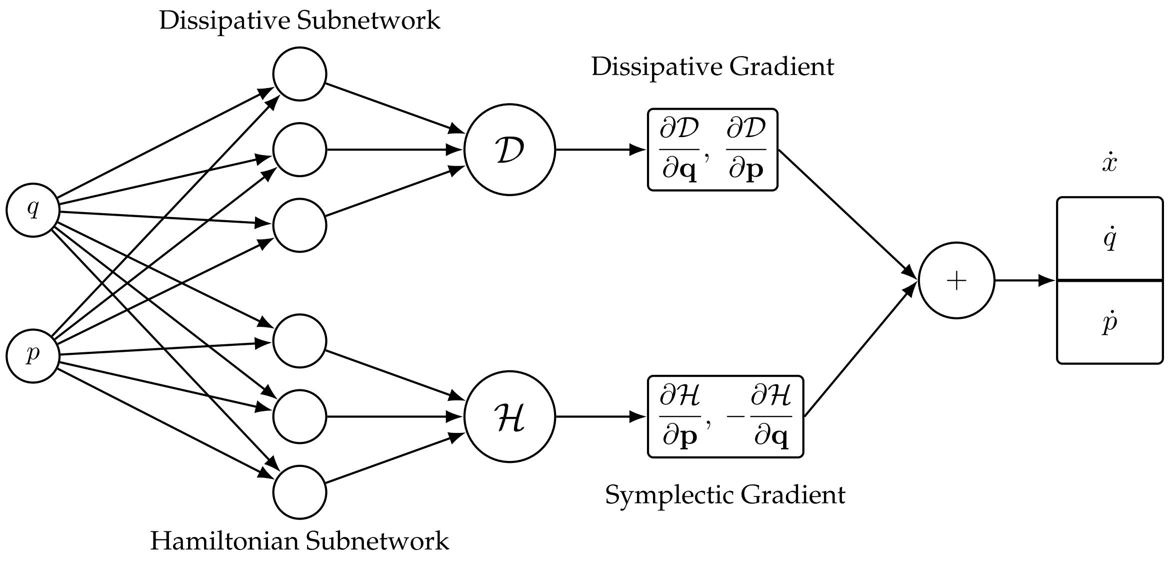

Dissipative Subnetwork
Dissipative Gradient
$\mathcal{D}$
$\frac{\partial \mathcal{D}}{\partial \mathbf{q}}, \frac{\partial \mathcal{D}}{\partial \mathbf{p}}$
q
p
$+$
$\dot{x}$
$\dot{q}$
$\dot{p}$
$\mathcal{H}$
$\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}$
Symplectic Gradient
Hamiltonian Subnetwork

Figure 2.5.: Illustration of DHNN, described in [87].

The loss function used in SHNN is defined as

$$L_{SHNN} = \left\| \frac{x_{n+1} - x_n}{h} - J^{-1} \nabla \mathcal{H}(s(x_n, x_{n+1})) \right\|_{L^2}^2 \tag{2.11}$$

where $s(x_0, x_1)$ corresponds to a symplectic integration scheme.

Zhu et al. [108] have performed a numerical error analysis on multiple parts of the network and illustrated the importance of using symplectic integrators. Theoretical analysis demonstrates that HNN variants utilizing symplectic integrators can guarantee to have well-defined network targets, whereas non-symplectic integrators can not. They also mathematically show that the expected network error is broken down into different sources in Figure 2.6 may not exist for non-symplectic integrators since not every vector-valued function is the gradient of another scalar function, causing the network target to not exist in the first place.

It is important to note that in the direct numerical integration of ODEs, implicit schemes such as the symplectic Euler or midpoint methods typically require iterative fixed-point solvers at each timestep which results in increased computational cost compared to explicit methods. However, during the training phase of HNNs, since the true trajectory pairs (x_n, x_{n+1}) can be made available, there is no need to solve for unknown future states at each step. Consequently, training HNNs with implicit symplectic methods incurs no additional computational overhead compared to explicit methods. However, it is important to note that during inference, the network must use the same integrator as was used during training. In particular, if an implicit integrator was used during training, then during the forward pass at inference time, it is necessary to solve an implicit equation involving

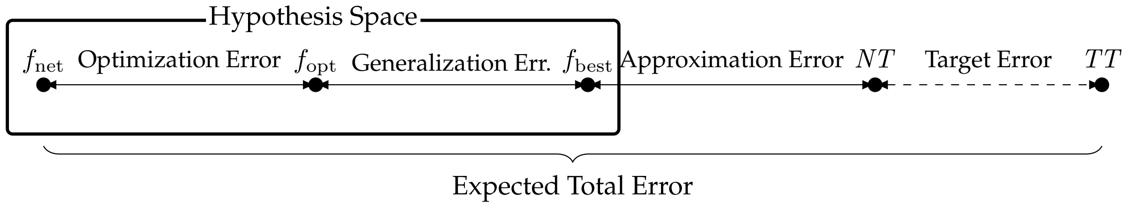


Figure 2.6.: Illustration of sources of error in learning Hamiltonian systems with neural networks as described by Zhu et al. [108]. f_{net} is the network function, f_{opt} is the optimal function with minimum loss, f_{best} is the closest function to the network target, NT is the network target and TT is the true target.

the neural network. This can require fixed-point iteration or root finding at each timestep, especially when modeling non-separable systems. For separable systems, some implicit methods can reduce to explicit schemes and thus do not require this additional solve.

Further research has been done on incorporating thermodynamics into the models where thermodynamics is considered to be a theory of theories [36]. An important example of this inclusion is Structure Preserving Neural Networks (SPNNs) [45] which are a type of NN that is trained to satisfy the 1st and 2nd laws of thermodynamics. These networks are based on the General Equation for the Non-Equilibrium Reversible-Irreversible Coupling (GENERIC) formalism [37]. GENERIC has been commonly used in dissipative systems and integration schemes [32].

This GENERIC formalism employs two operators and two potentials. The two operators are L and M which are bundles of vector fields and the two potentials are the energy potential $E(x)$ and the entropy potential $S(x)$. Similar to the DHNN architecture, this formalism also utilizes the energy field to model the conservative part of the dynamics and the entropy field to model the dissipative part. The time evolution of the system is given by

$$\frac{dx}{dt} = L\,\frac{\partial E}{\partial x} + M\,\frac{\partial S}{\partial x}. \tag{2.12}$$

In the conservative (reversible) part, the L operator is a Poisson matrix and is skew-symmetric. In the dissipative (irreversible) component, the operator M is the positive semi-definite and symmetric friction matrix to facilitate the energy dissipation. Furthermore, two degeneracy conditions are imposed in the formulation as

$$L\,\frac{\partial S}{\partial x} = M\,\frac{\partial E}{\partial x} = 0, \tag{2.13}$$

to induce the satisfaction of the first and second laws of thermodynamics. These conditions show that energy potential does not affect the entropy generation of the system and vice versa. The GENERIC formalism [37] shows this property by plugging the conditions into the equation as

$$\frac{\partial E}{\partial t} = \frac{\partial E}{\partial x}\frac{dx}{dt} = \frac{\partial E}{\partial x}\Big(L\,\frac{\partial E}{\partial x} + M\,\frac{\partial S}{\partial x}\Big) = 0, \tag{2.14}$$

which shows the conservation of energy in the system and

$$\frac{\partial S}{\partial t} = \frac{\partial S}{\partial x}\frac{dx}{dt} = \frac{\partial S}{\partial x}\Big(L\,\frac{\partial E}{\partial x} + M\,\frac{\partial S}{\partial x}\Big) = \frac{\partial S}{\partial x}\,M\,\frac{\partial S}{\partial x} \geq 0, \tag{2.15}$$

which shows the entropy generation. The L and M matrices of the system are already known. Using these matrices, SPNNs integration algorithm outputs the A and B matrices that approximate the discrete gradients of the energy and entropy potentials as

$$\frac{\Delta E}{\Delta x} \simeq A\,x, \qquad \frac{\Delta S}{\Delta x} \simeq B\,x.$$

Building on this formalism, Port-Metriplectic Neural Networks [43] were introduced. These networks add boundary terms for the energy and entropy potentials to model the interactions between subsystems. They illustrate this with a viscoelastic double pendulum system where each pendulum is modeled as a subsystem that exchanges energy and entropy with the surrounding systems.

Researchers have also applied the GENERIC formalism and SPNNs to graph neural networks [93, 44]. In these models, they impose inductive biases on both the geometry and the thermodynamics of the system. The architecture can be modified to apply these principles either locally on the node level or globally on the graph level. The global imposition is a more strict constraint than the local one. However, it requires the assembly of the global Poisson and dissipation matrices, which breaks the local structure of the graph networks and increases the computational cost. Local imposition preserves the node-by-node structure and can be more efficient. Aside from accurately modeling the system, these models can also provide information about the underlying physics of the system, as they output the L and M matrices, mentioned in the SPNN example.

Further research has been done on mesh based GNNs [69] where the simulation state is encoded as a graph. They use both mesh-space and world-space to do their calculations. Mesh space can model differential operators related to the underlying physics while world-space handles the external dynamics like collisions and contact. These models can run 1-2 orders of magnitude faster than the simulation on which it is trained and can support adaptive remeshing of the geometry.

Another research direction is taken on learning the Hamiltonian flow by using neural networks to approximate the right-hand-side of the ODE where the loss function includes the error in observed trajectory and predicted trajectory. This architecture is called Neural ODE (NODE) [18]. Building on this architecture, Symplectic ODE Nets (SymODEN) [107] were proposed in the control setting where external forces on the system are also considered. This proposal was followed by Dissipative SymODEN where the energy dissipation is also considered [106]. These additions allowed these models to be applied into different areas in robotics and control [28].

These models are frequently combined with the port-Hamiltonian [95, 60, 20] formalism to model dissipative or time-dependent systems with driving forces. This formalism explicitly models the total energy, dissipation and external forces on the system. The framework can be defined as

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = \left(\begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} - \mathbf{D}(\mathbf{q}) \right) \begin{bmatrix} \dfrac{\partial H}{\partial \mathbf{q}} \\ \dfrac{\partial H}{\partial \mathbf{p}} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{g}(\mathbf{q}) \end{bmatrix} \mathbf{u}, \tag{2.16}$$

where $\mathbf{u}$ represents the external forces, $\mathbf{D}(\mathbf{q})$ represents the dissipation matrix and $\mathbf{g}(\mathbf{q})$ is the input matrix, using the notation of Zhong et al. [106]. It can be seen that when the dissipation and input matrices are zero, the framework yields the original Hamiltonian formalism.

Port-Hamiltonian Neural Networks (port-HNNs) [25] implement this framework for dissipative and time-dependent systems. They replace the dissipation matrix $\mathbf{D}(\mathbf{q})$ with a constant damping term at the lower right quadrant, $\mathbf{N}$. They also replace the input matrix $\mathbf{g}(\mathbf{q})$ with a time-dependent force field $\mathbf{F}(t)$. These substitutions yield

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = \left(\begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & N \end{bmatrix} \right) \begin{bmatrix} \dfrac{\partial \mathcal{H}}{\partial \mathbf{q}} \\ \dfrac{\partial \mathcal{H}}{\partial \mathbf{p}} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{F}(t) \end{bmatrix}. \tag{2.17}$$

They use 3 separate networks. The first network is a regular HNN that models the conservative dynamics, the second network is a scalar network that models the damping term $\mathbf{N}$ and the third network takes the time, t, as input and models the force field $\mathbf{F}(t)$.

2.3. Sampling Networks

The main focus of research in the ML area has been on traditional iterative optimization algorithms such as stochastic gradient descent (SGD) [11], Adam [55] and RMSProp [46]. These methods can face various challenges like sensitivity to initialization and hyperparameters [91], poor conditioning [77], gradient issues during backpropagation [7] or slow convergence [50]. Instead of this iterative non-convex optimization, sampling all the hidden layer weights and biases and using a linear learned layer for the output is studied. This sampling can be random or according to some distribution [90]. Such networks are known as Random Feature Models (RFMs). Single layer RFMs such as shallow neural networks (SNNs) [84] and Extreme Learning Machines (ELMs) [47] share the idea of transforming inputs through a randomly selected nonlinear mapping and training only a simple linear output layer.

2.3.1. Sampling Where It Matters (SWIM)

Random Feature Models compared to gradient based optimization come with trade-offs. The methods are often data-agnostic and require the selection of a probability distribution which manifests as many hyperparameters that need to be tuned. Another disadvantage is that these methods usually require a large number of neurons to match the traditionally trained networks. Bolager et al. [10] introduced the Sampling Where It Matters (SWIM) as a data-driven neural network sampling algorithm. This approach significantly reduces the number of hyperparameters that require manual tuning such as the learning rate, number of epochs, batch size or choice of optimizer and makes the training process more efficient as the same hyperparameters can accommodate a broader range of target functions and network architectures. Furthermore, the hyperparameters are fixed for a given activation function in this method. In their paper, they prove that SWIM sampled networks are able to approximate large family continuous functions and are universal approximators. The heuristic for the weights and biases is to make them close to the gradients of the target function. Definition 2.2 gives the definition of a SWIM sampled network.

Definition 2.2 (SWIM Sampled Neural Network) *Let Φ be a feed-forward neural network. For $l = 1, \dots, L$, let $\{(x_{0,i}^{(1)}, x_{0,i}^{(2)})\}_{i=1}^{N_l} \subset \mathcal{X} \times \mathcal{X}$. If for each layer l and neuron i the weights and biases are*

$$w_{l,i} = s_1 \frac{x_{l-1,i}^{(2)} - x_{l-1,i}^{(1)}}{\|x_{l-1,i}^{(2)} - x_{l-1,i}^{(1)}\|^2}, \quad b_{l,i} = \langle w_{l,i},\, x_{l-1,i}^{(1)} \rangle + s_2, \tag{2.13}$$

*then Φ is called a **SWIM sampled network**, where $\|\cdot\|$ is the L^2-norm and $\langle \cdot, \cdot \rangle$ the inner product and $s_1, s_2 \in \mathbb{R}$ are constants that are used to place the activation functions into the correct range.*

For the ReLU activation, we choose $s_1 = 1$ and $s_2 = 0$, which positions the activation such that $\varphi(x^{(1)}) = 0$ and $\varphi(x^{(2)}) = 1$. In the case of the $\tanh$ activation, setting $s_1 = 2s_2$ and $s_2 = \ln(3)/2$ results in $\varphi(x^{(1)}) = 1/2$, $\varphi(x^{(2)}) = -1/2$ and $\varphi\left(\frac{1}{2}\left(x^{(1)} + x^{(2)}\right)\right) = 0$. In a regression problem with ReLU, the activation function linearly interpolates between the two points. For classification, tanh introduces a boundary if $x^{(1)}$ belongs to a different class than $x^{(2)}$. Figure 2.7 illustrates the placement of the point pairs and the activation functions. All weights up to the last linear layer are sampled so that $x_{l-1,i}^{(k)} = \Phi^{(l-1)}(x_{0,i}^{(k)})$ for $k \in \{1, 2\}$. The last layer is then solved to be

$$(W_{L+1}, b_{L+1}) = \underset{W_{L+1},\, b_{L+1}}{\arg\min}\ \mathcal{L}(W_{L+1}\Phi^{(L)}(\cdot) - b_{L+1}), \tag{2.18}$$

using the least squares method as mentioned in Section 2.1.4.

The sampling of the data points can be done uniformly or by following a heuristic that favors points at large gradients. This is done by assigning a probability density to the data points that takes into account the value of target function at that data point. This probability density is given as

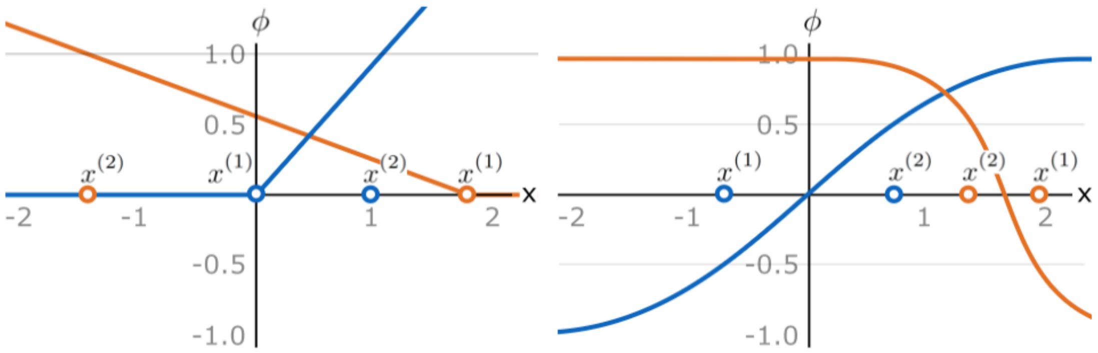


Figure 2.7.: Placement of point pairs $x^{(1)}$, $x^{(2)}$ for activation functions ReLU and $\tanh$ respectively, taken from [10].

$$p_l^{\varepsilon}\left(x_0^{(1)}, x_0^{(2)} \mid \{W_j, b_j\}_{j=1}^{l-1}\right) = \frac{\|f_l(x_{l-1}^{(2)}) - f_l(x_{l-1}^{(1)})\|}{\max\left(\|x_{l-1}^{(2)} - x_{l-1}^{(1)}\|,\, \epsilon\right)}, \tag{2.19}$$

in a supervised learning setting.

Part III.

Methods and Numerical Experiments

3. Methods and Results

In this chapter we introduce our framework for learning dissipative Hamiltonian systems by sampling networks. In Section 3.1, we explain our methods to construct the sampled networks to model the Hamiltonian and dissipative terms of the system. We aim to approximate Hamiltonian functions using sampled networks by using subnetworks and physical priors. Section 3.2 contains our numerical experiments and results.

3.1. Framework for Learning Dissipative Hamiltonian Systems by Sampling Networks

We propose a framework for learning Hamiltonian systems by sampling networks by embedding the dissipation term into our model architecture. Our model is based on the Random HNN by Rahma et al. [73, 74]. Similar to the original HNN paper, the inputs of the Random HNN are the position and momentum of the system, $x = (q, p)$ and the output is the Hamiltonian, $\mathcal{H}(x)$. The Hamiltonian is approximated by the neural network as

$$\widehat{\mathcal{H}}(x) = \Phi(x) = W_{L+1}\Phi^{(L)}(x) - b_{L+1} \approx \mathcal{H}(x). \tag{3.1}$$

Differentiating Equation (3.1) with respect to the input state x and exploiting the linearity of the gradient operator, we obtain

$$\begin{aligned}\nabla\widehat{\mathcal{H}}(x) &= \nabla\left(W_{L+1}\Phi^{(L)}(x) - b_{L+1}\right)\\ &= W_{L+1}\nabla\Phi^{(L)}(x).\end{aligned} \tag{3.2}$$

To determine the readout weights W_{L+1}, we enforce consistency between the network's induced vector field and the ground truth gradients derived from the training dataset $\mathcal{D} = \{(x_i, \dot{x}_i)\}_{i=1}^{K}$. By evaluating the network gradients and the target values (using $\nabla\mathcal{H} = \mathcal{J}^{-1}\dot{x}$) across all training samples, we construct the linear system

$$\begin{bmatrix}\nabla\widehat{\mathcal{H}}(x_1)\\ \vdots\\ \nabla\widehat{\mathcal{H}}(x_K)\\ \widehat{\mathcal{H}}(x_0)\end{bmatrix} = \begin{bmatrix}\nabla\Phi^{(L)}(x_1)\\ \vdots\\ \nabla\Phi^{(L)}(x_K)\\ \Phi^{(L)}(x_0)\end{bmatrix} \cdot W_{L+1}^{T} \overset{!}{=} \begin{bmatrix}\mathcal{J}^{-1}\dot{x}_1\\ \vdots\\ \mathcal{J}^{-1}\dot{x}_K\\ \mathcal{H}(x_0)\end{bmatrix}. \tag{3.3}$$

Specifically, each gradient block $\nabla\Phi^{(L)}(x_i)$ in the feature matrix is constructed by stacking the partial derivatives with respect to the position coordinates q and momentum coordinates p as

$$\left[\nabla\Phi^{(L)}(x_i)\right] = \begin{bmatrix}\nabla_q\Phi^{(L)}(x_i)\\ \nabla_p\Phi^{(L)}(x_i)\end{bmatrix} = \underbrace{\begin{bmatrix}\nabla_{q_1}\Phi^{(L)}(x_i)\\ \vdots \\ \nabla_{q_d}\Phi^{(L)}(x_i)\\ \nabla_{p_1}\Phi^{(L)}(x_i)\\ \vdots \\ \nabla_{p_d}\Phi^{(L)}(x_i)\end{bmatrix}}_{\in\mathbb{R}^{2d\times N_L}}, \tag{3.4}$$

where N_L denotes the width of the final hidden layer. This formulation allows us to solve for the optimal weights W_{L+1} directly via linear least squares. The bias term b_{L+1} is set to fix the integration constant by adding the final row in the linear system, assuming that the true Hamiltonian is known at the initial condition x_0. The whole linear system is then given by

$$\underbrace{\begin{bmatrix}\nabla\Phi^{(L)}(x_1) & 0\\ \vdots & \vdots\\ \nabla\Phi^{(L)}(x_K) & 0\\ \Phi^{(L)}(x_0) & 1\end{bmatrix}}_{A\in\mathbb{R}^{(2dK+1)\times(N_L+1)}} \cdot \underbrace{\begin{bmatrix}W_{L+1}^T\\ b_{L+1}\end{bmatrix}}_{w\in\mathbb{R}^{(N_L+1)}} \overset{!}{=} \underbrace{\begin{bmatrix}\mathcal{J}^{-1}\dot{x}_1\\ \vdots\\ \mathcal{J}^{-1}\dot{x}_K\\ \mathcal{H}(x_0)\end{bmatrix}}_{u\in\mathbb{R}^{(2dK+1)}}. \tag{3.5}$$

We extend this framework with Random Dissipative Hamiltonian Neural Networks (RDHNN) by the addition of a dissipative subnetwork to the model architecture similar to the DHNN by Sosanya et al. [87]. We denote the conservative subnetwork activations as $\phi(x)$ and the dissipative subnetwork activations as $\psi(x)$. The network has two separate heads, one for the Hamiltonian and one for the dissipative term that are connected to separate hidden layers. The $\phi(x)$ and $\psi(x)$ are passed through linear layers to yield the Hamiltonian and dissipation of the system. The symplectic gradient of the Hamiltonian and the dissipative gradient of the dissipative prediction are then combined to yield the time derivative of the system. Figure 3.1 shows a schematic illustration of this architecture.

