Learning KPZ Dynamics

Politecnico di Torino

Master in Physics of Complex Systems - International Track

Academic Year 2024-2025

Author:
Mensi Lorenzo

Supervisors:
Sergio Chibbaro, Alberto Rosso, Cyril Furtlehner

Internal Supervisor:
Alfredo Braunstein

July 23, 2025

Abstract

Non-linear stochastic processes are notoriously difficult to model, and inferring the dynamical equations from observations alone can be extremely challenging. To address this, our work develops an entirely data-driven Neural Network framework that learns a transform to linearize the system's dynamics. This is achieved by mapping the observations onto a latent space where the dynamical evolution is of a linear form. The result is a highly interpretable model, as the learned transformation can be related to known functions and the dynamics to corresponding linear operators. While neural network-based approaches have demonstrated considerable success in modeling deterministic dynamical systems, extending them to the stochastic regime represents a novel research frontier. We develop new ideas and validate them on the Kardar-Parisi-Zhang (KPZ) equation, a paradigmatic model for non-linear stochastic growth. This system is of particular interest because there is a known analytical solution, the Cole-Hopf transformation, which linearizes the dynamics. This allows for a rigorous comparison between our learnt components and the theoretical solution.

Contents

Introduction

At the heart of many complex systems lies the challenging interplay between non-linear dynamics and intrinsic randomness. A model for this class of phenomena is the Kardar-Parisi-Zhang (KPZ) equation [1]:

$$\partial_t h(x,t) = \nu \partial_x^2 h(x,t) + \frac{\lambda}{2}(\partial_x h(x,t))^2 + \eta(x,t). \tag{1}$$

The KPZ equation (1) is a cornerstone model in statistical mechanics for describing the kinetic roughening of growing interfaces and related non-equilibrium phenomena. Its significance extends across various physical contexts, including surface deposition [2], flame front propagation [3], colony growth in biology [4], directed polymers in random media [5], and even certain aspects of Burgers turbulence [6]. The equation is particularly interesting because it fundamentally captures the interplay between three competing physical effects: surface tension-driven smoothing ($\nu\partial_x^2 h$), non-linear growth dynamics ($\lambda(\partial_x h)^2$), and random fluctuations (η). As such, it serves as a paradigm for a broad universality class, characterizing systems far from equilibrium whose behavior exhibits universal scaling exponents.

Given its complexity, we take the KPZ equation as a typical example of a class of systems where the interplay between non-linearity and stochasticity makes their behavior difficult to model and predict, thus creating a need for powerful analytical and numerical tools. To model timeseries governed by complex dynamics, we propose a methodology that linearizes their evolution. The approach consists of mapping the data, $\vec{h}^t \in \mathbb{R}^N$ (from a discretized KPZ equation), onto a latent space, $\vec{z}^t \in \mathbb{R}^N$, which is designed specifically so that the dynamics within it are linear. For the KPZ equation, this can be done exactly via the Cole-Hopf transformation, $Z(x,t) = \exp\left(\frac{\lambda}{2\nu} h(x,t)\right)$ [7][8]. This analytical, local transformation maps the KPZ process into a simpler, linear (though with multiplicative noise) equation:

$$\partial_t Z(x,t) = \nu \partial_x^2 Z(x,t) + Z(x,t)\eta(x,t). \tag{2}$$

We leverage the expressive power of Neural Networks as universal function approximators to discover the transformations $\Phi : \vec{h}^t \to \vec{z}^t$ and its inverse $\Phi^{-1} : \vec{z}^t \to \vec{h}^t$, in a completely data driven way, without requiring prior knowledge of the specific equations that generate $\vec{h}^t$. These transformations form an autoencoder structure [9], with Φ acting as the encoder and Φ^{-1} as the decoder. Their purpose in this framework is not to advance the dynamics; rather, the encoder learns the latent representation of the data, and the decoder learns to map latent variables back to the original space. The dynamical

evolution is carried out by a separate linear module. This type of architecture has been successful for non-linear deterministic systems [10]. We develop new ideas to deal with the added challenge of stochasticity [11].

Initially, we aim to develop and validate our model within the framework of the KPZ system, where the existence of an analytical solution allows for a rigorous evaluation of its performance and limitations. The ultimate objective, however, is to create a general methodology applicable to systems where such simplifying transformations are not known, enabling the discovery of a linearizing transformation purely from data. A key advantage of this approach is that the resulting model is highly interpretable, as the learnt transformation and latent dynamics can be directly analyzed to gain analytical insight into the system.

Since the Cole-Hopf transform is local, i.e. the latent variable z_i^t only depends on the input h_i^t and not the other components of the vector $\vec{h}^t$, we choose to employ scalar Neural Networks. These networks take as input a real value h_i^t and produce as output another real value z_i^t. The model operates as follows:

- The initial conditions $\vec{h}^{t=0}$ are mapped onto the latent space using a Neural Network, which acts as the encoder Φ. The encoding is performed piece-wise, such that $\vec{z}^{\,t=0} = [\Phi(h_1^{t=0}), ..., \Phi(h_N^{t=0})]$.
- A dedicated module then produces the trajectory $[\vec{z}^{\,\Delta t}, ..., \vec{z}^{\,t}, \vec{z}^{\,t+\Delta t}, ...]$ in the latent space. The choice of how to model this latent space evolution is non-trivial. We propose a novel approach where the dynamics is represented as a linear Itô stochastic process [12].
- The latent states are then decoded using the decoder Φ^{-1}, implemented as another Neural Network. The statistics of the predictions $\hat{\vec{h}}^t = [\Phi^{-1}(z_1^t), ..., \Phi^{-1}(z_N^t)]$ should match those of the ground truth observations $\vec{h}^t$.

This thesis begins by introducing the fundamentals of Neural Networks (Chap. 1. We then introduce the physical system at the heart of our study in Chap. 2, which details the key features of the KPZ equation (Sect. 2.1) and the numerical procedure used to generate our dataset (Sect. 2.3).

Subsequently, in Chap. 3, we present the training of our model under two distinct conditions. The first scenario (Sect. 3.4) involves providing the model with the exact values of the noise $\vec{\eta}^{\,t}$ used to generate the ground truth dataset, effectively making the system's evolution deterministic. In the second, more challenging scenario (Sect. 3.5), we explore the case where these noise values are not provided, and the model must infer both the coordinate transformation and the latent dynamics in the presence of stochasticity.

Chapter 1

Neural Networks as Universal Function Approximators

1.1 Brief Introduction to Neural Networks

A Neural Network is a computational system designed to learn patterns and relationships directly from data. Rather than being programmed with explicit, task-specific rules, it learns by analyzing a large number of examples. This makes it a powerful and flexible model that can learn to approximate almost any relationship between a set of inputs and their corresponding outputs. The theoretical underpinning for this capability is the Universal Approximation Theorem [13], which states that a neural network with sufficient complexity has the potential to represent any continuous function. This guarantees that a suitable configuration of the network exists to solve a problem, and the challenge lies in finding it.

The fundamental building block of a neural network is the artificial *neuron*, or node. These nodes are simple processing units organized into a series of layers. A typical network consists of an input layer, which receives the initial data, one or more hidden layers, where the actual learning and feature abstraction occurs, and an output layer, which produces the final result. The information, often represented as a vector $\vec{x}$, flows from one layer to the next, undergoing a transformation at each step (Fig.1.1).

Each neuron in a layer receives inputs from neurons in the preceding layer. These connections are governed by *weights*, which are numerical values that signify the importance of each connection. A high weight means the input is considered more important, while a low weight means it is less so. The neuron sums up all these weighted inputs. This weighted sum is then passed through an *activation function*, which is a non-linear function that determines the neuron's output signal. This "firing" signal is then passed to the neurons in the subsequent layer. The weights are the critical tunable parameters of the network; the entire learning process is focused on finding the optimal values for these weights [14].

The process of finding these optimal weights is called training. It begins with the network making a guess, or prediction, for a given input. This guess is then compared to the known correct answer. To quantify how "wrong" the network's guess is, we use a *loss function*. This function calculates a score, known as the error or loss, which is high when the prediction is far from the

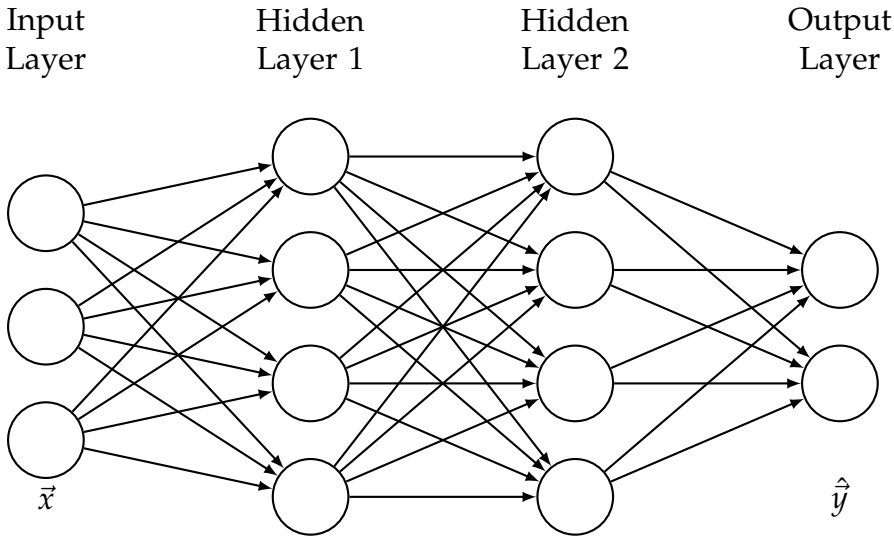


FIGURE 1.1: A diagram of a feed-forward neural network. The input vector $\vec{x}$ is passed to the input layer. Each node in the subsequent hidden layers computes a weighted sum of the outputs from the previous layer, with the arrows representing the weights. This sum is then passed through an activation function. The process repeats until the output layer produces the final prediction, $\hat{\vec{y}}$.

true value and low when it is close. The ultimate goal of training is to adjust the network's weights to make this loss as small as possible across all the training examples.

To minimize the loss, the network uses an automated process, most commonly an algorithm called *backpropagation*. This clever procedure works by calculating the contribution of each individual weight to the total error. It then adjusts each weight slightly in the direction that will most effectively reduce the error. This cycle of making a guess, calculating the loss, and adjusting the weights is repeated thousands or even millions of times. With each iteration, the network's predictions become progressively more accurate as its weights converge towards an optimal configuration. Through this iterative process of error minimization, the neural network learns the underlying mapping from inputs to outputs without ever being told the explicit rules.

1.2 The Mathematics of Neural Networks

To understand how a neural network learns, it's essential to delve into the mathematical operations that govern its behavior. The process can be broken down into two main phases: the *forward pass*, where the network makes a prediction, and the *backward pass* (backpropagation) [15], where it learns from its mistakes.

1.2.1 Forward Pass

The forward pass is the process of information flowing from the input layer through the hidden layers to the output layer. For a given input vector $\vec{x}$, the network computes an output, or prediction, $\hat{\vec{y}}$. Let's consider a single neuron j in layer l. Each neuron processes the information from the previous layer by first computing a weighted sum of all incoming signals to produce a value $z_j^{[l]}$, and then passing this value through a non-linear activation function to generate its own output signal, $a_j^{[l]} = g(z_j^{[l]})$.