Here, the system matrix $\mathbf{A}$ and the target vector $\mathbf{y}$ are constructed to embed the symplectic structure of the Hamiltonian dynamics as

$$\mathbf{A} = \begin{bmatrix}\nabla_p\phi(x) & \nabla_q\psi(x) & 0\\ -\nabla_q\phi(x) & \nabla_p\psi(x) & 0\\ \phi(x_0) & 0 & 1\end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix}\dot{q}\\ \dot{p}\\ H(x_0)\end{bmatrix} \tag{3.6}$$

In this formulation, $\phi(x)$ and $\psi(x)$ represent the randomized basis functions for the Hamiltonian and Dissipative terms. The last row of matrix $\mathbf{A}$ explicitly incorporates the initial energy value $H(x_0)$ to fix the integration constant. Consequently, this method not only approximates the vector field $\dot{x} = [\dot{q}, \dot{p}]^\top$ but also implicitly learns the scalar energy

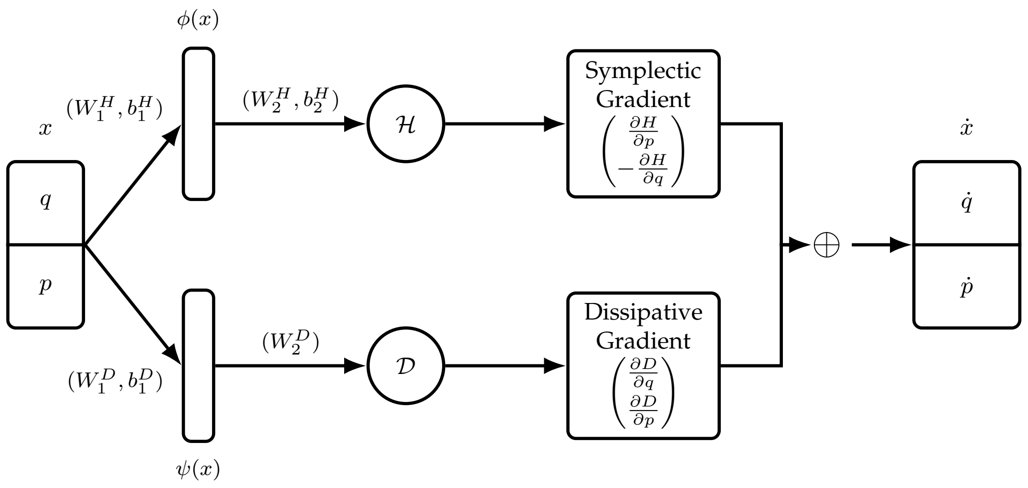

$\phi(x)$
x
(W_1^H, b_1^H)
(W_2^H, b_2^H)
$\mathcal{H}$
Symplectic Gradient
$\begin{pmatrix} \frac{\partial H}{\partial p} \\ -\frac{\partial H}{\partial q} \end{pmatrix}$
$\dot{x}$
q
p
$\oplus$
$\dot{q}$
$\dot{p}$
(W_2^D)
(W_1^D, b_1^D)
$\mathcal{D}$
Dissipative Gradient
$\begin{pmatrix} \frac{\partial D}{\partial q} \\ \frac{\partial D}{\partial p} \end{pmatrix}$
$\psi(x)$

Figure 3.1.: Diagram of the RDHNN architecture. The input $x = (q, p)$ is passed through the conservative and dissipative subnets to yield the Hamiltonian and dissipation of the system. The symplectic gradient of the Hamiltonian and the dissipative gradient of the dissipative prediction are then combined to yield the time derivative of the system. The parameters $W_1^H, b_1^H, W_1^D, b_1^D$ are sampled using the SWIM algorithm and the parameters W_2^H, b_2^H, W_2^D are determined by solving the linear system.

function $H(x)$ and the dissipation function $D(x)$. Algorithm 2 shows the algorithm for training the RDHNN model using the SWIM algorithm.

Algorithm 2 Algorithm for training RDHNNs using SWIM sampling. The hidden layer parameters are sampled using the SWIM algorithm while the last layer parameters are determined by solving the linear system based on the gradients.

```
1: Data: $\mathcal{D} = \{(x_k, \dot{x}_k) \mid k = 1, \dots, K\}$
2: $\Phi^0(X) = X$
   Step 1: SWIM Sampling of Hidden Layer Parameters for both subnetworks.
3: for $l = 1, \dots, L$ do
4:     $W_L^H, b_l^H \leftarrow$ sampler(SWIM)
5:     $W_l^D, b_l^D \leftarrow$ sampler(SWIM)
6: end for
   Step 2: Construct the linear system
7: Initialize $A \in \mathbb{R}^{(2dK+1)\times(N_L+1)}$ and $u \in \mathbb{R}^{(2dK+1)}$
8: Compute $\nabla\phi^{(L)}(x_k), \nabla\psi^{(L)}(x_k) \in \mathbb{R}^{2d\times N_L}$ via Autodiff
9: Calculate target $u = \mathcal{J}^{-1}\dot{x}_k = (-\dot{p}_k, \dot{q}_k)^T$
   Step 3: Add the final row to A and u to fix the integration constant
10: $A \leftarrow \begin{bmatrix} & A & \\ \phi(x_0) & 0 & 1 \end{bmatrix}$
11: $u \leftarrow \begin{bmatrix} u \\ \mathcal{H}(x_0) \end{bmatrix}$
   Step 4: Solve for final layer weights
12: $W_2^H, b_2^H, W_2^D \leftarrow \mathrm{argmin}_w \|Aw - u\|_2^2$
13: return $\Phi$
```

Having two separate heads for the Hamiltonian and dissipation is more flexible and allows the hidden layers to be optimized independently when trained with an iterative optimizer. However, since the uniform SWIM algorithm samples parameters in an unsupervised manner, the performance is expected to be theoretically equivalent to having a shared hidden layer. In order to test this, an RDHNN with a shared hidden layer (SharedRDHNN) is also trained and compared to the RDHNN with separate hidden layers (RDHNN). Figure 3.2 shows a schematic illustration of the SharedRDHNN architecture.

The results have shown that the SharedRDHNN achieves equivalent performance to the RDHNN with separate hidden layers when trained with the SWIM algorithm although it has half the number of hidden layer parameters.

The fact that both RDHNN and SharedRDHNN predict the conservative and dissipative components allows us to generalize the dissipation in the system by scaling the dissipative component by a factor of α while keeping the conservative component unchanged.

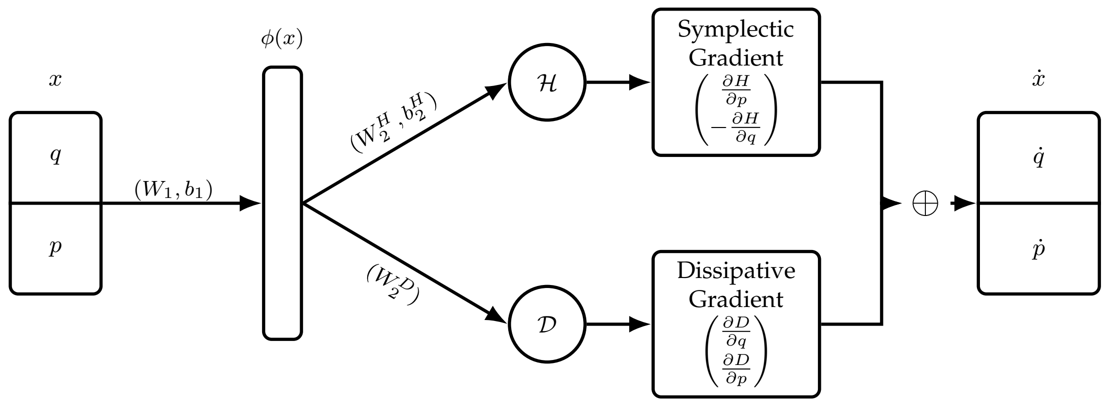

x
q
p
(W_1, b_1)
$\phi(x)$
(W_2^H, b_2^H)
(W_2^D)
$\mathcal{H}$
$\mathcal{D}$
Symplectic Gradient
$\begin{pmatrix} \frac{\partial H}{\partial p} \\ -\frac{\partial H}{\partial q} \end{pmatrix}$
Dissipative Gradient
$\begin{pmatrix} \frac{\partial D}{\partial q} \\ \frac{\partial D}{\partial p} \end{pmatrix}$
$\oplus$
$\dot{x}$
$\dot{q}$
$\dot{p}$

Figure 3.2.: Schematic illustration of the SharedRDHNN architecture. The input $x = (q, p)$ is passed through the shared hidden layer to yield the Hamiltonian and dissipation of the system. The symplectic gradient of the Hamiltonian and the dissipative gradient of the dissipative prediction are then combined to yield the time derivative of the system. The parameters W_1, b_1 are sampled using the SWIM algorithm and the parameters W_2^H, b_2^H, W_2^D are determined by solving the linear system.

Through scaling, we can predict the dynamics of the system in higher or lower dissipation environments.

As our baseline model, we use the Random HNN model only in the conservative cases since it forces the Hamiltonian to be conserved. Due to this, in the dissipative cases, we use the Random ODE-Net model to approximate the dynamics. Unlike the physics-informed models, the Random ODE-Net model does not learn the Hamiltonian and dissipation terms explicitly, nor does it incorporate the in-graph gradients of the Hamiltonian and dissipation terms. Instead, it learns the dynamics of the system directly. Its output is the time derivative of the system, $\dot{x} = [\dot{q}, \dot{p}]^\top$.

3.2. Numerical Experiments

All experiments shown in this section were conducted on the `x-wing0` compute node of the HomeOne cluster at the Technical University of Munich unless otherwise specified. The system is equipped with an AMD EPYC 7402 CPU (2.80 GHz, 24 cores) and 256 GiB of RAM. Training was performed using a single NVIDIA GeForce RTX 3080 GPU with 10 GiB of VRAM. The models were implemented in PyTorch with CUDA acceleration enabled. The experiments ran on a single compute node without any distributed training.

Due to the randomness of the sampling algorithms, the models were trained 100 times and the median of the results was taken to get the most representative results. When evaluating the models, we report the test errors, training times and speedup factors along with the trajectory errors for the models when integrated over several thousand timesteps using symplectic integrators. We also utilize the relative error of the models that is defined as the ratio of the error to the ground truth norm. In all dissipative experiments, we integrate the system until it reaches a steady state unless otherwise specified when plotting the trajectory errors. An 80/20 split was used for the training and testing datasets in both SWIM and Adam trained models. All models in this work utilize the Softplus activation function for the hidden layers with ReLU initialization.

3.2.1. Single Pendulum

In order to test our models with a simple non-linear system, we can consider the single pendulum system with and without damping. The system is defined with a single degree of freedom, the angle θ of the pendulum. The Hamiltonian is defined as

$$\mathcal{H}(x) = mgl(1 - \cos(q)) + \frac{p^2}{2ml^2} \tag{3.7}$$

with the partial derivatives

$$\nabla_q \mathcal{H}(x) = mgl \sin(q), \nabla_p \mathcal{H}(x) = \frac{p}{ml^2}, \tag{3.8}$$

where m is the mass of the pendulum, l is the length of the pendulum, g is the acceleration due to gravity, q is the angle of the pendulum and p is the angular momentum.

For simplicity, we set $m = l = g = 1$ and aim to approximate the Hamiltonian

$$\mathcal{H}(x) = (1 - \cos(q)) + \frac{p^2}{2}, \tag{3.9}$$

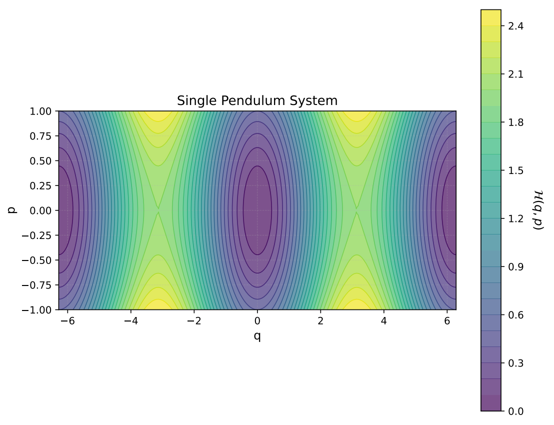
for domains $[-2\pi, 2\pi] \times [-1, 1]$ and $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$. Figure 3.3 shows the corresponding ground truth Hamiltonian phase plots for the system. In order to model dissipation, we introduce a damping term on the angular momentum, p. The update term for the angular momentum is then given by

$$\dot{p} = \nabla_q \mathcal{H}(x) - \rho p = -mgl \sin(q) - \rho p, \tag{3.10}$$

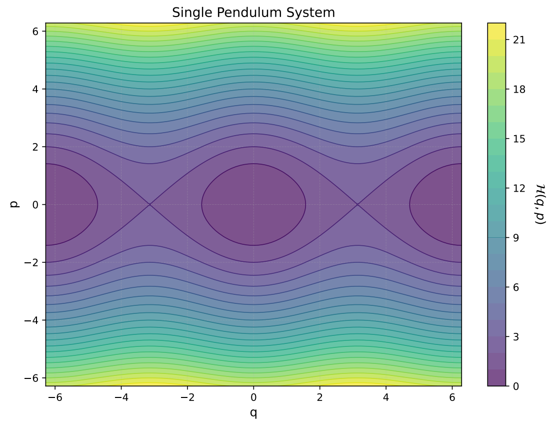
where ρ is the damping coefficient. In our experiments, we trained 4 models on the system across different layer widths and dataset sizes for both the conservative and dissipative cases. The models used are the Vanilla HNN, MLP, RDHNN and SharedRDHNN.

Figures 3.4 and 3.5 show the scaling results for the conservative and dissipative cases respectively, for the SWIM trainer. In the hidden layer width scaling experiments, the dataset size, $|D|$ is set to 10,000 while in the dataset size scaling experiments the hidden layer width is 1500.

In the conservative system, all models reach low approximation errors on both domains, with significantly lower error on the narrower domain. We observe all HNN based models



$[-2\pi, 2\pi] \times [-1, 1]$



$[-2\pi, 2\pi] \times [-2\pi, 2\pi]$

Figure 3.3.: Ground truth Hamiltonian phase plots for the single pendulum system in Equation (3.9) for the domains $[-2\pi, 2\pi] \times [-1, 1]$ and $[-2\pi, 2\pi] \times [-6, 6]$ respectively with iso-contours of the Hamiltonian.

outperforming the MLP across all layer widths and dataset sizes. Furthermore, all HNN based models show comparable performance across the board. However, we observe the HNN model outperforms the others for smaller dataset sizes. This can be explained by its less complex architecture where the RDHNN and SharedRDHNN models also learn a dissipative field that offers no theoretical benefit in the conservative scenario. Figure 3.6 shows the rollout errors for the conservative system. The Hamiltonian of the MLP predictions diverge from the true Hamiltonian at a significantly higher rate compared to the physics-informed models when integrated over time in the evaluation phase. In the MLP, the system slowly gains energy over time and diverges. The HNN based models depict comparable behavior and errors, with RDHNN and SharedRDHNN having the edge over the HNN.

In the dissipative system, as expected, we observe the HNN failing to adapt to the dissipation and learn the correct flow field. Both RDHNN and SharedRDHNN models learn the field with very low approximation errors and outperform the MLP. On Figure 3.7, we plot the phase space mean squared error of the three models in comparison for both SWIM and Adam optimizers. In both comparisons, models that utilized the Adam optimizer were trained for 2 million iterations. Compared to the MLP, SharedRDHNN performs better with more stable long term behavior when trained with both SWIM and Adam.

Despite having a single hidden layer, SharedRDHNN is able to match the performance of RDHNN when trained with SWIM. This is an expected consequence of the SWIM trainer. Since both subnetworks are sampled from the same distribution and with uniform probability densities having the second network does not offer any additional benefit.