The first operation within the neuron is to compute a weighted sum of its inputs. It is calculated as:

$$z_j^{[l]} = \sum_i w_{ji}^{[l]} a_i^{[l-1]} + b_j^{[l]}$$

where:

- $a_i^{[l-1]}$ is the activation (output) of the i-th neuron in the previous layer, $l-1$. For the first hidden layer, $a_i^{[0]}$ would be the i-th component of the input vector $\vec{x}$.
- $w_{ji}^{[l]}$ is the weight of the connection from the i-th neuron in layer $l-1$ to the j-th neuron in layer l.
- $b_j^{[l]}$ is the bias term for the j-th neuron in layer l. The bias allows the activation function to be shifted to the left or right, which can be critical for successful learning.

In a more compact vector notation, for the entire layer l, this can be written as:

$$\vec{z}^{[l]} = \mathbf{W}^{[l]} \vec{a}^{[l-1]} + \vec{b}^{[l]}$$

Here, $\mathbf{W}^{[l]}$ is the weight matrix for layer l, where each entry W_{ji} is the weight $w_{ji}^{[l]}$. $\vec{a}^{[l-1]}$ is the vector of activations from the previous layer, and $\vec{b}^{[l]}$ is the bias vector for the current layer.

Once the weighted sum $\vec{z}^{[l]}$ is computed, it is passed through a non-linear activation function, $g(\cdot)$, to produce the output of the layer, $\vec{a}^{[l]}$:

$$\vec{a}^{[l]} = g(\vec{z}^{[l]})$$

This function is applied element-wise to the vector $\vec{z}^{[l]}$. The choice of activation function is crucial; common examples include the Sigmoid, ReLU (Rectified Linear Unit), and Tanh functions. This process is repeated for each layer in the network until the final output layer is reached. The activation of the final layer, $\vec{a}^{[L]}$, is the network's prediction, $\hat{\vec{y}}$.

1.2.2 Backpropagation

After the forward pass, the network's prediction $\hat{\vec{y}}$ is compared to the true target value $\vec{y}$ using a loss function, $L(\hat{\vec{y}}, \vec{y})$. The goal of training is to minimize this loss. Backpropagation is the algorithm that computes the gradient of the loss function with respect to the network's parameters (weights and biases), allowing for their update in a direction that minimizes the loss.

At its core, backpropagation relies on the chain rule from calculus. The algorithm starts at the output layer and propagates the error gradient backward through the network. Let's start by computing the derivative of the loss with respect to the pre-activation of the output layer, $\vec{z}^{[L]}$:

$$\frac{\partial L}{\partial \vec{z}^{[L]}} = \frac{\partial L}{\partial \vec{a}^{[L]}} \odot g'(\vec{z}^{[L]})$$

where $\odot$ denotes the element-wise product, and $g'(\vec{z}^{[L]})$ is the derivative of the activation function evaluated at $\vec{z}^{[L]}$. This term, often denoted $\delta^{[L]}$, represents the "error" originating at each neuron in the output layer.

From this error term, we can find the gradients for the weights and biases of the final layer L. To find the gradient for a single weight $w_{ji}^{[L]}$, we use the chain rule:

$$\frac{\partial L}{\partial w_{ji}^{[L]}} = \frac{\partial L}{\partial z_j^{[L]}} \frac{\partial z_j^{[L]}}{\partial w_{ji}^{[L]}}$$

The first part, $\partial L / \partial z_j^{[L]}$, is simply the j-th component of the error vector $\delta^{[L]}$. The second part, $\partial z_j^{[L]} / \partial w_{ji}^{[L]}$, is the derivative of the weighted sum with respect to that weight, which is just the corresponding input activation from the previous layer, $a_i^{[L-1]}$. This means $\partial L / \partial w_{ji}^{[L]} = \delta_j^{[L]} a_i^{[L-1]}$.

To compute this for the entire weight matrix $\mathbf{W}^{[L]}$, we perform this operation for all j and i. This is captured concisely by the outer product of the error vector and the activation vector from the previous layer:

$$\frac{\partial L}{\partial \mathbf{W}^{[L]}} = \frac{\partial L}{\partial \vec{z}^{[L]}} (\vec{a}^{[L-1]})^T$$

For the biases, the derivation is simpler. The gradient of the loss with respect to the bias vector $\vec{b}^{[L]}$ is:

$$\frac{\partial L}{\partial \vec{b}^{[L]}} = \frac{\partial L}{\partial \vec{z}^{[L]}} \frac{\partial \vec{z}^{[L]}}{\partial \vec{b}^{[L]}} = \frac{\partial L}{\partial \vec{z}^{[L]}} \cdot 1 = \frac{\partial L}{\partial \vec{z}^{[L]}}$$

Since the bias is simply added to the weighted sum, its derivative is 1. Therefore, the gradient for the biases is just the error vector itself.

Next, the error is propagated to the previous layer, $L-1$. We compute the derivative of the loss with respect to the activation of layer $L-1$:

$$\frac{\partial L}{\partial \vec{a}^{[L-1]}} = (\mathbf{W}^{[L]})^T \frac{\partial L}{\partial \vec{z}^{[L]}}$$

This allows us to find the derivative with respect to the pre-activations of layer $L-1$:

$$\frac{\partial L}{\partial \vec{z}^{[L-1]}} = \frac{\partial L}{\partial \vec{a}^{[L-1]}} \odot g'(\vec{z}^{[L-1]})$$

This process is repeated, moving backward through the network. For any layer l, the general equations for backpropagation are:

$$\frac{\partial L}{\partial \vec{z}^{[l]}} = \frac{\partial L}{\partial \vec{a}^{[l]}} \odot g'(\vec{z}^{[l]}) = \left((\mathbf{W}^{[l+1]})^T \frac{\partial L}{\partial \vec{z}^{[l+1]}} \right) \odot g'(\vec{z}^{[l]})$$

And the gradients for the parameters of layer l are:

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \frac{\partial L}{\partial \vec{z}^{[l]}} (\vec{a}^{[l-1]})^T$$

$$\frac{\partial L}{\partial \vec{b}^{[l]}} = \frac{\partial L}{\partial \vec{z}^{[l]}}$$

Once these gradients have been computed for every layer, the network's parameters are updated to minimize the loss. The gradients point in the direction of the steepest ascent of the loss function, so to decrease the loss, the parameters are adjusted in the opposite direction. This is the core principle of optimization algorithms like *Gradient Descent*. Specifically, each parameter is updated according to the following rules:

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha \frac{\partial L}{\partial \mathbf{W}^{[l]}}$$

$$\vec{b}^{[l]} := \vec{b}^{[l]} - \alpha \frac{\partial L}{\partial \vec{b}^{[l]}}$$

where α is a hyperparameter known as the *learning rate* that controls the size of the update step. This iterative cycle of forward pass, backpropagation, and parameter updates is what allows the network to learn from the data.

1.3 Optimizing the Learning Process

The gradient descent update rule described previously provides a solid foundation for how a network learns. However, applying it naively presents a significant challenge in practice. The gradients are calculated by averaging the loss over the entire training dataset. For modern datasets, which can contain millions of examples, performing a full forward and backward pass over all

data just to make a single update to the parameters is computationally prohibitive. This method, known as *Batch Gradient Descent*, is therefore too slow for most real-world applications.

1.3.1 Mini-Batching

To solve this problem, we use a technique called *Mini-Batch Gradient Descent* [16]. Instead of using the entire dataset for each update, we compute the gradient on a small, random subset of the data called a "mini-batch" (e.g., 32, 64, or 128 examples). This provides an estimate, or a "noisy" approximation, of the true gradient. While this approximation isn't perfect, it's good enough to guide the parameters in the right general direction, and it's vastly more computationally efficient.

However, even with mini-batching, standard gradient descent has limitations. It uses a single, fixed learning rate (α) for all parameters, which can be difficult to tune. Furthermore, it can struggle with certain types of loss landscapes, such as long, narrow ravines, where it oscillates without making much progress toward the minimum.

1.3.2 The Adam Optimizer

To address the challenges of standard gradient descent, more sophisticated optimization algorithms have been developed. One of the most successful and widely used is the *Adam* optimizer [17], which stands for *Adaptive Moment Estimation*. Adam adapts the learning rate for each parameter individually and leverages past gradients to accelerate convergence. It achieves this by maintaining two exponentially decaying moving averages for each parameter:

1. **The First Moment (Momentum):** This is the moving average of the gradients, which acts like a ball rolling down a hill, accumulating momentum. It is represented by a vector $\vec{m}$ and updated at each timestep t as:

$$\vec{m}_t = \beta_1 \vec{m}_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial \mathbf{W}_t}$$

2. **The Second Moment (Adaptive Learning Rate):** This is the moving average of the *squared* gradients, which scales the learning rate on a per-parameter basis. It is represented by a vector $\vec{v}$:

$$\vec{v}_t = \beta_2 \vec{v}_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial \mathbf{W}_t} \right)^2$$

In these equations, β_1 and β_2 are hyperparameters that control the decay rates of the moving averages (typically close to 1, e.g., 0.9 and 0.999 respectively). The term $\frac{\partial L}{\partial \mathbf{W}_t}$ is the gradient of the loss with respect to the weights at the current timestep t. The squaring in the second moment equation is performed element-wise.

Because these moving averages are initialized as vectors of zeros, they are biased towards zero during the initial steps of training. Adam corrects for this bias by computing:

$$\hat{\vec{m}}_t = \frac{\vec{m}_t}{1-\beta_1^t} \quad \text{and} \quad \hat{\vec{v}}_t = \frac{\vec{v}_t}{1-\beta_2^t}$$

Finally, these bias-corrected estimates are used in the parameter update rule. The update for the weight matrix $\mathbf{W}$ is:

$$\mathbf{W}_{t+1} := \mathbf{W}_t - \alpha \frac{\hat{\vec{m}}_t}{\sqrt{\hat{\vec{v}}_t} + \epsilon}$$

Here, the division is performed element-wise: each component of the momentum vector $\hat{\vec{m}}_t$ is divided by the square root of the corresponding component of the adaptive learning rate vector $\hat{\vec{v}}_t$. The hyperparameter α is the initial learning rate, and ϵ is a very small number (e.g., 10^{-8}) to prevent division by zero. This combination of momentum and per-parameter learning rates makes Adam robust, efficient, and a default choice for training deep neural networks.

1.4 Higher-Order Derivatives and Automatic Differentiation

While first-order gradients are the foundation of network training, higher-order derivatives can also provide crucial insights.

A practical solution to compute them is *Automatic Differentiation* (AD) [18]. AD works by first deconstructing any computation into a sequence of elementary operations (e.g., addition, multiplication, exponential) stored in a computational graph. It then computes derivatives by systematically applying the chain rule to this graph. There are two primary modes:

1. Forward Mode: This mode computes the derivative by traversing the graph from inputs to outputs. It calculates how a small change in a single input affects every node in the graph. It is efficient for functions with few inputs and many outputs.

2. Reverse Mode: This mode traverses the graph from the final output back to the inputs. It efficiently calculates how a single output (like the loss) is affected by every input and parameter. Backpropagation is an application of reverse-mode AD.