Another important observation is the difference in training times between the Adam and

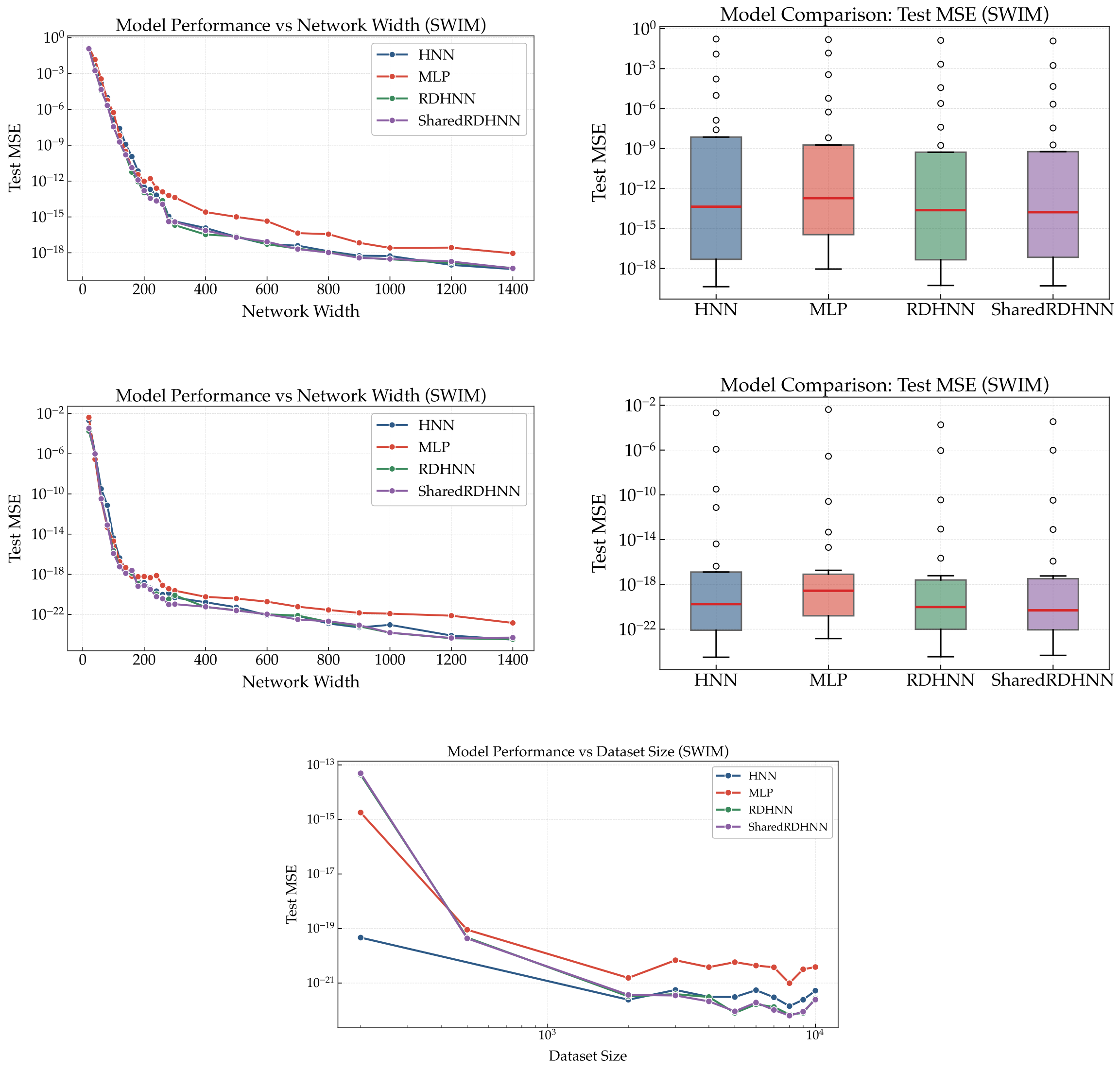


Figure 3.4.: Hidden layer width scaling results for the conservative single pendulum system in Equation (3.9) for the SWIM trainer. The top row shows the domain $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ and the second row shows the domain $[-2\pi, 2\pi] \times [-1, 1]$. The experiments are performed with a dataset size of 10000 points. The bottom row shows the dataset scaling results with a hidden layer width of 1500 on the domain $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$.

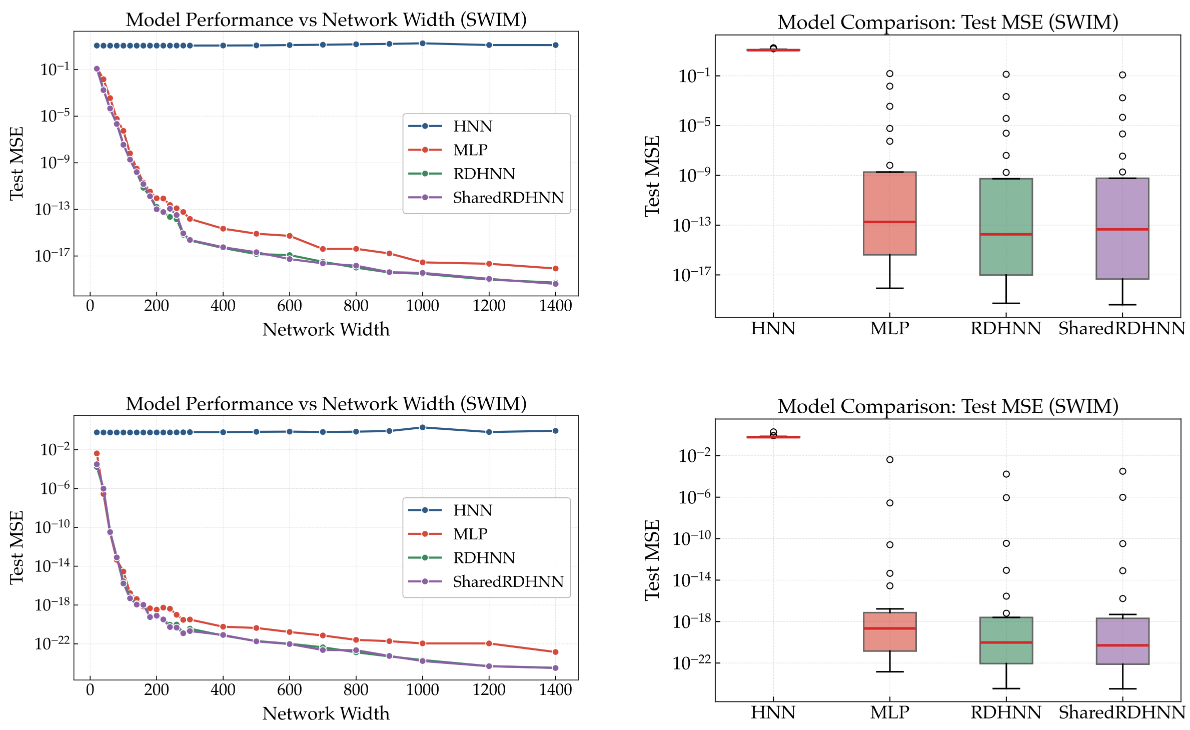


Figure 3.5.: Hidden layer width scaling results for the dissipative single pendulum system in Equation (3.9) with a damping coefficient of 2, for the SWIM trainer. The top row shows the domain $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ and the second row shows the domain $[-2\pi, 2\pi] \times [-1, 1]$. The experiments are performed with a dataset size of 10000 points.

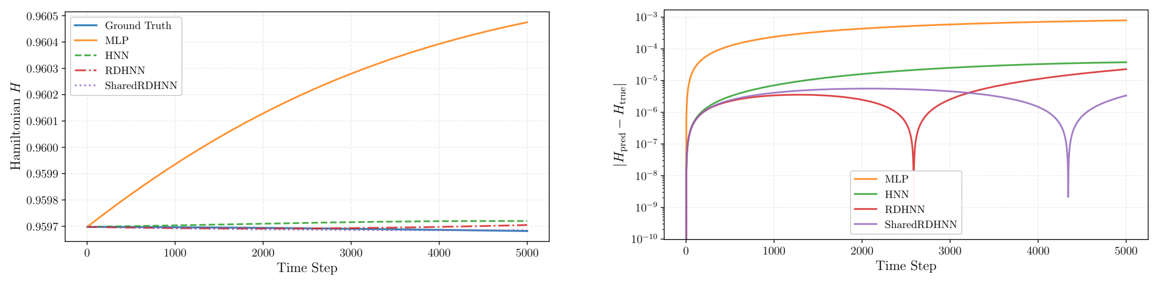


Figure 3.6.: Predicted Hamiltonian values (left) and corresponding prediction errors (right) for the conservative single pendulum system with hidden layer width of 100 and SWIM trainer.

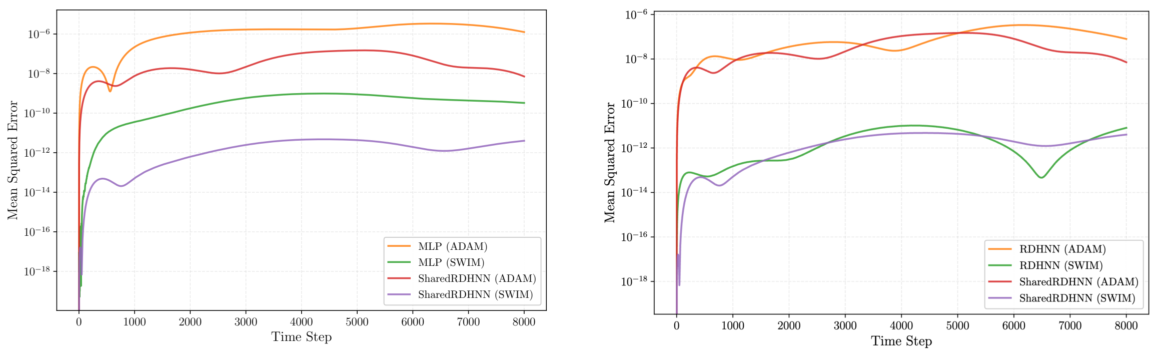


Figure 3.7.: Phase space error distributions for the dissipative single pendulum system (SWIM and Adam optimizers, $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$, damping = 2). Left: MLP vs SharedRDHNN. Right: SharedRDHNN vs RDHNN. Adam training is done over 2 million steps.

Table 3.1.: Comparison of training times (t) and relative test errors (ϵ) for models with a hidden layer width of 100.

Model	t_{Adam} (s)	t_{SWIM} (s)	Speedup	ϵ_{Adam}	ϵ_{SWIM}
MLP	2566.93	0.0144	1.8×10^5	1.7×10^{-4}	1.5×10^{-6}
RDHNN	9418.16	0.0426	2.2×10^5	6.6×10^{-5}	7.6×10^{-7}
SharedRDHNN	9219.32	0.0260	3.5×10^5	7.2×10^{-5}	7.3×10^{-7}

SWIM optimizers. Table 3.1 shows the training times and relative test errors for the models with a hidden layer width of 100. We observe that the SWIM optimizer is significantly faster than the Adam optimizer with speedup factors ranging from 2.2×10^5 to 3.5×10^5. This is an expected consequence of the SWIM optimizer's ability to train the models in a more efficient way since it only trains the last linear layer and it does not need iterative optimization since it is a convex problem. In this comparison, the Adam optimizer was trained for 2 million steps with a batch size of 512 and with learning rate scheduling between 10^{-3} and 10^{-5} that corresponds to over 120,000 epochs. The corresponding loss curves can be found in Figure A.3. Furthermore, especially in Adam trained models, we observe the physics-informed models are significantly slower than the MLP. As we have to compute the gradients of the Hamiltonian with respect to the inputs after each forward pass, the training time is 3-4 times slower than the MLP with iterative optimization. Also, the loss functions of the physics-informed models are more complex and require more computational resources to compute, which further slows down the training process.

We plot the conservative and dissipative prediction fields of SharedRDHNN and RDHNN in Figure 3.8. We observe that although SWIM trained models are more accurate, the implicitly learned dissipative and conservative fields are not as well defined as their Adam trained counterparts. While SWIM trained models are very efficient to build an accurate prediction, their learned conservative and dissipative fields appear highly irrotational. To quantitatively compare these fields, the domain $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ is discretized into a grid of 100x100 points and the predictions of both subnetworks are compared. For the Adam trained SharedRDHNN, the mean norms of the conservative and dissipative predictions within the grid are 3.63 and 5.56 respectively. For the SWIM trained case, we see significantly higher activations with around 7.70×10^6 in both fields.

To resolve this, we modified the vector field to have zero-mean by subtracting the mean of the predictions from all elements. This revealed a more flow-like behavior in the vector field, as shown in Figure 3.9. However, we still do not observe the rotational and irrotational decomposition in the vector field. This behavior differs from the damped mass-Spring system or the RLC circuits in explained in Section 3.2.2, where the Helmholtz decomposition is observed evidently with little to no error. We characterize the discrepancy in field reconstruction accuracy between the mass-spring system and the pendulum by the algebraic properties of their respective vector fields.

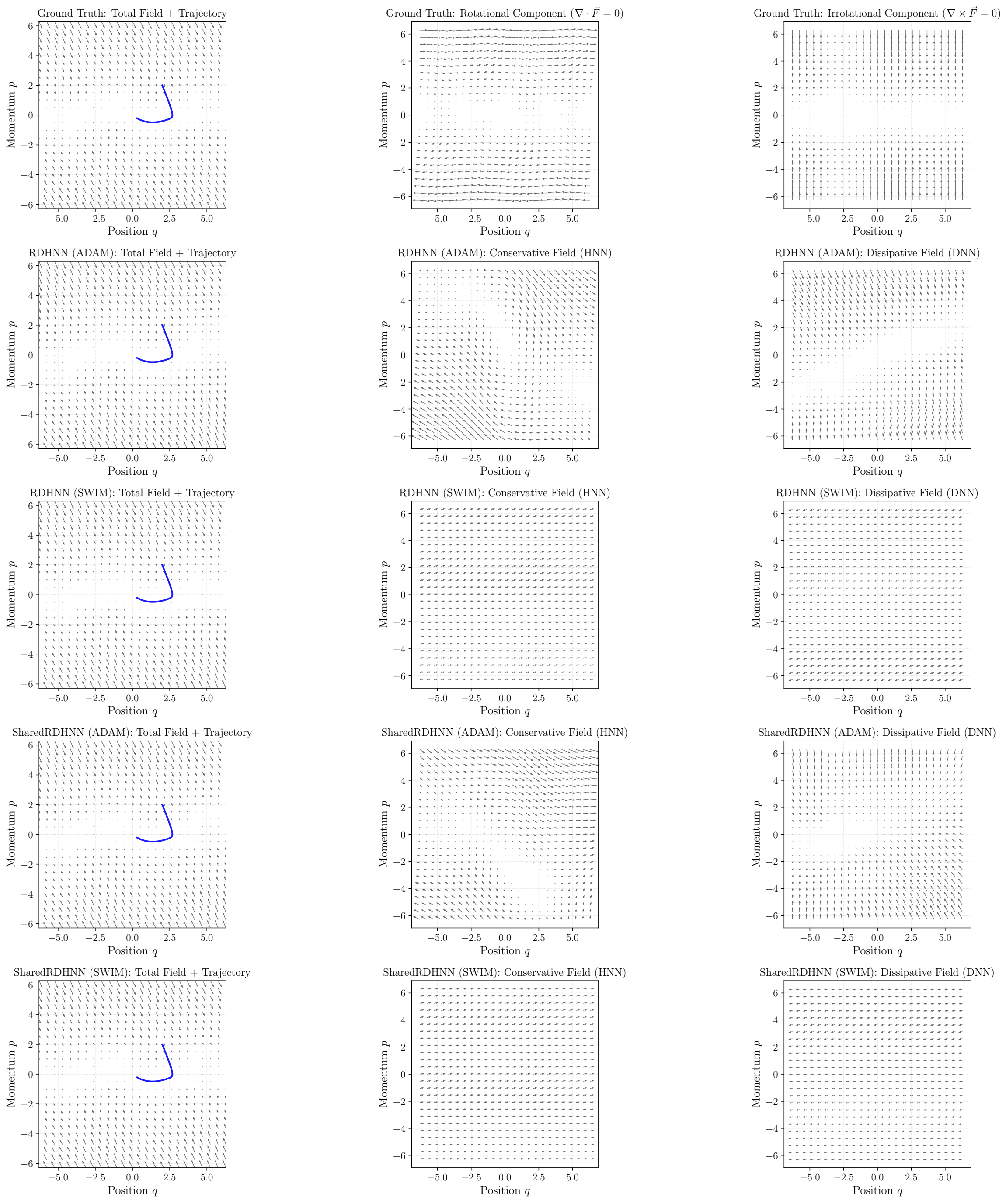


Figure 3.8.: Vector fields and integrated trajectories for the MLP, RDHNN and SharedRDHNN models on the dissipative single pendulum ($[-2\pi, 2\pi] \times [-2\pi, 2\pi]$, damping = 2).

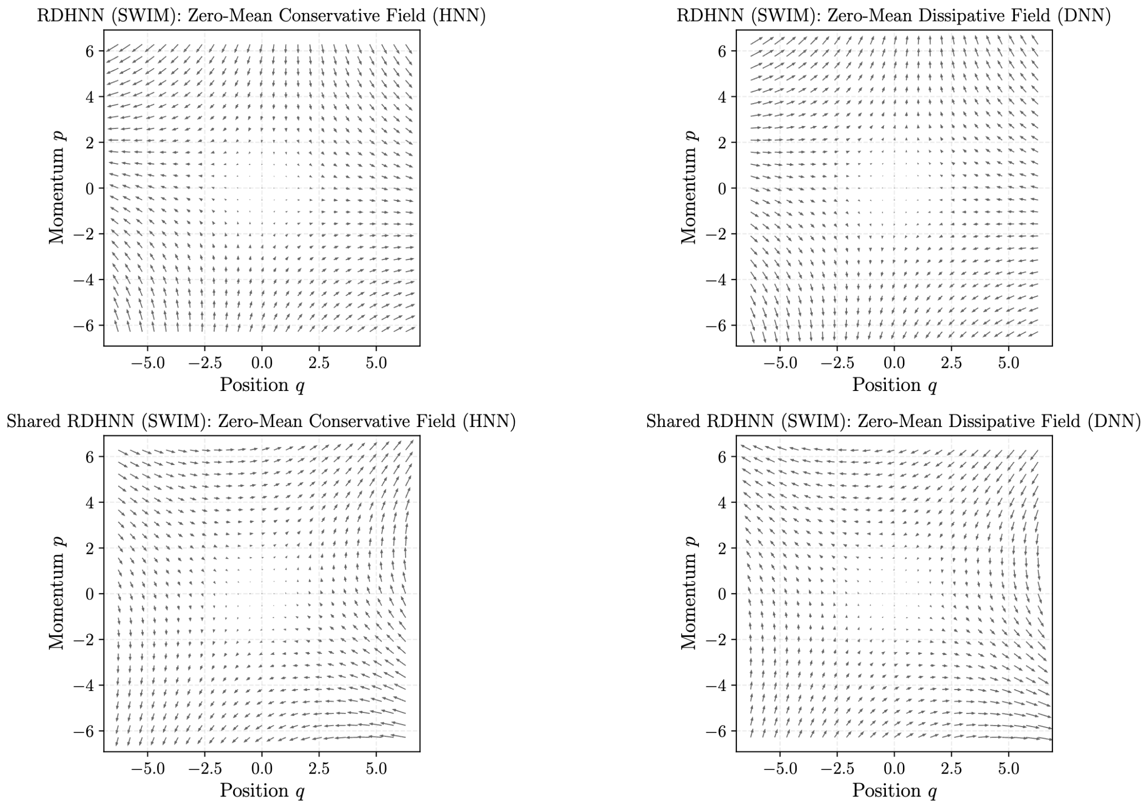


Figure 3.9.: Vector fields and integrated trajectories for the RDHNN and SharedRDHNN models on the dissipative single pendulum ($[-2\pi, 2\pi] \times [-2\pi, 2\pi]$, damping = 2).

Linear Case: Unique Matrix Decomposition

For the damped mass-spring system, the true dynamics are linear and governed by a constant matrix $\mathbf{A} \in \mathbb{R}^{2d \times 2d}$. The vector field is given by:

$$\dot{x} = \mathbf{A}x = \begin{bmatrix} 0 & 1 \\ -k & -\rho \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix}. \tag{3.11}$$

The model attempts to approximate this using a quadratic Hamiltonian $\mathcal{H}(x) = \frac{1}{2}x^T\mathbf{M}x$ and a constant dissipation matrix $\mathbf{D}$. The choice of a quadratic Hamiltonian and constant dissipation arises from the physical definitions of the harmonic oscillator. The total energy of a linear spring is given by $\mathcal{H} = \frac{p^2}{2m} + \frac{kq^2}{2}$, which can be expressed in the quadratic form $\frac{1}{2}x^T\mathbf{M}x$. Furthermore, linear viscous damping ($F \propto -\dot{q}$) implies that the dissipation operator acting on the gradient must be state-independent (constant $\mathbf{D}$) to preserve the linearity of the vector field. The learned dynamics take the form:

$$\dot{x}_{pred} = (\mathcal{J} - \mathbf{D})\nabla\mathcal{H}(x) = (\mathcal{J} - \mathbf{D})\mathbf{M}x. \tag{3.12}$$

Assuming the network correctly identifies the mass matrix $\mathbf{M} \approx \mathbf{I}$ since we set $m = k = 1$, the learning problem reduces to decomposing the system matrix $\mathbf{A}$ into conservative and dissipative components:

$$\mathbf{A} \overset{!}{=} \mathcal{J} - \mathbf{D}. \tag{3.13}$$

This decomposition is algebraically unique since any square matrix $\mathbf{A}$ can be uniquely separated into its skew-symmetric and symmetric parts through the Toeplitz decomposition as

$$\mathcal{J} = \frac{1}{2}(\mathbf{A} - \mathbf{A}^T) \quad \text{(Conservative, Skew-Symmetric)}, \tag{3.14}$$

$$-\mathbf{D} = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T) \quad \text{(Dissipative, Symmetric)}. \tag{3.15}$$

Since the mathematical structure offers a one-to-one mapping between the system matrix $\mathbf{A}$ and the tuple $(\mathcal{J}, \mathbf{D})$, the network has no freedom to "mix" the fields.

Non-Linear Case: Functional Gauge Ambiguity

For the simple pendulum, the dynamics are non-linear due to the sinusoidal restoring force to be

$$\dot{x} = f(x) = \begin{bmatrix} p \\ -mgl\sin(q) - cp \end{bmatrix}. \tag{3.16}$$

The RDHNN models this field via state-dependent functions as

$$\dot{x}_{pred} = (\mathcal{J} - \mathbf{D}_\phi(x))\nabla\mathcal{H}_\theta(x). \tag{3.17}$$

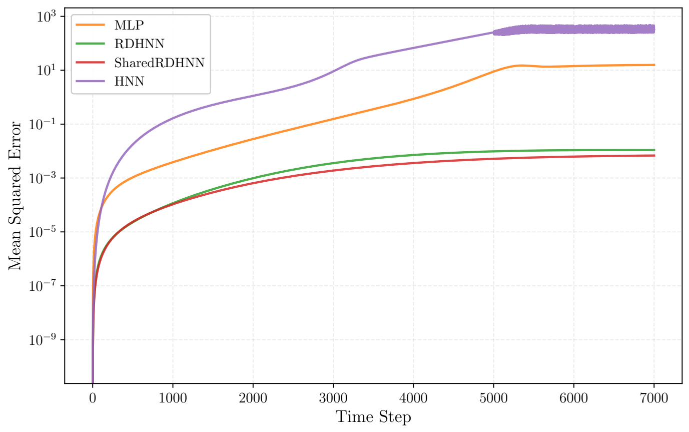


Figure 3.10.: Mean square error of the on the dissipative pendulum system where the models are trained on the domain $[-2, 2] \times [-2, 2]$ and tested on the domain $[-5, 5] \times [-5, 5]$. The models have a hidden layer width of 100 and are trained with the SWIM optimizer.

Unlike the linear case, there is no unique algebraic decomposition for an arbitrary vector field $f(x)$ into symplectic and dissipative components.

Let $(\mathcal{H}^*, \mathbf{D}^*)$ be the ground truth solution. The network may find an alternative solution $(\tilde{\mathcal{H}}, \tilde{\mathbf{D}})$ such that:

$$(\mathcal{J} - \tilde{\mathbf{D}}(x))\nabla\tilde{\mathcal{H}}(x) = (\mathcal{J} - \mathbf{D}^*(x))\nabla\mathcal{H}^*(x) = f(x). \tag{3.18}$$

For example, if the network learns a distorted Hamiltonian $\tilde{\mathcal{H}}$ that includes a portion of the dissipative force (e.g., a "tilted" potential), it can compensate by learning a warped dissipation matrix $\tilde{\mathbf{D}}(x)$ that rotates the gradient back to the correct direction. Since the least squares solution that we calculate minimizes the error $\|\dot{x}_{pred} - \dot{x}_{true}\|^2$ only penalizes the *net* vector field, this "cancellation error" is invisible to the optimization process which leads to physically incorrect individual fields despite accurate total trajectory prediction.

In order to test the generalization capabilities of the models, we conducted further experiments where we trained the models on the domain $[-2, 2] \times [-2, 2]$ and tested on the unseen domain $[-5, 5] \times [-5, 5]$ to perform an extreme scaling test since all the models could capture the training domain very well. Figure 3.10 shows the mean square error of the models on the test domain. We observe that the RDHNN and SharedRDHNN models are able to generalize to the larger domain with significantly lower error than the MLP. This is expected as physics-enhanced models tend to better extrapolate, using the Hamiltonian bias in this case.

3.2.2. Mass-Spring System and RLC Circuits

We consider a system where a mass attached to a spring and damper, connected to a wall, is displaced from its equilibrium position. The Hamiltonian of the mass-spring system is given by

$$\mathcal{H}(q,p) = \frac{1}{2}kq^2 + \frac{p^2}{2m}, \tag{3.19}$$

with the partial derivatives

$$\nabla_q \mathcal{H}(q,p) = kq, \nabla_p \mathcal{H}(q,p) = \frac{p}{m}, \tag{3.20}$$

where k is the spring constant, m is the mass, q is the position and p is the momentum. For simplicity, we set $k = m = 1$ and aim to approximate the Hamiltonian

$$\mathcal{H}(q,p) = \frac{1}{2}q^2 + \frac{p^2}{2}, \tag{3.21}$$

for the domain $[-2,2] \times [-2,2]$. Similar to the single pendulum system, we introduce a damping term on the momentum, p. The update term for the momentum is given by

$$\dot{p} = \nabla_q \mathcal{H}(q,p) - \rho p = kq - \rho p, \tag{3.22}$$

where ρ is the damping coefficient. Another system we consider along with this system is the RLC circuit where a resistor R, capacitor C and inductor L are connected in series. In this representation, q represents the charge of the capacitor. The Hamiltonian of the RLC circuit is given by

$$\mathcal{H}(q,p) = \frac{1}{2}\frac{q^2}{C} + \frac{p^2}{2L}, \tag{3.23}$$

with the partial derivatives

$$\nabla_q \mathcal{H}(q,p) = \frac{q}{C} = v_c, \nabla_p \mathcal{H}(q,p) = \frac{p}{L} = i, \tag{3.24}$$

where v_c is the voltage across the capacitor and i is the current through the inductor. With the identifications

$$m \leftrightarrow L, \quad \text{and} \quad k \leftrightarrow \frac{1}{C}, \tag{3.25}$$

it can be seen that the two systems are dynamically equivalent to each other. Figure 3.11 shows the corresponding ground truth Hamiltonian phase plots for the systems.

Figure 3.12 shows the scaling results for the conservative and dissipative cases respectively for the SWIM trainer while Figure 3.13 shows the results for the RLC circuit. In the hidden layer width scaling experiments, the dataset size, $|D|$ is set to 10,000 while in the

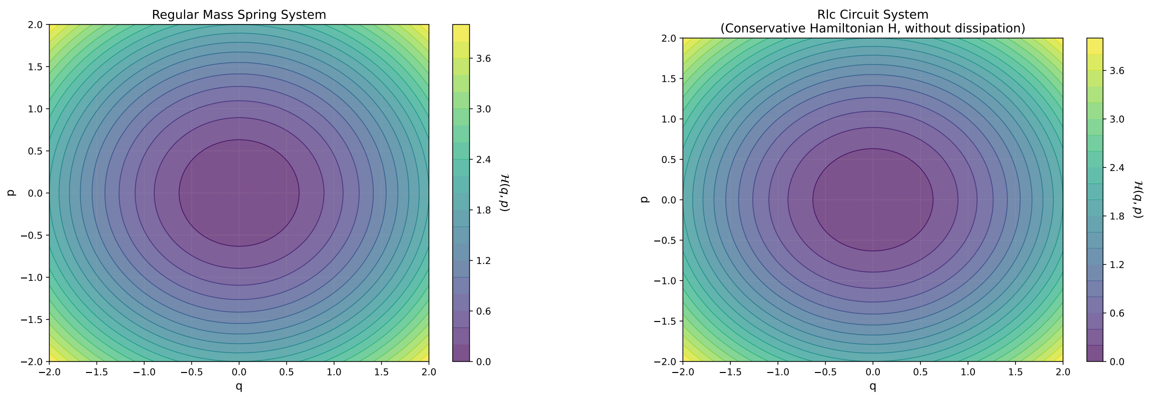


Figure 3.11.: Ground truth Hamiltonian phase plots for the mass-spring system on the left and the RLC circuit on the right showing iso-contours of the Hamiltonian in their respective phase spaces.

dataset size scaling experiments the hidden layer width is 1500. We observe all models learning the system dynamics well and achieving low approximation errors with the MLP having the lowest error. In the conservative case, all HNN based models have similar performance. Since we trained the SWIM trained models 100 times and took the median of the results, the performance of the RDHNN and SharedRDHNN models are virtually identical.

In Figure 3.12, we plot the errors of the models on the hidden layer scaling experiments, grouped by the trainer type on the bottom row. The results show that the SWIM trained models are able to train at a fraction of the time required by the Adam trained models while massively outperforming them in terms of error. In this specific experiments, Adam trained models were trained for 2 million steps with learning rate scheduling.

We plot the dissipative and conservative vector fields for the RDHNN and SharedRDHNN models on the dissipative mass-spring system in Figure 3.14. Similar to the single pendulum system, we observe quite high activations in both fields that cancel each other out. When we zero-mean each field, the flow-like behavior becomes more apparent. This is shown on Figure 3.15 where the zero mean fields are plotted for the SWIM trained models. It can be seen that both models replicate the ground truth vector field quite well where the rotational and irrotational components are clearly visible. However, we have seen that the learned decomposition also heavily depends on the seeds for the sampling algorithm as successive runs ended up with slightly different decompositions that have comparable errors. The most common error mode in the learned decomposition is predicting a slightly tilted irrotational component as shown in Figure 3.16.

Similar to the single pendulum system, we observe a significant speedup in the training time of the SWIM trained models compared to the Adam trained models along with a

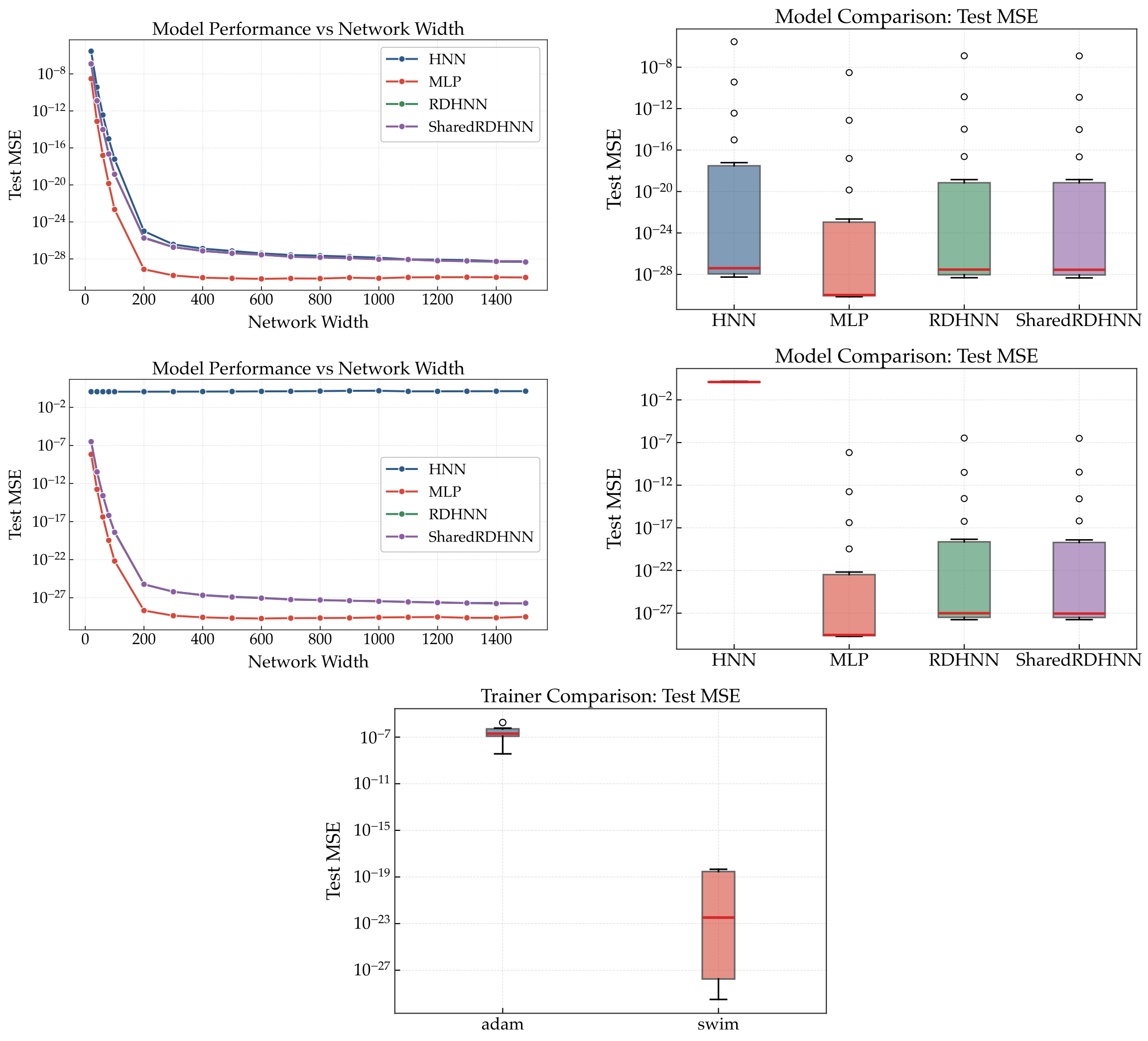


Figure 3.12.: Hidden layer width scaling results for the mass-spring system in Equation (3.21) for the SWIM trainer. The top row shows the results of the conservative system and the second row shows the results of the dissipative system. The experiments are performed with a dataset size of 10000 points. The bottom row shows the trainer comparison results between the Adam and SWIM optimizers in the dissipative system.

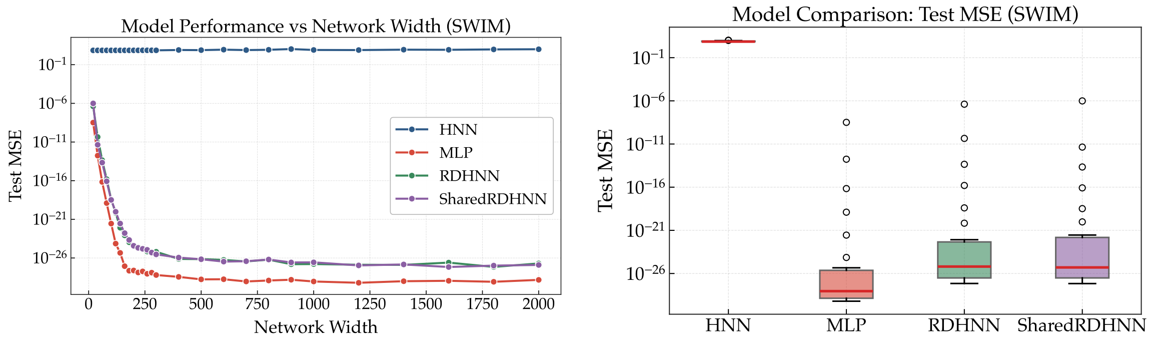


Figure 3.13.: Hidden layer width scaling and model comparison results for the dissipative RLC circuit using the SWIM optimizer with a dataset size of 10000 points.

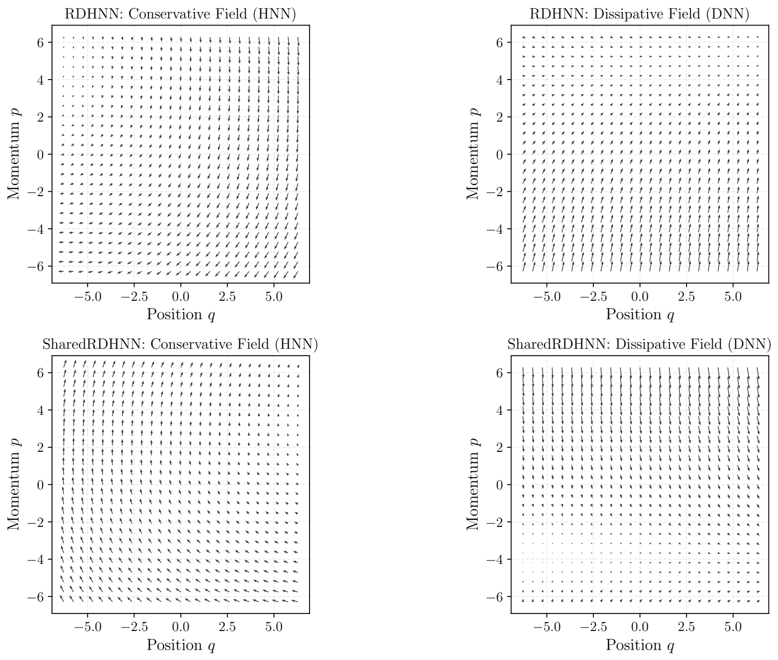


Figure 3.14.: Vector fields for the RDHNN and SharedRDHNN models on the dissipative mass-spring system ($[-2, 2] \times [-2, 2]$, damping = 2).

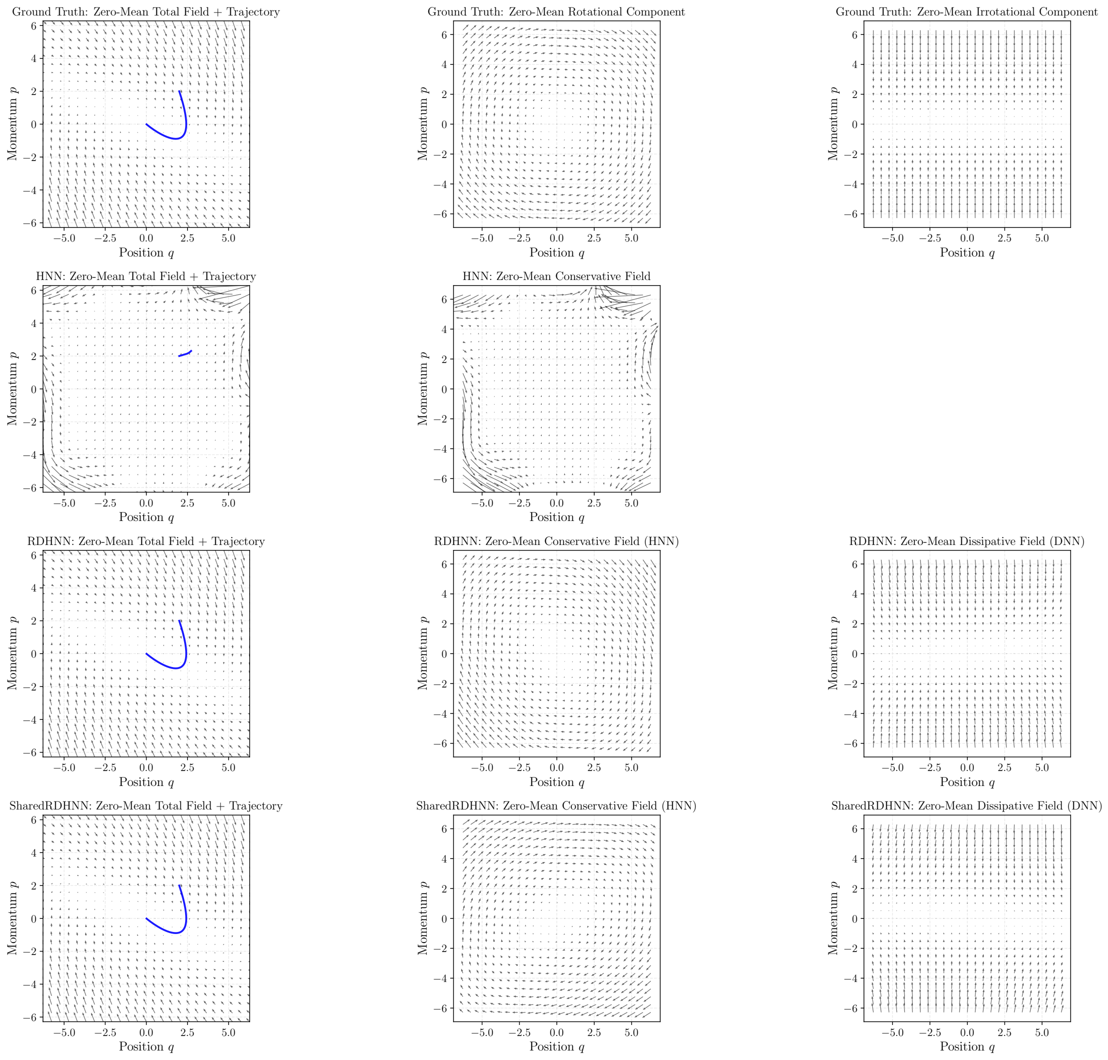


Figure 3.15.: Vector fields and integrated trajectories for the SWIM trained RDHNN and SharedRDHNN models on the dissipative mass-spring system ($[-2, 2] \times [-2, 2]$, damping = 2).