The power of AD frameworks is that the function that computes the gradient is itself just another function within the computational graph. This means we can re-apply AD to the gradient computation itself to get second-order derivatives. For example, by running reverse-mode AD on the loss function, we obtain the gradient vector. By then running another pass of AD on the gradient computation, we can obtain second-order information.

Crucially, the power of AD is not limited to derivatives with respect to parameters. It can also compute derivatives of the network's output with respect to its inputs (e.g., $\frac{\partial \hat{y}}{\partial x}$). This capability is the foundation for advanced applications like *Physics-Informed Neural Networks* (PINNs) [19], where a model is trained to satisfy a known differential equation by directly penalizing its derivatives' behavior in the loss function.

Chapter 2

Integrating the KPZ equation

2.1 Theoretical overview

In 1+1 dimensions (one spatial dimension x and one temporal dimension t), the Kardar-Parisi-Zhang (KPZ) equation [1] describes the evolution of an interface height $h(x,t)$ as:

$$\partial_t h(x,t) = \nu \partial_x^2 h(x,t) + \frac{\lambda}{2}(\partial_x h(x,t))^2 + \eta(x,t). \tag{2.1}$$

Here, $h(x,t)$ is the height of the interface at position x and time t (Fig. 2.1) . The term $\nu > 0$ is a coefficient related to surface tension, promoting smoothing of the interface, analogous to viscosity. The parameter λ is the prefactor of the non-linear term, $\frac{\lambda}{2}(\partial_x h(x,t))^2$. This term is crucial as it represents the growth component normal to the interface [20]. If $\lambda = 0$ the average height of the interface remains constant, and (2.1) reduces to the Edward-Wilkinson (EW) equation [21]. Finally, $\eta(x,t)$ is a stochastic noise term, typically assumed to be Gaussian white noise, uncorrelated in space and time, with zero mean and correlator:

$$\langle \eta(x,t) \rangle = 0, \quad \langle \eta(x,t)\eta(x',t') \rangle = 2T\delta(x-x')\delta(t-t'),$$

where $\langle \dots \rangle$ denotes the average over thermal fluctuations, and T quantifies the noise strength.

A central theme of this work is to map non-linear stochastic processes onto linear ones. In the case of the KPZ equation, this can be done exactly via the Cole-Hopf transformation [7][8]:

$$Z(x,t) = \exp\left(\frac{\lambda}{2\nu} h(x,t)\right), \tag{2.2}$$

which maps 2.1 onto a stochastic diffusion process with multiplicative noise:

$$\partial_t Z(x,t) = \nu \partial_x^2 Z(x,t) + Z(x,t)\eta(x,t). \tag{2.3}$$

In this report, we investigate the evolution of an ensemble $\{[h(x,t)]_{t=0}^{t=\tau}\}^B$ of B trajectories, each evolving from $t = 0$ to $t = \tau$. Every trajectory starts from a flat initial condition, i.e., $h(x, t = 0) = 0$, $\forall x$. We consider the case of interfaces of finite length L, $x \in [0, L]$, and we employ Periodic Boundary

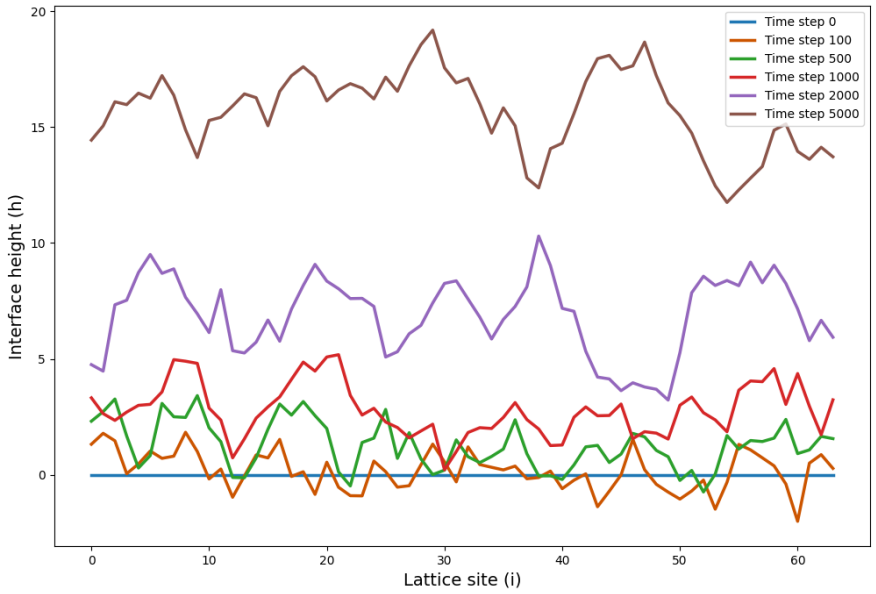


FIGURE 2.1: Example evolution of a discretized KPZ interface $\vec{h}^t$, starting from a flat initial condition $\vec{h}^{t=0} = \vec{0}$. As the non-linearity is present ($\lambda = 1$), the average level of the interface grows in time. In the early stages, the width (variance) grows as a power-law. The interface was discretized using $N = 64$ lattice points.

Conditions (P.B.C), $h(0,t) = h(L,t)$. As the equation has a stochastic nature, our primary focus is on statistically relevant quantities that characterize the growth process.

We define the spatial average of the height at a given time t of an interface as: $\overline{h(x,t)} = \frac{1}{L}\int_0^L h(x,t)\,dx$. The centered moments of the height distribution for a given interface are defined as:

$$w_n[h(x,t)] = \frac{1}{L}\int_0^L (h(x,t) - \overline{h(x,t)})^n\,dx$$

.

To obtain robust statistics, we perform an ensemble average over the B different interface realizations at time t: $\overline{h}(t) = \langle\overline{h(x,t)}\rangle$, $w_n(t) = \langle w_n[\vec{h}^t]\rangle$. This ensemble (thermal) averaging procedure smooths out the stochastic fluctuations inherent in a single growth process, revealing the underlying universal behavior.

We are mainly interested in the evolution of the interface width, or roughness, which is the second moment $w_2(t)$ (Fig. 2.2). This quantity captures the dynamics of the interface fluctuations. The early time behavior of the width is expected to follow a power law in time, $w_2(t) \sim t^{2\beta}$, where β is the growth exponent. The value of β depends on the underlying growth mechanism. For the KPZ equation $\beta = 1/3$, resulting in a width that scales as $w_2(t) \sim t^{2/3}$. This value is universal, holding for a wide variety of models that share the KPZ equation's essential stochastic growth character.

At long times, for a finite system with periodic boundary conditions, the

interface width saturates to a steady-state value, w_2^{sat}, which depends on the system size L. The saturation time, t_{sat}, also scales with L. For the KPZ equation $t_{sat} \sim L^{3/2}$. For the specific parameters $\nu = T = 1/2$, the saturation value of the KPZ process is expected to plateau at $w_2(t \to +\infty) = w_2^{sat} = L/12$. [22].

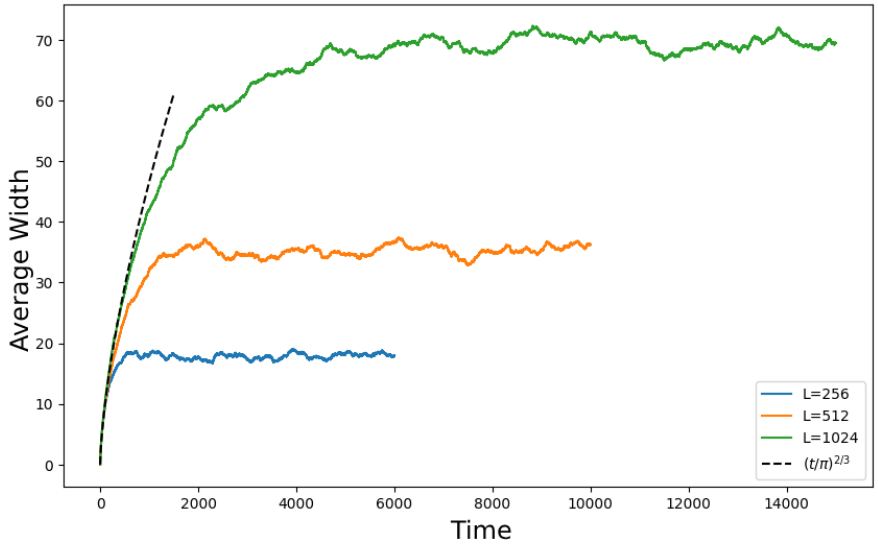
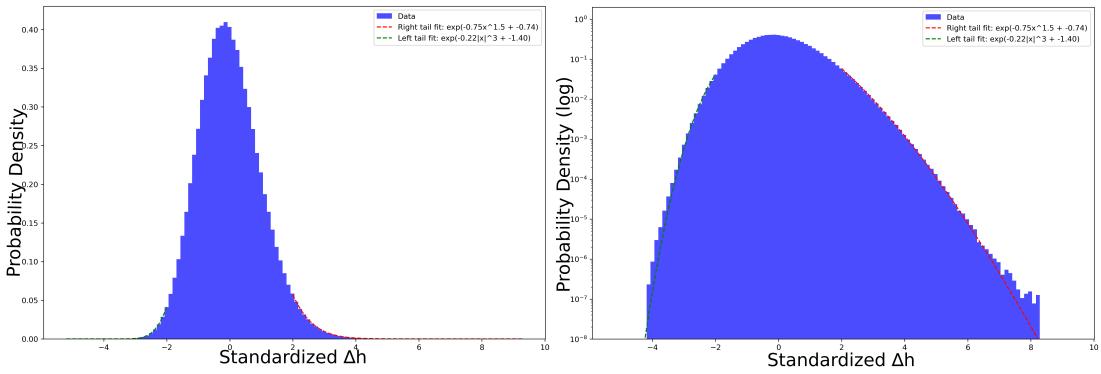


FIGURE 2.2: Simulation: evolution of the width $w_2(t)$ (variance) for KPZ interfaces with different lengths L. The early time evolution is characterized by a power-law with exponent $2/3$. As the simulated systems have finite size, w_2 eventually saturates to a plateau value w_2^{sat}, which depends on the system size. The time t_{sat} to reach the plateau is also proportional to the system's size, scaling as $L^{3/2}$. The time axis is measured in arbitrary units (a.u.), where one a.u. corresponds to $1/\Delta t$ timesteps.

2.2 Tracy-Widom distribution

The Tracy–Widom distribution, first derived in the context of random matrix theory by Tracy and Widom [23], plays a fundamental role in the study of the Kardar–Parisi–Zhang (KPZ) universality class. In the one-dimensional KPZ equation, the height fluctuations under proper rescaling have been shown to follow a Tracy–Widom distribution [24, 25]. Unlike the Gaussian distribution, which exhibits symmetric exponential decay in its tails, the Tracy–Widom distribution is highly non-Gaussian and features strongly asymmetric tails [26]. Specifically, the left tail (corresponding to rare, unusually low values) exhibits super-exponential decay, given by $\approx \exp\left(-|x|^3\right)$ whereas the right tail (corresponding to extreme positive fluctuations) follows a slower stretched exponential decay of the form $\approx \exp\left(-x^{\frac{3}{2}}\right)$ This asymmetry and deviation from Gaussian behavior reflect the underlying non-equilibrium nature of the KPZ growth process and its deep connection to random matrix theory [27]. The emergence of the Tracy–Widom distribution in KPZ systems highlights the universality of

extreme fluctuations in correlated stochastic processes. The random variable $\Delta h = h(x,t) - h(x,t')$ should exhibit these peculiar tail features [28].



(A) Empirical distribution of standardized Δh with tail fits

(B) Semi-log plot of the distribution of the standardized Δh

FIGURE 2.3: Simulation: distribution of the variable Δh, standardized to have a mean of 0 and a variance of 1. This procedure preserves the characteristic asymmetric shape of the Tracy-Widom distribution found in the KPZ universality class. The distribution exhibits asymmetric tails, with the left tail decaying faster than the right. This is consistent with the expected asymptotic behaviors, which follow the form $\approx \exp\left(-c_1|x|^3\right)$ for the left tail and $\approx \exp\left(-c_2 x^{3/2}\right)$ for the right.

2.3 Numerical Integration

For numerical integration, Eq. (2.1) must be discretized in both space and time. We consider the interface at a set of N discrete points $x_i = (i-1)\Delta x$ for $i = 1, \ldots, N$, where $\Delta x = L/N$ is the spatial discretization step. The height at these points at time t is denoted by $h_i^t \equiv h(x_i, t)$. We therefore denote the (column) state vector of the interface at time t as $\vec{h}^t = [h_1^t, \ldots, h_N^t] \in \mathbb{R}^N$. The spatial averages are computed by summing over the lattice points, $\overline{h(x,t)} \approx \frac{1}{N}\sum_{i=1}^{N} h_i^t$, while ensemble averages are computed by summing over the interfaces which share the same time index t, $w_n(t) = \langle w_n[\vec{h}^t] \rangle \approx \frac{1}{B}\sum_{\vec{h}^t} w_n[\vec{h}^t]$. For an explicit time-stepping scheme, all spatial derivatives and terms on the right-hand side of the discretized equation (which determine the evolution from $t - \Delta t$ to t) are evaluated at the previous time step, $t - \Delta t$.

The Laplacian, $\partial_x^2 h$, acting on the discretized height $h_i^{t-\Delta t}$ is approximated using a standard central difference scheme:

$$\partial_x^2 h_i^{t-\Delta t} = \frac{h_{i+1}^{t-\Delta t} + h_{i-1}^{t-\Delta t} - 2h_i^{t-\Delta t}}{\Delta x^2}.$$

The discretization of the non-linear term $(\partial_x h)^2$ is more delicate. We consider two alternatives:

- Standard central discretization:

$$(\partial_x h_i^{t-\Delta t})^2 = \left(\frac{h_{i+1}^{t-\Delta t} - h_{i-1}^{t-\Delta t}}{2\Delta x} \right)^2 \tag{2.4}$$

- Symmetric three-point discretization, proposed by Marguet et al. [29], to better preserve underlying symmetries of the KPZ plateau state:

$$(\partial_x h_i^{t-\Delta t})^2 = \frac{1}{3\Delta x^2} \left[(h_{i+1}^{t-\Delta t} - h_i^{t-\Delta t})^2 + (h_i^{t-\Delta t} - h_{i-1}^{t-\Delta t})^2 + (h_{i+1}^{t-\Delta t} - h_i^{t-\Delta t})(h_i^{t-\Delta t} - h_{i-1}^{t-\Delta t}) \right] \tag{2.5}$$

These two discretizations yield different numerical behaviors. For instance, if $h_{i-1}^{t-\Delta t} = h_{i+1}^{t-\Delta t}$, the central approximation (Eq. 2.4) interprets $h_i^{t-\Delta t}$ as an extremum, resulting in a zero derivative. The three-point scheme (Eq. 2.5) generally yields a non-zero value by averaging backward and forward derivative contributions. From our tests, the central discretization scheme often captures the KPZ $t^{2/3}$ behavior more accurately, even for shorter chains. Conversely, the three-point discretization tends to better represent the correct plateau statistics, independently of simulation parameters.

Finally, for time discretization, we employ an explicit Euler-Maruyama scheme [30]. The update rule for h_i^t from $h_i^{t-\Delta t}$ is:

$$h_i^t = h_i^{t-\Delta t} + \Delta t \left(\nu \partial_x^2 h_i^{t-\Delta t} + \frac{\lambda}{2} (\partial_x h_i^{t-\Delta t})^2 \right) + \sqrt{\frac{2T\Delta t}{\Delta x}} \mathcal{N}_i(0,1), \tag{2.6}$$

where $\mathcal{N}_i(0,1)$ are independent Gaussian random variables with zero mean and unit variance, sampled for each site i and each time step Δt. The coefficient $\sqrt{2T\Delta t/\Delta x}$ ensures that the discrete noise correctly approximates the continuous noise $\eta(x,t)$.

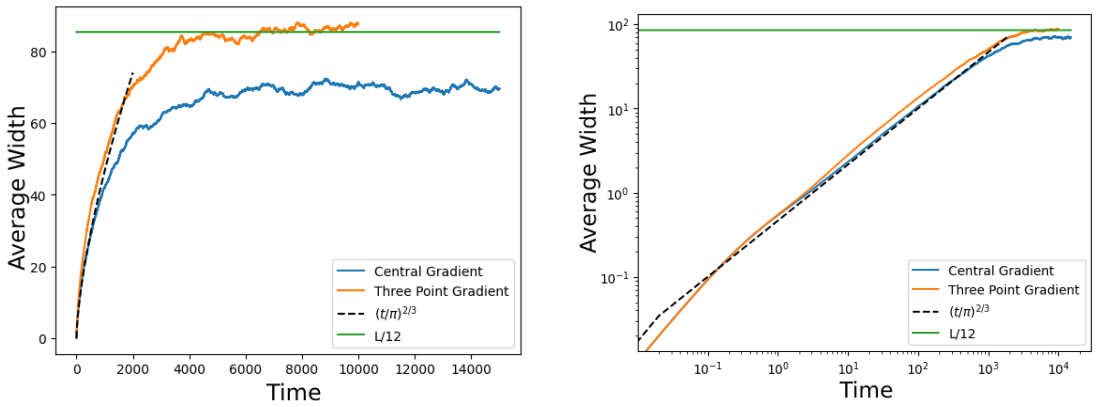
2.4 Numerical results

We numerically investigate the Kardar-Parisi-Zhang (KPZ) equation. Unless otherwise specified, simulations use parameters $\Delta x = 1$, $\Delta t = 0.02$, $\nu = T = 0.5$, under periodic boundary conditions.

Discretization of the non-linear term We investigate the differences between the Central discretization scheme (2.4) and the Three point approximation (2.5). A direct comparison of the two schemes, using identical parameters ($L = 1024, \lambda = 2$), reveals a fundamental trade-off (Figure 2.4).

The Three-Point Symmetric scheme excels at reproducing plateau statistics. As shown in Figure 2.4, the interface width $w_2(t)$ correctly saturates at the expected theoretical value $w_2^{\text{sat}} = L/12$. In contrast, the standard central difference scheme systematically underestimates this saturation plateau; increasing the value of λ leads to an even lower estimate of the saturation value, which should be independent of λ. Conversely, the standard scheme more faithfully

captures the KPZ $t^{2/3}$ regime. It consistently reproduces the expected early-time growth exponent even for shorter system sizes. The Three-Point scheme, while superior for plateau properties, shows less consistent early-time behavior, often exhibiting different effective exponents before potentially converging to the KPZ scaling regime.



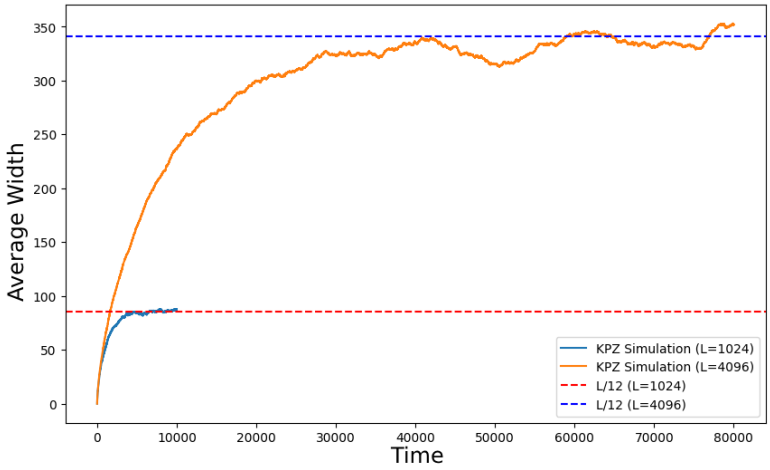
(A) Linear scale (B) Log-log scale

FIGURE 2.4: Time evolution of the interface width $w_2(t)$ (variance) for the standard Central difference (2.4) and Three-Point Symmetric (2.5) discretization schemes. The Three-Point scheme correctly predicts the saturation value, while the standard scheme better captures the early-time $t^{2/3}$ growth. The time axis is measured in arbitrary units (a.u.), where one a.u. corresponds to $1/\Delta t$ timesteps.

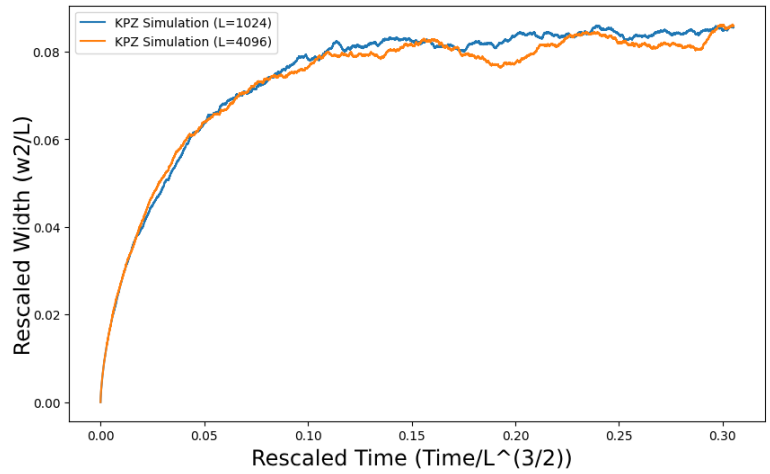
Effect of System Size (L): As expected, longer chains are better suited for studying scaling laws, as they provide a wider time window for the KPZ growth regime. Simulations with $L = 1024$ and $L = 4096$ (Figure 2.5) confirm that, in the case of the Three Point discretization (2.5), the saturation plateau is correctly reproduced regardless of system size. The data collapses well when rescaling the axes by w_2/L and $t/L^{3/2}$, consistent with theoretical scaling predictions.

Effect of Non-linearity (λ): For the standard central difference scheme, increasing λ was observed to lower the saturation plateau, an unexpected artifact. The Three-Point scheme corrects this, as the plateau value remains independent of λ. However, for large systems ($L = 4096$) and strong non-linearity ($\lambda = 2$), we observe an anomalous initial growth regime with an effective exponent greater than $2/3$.