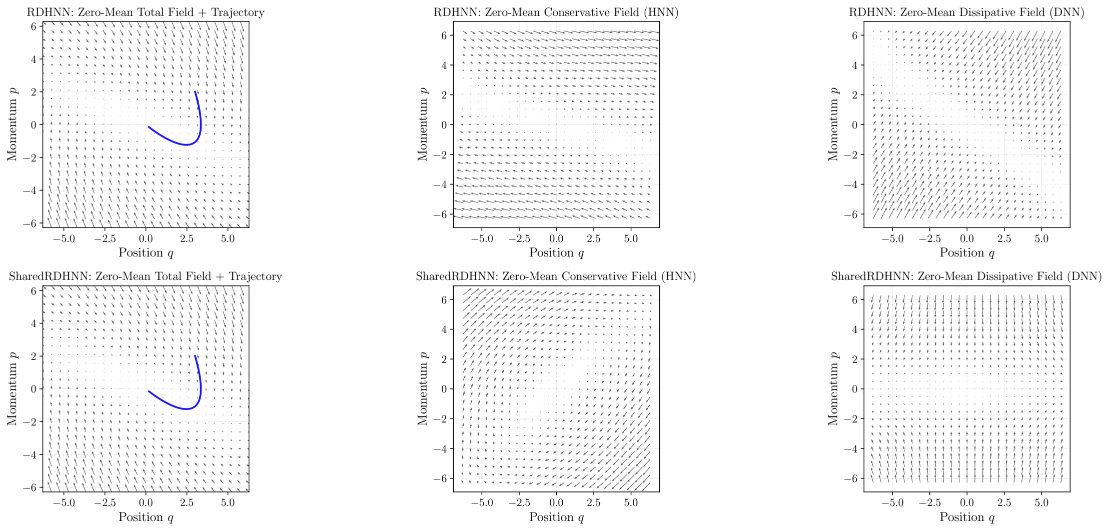


Figure 3.16.: Tilted decompositions learned for the SWIM trained RDHNN and SharedRDHNN models on the dissipative mass-spring system with different seeds.

Table 3.2.: Comparison of training times (t) and relative test errors (ϵ) for models with width $W = 100$ on the dissipative mass-spring system. Adam trained models were trained for 2 million steps with learning rate scheduling.

Model	t_{Adam} (s)	t_{SWIM} (s)	Speedup	ϵ_{Adam}	ϵ_{SWIM}
MLP	2734.47	0.0127	2.1×10^5	3.1×10^{-5}	4.1×10^{-12}
RDHNN	9246.49	0.0557	1.7×10^5	2.6×10^{-4}	3.4×10^{-10}
SharedRDHNN	9701.73	0.0233	4.2×10^5	1.9×10^{-4}	3.2×10^{-10}

significant improvement in the error of the models. Table 3.2 shows the training times and relative test errors for the models with a hidden layer width of 100 and a dataset size of 10000 points on the dissipative mass-spring system.

3.2.3. Morse and Duffing Oscillators

In this section, we investigate the performance of our models on two well-known nonlinear oscillators, the Morse [56, 64] and Duffing [103, 21] oscillators. Unlike the simple harmonic oscillator or linear mass-spring systems, these oscillators feature exponential and higher-order polynomial terms in their potential energy functions. This increased complexity in their Hamiltonian formulations provides a more rigorous test of the models' ability to capture nonlinear dynamics beyond the linear regime.

The Morse oscillator is based on a nonlinear potential energy function called the Morse potential. The Morse potential is used to model the potential energy of a diatomic molecule or the interaction between an atom with a surface. This potential is given by

$$V(q) = D_w(1 - e^{-\alpha q})^2, \tag{3.26}$$

where D_w is the depth of the potential well, α is the parameter that controls the width of the well and q is the displacement from the equilibrium position. The Hamiltonian of the Morse oscillator is given by

$$\mathcal{H}(q,p) = \frac{p^2}{2m} + D_w(1 - e^{-\alpha q})^2, \tag{3.27}$$

where m is the mass with the partial derivatives

$$\nabla_q \mathcal{H}(q,p) = -2D_w\alpha(e^{-\alpha q} - e^{-2\alpha q}), \nabla_p \mathcal{H}(q,p) = \frac{p}{m}. \tag{3.28}$$

For our experiments, we set $D_w = \alpha = m = 1$. The Hamiltonian is then given by

$$\mathcal{H}(q,p) = \frac{p^2}{2} + (1 - e^{-q})^2. \tag{3.29}$$

The Duffing oscillator can be described as a mass-spring system with a nonlinear spring, damping and a forcing term. The equation of motion for the Duffing oscillator is given by

$$\ddot{q} + \delta\dot{q} + \alpha q + \beta q^3 = \gamma\cos(\omega t), \tag{3.30}$$

where δ is the damping coefficient, α is the spring constant, β is the nonlinear spring constant, ω is the forcing frequency and γ is the amplitude of the forcing term. The Hamiltonian of the Duffing oscillator without the forcing term or damping is given by

$$\mathcal{H}(q,p) = \frac{p^2}{2m} + \frac{\alpha}{2}q^2 + \frac{1}{4}\beta q^4, \tag{3.31}$$

with the partial derivatives

$$\nabla_q \mathcal{H}(q,p) = \alpha q + \beta q^3, \nabla_p \mathcal{H}(q,p) = \frac{p}{m}, \tag{3.32}$$

where m is the mass. For our experiments, we set $m = 1$, $\alpha = -1$, $\beta = 5$ and $\delta = 0$ For the dissipative cases, we set $\delta = 0.12$. The Hamiltonian is then given by

$$\mathcal{H}(q,p) = \frac{p^2}{2} - \frac{1}{2}q^2 + \frac{5}{4}q^4. \tag{3.33}$$

Figure 3.17 shows the phase space plots for the Morse and Duffing oscillators with the parameters from Equation (3.29) and Equation (3.33) respectively.

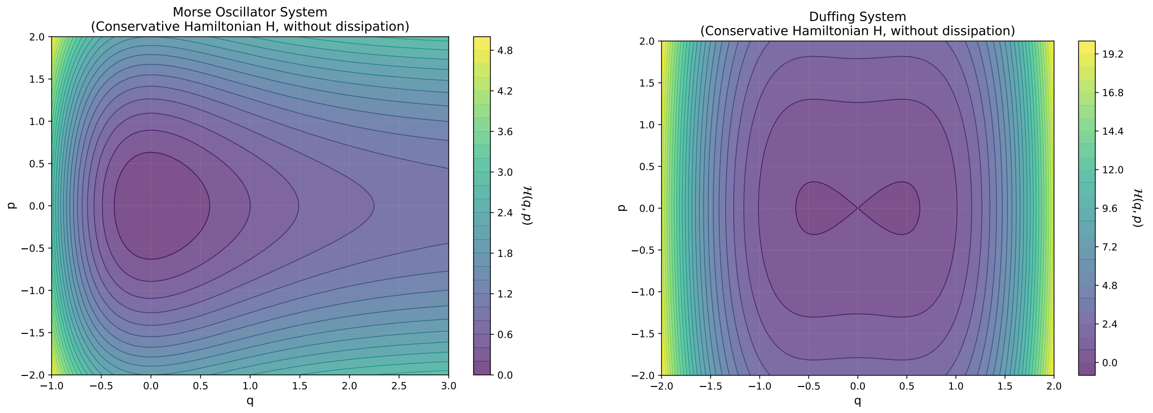


Figure 3.17.: Phase space plots for the Morse Oscillator (left) with the parameters from Equation (3.29) and Duffing Oscillator (right) with parameters from Equation (3.33).

Figure 3.18 shows the hidden layer width scaling MSE and model comparison results for the Morse oscillator system with the parameters from Equation (3.29). In the conservative case, we observe all physics based models performing similarly with the MLP having slightly higher error. This trend is followed in the dissipative case also where all models learn the system dynamics well with MLP having slightly higher error.

We then test the rollout prediction error of the models on the same dissipative system and integrate until the system reaches steady state. Figure 3.19 shows the rollout prediction errors in this setting for q and p, trained with the SWIM optimizer. Over the course of the integration, we observe a marginal performance improvement in SharedRDHNN and RDHNN over the MLP.

To illustrate the impact of different trainers, we show the rollout prediction errors for the Adam and SWIM optimizers in Figure 3.20. For every model and every hidden layer width, we observe the SWIM optimizer significantly outperforming the Adam optimizer by several orders of magnitude in terms of rollout prediction error.

The results on the Morse Oscillator (Table 3.3) provide the strongest evidence for the fundamental difference in scaling behavior between gradient-based optimization and the proposed SWIM method. While Adam exhibits diminishing returns and stagnated error rates as network width increases, SWIM behaves akin to a high-order numerical method. For the proposed method, increasing the network width directly translates into a five-order-of-magnitude reduction in error, dropping from $\sim 10^{-5}$ to $\sim 10^{-10}$. This issue in scaling Adam is further amplified by the lack of hyperparameter tuning and the use of learning rate scheduling specific to the hidden layer width. This need for hyperparameter tuning is not present in the SWIM optimizer, which significantly increases the scalability of the training process.

We plot the zero-mean predicted conservative and dissipative vector fields for the SWIM

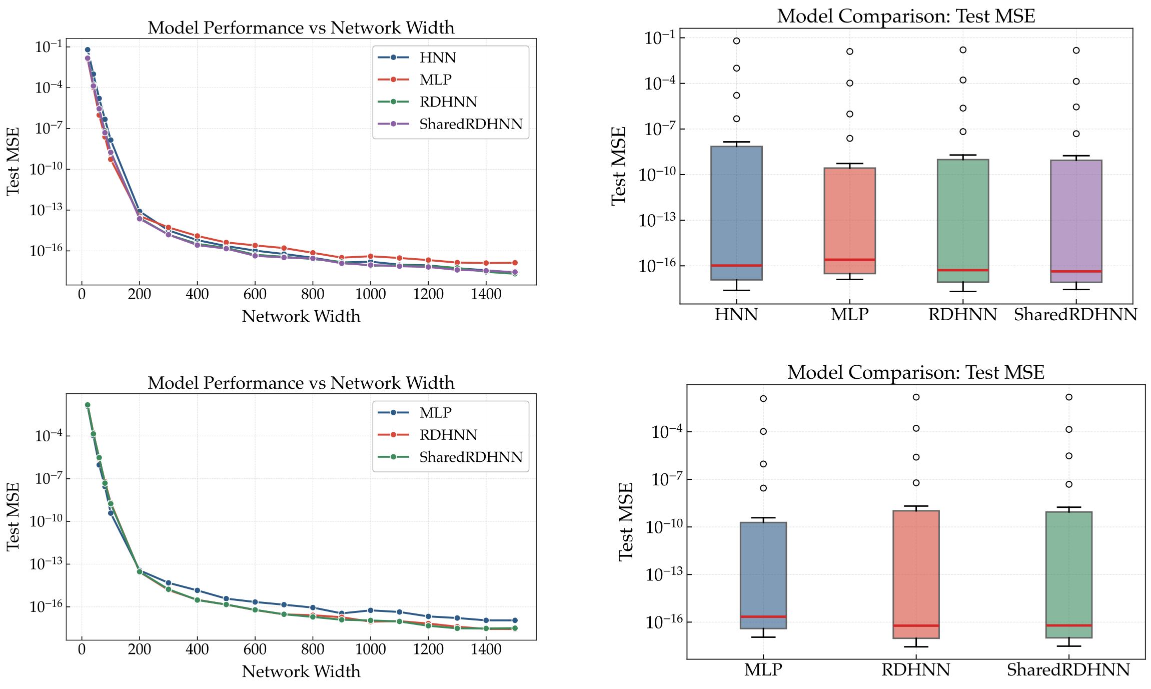


Figure 3.18.: Hidden layer width scaling MSE and model comparison results for the Morse oscillator system for the SWIM trainer. The top row shows the conservative system and the bottom row shows the dissipative system with a damping coefficient of 2. The experiments are performed with a dataset size of 10000 points.

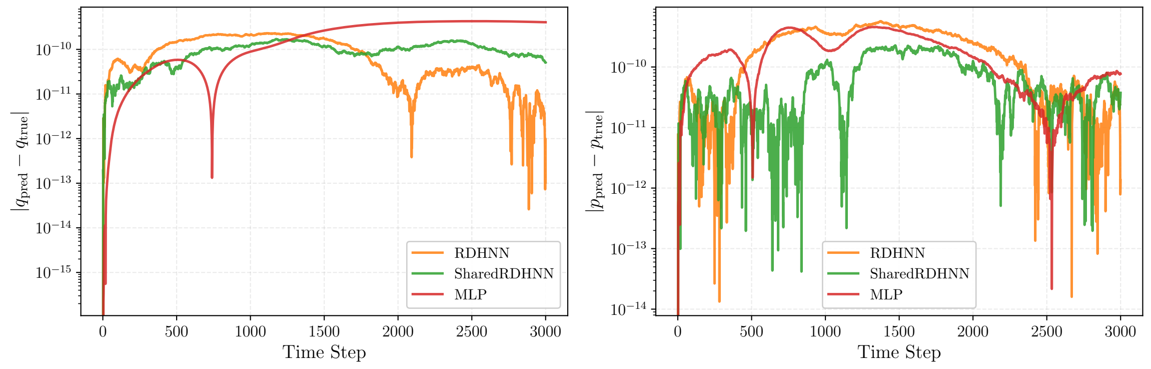


Figure 3.19.: Rollout prediction errors for the dissipative Morse oscillator system with a damping coefficient of 2 from Equation (3.29), trained with the SWIM optimizer. Left figure shows the error in q and the right figure shows the error in p.

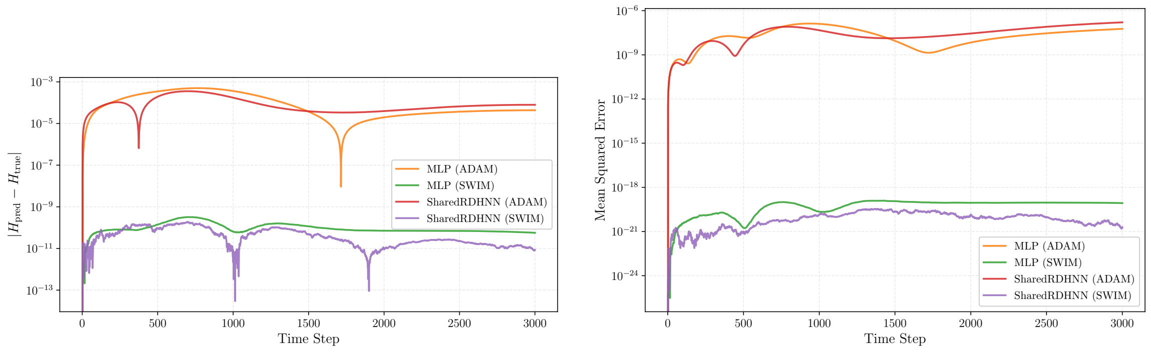


Figure 3.20.: Comparison of SWIM and Adam optimizer performance for the dissipative Morse oscillator with a hidden layer width of 1500 and a dataset size of 10000 points. The left plot shows the Hamiltonian error over time, while the right plot depicts the phase space MSE.

Table 3.3.: Comparison of training metrics for the Morse Oscillator system at widths $W = 100$ and $W = 1500$. t denotes training time (seconds) and ϵ denotes relative test error.

Model	Width	t_{Adam} (s)	t_{SWIM} (s)	Speedup	ϵ_{Adam}	ϵ_{SWIM}
MLP	100	2894.80	0.0125	2.3×10^5	6.9×10^{-5}	1.0×10^{-5}
MLP	1500	3623.98	0.3269	1.1×10^4	2.5×10^{-4}	1.4×10^{-9}
RDHNN	100	6992.80	0.0436	1.6×10^5	2.3×10^{-4}	1.8×10^{-5}
RDHNN	1500	10968.53	0.9220	1.2×10^4	5.1×10^{-4}	4.7×10^{-10}
SharedRDHNN	100	9758.28	0.0235	4.2×10^5	1.4×10^{-4}	1.8×10^{-5}
SharedRDHNN	1500	10920.77	0.9198	1.2×10^4	3.3×10^{-4}	4.4×10^{-10}

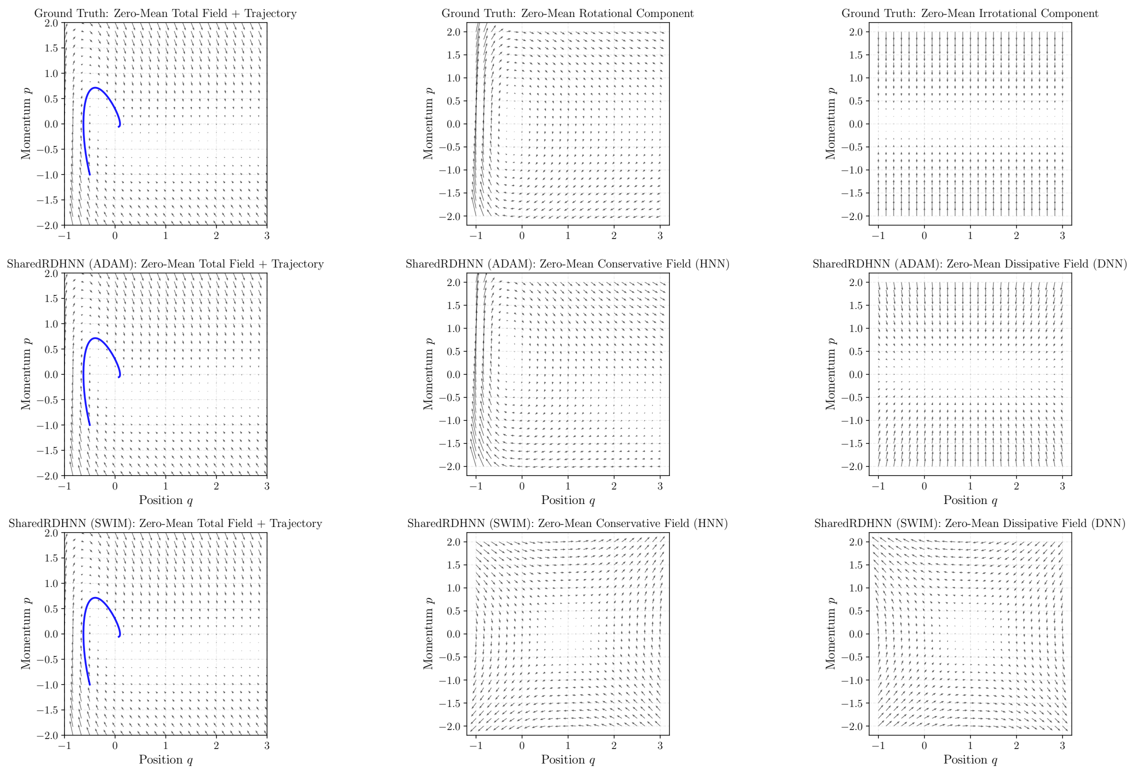


Figure 3.21.: Predicted vector fields for the SWIM and Adam optimizers for the dissipative Morse oscillator system with a damping coefficient of 2. Dataset size is 10000 points and hidden layer width is 1500.

and Adam optimizers for the dissipative Morse oscillator system in Figure 3.21. Similar to the previous experiments, we observe that the learned vector fields are highly dependent on the seed when trained with the SWIM optimizer. Although different seeds yield overall similar training errors, the decomposed vector fields differ visually. When trained with the Adam optimizer, the decomposed vector fields illustrate the flow characteristics similar to the ground truth where a clear separation between the rotational and the irrotational components is visible.

Having established the strong performance of SWIM trained models on the Morse oscillator, we now turn to the Duffing oscillator to examine whether these results generalize to systems with polynomial nonlinearities. While the Morse potential involves exponential terms, the Duffing oscillator features a cubic restoring force, providing a complementary test case for evaluating the proposed methods.