Numerical Stability: Stability tests show that for $\lambda \leq 2$, the results are robust for the time steps used ($\Delta t = 0.02, \Delta t = 0.001$). However, as λ approaches 3, simulations with the larger time step ($\Delta t = 0.02$) become unstable and diverge, highlighting the limits of the numerical scheme.



(A) Linear Plot



(B) Rescaled plot

FIGURE 2.5: Interface width $w_2(t)$ for system sizes $L = 1024$ and $L = 4096$ using the Three-Point scheme. The rescaled plot (b) shows excellent data collapse, confirming the scaling relation $w_2 \sim L$ and $t \sim L^{3/2}$.

Chapter 3

Machine Learning Results

3.1 Main Goals and Learning Philosophy

In this chapter we detail the specifics of the model and how to train it. The model is comprised of two Feed Forward Neural Networks (FFNs) [31], which form our autoencoder structure, and a linear module, which evolves the states in latent space. We construct this module to be compatible with the expected analytical solution (2.3) given by the Cole-Hopf transform (2.2). The linear update rule in latent space is therefore given by

$$\vec{z}^{\,t+\Delta t} = \mathbf{K}\vec{z}^{\,t} + \mathbf{M}(\vec{z}^{\,t} \odot \vec{\eta}^{\,t}). \tag{3.1}$$

$\mathbf{K}$ and $\mathbf{M}$ represent learnable linear weights (matrices), while $\vec{\eta}^{\,t}$ is a vector of i.i.d. gaussian variables. $\odot$ denotes piece-wise product $\vec{z}^{\,t} \odot \vec{\eta}^{\,t} = [z_1^t \eta_1^t, ..., z_N^t \eta_N^t]$. The learnt values of these matrices directly affect the statistics of the generated trajectories. Specifically the matrix $\mathbf{K} - \mathbf{I}$ (where $\mathbf{I}$ is the Identity) is the drift matrix, determining the expected value of the increment $\Delta\vec{z}$, while the diffusion matrix $\mathbf{M}$ affects the covariance structure of the increments.

Our goal is to learn the invertible transformation $\vec{z}^{\,t} = \Phi(\vec{h}^t)$, its inverse $\vec{h}^t = \Phi^{-1}(\vec{z}^{\,t})$ and the latent matrices $\mathbf{K}$ and $\mathbf{M}$. We assume that the dataset $\mathbb{D}$ is comprised of B timeseries which are τ steps long $\{[\vec{h}^{t=0}, ..., \vec{h}^{t=\tau\Delta t}]\}^B, \vec{h}^t \in \mathbb{R}^N$. In its fully trained state the model is given the B initial conditions $\{\vec{h}^{t=0}\}^B$, and is able to reproduce a new dataset $\hat{\mathbb{D}}$ of predicted trajectories $\{[\hat{\vec{h}}^{t=0}, ..., \hat{\vec{h}}^{t=\tau\Delta t}]\}^B, \hat{\vec{h}}^t \in \mathbb{R}^N$ whose statistics should be close to those of $\mathbb{D}$.
The dataset represents the evolution of an ensemble of KPZ trajectories (Sect. 2.1).
We propose a novel loss function, L, specifically designed to focus on learning increments: $\Delta\vec{h}^t = \vec{h}^{t+\Delta t} - \vec{h}^t$. The idea is that the overall statistical patterns observed in the dataset (its global statistics) emerge from these local, step-by-step dynamics. Therefore, by studying the evolution of these increments, we focus the learning procedure on uncovering the fundamental process that generates the data.

The training procedure deliberately places greater emphasis on the decoder Φ^{-1}. This choice is justified by how trajectories are generated: the encoder is used only once to process the initial conditions of a trajectory. Following this, the system's dynamics are evolved for τ steps directly within the latent space, without requiring additional encoding. The decoder, however, must be used

at each of these τ steps to reconstruct the state for comparison with the ground truth dataset $\mathbb{D}$.

3.2 Architecture, Algorithms and Initialization

The choice of neural network architecture is critical for successfully finding a mapping which is consistent with the imposed latent dynamics. In our case, the Cole-Hopf transformation (Eq. 2.2) provides strong analytical guidance, allowing us to make a more informed choice. Since this transformation is local (point-wise) we employ a scalar Feed-Forward Network (FFN) [32] that maps a single input value to one output value. Our network is built of 15 hidden layers with 5 neurons each.
The selection of the activation function [33] is equally important. Given that the target transformation is exponential, we employed a function that can represent unbounded values while maintaining non-zero gradients for effective training. For this reason, we chose the Sigmoid Linear Unit (SiLU)defined as $\text{SiLU}(x) = \frac{x}{1+e^{-x}}$. This function behaves similarly as the ReLU (Fig.3.1) but has the advantages of being smooth and continuously differentiable. This property is crucial as it has a direct impact on the derivatives of the network, which play an important role in the loss definition.

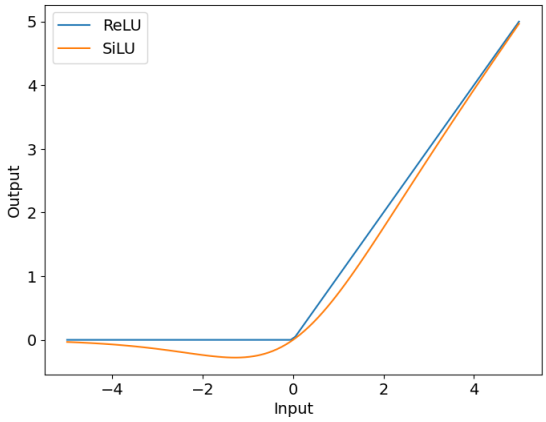


FIGURE 3.1: Differences between the ReLU (blue) and SiLU (orange) functions. While the ReLU derivative is 0 for all negative inputs the smoother SiLU function maintains a non-zero derivative. This allows for more consistent gradient flow and enables the network to more accurately represent derivatives of functions.

Trajectories were generated using a timestep $\Delta t = 0.05$. This choice is also significant, as too small a timestep would result in minimal change between consecutive states, potentially hindering the model's ability to learn the dynamics.

The model's parameters were trained using the Adam (Sect.1.3.2) optimization algorithm, with a learning rate that was dynamically adjusted via a scheduler [34]; the rate was reduced whenever the total loss metric plateaued, allowing for more refined convergence. The training data, consisting of multiple time-series trajectories, was first decomposed into a large set of one-step state transitions, where each transition is a pair $(\vec{h}^t, \vec{h}^{t+dt})$. Mini-batches (of size 32) were employed during training to estimate the gradient of the loss functions (Sect. 1.3.1).

The model's feed-forward network weights are initialized using the Kaiming He uniform method [35]. The latent matrix $\mathbf{K}$ (Eq. 3.1) is initialized to an identity matrix $\mathbf{I}$, and the latent diffusion matrix $\mathbf{M}$ to a gaussian random matrix with 0 mean and standard deviation $0.1 \approx O(\sqrt{\Delta t})$.

3.3 Itô's Lemma

The dynamics of the observable process h_i must be consistent with the dynamics of the latent process z_i as they are linked by the transformation $h_i = \Phi^{-1}(z_i)$. Given our scalar transformation Φ^{-1}, Itô's lemma [12] states that the infinitesimal increment dh_i is related to the increment dz_i by:

$$dh_i = \left.\frac{d\Phi^{-1}}{dz}\right|_{z=z_i} dz_i + \frac{1}{2}\left.\frac{d^2\Phi^{-1}}{dz^2}\right|_{z=z_i} (dz_i)^2 \tag{3.2}$$

where the derivatives of our learned mapping are evaluated at z_i. The inclusion of the second-order term, $(dz_i)^2$, is the key feature of Itô calculus, as it is of order $O(dt)$ and cannot be neglected.

This equation provides a powerful, model-independent consistency check, imposing a strict mathematical relationship between the observed dynamics of the ground truth increments (dh_i), the learned dynamics in latent space (dz_i), and the learned transformation itself (Φ^{-1} and its derivatives). This principle motivates the formulation of a key component of our loss function, which penalizes violations of Itô's Lemma.

A similar equation could be derived for the encoder, using the definition $z_i = \Phi(h_i)$. However, we opted against this approach. Our primary concern was that formulating the objective this way would result in the ground truth increments being multiplied by parameters of our model, i.e. the first and second derivatives of the encoder, potentially making the learning target less direct. Note that these equations are not mutually exclusive, and could be potentially implemented simultaneously.

3.4 Training with the noise as input: Deterministic Regime

The network is trained in a deterministic regime. This is achieved by providing the model with the exact sequence of the noise, $\vec{\eta}$, that was used to generate the

training data. This allows for direct comparison between the model's one-step predictions and the true next state of the system.
While this represents a simplified regime, it holds potential for useful real-world applications. For instance, it can be employed to identify transformations that effectively (or at least approximately) linearize the dynamics of non-linear stochastic processes for which the dynamical equations are known.
In this simplified regime, the dataset $\mathbb{D}$ is therefore comprised of B trajectories of τ steps $\{[\vec{h}^{t=0}, ..., \vec{h}^{t=\tau\Delta t}]\}^B$ as well as the normal gaussian vectors $\{[\vec{\eta}^{\,t=0}, ..., \vec{\eta}^{\,t=(\tau-1)\Delta t}]\}^B$ used to generate the states $\vec{h}^t$.

3.4.1 Loss definition

The total training loss is a weighted sum of the following components, calculated over a minibatch of trajectories denoted by $\mathbb{B}$. For each sample $(\vec{h}^t, \vec{h}^{t+\Delta t}) \in \mathbb{B}$, the corresponding loss terms are computed and then summed.

- **Reconstruction Loss (L_{recon})**: This term enforces that the encoder Φ and the decoder Φ^{-1} are functional inverses of each other. This is an important constraint, as the decoder should not contribute to forwarding the dynamics (the integration wouldn't be linear if it did). It is defined as the mean squared error (MSE) between the original input vector $\vec{h}^t$ and its reconstruction $\hat{\vec{h}}^t = \Phi^{-1}(\Phi(\vec{h}^t))$. The loss over a minibatch $\mathbb{B}$ is:

$$L_{\text{recon}} = \sum_{\vec{h}^t \in \mathbb{B}} \sum_i \left(h_i^t - \hat{h}_i^t\right)^2 \tag{3.3}$$

- **Prediction Loss (L_{pred})**: This loss directly penalizes errors in the one-step prediction of the system's state. A direct comparison is possible because the exact noise vector $\vec{\eta}^{\,t}$ used to generate $\vec{h}^{t+\Delta t}$ is provided as an input to the model. The loss is the MSE between the true next state $\vec{h}^{t+\Delta t}$ and the prediction $\hat{\vec{h}}^{t+\Delta t}$. The predicted state is generated by evolving the latent state $\vec{z}^{\,t} = \Phi(\vec{h}^t)$ and decoding the result: $\hat{\vec{h}}^{t+\Delta t} = \Phi^{-1}\left(\mathbf{K}\vec{z}^{\,t} + \mathbf{M}\left(\vec{z}^{\,t} \odot \vec{\eta}^{\,t}\right)\right)$. The prediction loss is thus:

$$L_{\text{pred}} = \sum_{\vec{h}^t \in \mathbb{B}} \sum_i \left(h_i^{t+\Delta t} - \hat{h}_i^{t+\Delta t}\right)^2 \tag{3.4}$$