We perform the same hidden layer width scaling experiments on the Duffing oscillator with the Hamiltonian from Equation (3.33). As with the Morse system, we use a dataset size of 10,000 points and vary the hidden layer width to evaluate model performance in both the conservative and dissipative settings. Figure 3.22 shows the model performance and comparisons for the conservative and dissipative systems respectively. In both cases, we observe all models learning the system dynamics well and performing similarly, where physics based models illustrate a slight performance advantage over the MLP for larger hidden layer widths.

To compare the performance of the two training approaches, we evaluate the phase space MSE over time for the dissipative Duffing system in Figure 3.23. Consistent with our findings on the Morse oscillator, SWIM trained models outperform their Adam trained counterparts by several orders of magnitude across all model architectures. This result reinforces the previous finding that SWIM's direct solution approach yields superior generalization on rollout predictions even for systems with fundamentally different nonlinear characteristics.

Figure 3.24 compares the zero-mean decomposed vector fields learned by SWIM and Adam trained SharedRDHNN models against the ground truth. The Adam trained model produces conservative and dissipative components that visually match the ground truth patterns. In contrast, the SWIM trained model learns a different decomposition that does not resemble the ground truth structure. However, when the conservative and dissipative fields are combined, the SWIM model achieves significantly lower prediction error. This observation, consistent with the Morse oscillator results, suggests that the decomposition learned by SWIM may not be physically interpretable even though it remains functionally accurate for predicting system dynamics.

The scalability results for the Duffing system (Table 3.4) reinforce the divergence in scaling behavior observed in previous experiments. Notably, Adam exhibits a clear case of optimization saturation. When the network width is increased from $W = 100$ to $W = 1500$, the test error actually degrades across all models. For instance, the SharedRDHNN error rises from 4.0×10^{-4} to 1.1×10^{-3}. It is important to note that for all Adam experiments, we used a fixed training budget of 2 million iterations with identical hyperparameters (learn-

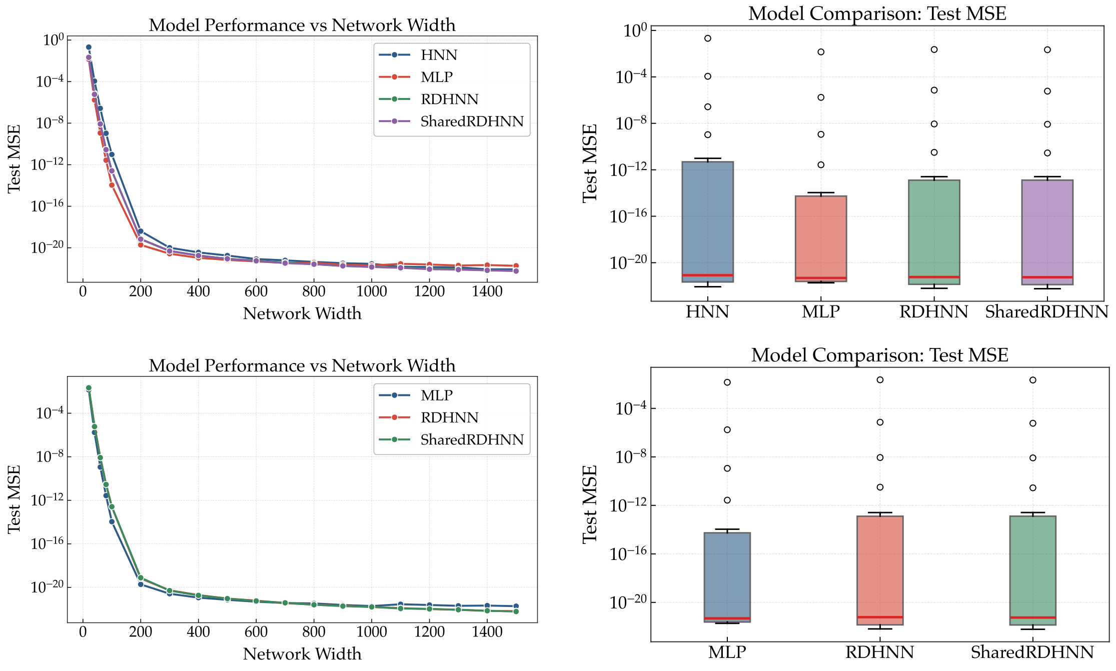


Figure 3.22.: Hidden layer width scaling MSE and model comparison results for the Duffing oscillator system with the SWIM trainer. The top row shows results for the conservative system and the bottom row shows results for the dissipative system with a damping coefficient of 2. Experiments use a dataset size of 10,000 points.

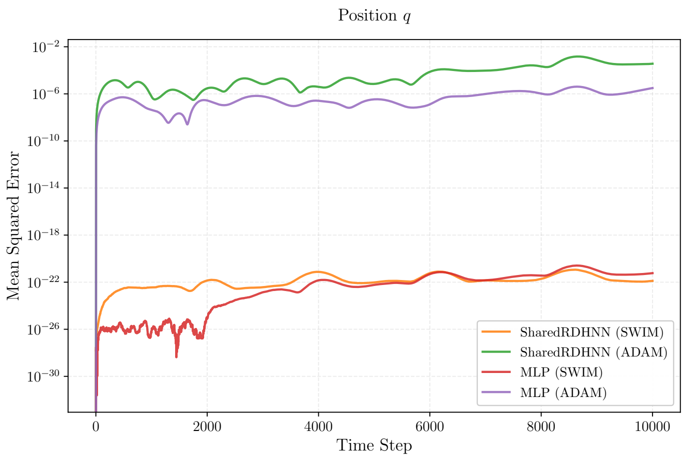


Figure 3.23.: Phase space MSE for the dissipative Duffing oscillator system. Dataset size is 10000 points and hidden layer width is 1500.

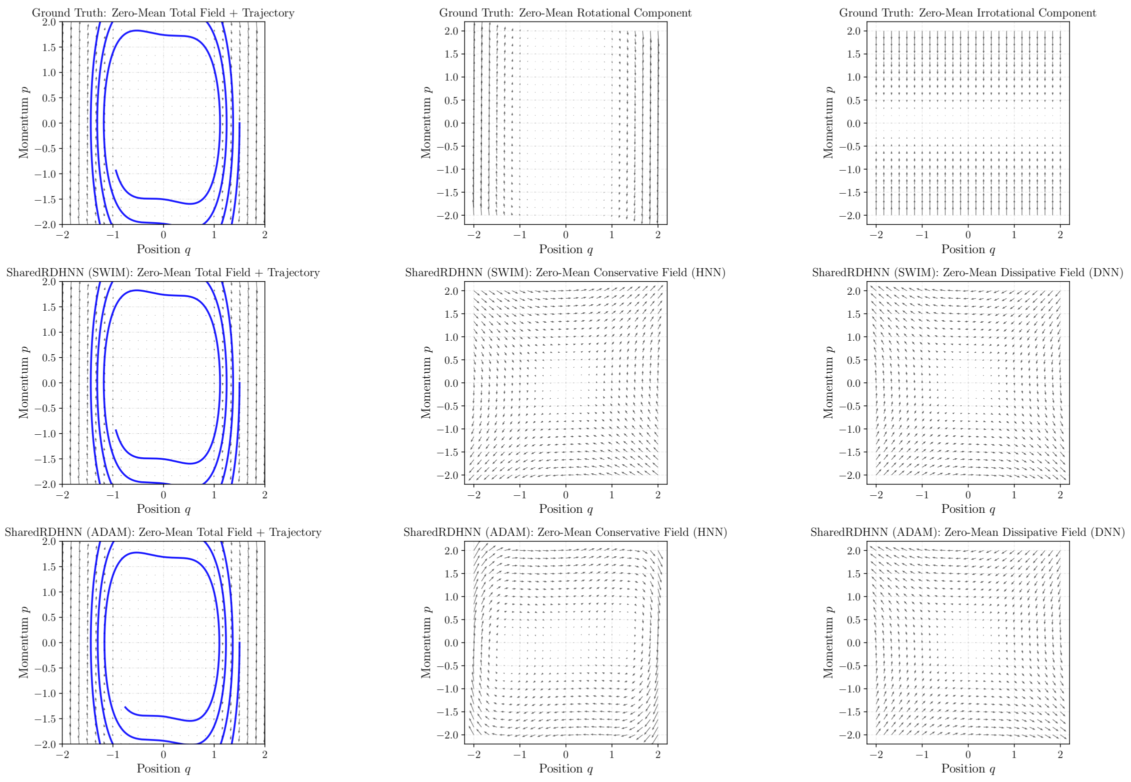


Figure 3.24.: Zero-mean vector fields and trajectories for the dissipative Duffing oscillator system. Dataset size is 10000 points and hidden layer width is 1500.

Table 3.4.: Scalability Analysis: Comparison of training metrics for the Duffing system at widths $W = 100$ and $W = 1500$. t denotes training time (seconds) and ϵ denotes relative test error.

Model	Width	t_{Adam} (s)	t_{SWIM} (s)	Speedup	ϵ_{Adam}	ϵ_{SWIM}
MLP	100	3050.52	0.0128	2.4×10^5	3.1×10^{-5}	1.0×10^{-8}
MLP	1500	3608.18	0.3269	1.1×10^4	1.6×10^{-4}	1.4×10^{-12}
RDHNN	100	9675.12	0.0432	2.2×10^5	1.0×10^{-4}	5.0×10^{-8}
RDHNN	1500	10887.06	0.9192	1.2×10^4	7.1×10^{-4}	8.1×10^{-13}
SharedRDHNN	100	9707.52	0.0233	4.2×10^5	4.0×10^{-4}	5.1×10^{-8}
SharedRDHNN	1500	10995.61	0.9204	1.2×10^4	1.1×10^{-3}	7.8×10^{-13}

ing rate, batch size, etc.) across all network widths. This fixed configuration, while practical for systematic comparison, may be insufficient for larger models that require longer training or adjusted learning rates to converge properly.

In contrast, SWIM requires no such hyperparameter tuning. The method directly solves for the output layer weights given the randomly sampled hidden layer, eliminating the need to adjust learning rates or training duration as the network size changes. This characteristic gives SWIM a distinct advantage in width scaling experiments, as it can leverage the increased capacity without additional configuration. The error drops by approximately five orders of magnitude, from $\sim 5 \times 10^{-8}$ to $\sim 8 \times 10^{-13}$, approaching machine precision. This demonstrates that SWIM successfully utilizes the over-parameterized feature space to resolve the underlying dynamics with high accuracy.

The computational disparity between the two approaches is substantial. While Adam requires over 3 hours to reach a suboptimal solution with error on the order of 10^{-3}, SWIM achieves almost machine-precision level accuracy in under a second. These results, combined with the findings from the Morse oscillator, suggest that SWIM provides a robust and efficient alternative to gradient-based optimization for learning Hamiltonian systems with diverse nonlinear characteristics.

We also attempted to train models on the driven Duffing oscillator, which includes an external periodic forcing term $\gamma \cos(\omega t)$. However, the current architecture only takes the state variables (q, p) as input and does not incorporate time as an explicit parameter. As a result, the models cannot capture time-dependent dynamics and are limited to autonomous systems. Extending the framework to handle non-autonomous systems with explicit time dependence remains a direction for future work.

3.2.4. Double Pendulum System

In this section, we consider the double pendulum system. This system consists of two pendulums where the second pendulum is attached to the end of the first pendulum while the first pendulum is attached to a wall. Different from the previous systems, there are two degrees of freedom in this system, the angles of the two pendulums. Although the system is created by stacking two single pendulums, it is not a simple extension of the single pendulum system as the system's behavior is significantly more chaotic and complex. In this system, the Hamiltonian is given by the sum of the kinetic and potential energies, $\mathcal{H}(q, p) = T + V$ where T and V are the kinetic and potential energies that are defined as

$$T = \frac{m_2 l_2^2 p_1^2 + (m_1 + m_2) l_1^2 p_2^2 - 2 m_2 l_1 l_2 p_1 p_2 \cos(q_1 - q_2)}{2 m_2 l_1^2 l_2^2 C}, \tag{3.34}$$

$$V = -(m_1 + m_2) g l_1 \cos(q_1) - m_2 g l_2 \cos(q_2), \tag{3.35}$$

where m_1 and m_2 are the masses of the first and second pendulums, l_1 and l_2 are the lengths of the first and second pendulums, g is the acceleration due to gravity, q_1 and q_2

are the angles of the first and second pendulums, p_1 and p_2 are the angular momenta of the first and second pendulums, and C is a helper term defined as

$$C = m_1 + m_2 \sin^2(q_1 - q_2) \tag{3.36}$$

We define the helper terms h_1 and h_2 to simplify the partial derivatives of the Hamiltonian as

$$h_1 = \frac{p_1 p_2 \sin(q_1 - q_2)}{l_1 l_2 \, C}, \tag{3.37}$$

$$h_2 = \frac{m_2 l_2^2 p_1^2 + (m_1 + m_2) l_1^2 p_2^2 - 2 m_2 l_1 l_2 p_1 p_2 \cos(q_1 - q_2)}{2 l_1^2 l_2^2 \, C^2}. \tag{3.38}$$

The partial derivatives of the Hamiltonian are

$$\frac{\partial H}{\partial q_1} = (m_1 + m_2) g l_1 \sin(q_1) + h_1 - h_2 \sin(2(q_1 - q_2)), \tag{3.39}$$

$$\frac{\partial H}{\partial q_2} = m_2 g l_2 \sin(q_2) - h_1 + h_2 \sin(2(q_1 - q_2)), \tag{3.40}$$

$$\frac{\partial H}{\partial p_1} = \frac{l_2 p_1 - l_1 p_2 \cos(q_1 - q_2)}{l_1^2 l_2 \, C}, \tag{3.41}$$

$$\frac{\partial H}{\partial p_2} = \frac{(m_1 + m_2) l_1 p_2 - m_2 l_2 p_1 \cos(q_1 - q_2)}{m_2 l_1 l_2^2 \, C}. \tag{3.42}$$

This Hamiltonian is shown on Figure 3.25 for different slices of the phase space. For systems with n degrees of freedom, the state space extends to $\mathbf{x} = (\mathbf{q}, \mathbf{p}) \in \mathbb{R}^{2n}$, where $\mathbf{q} = (q_1, \ldots, q_n)$ are generalized coordinates and $\mathbf{p} = (p_1, \ldots, p_n)$ are generalized momenta.

For the double pendulum ($n = 2$), the input is $\mathbf{x} = (q_1, q_2, p_1, p_2)^\top \in \mathbb{R}^4$, and the governing equations become:

$$\dot{q}_i = \frac{\partial \mathcal{H}_\theta}{\partial p_i} + \frac{\partial \mathcal{D}_\phi}{\partial q_i}, \quad i = 1, 2 \tag{3.43}$$

$$\dot{p}_i = -\frac{\partial \mathcal{H}_\theta}{\partial q_i} + \frac{\partial \mathcal{D}_\phi}{\partial p_i}, \quad i = 1, 2 \tag{3.44}$$

The RDHNN architecture remains structurally identical with two scalar-valued neural networks learning the Hamiltonian and Rayleigh dissipation functions over the extended state space. However, the input and output dimensions are increased to match the number of degrees of freedom as shown in Figure 3.26.

Figure 3.27 shows the hidden layer width scaling mean squared error results and model comparison for the double pendulum system when trained with the SWIM trainer. In all experiments, we use unit masses ($m_1 = m_2 = 1$), unit lengths ($l_1 = l_2 = 1$), and gravitational acceleration $g = 1$. The dataset size D is set to 10,000. As opposed to the single pendulum system, we see that in both the conservative and dissipative cases, all

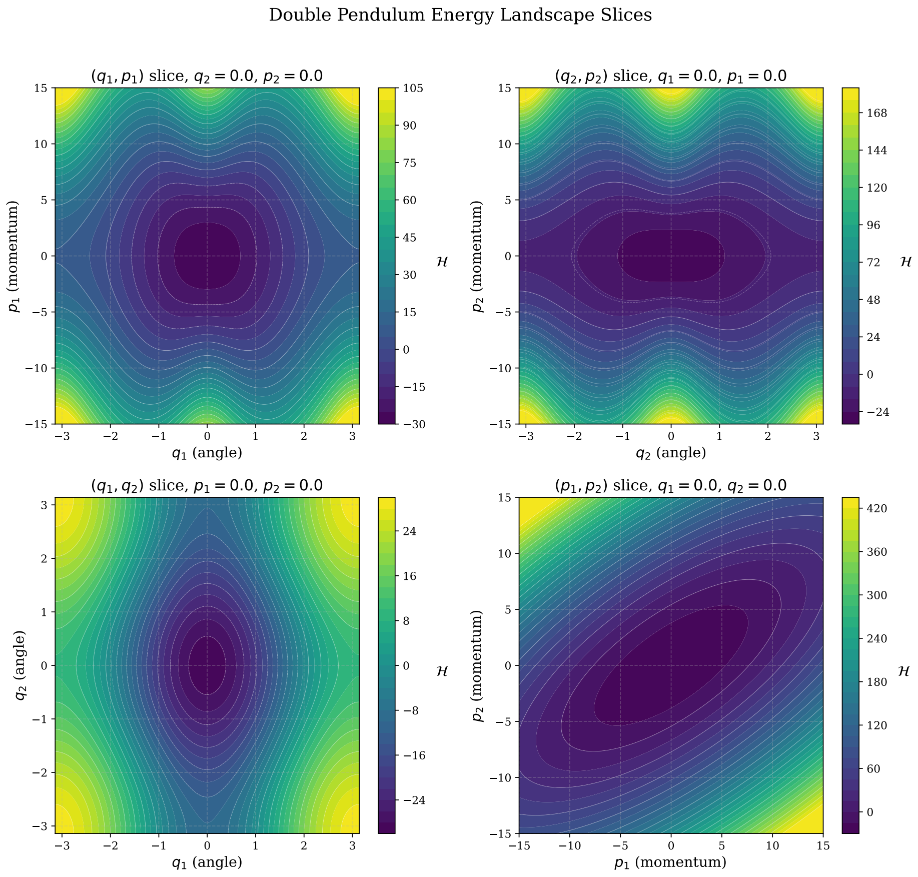


Figure 3.25.: Sample Hamiltonian landscape slices of the double pendulum system in the phase space. These slices illustrate the complex Hamiltonian structure resulting from the interaction between the two pendulum arms.

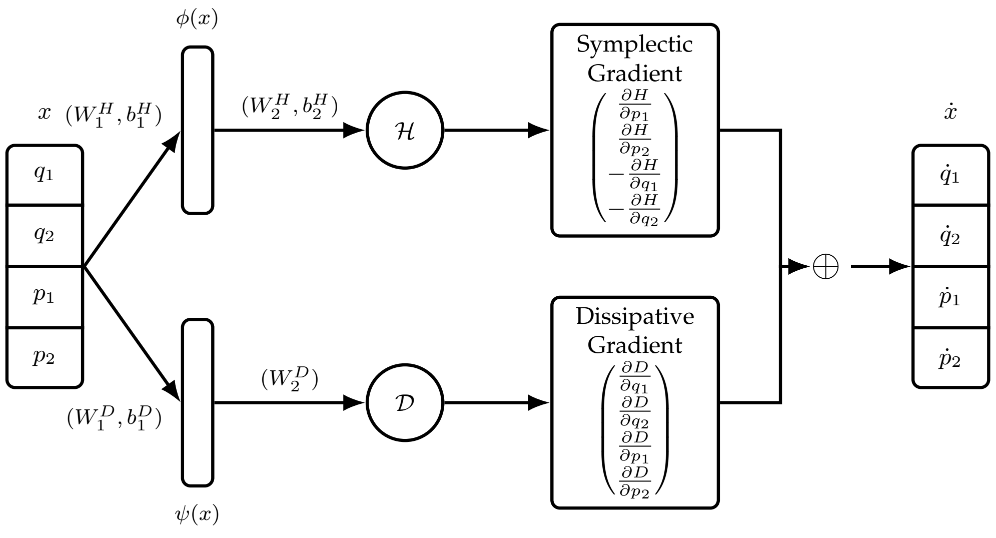

$\phi(x)$
x
(W_1^H, b_1^H)
(W_2^H, b_2^H)
$\mathcal{H}$
Symplectic Gradient
$\begin{pmatrix} \frac{\partial H}{\partial p_1} \\ \frac{\partial H}{\partial p_2} \\ -\frac{\partial H}{\partial q_1} \\ -\frac{\partial H}{\partial q_2} \end{pmatrix}$
$\dot{x}$
q_1
q_2
p_1
p_2
$\oplus$
$\dot{q}_1$
$\dot{q}_2$
$\dot{p}_1$
$\dot{p}_2$
(W_1^D, b_1^D)
(W_2^D)
$\mathcal{D}$
Dissipative Gradient
$\begin{pmatrix} \frac{\partial D}{\partial q_1} \\ \frac{\partial D}{\partial q_2} \\ \frac{\partial D}{\partial p_1} \\ \frac{\partial D}{\partial p_2} \end{pmatrix}$
$\psi(x)$

Figure 3.26.: Diagram of the RDHNN architecture for the double pendulum system. The input $x = (q_1, q_2, p_1, p_2)$ represents the two pendulum angles and their conjugate momenta. The conservative subnet outputs the Hamiltonian $\mathcal{H}$, whose symplectic gradient provides the conservative dynamics. The dissipative subnet outputs $\mathcal{D}$, whose gradient contributes damping.

models significantly struggle to learn the system dynamics where the test MSE is orders of magnitude higher than that of other systems investigated in this work. In both cases, all physics-based models perform similarly, with the MLP achieving lower error at smaller hidden layer widths. As the width increases, all models converge to a similar error. It is important to note that the physics-based models utilize the increasing hidden layer width better than the MLP, as the inductive bias of said models makes it less likely to overfit to the data and more likely to generalize well.