- **Itô Consistency Loss (L_{Ito})**: This loss matches the true state increment, $\Delta\vec{h} = \vec{h}^{t+\Delta t} - \vec{h}^t$, with a predicted increment derived from an Itô expansion at order $O(\Delta t)$. The predicted increment for each component, $\Delta\hat{h}_i$, is:

$$\Delta\hat{h}_i = \left.\frac{d\Phi^{-1}}{dz}\right|_{z=z_i^t} \Delta z_i + \frac{1}{2} \left.\frac{d^2\Phi^{-1}}{dz^2}\right|_{z=z_i^t} (\Delta z_i)^2 \tag{3.5}$$

Where $\Delta\vec{z} = \vec{z}^{t+\Delta t} - \vec{z}^{t}$. The loss L_{Ito} is the MSE between the true and predicted increments:

$$L_{\text{Ito}} = \sum_{\vec{h}^t \in \mathbb{B}} \sum_i \left(\Delta h_i - \Delta\hat{h}_i\right)^2 \tag{3.6}$$

The total training loss, L_{total}, is the weighted sum of these three components:

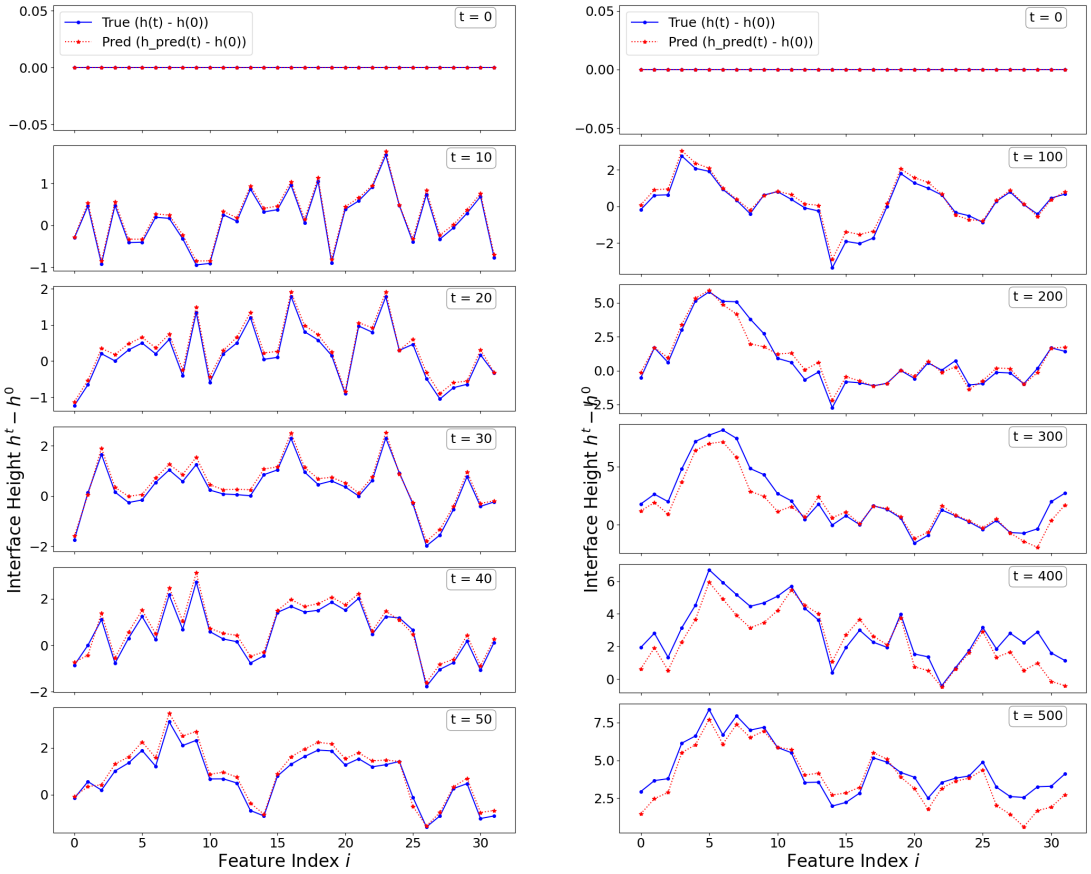
$$L_{\text{total}} = w_{\text{recon}} L_{\text{recon}} + w_{\text{pred}} L_{\text{pred}} + w_{\text{Ito}} L_{\text{Ito}} \tag{3.7}$$

The weights used in the final model are $w_{\text{pred}} = w_{\text{Ito}} = 1$ and $w_{\text{recon}} = 0.1$.

It is important to note that all loss components contribute to the training of both the encoder and the decoder. Even in losses that only apparently feature the decoder Φ^{-1} (like L_{Ito}), the dependence on the encoder's parameters is contained within the latent state vector $\vec{z}^{t}$. Therefore, during backpropagation, gradients flow through the entire network, ensuring all parts of the model are optimized concurrently. All required gradients, including the derivatives of the decoder Φ^{-1} needed for the Itô loss, are calculated efficiently via the automatic differentiation (autograd) [18] capabilities of the deep learning framework (Sect. 1.4.)

3.4.2 Results

Dataset Configuration. Various training datasets were generated using the fixed parameters $\lambda = 1$, $\nu = T = 1/2$, $\Delta x = 1$, $\Delta t = 0.05$ and $N = 32$. The datasets differ in four key aspects: the numerical discretization scheme used for the non-linear term (see Equations 2.4 and 2.5), the total trajectory length τ, the number of trajectories B, and the number of initial time steps D discarded. These initial steps were discarded to provide a more challenging setting, were the system has evolved from its initial flat condition. Varying the discretization of the non-linear terms assesses the model's robustness, as different approaches yield trajectories with distinct statistical properties (Sect. 2.4).



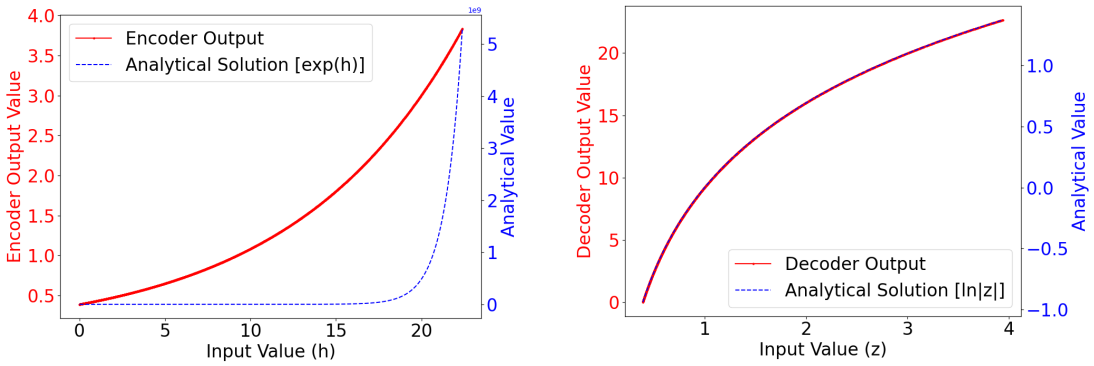
(A) t=0 to t=50 (B) t=0 to t=500

FIGURE 3.2: Comparison of the ground truth displacement, $\vec{h}^t - \vec{h}^0$ (blue), versus the model's prediction, $\hat{\vec{h}}^t - \hat{\vec{h}}^0$ (red), at representative time steps. To generate the prediction, the model was given the true initial conditions and the entire corresponding sequence of stochastic noise used to generate the test dataset. The results show that while point-wise error grows over time, the prediction correctly reproduces the key structural features of the true state.

Model Evaluation. We evaluated the model's prediction capabilities on novel test sets, prepared in a similar fashion as the training set. The model is provided with the initial states $\{\vec{h}^{t=0}\}^B$ and the complete sequence of noise vectors, $\{[\vec{\eta}^0, \ldots, \vec{\eta}^{(\tau-1)\Delta t}]\}^B$, which were used to generate every trajectory in the test set. The model first encodes the initial state onto the latent space, $\vec{z}^0 = \Phi(\vec{h}^0) = [\Phi(h_1^t), ..., \Phi(h_N^t)]$, and then predicts the entire latent trajectory $[\vec{z}^{\Delta t}, \ldots, \vec{z}^{\tau\Delta t}]$ using the learnt matrices $\mathbf{K}$ and $\mathbf{M}$ and the noise inputs $[\vec{\eta}^0, \ldots, \vec{\eta}^{(\tau-1)\Delta t}]$. This latent trajectory is subsequently decoded back to the ground truth space to obtain the predictions, $[\hat{\vec{h}}^{\Delta t}, \ldots, \hat{\vec{h}}^{\tau\Delta t}]$. Performance is then quantified by the Mean Squared Error (MSE) between the predicted and ground truth trajectories. This calculation is performed for every trajectory in

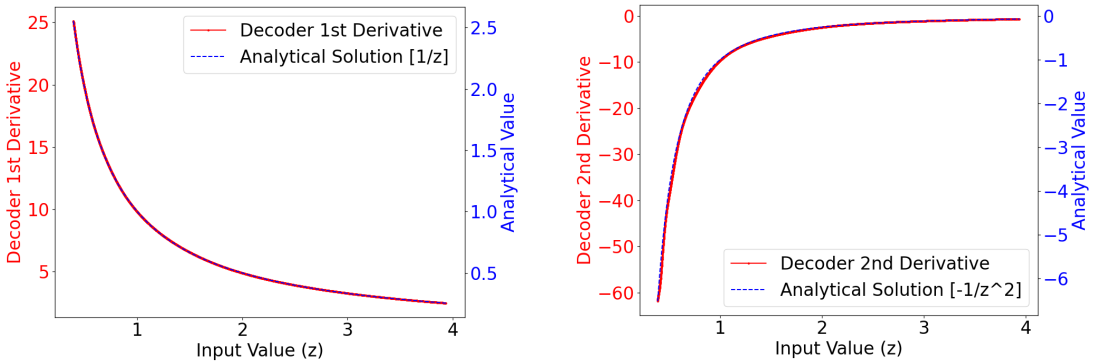
the test set; the resulting MSEs are then sorted. The results presented correspond to the median MSE value. Figure 3.2 shows snapshots comparing the model's predictions to the true state values at several time horizons.

Mapping and Latent Dynamics. We first evaluate the transformations learnt by the feed-forward networks: the encoder Φ and the decoder Φ^{-1}. While the Cole-Hopf transformation (2) suggests analytical forms of $\Phi(h_i) = e^{h_i}$ and $\Phi^{-1}(z_i) = \ln(z_i)$, our learnt functions are not these exact solutions. Nevertheless, after rescaling and shifting, they exhibit a close resemblance in shape and behavior (Fig. 3.3). Specifically, the learned encoder Φ maps the input data $\vec{h}$ to a more constrained codomain (values of $\vec{z}$) compared to the exponential function, which grows rapidly over the dataset's domain (Fig. 3.3a). This compression likely facilitates the decoder's task, allowing it to more accurately approximate the analytical target function (Fig. 3.3b). When examining the derivatives, we find that the encoder's first and second derivatives, $\frac{d\Phi}{dh}$ and $\frac{d^2\Phi}{dh^2}$, are approximately 10 and 100 times smaller, respectively, than the output values of the encoder. Despite this, derivatives maintain a shape consistent with the analytical solution (Fig. 3.4), though accuracy diminishes for higher-order derivatives. Models trained on datasets with longer trajectories (higher value of τ) seem to show better agreement with the analytical transformation, after rescaling. Although the model sometimes finds an encoding transformation closer to $\Phi(h_i) = -\exp(h_i)$, this is not problematic, because the latent dynamic (Eq. 3.1) is invariant under the transformation $\vec{z} \to -\vec{z}$.