When the models were tested on their trajectory predictions when integrated over time, since the system is highly chaotic and the models have relatively high errors in their predictions, all models diverge from the true trajectory rapidly. Depending on the arbitrarily chosen initial conditions, their comparative performance varies greatly. This makes it difficult to draw meaningful conclusions from the predicted trajectories.

To further investigate the scaling behavior of SWIM relative to Adam on this challenging system, we compare training metrics at two representative network widths in Table 3.5. The results reveal distinct scaling behaviors between the two optimizers. Increasing the network width from $W = 80$ to $W = 1500$ acts as an effective mechanism for reducing error in SWIM-trained models. For the RDHNN, the error drops from approximately 15% to 3.8% which represents a nearly $4\times$ improvement. In contrast, Adam's error remains relatively stagnant at around 1.3%. As with the other experiments, Adam was trained with

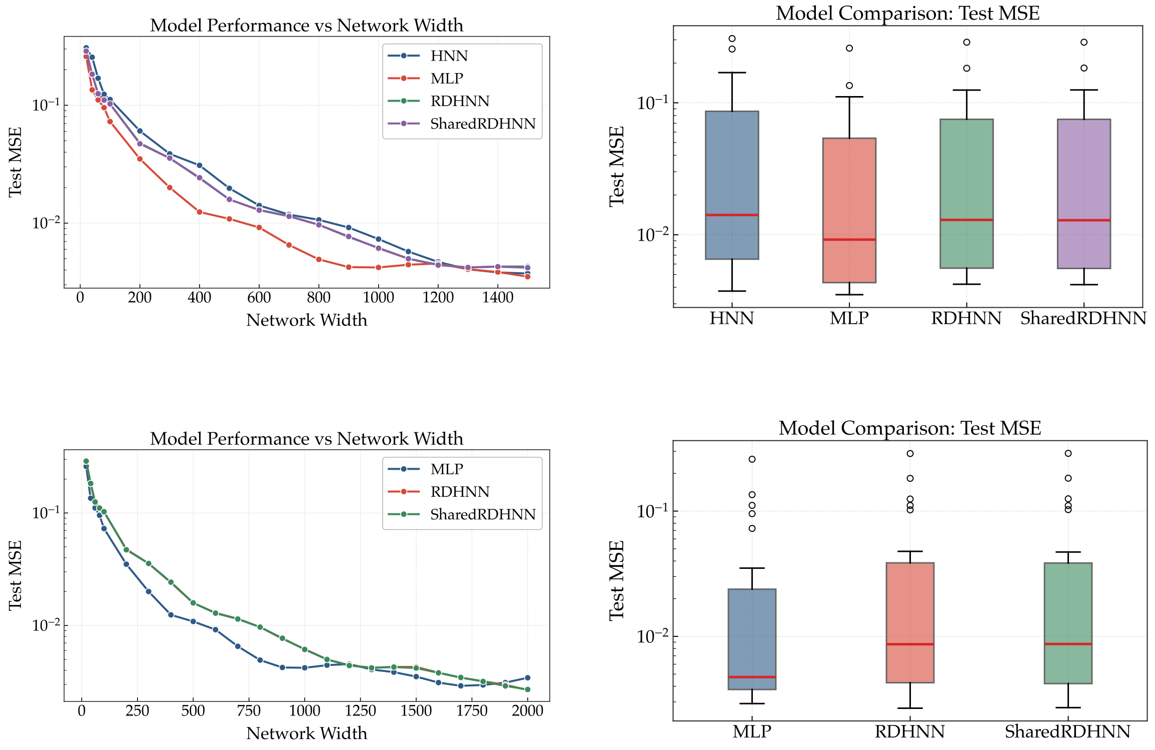


Figure 3.27.: Hidden layer width scaling mean squared error results and model comparison for the double pendulum system with the SWIM trainer. The top row shows the conservative system and the bottom row shows the dissipative system with a damping coefficient of 2. The experiments are performed with a dataset size of 10000 points.

Table 3.5.: Scalability analysis: comparison of training metrics for the double pendulum system at widths $W = 80$ and $W = 1500$. t denotes training time (seconds) and ϵ denotes relative test error.

Model	Width	t_{Adam} (s)	t_{SWIM} (s)	Speedup	ϵ_{Adam}	ϵ_{SWIM}
MLP	80	528.96	0.0194	2.7×10^4	1.7×10^{-2}	1.3×10^{-1}
MLP	1500	1269.81	0.4549	2.8×10^3	1.5×10^{-2}	4.0×10^{-2}
RDHNN	80	4262.23	0.1245	3.4×10^4	1.3×10^{-2}	1.5×10^{-1}
RDHNN	1500	6021.06	2.2245	2.7×10^3	1.3×10^{-2}	3.8×10^{-2}
SharedRDHNN	80	4242.24	0.0748	5.7×10^4	1.8×10^{-2}	1.5×10^{-1}
SharedRDHNN	1500	3071.55	2.1638	1.4×10^3	1.5×10^{-2}	3.8×10^{-2}

a fixed budget of 2 million iterations and identical hyperparameters across all network widths. This fixed configuration may be insufficient for the larger models, and the stagnant error suggests that without width-specific tuning of learning rates or training duration, the iterative optimizer struggles to leverage the additional capacity.

At low widths ($W = 80$), SWIM exhibits an error roughly $10\times$ higher than Adam. However, in the over-parameterized regime ($W = 1500$), this gap narrows to a factor of approximately 3. This trend suggests that SWIM offers a uniquely scalable approach as it allows one to trade minimal computational time (on the order of seconds) for substantial gains in accuracy through wider networks. The persistent accuracy gap can be attributed to the random feature approximation inherent to SWIM. Sampled activations in the hidden layer may not be well-suited to capture the complex, highly nonlinear dynamics of the chaotic double pendulum system.

Despite SWIM's lower accuracy, the speedup factors remain striking. Even for the largest networks ($W = 1500$), SWIM completes training in approximately 2 seconds compared to over an hour for Adam, representing speedups of 10^3–$10^4\times$. The double pendulum system thus represents a challenging benchmark where chaotic dynamics limit the effectiveness of randomly sampled features, yet SWIM's computational efficiency still enables rapid exploration of model architectures and hyperparameters that would be infeasible with gradient-based optimization.

Part IV.

Conclusion

4. Conclusion

This thesis investigated the application of random feature neural networks to learning dissipative Hamiltonian systems. Building on the Random HNN framework, we extended the approach to accommodate dissipation through the Random Dissipative Hamiltonian Neural Network architecture. The hidden layer parameters are sampled using the SWIM algorithm while the output layer weights are determined by solving a linear least squares problem. We additionally proposed the SharedRDHNN variant with a shared hidden layer for both the Hamiltonian and dissipation heads. The proposed methods were evaluated on systems of varying complexity, including the mass-spring system, single pendulum, Morse and Duffing oscillators, and the chaotic double pendulum.

SWIM trained models achieve computational speedups of 10^3 to 10^5 times compared to Adam across all tested systems in the training time. On systems with moderate complexity such as the Morse and Duffing oscillators, SWIM trained models achieve superior accuracy with errors approaching machine precision at large network widths. The SharedRDHNN performs equivalently to the RDHNN when trained with SWIM, confirming that separate hidden layers are unnecessary under this sampling scheme.

A key advantage of the SWIM-based approach is the elimination of hyperparameter tuning. Unlike gradient-based methods that require careful selection of learning rates, batch sizes, and training durations for each network configuration, SWIM uses fixed hyperparameters determined solely by the activation function. The experiments also revealed that Adam's performance can stagnate or degrade at larger widths when hyperparameters are held fixed, whereas SWIM consistently benefits from increased network capacity.

Several directions emerge for future work. The current framework assumes autonomous systems without explicit time dependence. Extending the architecture to incorporate time as an input would enable learning of non-autonomous systems with external forcing, such as the driven Duffing oscillator. Additionally, a hybrid approach combining SWIM initialization with gradient-based fine-tuning could achieve the benefits of both methods, using rapid SWIM training for initial convergence followed by Adam for precision improvements. Extending the framework to systems with more degrees of freedom and investigating scaling behavior in higher dimensions present important directions for broader applicability. Finally, adapting the SWIM sampling approach to graph neural network architectures could enable efficient learning of multi-body Hamiltonian systems with interacting components.

Bibliography

[1] Christine Allen-Blanchette. Hamiltonian generative adversarial networks. *arXiv preprint arXiv:2308.11216*, 2023.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 3 edition, 1999.

[3] Vladimir I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer, New York, 2 edition, 1989.

[4] Luis Barreira and Claudia Valls. *Dynamical Systems: An Introduction*. Universitext. Springer London, London, 2013.

[5] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2018.

[6] Thomas Beckers, Jacob Seidman, Paris Perdikaris, and George J. Pappas. Gaussian process port-hamiltonian systems: Bayesian learning with physics prior. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, page 1447–1453. IEEE, December 2022.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[8] Tom Bertalan, Felix Dietrich, Igor Mezić, and Ioannis G. Kevrekidis. On learning hamiltonian systems from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(12):121107, December 2019. arXiv:1907.12715 [physics].

[9] Harsh Bhatia, Gregory Norgard, Valerio Pascucci, and Peer-Timo Bremer. The helmholtz–hodge decomposition—a survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(8):1386–1404, 2013.

[10] Erik Lien Bolager, Iryna Burak, Chinmay Datar, Qing Sun, and Felix Dietrich. Sampling weights of deep neural networks, November 2023.

[11] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT*, pages 177–186, 2010.

[12] Tomas Brauner. Analytical Mechanics and Field Theory, March 2025.

[13] Michael Brin and Garrett Stuck. Introduction to Dynamical Systems. 2002.

[14] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

[15] J Candy and W Rozmus. A symplectic integration algorithm for separable hamiltonian functions. *Journal of Computational Physics*, 92(1):230–256, 1991.

[16] Claudio Canuto, M. Yousuff Hussaini, Alfio Quarteroni, and Thomas A. Zang. *Spectral Methods: Fundamentals in Single Domains*. Springer, 2006.

[17] Junyi Chai, Hao Zeng, Anming Li, and Eric W.T. Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

[18] Tian Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, pages 6571–6583, 2018.

[19] Zhong Chen, Jing Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In *8th International Conference on Learning Representations (ICLR)*, 2020. arXiv preprint arXiv:1909.13334.

[20] K. Cherifi. An overview on recent machine learning techniques for port hamiltonian systems. *Physica D: Nonlinear Phenomena*, 411:132620, 2020.

[21] Amitava Choudhuri, Madan Mohan Panja, and Benoy Talukdar. On the solution and lagrangian representation of duffing oscillator with damping, 2025.

[22] David Cline. *Variational Principles in Classical Mechanics*. University of Rochester River Campus Libraries, 2017.

[23] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian Neural Networks, July 2020.

[24] Marco David and Florian Méhats. Symplectic Learning for Hamiltonian Neural Networks. *Journal of Computational Physics*, 494:112495, December 2023.

[25] Shaan Desai, Marios Mattheakis, David Sondak, Pavlos Protopapas, and Stephen Roberts. Port-Hamiltonian Neural Networks for Learning Explicit Time-Dependent Dynamical Systems. 104(3).

[26] Matthew Dixon and Sebastian Reich. Symplectic time-stepping for particle methods. *Journal / Conference / Report — unknown (preprint)*, 2004. Preprint available at `https://www.math.uni-potsdam.de/~sreich/04_1.pdf`.

[27] Denis Donnelly and Edwin Rogers. Symplectic integrators: An introduction. *American Journal of Physics*, 73(10):938–945, 10 2005.

[28] Thai Duong and Nikolay Atanasov. Hamiltonian-based neural ode networks on the se(3) manifold for dynamics learning and control, 2021.

[29] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. Springer, 3rd edition, 2002.

[30] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[31] Herbert Goldstein, Charles P. Poole, and John L. Safko. *Classical Mechanics*. Addison–Wesley, Boston, MA, 3 edition, 2002.

[32] David González, Francisco Chinesta, and Elías Cueto. Thermodynamically consistent data-driven computational mechanics. *Continuum Mechanics and Thermodynamics*, 31(1):239–253, 2019.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[34] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[35] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks, September 2019.

[36] Miroslav Grmela, Václav Klika, and Michal Pavelka. Gradient and generic evolution towards reduced dynamics. *arXiv preprint arXiv:1912.07693*, 2019.

[37] Miroslav Grmela and Hans Christian Öttinger. Dynamics and thermodynamics of complex fluids. i. development of a general formalism. *Physical Review E*, 56(6):6620–6632, 1997.

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

[39] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Symplectic integration of hamiltonian systems. In *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer, Berlin, Heidelberg, 2006.

[40] William Rowan Hamilton. On a general method in dynamics. *Philosophical Transactions of the Royal Society*, 124:247–308, 1834.

[41] William Rowan Hamilton. Second essay on a general method in dynamics. *Philosophical Transactions of the Royal Society*, 125:95–144, 1835.

[42] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[43] Quercus Hernández, Alberto Badías, Francisco Chinesta, and Elías Cueto. Port-metriplectic neural networks: Thermodynamics-informed machine learning of complex physical systems. *Computational Mechanics*, 72(3):553–561, September 2023.

[44] Quercus Hernández, Alberto Badías, Francisco Chinesta, and Elías Cueto. Thermodynamics-informed graph neural networks. *IEEE Transactions on Artificial Intelligence*, 5(3):967–976, March 2024.

[45] Quercus Hernández, Alberto Badias, David Gonzalez, Francisco Chinesta, and Elias Cueto. Structure-preserving neural networks. *Journal of Computational Physics*, 426:109950, February 2021.

[46] Geoffrey Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. Lecture 6.5 of Neural Networks for Machine Learning, Coursera, 2012.

[47] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 985–990 vol.2, 2004.

[48] Shenglin Huang, Zequn He, Bryan Chen, and Celia Reina. Variational Onsager Neural Networks (VONNs): A thermodynamics-based variational learning strategy for non-equilibrium PDEs. *Journal of the Mechanics and Physics of Solids*, 163:104856, June 2022.

[49] Raban Iten, Tony Metger, Henrik Wilming, Lídia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *Physical Review Letters*, 124(1):010508, 2020.

[50] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307, 1988.

[51] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, February 2021.

[52] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[53] Z.-Y. Khoo, J. S. C. Low, and S. Bressan. Separable hamiltonian neural networks. *arXiv preprint arXiv:2309.01069*, 2023.

[54] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[55] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[56] Vladimír Krajňák and Stephen Wiggins. Dynamics of the morse oscillator: Analytical expressions for trajectories, action-angle variables, and chaotic dynamics. *International Journal of Bifurcation and Chaos*, 29(11):1950157, October 2019.

[57] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.

[58] Zichao Long, Yiping Lu, and Bin Dong. PDE-Net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[59] Lu Lu, Pengzhan Jin, Guofei Pang, Zongyi Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218–229, 2021.

[60] B.M. Maschke and A.J. van der Schaft. Port-controlled hamiltonian systems: Modelling origins and systemtheoretic properties. *IFAC Proceedings Volumes*, 25(13):359–365, 1992. 2nd IFAC Symposium on Nonlinear Control Systems Design 1992, Bordeaux, France, 24-26 June.

[61] E. Minguzzi. Rayleigh's dissipation function at work. *European Journal of Physics*, 36(3):035014, 2015.

[62] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T. Dudley. Deep learning for healthcare: Review, opportunities and challenges. *Briefings in Bioinformatics*, 19(6):1236–1246, November 2018.

[63] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv e-prints*, December 2013.

[64] P. M. Morse. Diatomic molecules according to the wave mechanics. ii. vibrational levels. *Physical Review*, 34(1):57, 1929.

[65] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[66] Christian Offen and Sina Ober-Blöbaum. Symplectic integration of learned Hamiltonian systems. 32(1).

[67] Houman Owhadi and Gene Ryan Yoo. Kernel flows: From learning kernels from data into the abyss. *Journal of Computational Physics*, 389:22–47, July 2019.

[68] Michael E. Peskin and Daniel V. Schroeder. *An Introduction to Quantum Field Theory*. Addison–Wesley, Reading, MA, 1995. Hamiltonian density and canonical quantization in QFT.

[69] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning Mesh-Based Simulation with Graph Networks, June 2021.

[70] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[71] Hong Qin, Jian Liu, Jianyuan Xiao, Ruili Zhang, Yajuan He, Yang Wang, Yan Sun, Joshua W. Burby, Laura Ellison, and Ying Zhou. Canonical symplectic particle-in-cell method for long-term large-scale simulations of the vlasov–maxwell equations. *Nuclear Fusion*, 56(1):014001, 2015.

[72] Atamert Rahma. Sampling Neural Networks to Approximate Hamiltonian Functions. May 2024.

[73] Atamert Rahma, Chinmay Datar, Ana Cukarska, and Felix Dietrich. Rapid training of Hamiltonian graph networks without gradient descent, June 2025.

[74] Atamert Rahma, Chinmay Datar, and Felix Dietrich. Training Hamiltonian neural networks without backpropagation, November 2024.

[75] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[76] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[77] Levent Sagun, Utku Evci, V. Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks, 2018.

[78] Rick Salmon. Hamiltonian fluid mechanics. *Annual Review of Fluid Mechanics*, 20(1):225–256, 1988.

[79] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian Graph Networks with ODE Integrators. (arXiv:1909.12790), September 2019.

[80] J. M. Sanz-Serna. Symplectic integrators for hamiltonian problems: an overview. *Acta Numerica*, 1:243–286, 1992.

[81] Jesús María Sanz-Serna. Symplectic integrators for hamiltonian problems: An overview. *Acta Numerica*, 1:243–286, 1992.

[82] Sheikh Saqlain, Wei Zhu, Efstathios G. Charalampidis, and Panayotis G. Kevrekidis. Discovering governing equations in discrete systems using physics-informed neural networks. *arXiv e-prints*, 2022. arXiv:2212.00971.

[83] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[84] W.F. Schmidt, M.A. Kraaijveld, and R.P.W. Duin. Feedforward neural networks with random weights. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4, 1992.

[85] David Shepherd, James Miles, Matthias Heil, and Milan Mihajlović. An adaptive step implicit midpoint rule for the time integration of newton's linearisations of non-linear problems with applications in micromagnetics. *Journal of Scientific Computing*, 80(2):1058–1082, August 2019.

[86] Mohsen Soori, Behrooz Arezoo, and Roza Dastres. Artificial intelligence, machine learning and deep learning in advanced robotics, a review. *Cognitive Robotics*, 3:54–70, 2023.

[87] Andrew Sosanya and Sam Greydanus. Dissipative Hamiltonian Neural Networks: Learning Dissipative and Conservative Dynamics Separately, January 2022.

[88] Steven Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. A Chapman & Hall Book. CRC Press, Boca Raton London New York, second edition, first issued in hardback edition, 2019.