(A) Encoder Output Φ (B) Decoder output Φ^{-1}

FIGURE 3.3: Comparison of the learned (red) encoder Φ and decoder Φ^{-1} with their analytical counterparts (blue) from the Cole-Hopf transform (2). The plots use a dual y-axis to show that the learned functions were rescaled for comparison. The encoder maps input values to a significantly smaller codomain (z) than its analytical counterpart. The learned decoder closely approximate the shape of the analytical function after rescaling. The values on the x-axis are representative of those in the test set and correspond to ensemble averages of $\vec{h}^t$ (for the encoder) or $\vec{z}^t$ (for the decoder) at different timestesp t.



(A) Decoder First derivative $\frac{d\,\Phi^{-1}}{dz}$ (B) Decoder Second derivative $\frac{d^2\,\Phi^{-1}}{dz^2}$

FIGURE 3.4: Comparison of the learned derivatives of the decoder, $\frac{d\,\Phi^{-1}}{dz}$ and $\frac{d^2\,\Phi^{-1}}{dz^2}$ with their analytical counterparts from the Cole-Hopf transform (2). The plots use a dual y-axis to show that the learned functions were rescaled for comparison. The decoder's derivative, after rescaling, have the same shape as the expected analytical solution. The values on the x-axis are representative of those in the test and correspond to the ensemble average of the latent states $\vec{z}^{\,t}$ at different timestep t.

We also analyzed the latent dynamical matrices, observing that the drift matrix, $\mathbf{K} - \mathbf{I}$, approximates a discrete Laplacian rescaled by $\nu = 1/2$, and the diffusion matrix, $\mathbf{M}$, is proportional to the identity matrix (Fig. 3.5). This is consistent to what we expect from the analytical solution (2.3). The diagonal entries of $\mathbf{M}$ are less than 1, a characteristic that may be linked to the properties of the learned encoder Φ and the ratio between its derivatives and its output. The model is able to understand the boundary conditions, as the values on the corners of the drift matrix (Fig. 3.5) are non 0.

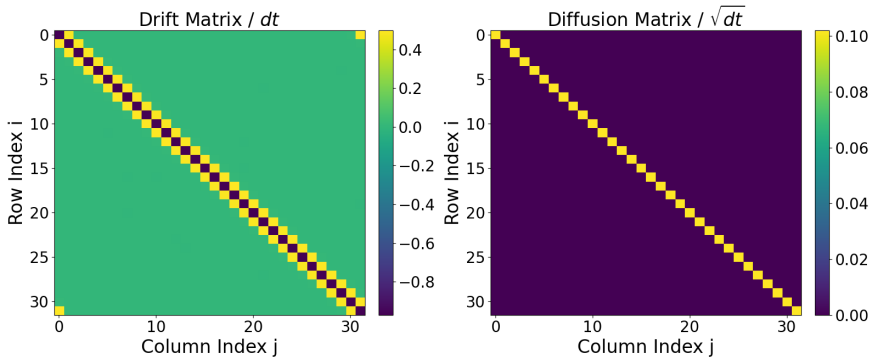


FIGURE 3.5: Heatmap visualization of the learned latent matrices: the drift $\mathbf{K} - \mathbf{I}$ and diffusion $\mathbf{M}$, rescaled by Δt and $\sqrt{\Delta t}$ respectively. The structure aligns with the analytical solution; the drift matrix closely resembles a discrete Laplacian scaled by $1/2$, while the diffusion matrix is diagonal with constant entries that are less than one. This is not problematic and is consistent with the properties of the learnt transformations Φ and Φ^{-1}. The non-zero values in the top right and bottom left corners of the drift matrix highlight the ability of the model to understand the periodic boundary conditions

Prediction Task. We evaluate the predictive performance of models trained on datasets generated with different discretization schemes and parameters by measuring the Mean Squared Error (MSE) between the model's prediction, $\hat{\vec{h}}^t$, and the ground truth, $\vec{h}^t$. Table 3.1 summarizes the median MSE at various timesteps (t) for different discretization schemes, trajectory length (τ), initial discards (D), and number of trajectories (B). Our results indicate that increasing the number of trajectories B from 1000 to 10 000 improves performance, though at the cost of significantly slower training times. The number of initial discards, D, appears to have a negligible effect on the model's predictive accuracy. We observe a substantial performance gap between the Central and Three-point schemes for $\tau = 1000$ and $\tau = 5000$, with the Central scheme performing better. However, for $\tau = 2000$, the models exhibit similar performance. The choice of τ values was informed by the observation that the width (variance) w_2 of the chains plateaus at approximately 1000 timesteps for a chain length of $L = 32$. The overall performance is consistent with findings in related literature [10].

TABLE 3.1: Median Mean Squared Error (MSE) Across Discretization Schemes and Parameters. The table displays the median MSE for trajectories generated with different discretization schemes, timesteps (τ), initial discards (D), and number of trajectories (B). The model more accurately reproduces trajectories from the Central discretization scheme. Both D and B have a minimal impact on model performance.

Scheme	τ	D	B	**Median MSE at Timestep (t)**			
				50	**100**	**500**	**1000**
Central	1k	50	1k	5.47×10^{-2}	1.30×10^{-1}	7.25×10^{-1}	1.25
Central	1k	50	10k	5.12×10^{-2}	1.24×10^{-1}	6.71×10^{-1}	1.24
Three-point	1k	50	1k	1.06×10^{-1}	2.37×10^{-1}	1.14	1.96
Central	2k	0	1k	7.15×10^{-2}	1.76×10^{-1}	8.68×10^{-1}	1.40
Central	2k	50	1k	7.99×10^{-2}	1.92×10^{-1}	1.06	1.79
Three-point	2k	50	1k	7.82×10^{-2}	1.86×10^{-1}	9.44×10^{-1}	1.64
Central	5k	50	1k	6.40×10^{-2}	1.54×10^{-1}	7.91×10^{-1}	1.58
Three-point	5k	50	1k	1.04×10^{-1}	2.47×10^{-1}	1.28	2.21

Figure 3.6 illustrates the evolution of the accuracy and relative accuracy of the model as a function of the timesteps. The relative MSE seems to decrease as the width (variance) $w_2(t)$ begins to saturate.

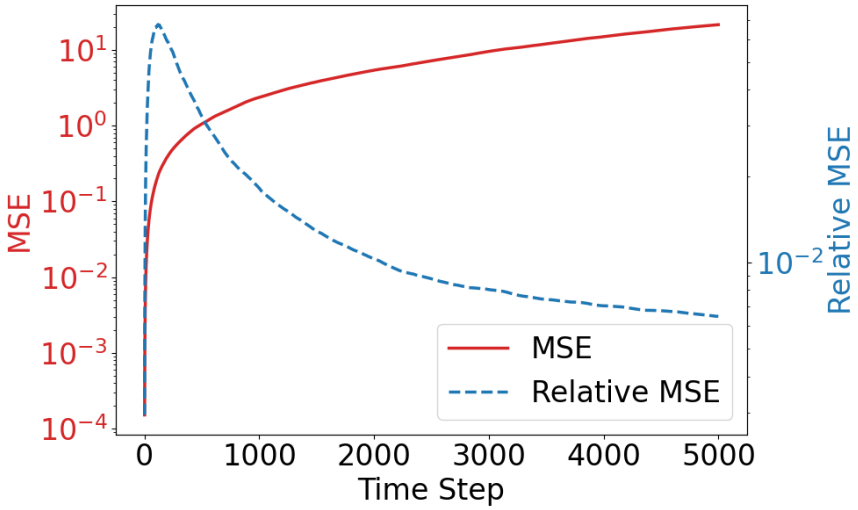
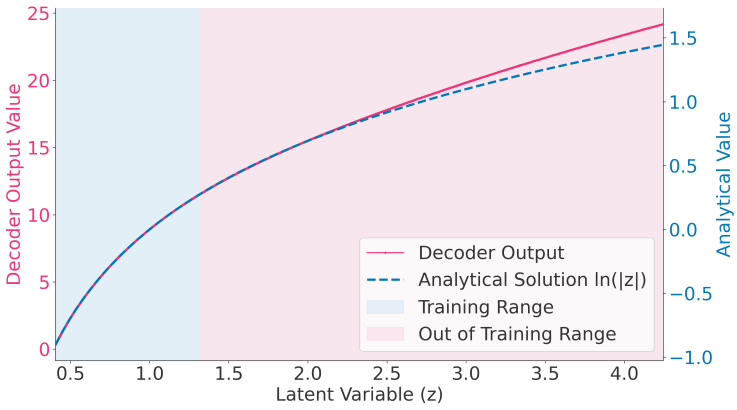
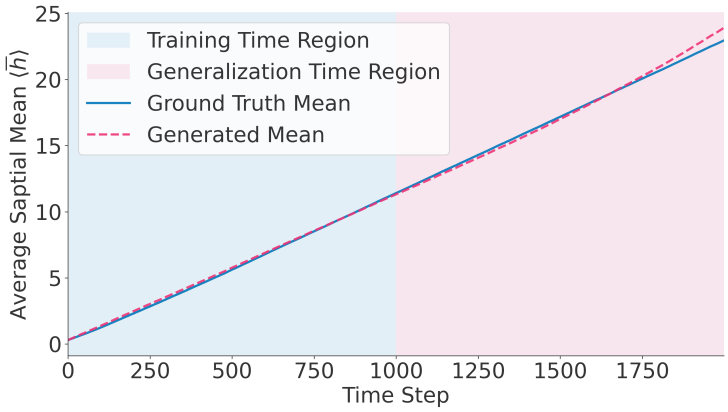


FIGURE 3.6: Temporal evolution of the model's prediction error. The red curve shows the absolute Mean Squared Error (MSE) between the network's prediction, $\hat{\vec{h}}^t$, and the ground truth, $\vec{h}^t$. The blue curve presents the MSE normalized by the mean squared displacement from the initial state, $\langle \overline{(\vec{h}^t - \vec{h}^0)^2} \rangle$, which serves as a baseline for predictive skill. The normalized error remains well below 1.0 across all timesteps, demonstrating that the model consistently makes skillful predictions and maintains long-term stability.

Trajectory Generation and Generalization Beyond Training Range We evaluated the model's ability to generate novel trajectories by sampling the noise increments, $\vec{\eta}^{\,t}$, from a standard Gaussian distribution. The statistical properties of the generated trajectories matched those of the ground truth data, with lower accuracy on higher order cumulants. We also assessed the model's performance on trajectories longer than the training horizon, τ. As illustrated in Figure 3.7, the model demonstrated a degree of generalization beyond the training data. However, as the time extended further outside the training range (the red-shaded region), the decoder's predictions eventually diverged from the expected analytical behavior. While the learned matrices, $\mathbf{K}$ and $\mathbf{M}$, should correctly evolve the latent variables $\vec{z}^{\,t}$ beyond τ, the decoder itself is not trained to function outside this range. Therefore, the extent of successful generalization depends on how quickly the latent variables evolve into a region unfamiliar to the decoder.