[89] Sauro Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001.

[90] P. N. Suganthan and R. Katuwal. On the origins of randomization-based feedforward neural networks. *Applied Soft Computing*, 105:107239, 2021.

[91] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning (ICML)*, 2013.

[92] Molei Tao. Explicit symplectic approximation of nonseparable hamiltonians: Algorithm and long time performance. *Physical Review E*, 94(4), October 2016.

[93] Alicia Tierz, Iciar Alfaro, David González, Francisco Chinesta, and Elías Cueto. Graph neural networks informed locally by thermodynamics, January 2025.

[94] Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian Generative Networks, February 2020.

[95] Arjan van der Schaft and Dimitri Jeltsema. Port-hamiltonian systems theory: An introductory overview. *Foundations and Trends® in Systems and Control*, 1(2-3):173–378, 2014.

[96] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education, 2nd edition, 2007.

[97] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[98] Hermann von Helmholtz. Über integrale der hydrodynamischen gleichungen, welche den wirbelbewegungen entsprechen. *Journal für die Reine und Angewandte Mathematik*, 55:25–55, 1858.

[99] J.-L. Wu, H. Xiao, and E. Paterson. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *Journal of Computational Physics*, 406:109209, 2020.

[100] Shanshan Xiao, Jiawei Zhang, and Yifa Tang. Generalized Lagrangian Neural Networks, January 2024.

[101] Hao Xu and Jia Pan. Hhd-gp: Incorporating helmholtz-hodge decomposition into gaussian processes for learning dynamical systems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 67282–67318. Curran Associates, Inc., 2024.

[102] Haruo Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150(5):262–268, 1990.

[103] Rafik Zeraoulia and Chaima Zeraoulia. Exploring the duffing equation: Numerical analysis, discrete dynamics,and ecological modeling, 2023.

[104] Rui Zhang, Jian Liu, Yiheng Tang, Hong Qin, Jianyuan Xiao, and Ben Zhu. Canonicalization and symplectic simulation of the gyrocenter dynamics in time-independent magnetic fields. *Physics of Plasmas*, 21(3):032504, 2014.

[105] L. Zheng and X. Zhang. Chapter 8 - numerical methods. In Liancun Zheng and Xinxin Zhang, editors, *Modeling and Analysis of Modern Fluid Problems*, Mathematics in Science and Engineering, pages 361–455. Academic Press, 2017.

[106] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative SymODEN: Encoding Hamiltonian Dynamics with Dissipation and Control into Deep Learning, April 2020.

[107] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control, March 2024.

[108] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep Hamiltonian networks based on symplectic integrators, April 2020.

[109] Yinhao Zhu, Nicholas Zabaras, P. S. Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[110] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7th edition, 2013.

Appendix

A. Detailed Descriptions

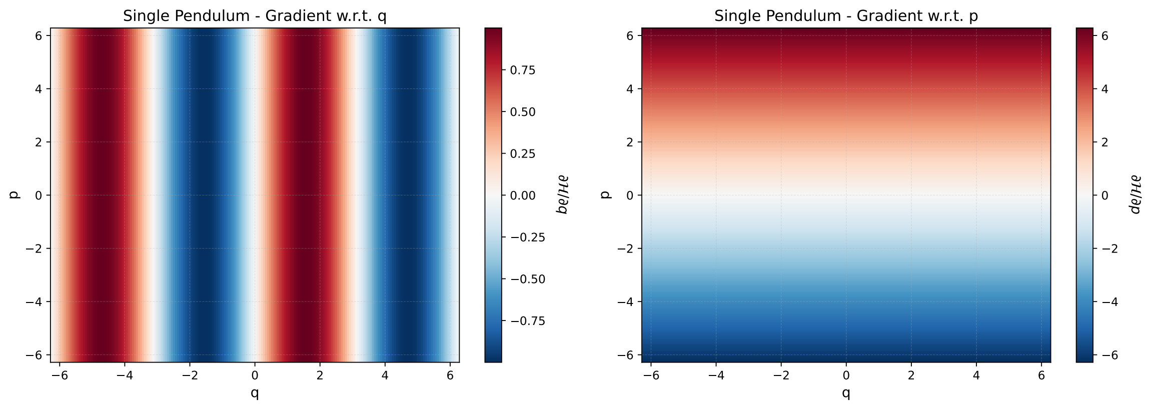


Figure A.1.: Gradient of the ground truth Hamiltonian for the single pendulum system in Equation (3.9).

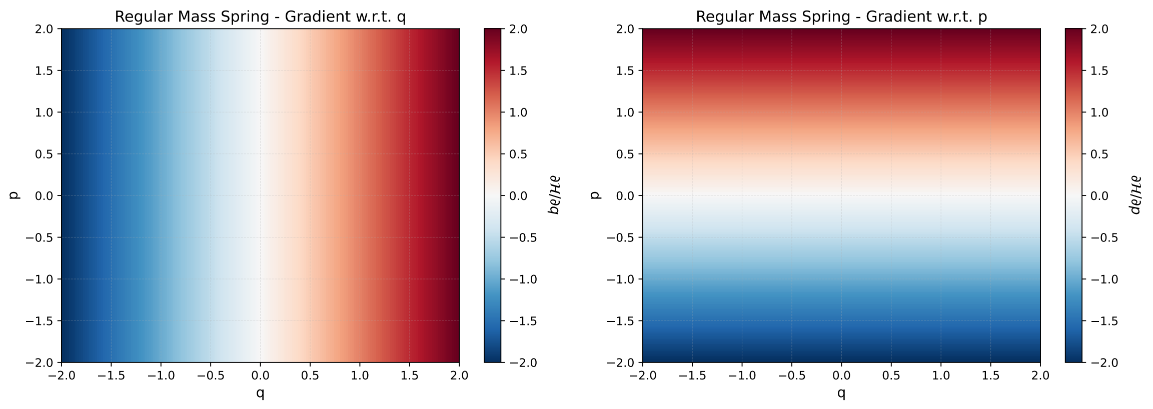


Figure A.2.: Gradient of the ground truth Hamiltonian for the mass spring system for the domain $[-2\pi, 2\pi] \times [-1, 1]$.

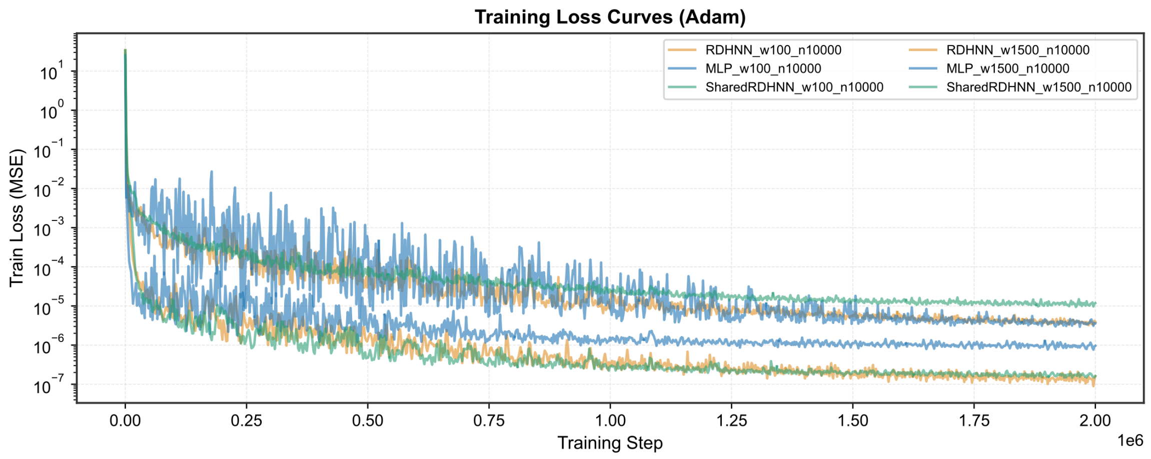


Figure A.3.: Loss curves for the dissipative single pendulum system for the domain $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ with a damping coefficient of 2.

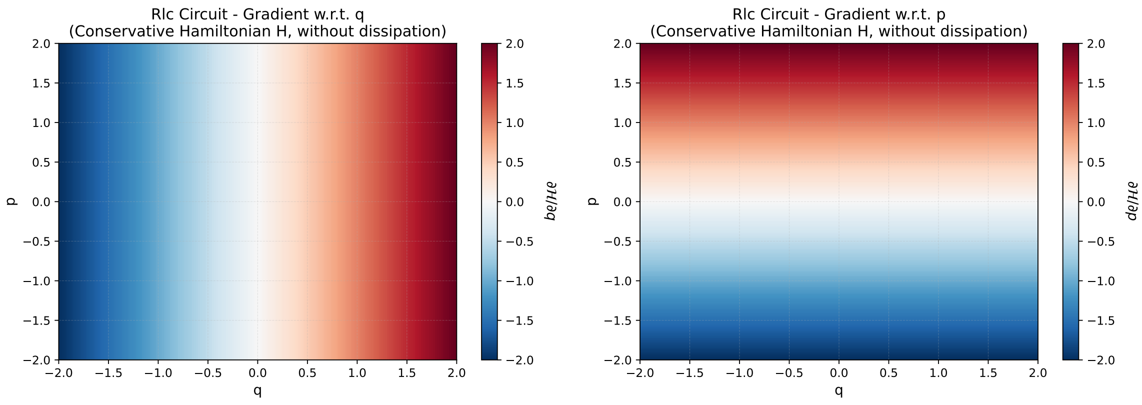


Figure A.4.: Gradient of the ground truth Hamiltonian for the RLC circuit for the domain $[-2, 2] \times [-2, 2]$.

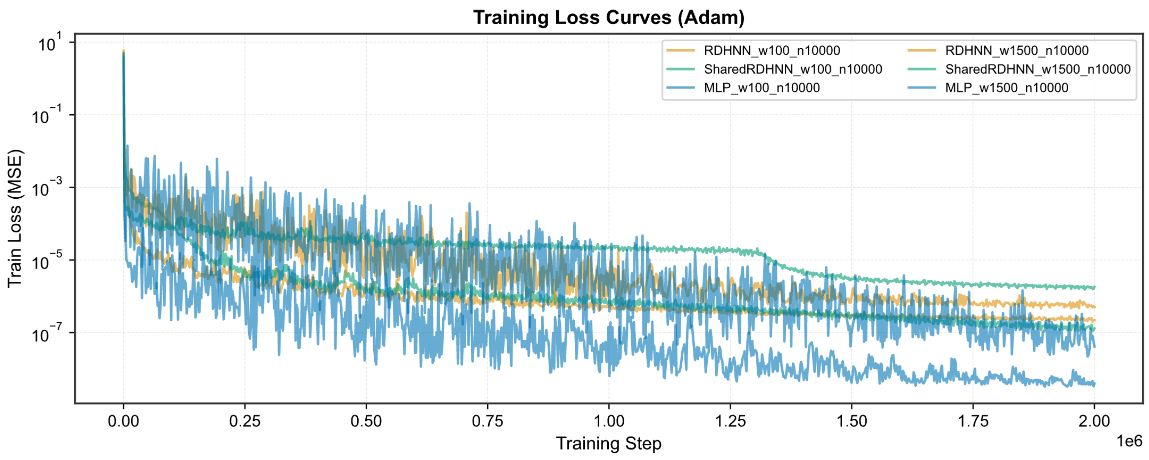


Figure A.5.: Loss curves for the dissipative mass-spring system for the domain $[-2, 2] \times [-2, 2]$ with a damping coefficient of 2.

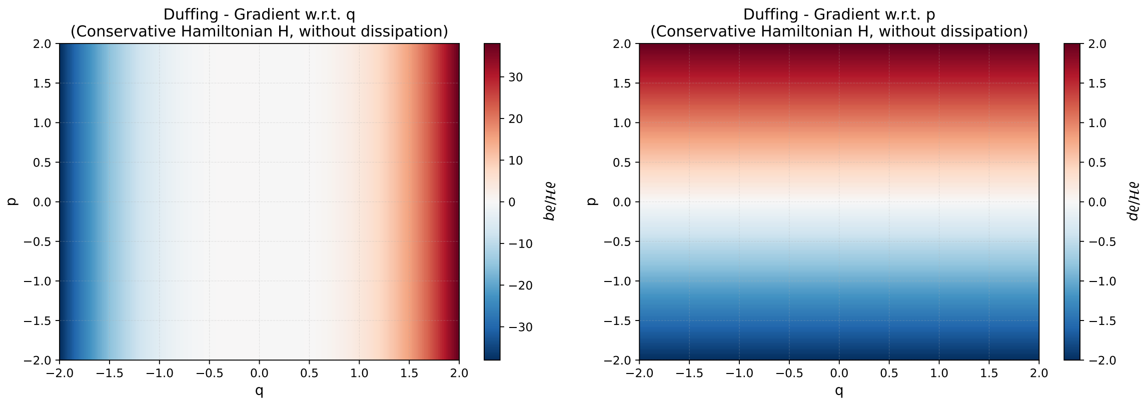


Figure A.6.: Gradient of the ground truth Hamiltonian for the Duffing oscillator from Equation (3.33).

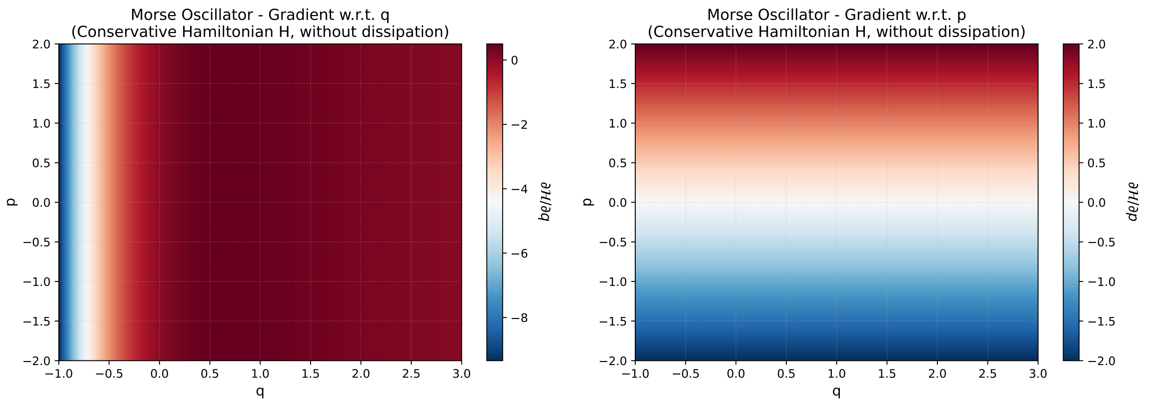


Figure A.7.: Gradient of the ground truth Hamiltonian for the Morse oscillator in the domain $q \in [-1, 3]$, $p \in [-2, 2]$ with the parameters from Equation (3.29).

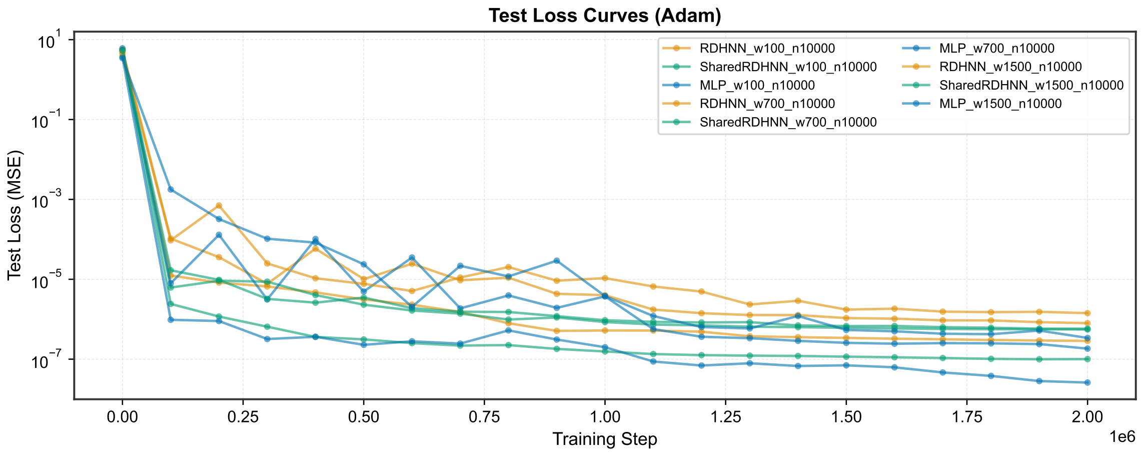


Figure A.8.: Comparison of loss curves for the damped Morse oscillator system from Equation (3.29) trained with Adam optimizer.

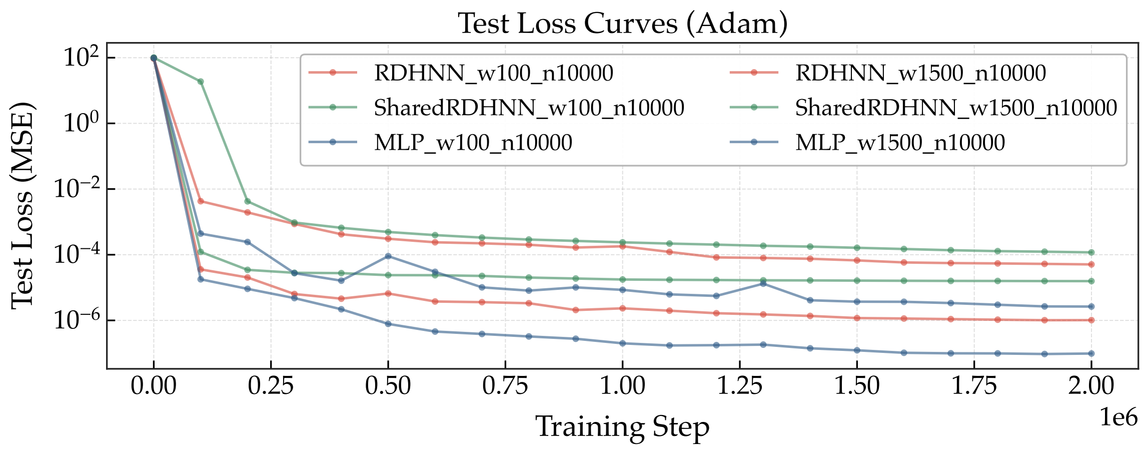


Figure A.9.: Loss curves for the dissipative Duffing oscillator system trained with the Adam optimizer.

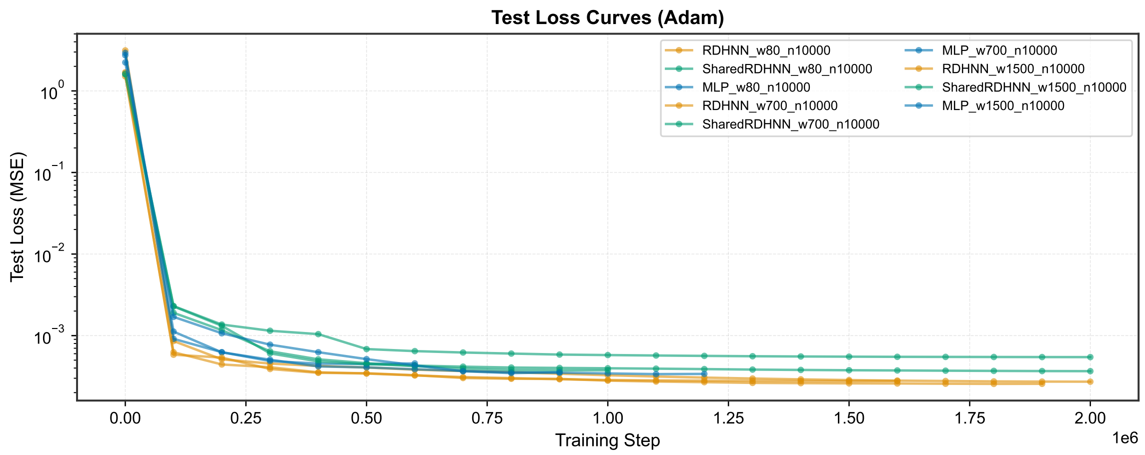


Figure A.10.: Loss curves for the double pendulum system with a damping coefficient of 2.