(A) Decoder Output $\Phi^{-1}(z)$



(B) Mean of generated trajectories

FIGURE 3.7: Performance outside the training range. The blue-shaded area ($t < \tau$) represents the domain seen during training, while the red-shaded area ($t > \tau$) indicates the region beyond it. The model shows some generalization until the decoder can no longer reproduce the analytical curve accurately.

3.5 Training without the noise as input: Stochastic Regime

In the stochastic training regime, the model receives only the time-series $\{[\vec{h}^{t=0}, ..., \vec{h}^{t=\tau\Delta t}]\}^B$ as input and has no knowledge of the exact values of the stochastic noise, $\vec{\eta}^{\,t}$, used to generate the data. To propel the system forward, the model must draw its own noise at each step from i.i.d. standard normal variables.

The primary challenge in this regime is that a direct comparison between predictions and the ground truth observations is no longer as meaningful. We define the expectation operator $\mathbb{E}[...]$ which denotes a type of ensemble averaging over all possible realizations of the drawn noise at a given step t. This can be thought as averaging over the distribution $P(\vec{h}^{t+\Delta t}|\vec{h}^{t})$. The state at time t is considered as *given*, hence it is not a random quantity. While we can analytically compute these averages for our latent dynamics, as its form (Eq. 3.1) is known, this cannot be done for the ground truth observations; we only have a single sample of the pair $(\vec{h}^{\,t+\Delta t}|\vec{h}^{t})$. This distinction informs our loss function design, as it affects the implementation of the constraint given by Itô's Lemma (3.2).

We modify our consistency equation $\Delta h_i = \Delta\hat{h}_i$ (3.5) and apply the expectation operator $\mathbb{E}[...]$ to the right hand side (r.h.s.) only. Averaging only one side of the equation is not ideal but without knowledge of the equations that generate Δh_i, the type of averaging we require cannot be performed on the l.h.s.

As we are in a situation were the signal is dominated by noise, because of the stochastic square root scaling $\propto \sqrt{\Delta t}$ which dominates over the deterministic term $\propto \Delta t$, it is important to find strategies to avoid overfitting the noise, like adopting a frugal architecture (an architecture with less parameters).

From our latent dynamics model, we can analytically compute the target moments of the latent increment, $\Delta\vec{z}^{\,t}$. The mean is given by $\mathbb{E}[\Delta\vec{z}^{\,t}] = (\mathbf{K} - \mathbf{I})\vec{z}^{\,t}$, and the covariance matrix is: $\mathbb{E}[\Delta\vec{z}^{\,t}\Delta\vec{z}^{\,t^T}] = \mathbf{M}\,\mathrm{diag}(\vec{z}^{\,t})^2\,\mathbf{M}^T + O(\Delta t^2)$, where $\mathrm{diag}(\vec{z}^{\,t})$ is a diagonal matrix with the elements of $\vec{z}^{\,t}$ on its diagonal. T denotes transposition.

3.5.1 Loss definition

The total training loss is a weighted sum of the following components, calculated over a minibatch of trajectories denoted by $\mathbb{B}$.

- **Reconstruction Loss (L_{recon})**: Unchanged. This loss remains crucial as it ensures the encoder Φ and decoder Φ^{-1} are one the inverse of the other. Denoting the model's reconstruction with $\hat{\vec{h}}^{t} = \Phi^{-1}(\Phi(\vec{h}^{t}))$, its definition remains:

$$L_{\text{recon}} = \sum_{\vec{h}^t \in \mathbb{B}} \sum_i \left(h_i^t - \hat{h}_i^t\right)^2 \tag{3.8}$$

- **Mean Itô Consistency Loss ($L_{\text{Ito-mean}}$)**: This loss adapts the Itô consistency to align the true, single-instance increment with the ensemble average of the predicted increment. For each datapoint $\vec{h}^{t}$ in the batch,

the loss penalizes the squared difference between the true observed state change, Δh_i, and its expected value predicted by the model, $\mathbb{E}[\Delta \hat{h}_i]$. The predicted mean increment is defined as: $\mathbb{E}[\Delta \hat{h}_i] = \frac{d\Phi^{-1}}{dz}\Big|_{z=z_i^t} \mathbb{E}[\Delta z_i] + \frac{1}{2} \frac{d^2\Phi^{-1}}{dz^2}\Big|_{z=z_i^t} \mathbb{E}[(\Delta z_i)^2]$. The loss is the mean squared error between the true and predicted mean increments:

$$L_{\text{Ito-mean}} = \sum_{\vec{h}^t \in \mathbb{B}} \sum_i \left(\Delta h_i - \mathbb{E}[\Delta \hat{h}_i] \right)^2 \tag{3.9}$$

To provide a more stable training signal, an additional term ($L_{\overline{\text{Ito-mean}}}$) compares the spatial (feature) averages of the true and predicted increments:

$$L_{\overline{\text{Ito-mean}}} = \sum_{\vec{h}^t \in \mathbb{B}} \left(\frac{1}{N} \sum_i \Delta h_i - \frac{1}{N} \sum_i \mathbb{E}[\Delta \hat{h}_i] \right)^2 \tag{3.10}$$

- **Covariance Itô Consistency Loss ($L_{\text{Ito-cov}}$)**: This loss ensures that the model captures the correlation structure of the state increments. The loss function is designed to match the empirical outer product from the data, $\Delta \vec{h}_{\text{true}} \Delta \vec{h}_{\text{true}}^T$, with the expectation of its model-generated counterpart. Using the leading-order approximation, the predicted expectation of the covariance is given by: $\mathbb{E}[\Delta h_i \Delta h_j]_{\text{pred}} = \frac{d\Phi^{-1}}{dz}\Big|_{z=z_i^t} \frac{d\Phi^{-1}}{dz}\Big|_{z=z_j^t} \mathbb{E}[\Delta z_i \Delta z_j]$. The latent covariance $\mathbb{E}[\Delta z_i \Delta z_j]$ is the (i,j)-th element of the matrix $\mathbf{M}\,\text{diag}(\vec{z}^t)^2\,\mathbf{M}^T$. The loss penalizes the element-wise difference between the true and predicted covariance matrices:

$$L_{\text{Ito-cov}} = \sum_{\vec{h}^t \in \mathbb{B}} \sum_{i,j} \left((\Delta h_i \Delta h_j)_{\text{true}} - \mathbb{E}[\Delta h_i \Delta h_j]_{\text{pred}} \right)^2$$

A spatially-averaged (feature-averaged) version of this loss ($L_{\overline{\text{Ito-cov}}}$) is also used. It compares the average value of the covariance matrices to provide a more stable training signal:

$$L_{\overline{\text{Ito-cov}}} = \sum_{\vec{h}^t \in \mathbb{B}} \left(\frac{1}{N^2} \sum_{i,j} (\Delta h_i \Delta h_j)_{\text{true}} - \frac{1}{N^2} \sum_{i,j} \mathbb{E}[\Delta h_i \Delta h_j]_{\text{pred}} \right)^2 \tag{3.11}$$

The total training loss, L_{total}, is the weighted sum of these five components:

$$L_{\text{total}} = w_{\text{recon}} L_{\text{recon}} + w_{\text{Ito-mean}} L_{\text{Ito-mean}} + w_{\overline{\text{Ito-mean}}} L_{\overline{\text{Ito-mean}}} + w_{\text{Ito-cov}} L_{\text{Ito-cov}} + w_{\overline{\text{Ito-cov}}} L_{\overline{\text{Ito-cov}}}$$

The weights are set to $w_{\text{recon}} = 0.1$, $w_{\text{Ito-mean}} = w_{\overline{\text{Ito-mean}}} = 1$, and $w_{\text{Ito-cov}} = w_{\overline{\text{Ito-cov}}} = 100$.

The loss terms $L_{\text{Ito-mean}}$ and $L_{\text{Ito-cov}}$ should plateau at a non 0 value, as it's unreasonable to expect exact matching between a single random realization and an average. The plateau value is given respectively by the variance of Δh_i

and $\Delta h_i \Delta h_j$. Therefore the theoretical lower bound for the loss terms is $O(\Delta t)$ for $L_{\text{Ito-mean}}$ and $O(\Delta t^2)$ for $L_{\text{Ito-cov}}$.

3.5.2 Results

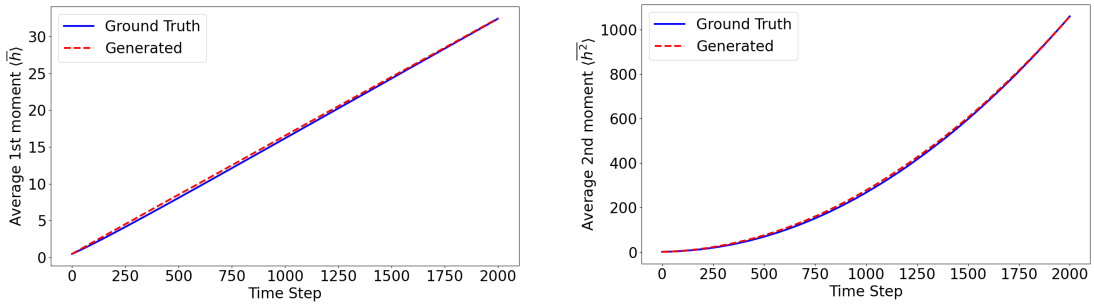
Dataset Configuration and Model Evaluation. The model was trained on the same datasets used in the deterministic regime (Sect. 3.4). The noise vectors $\vec{\eta}$ used to generate the dataset were not provided as input to the model during training or testing. The model was evaluated by first initializing the latent state $\vec{z}^{\,t=0}$ with the initial conditions $\{\vec{h}^{\,t=0}\}^B$, then generating a full latent trajectory $\{[\vec{z}^{\,\Delta t}, \ldots, \vec{z}^{\,\tau\Delta t}]\}^B$ by sampling noise at every step from a standard normal distribution. This trajectory was subsequently decoded back into the physical space to form a generated dataset $\hat{\mathbb{D}}$ of trajectories $\{[\hat{\vec{h}}^{\,\Delta t}, \ldots, \hat{\vec{h}}^{\,\tau\Delta t}]\}^B$. Finally, the statistics of the generated dataset were compared with those of the original dataset $\mathbb{D}$.

Generative Task: Statistical Moments In the stochastic regime, the performance of the models exhibits greater variability compared to the deterministic case. Consequently, while successful convergence to a high-performance state is observed in some instances, it is not consistently guaranteed. We are actively investigating strategies to enhance the stability and reliability of the training process.

We present the instances in which convergence to a satisfying performance was obtained. In theses cases the model performs as well in the generative task as models trained in the deterministic case (Sect 3.4). The time evolution of the uncentered moments generated by the model align closely with the of the test set data (Figure 3.8). Similarly, the interface width $w_2(t)$ (variance), shows excellent agreement in its initial growth phase. As illustrated in Figure 3.9, the model consistently captures the characteristic KPZ dynamics, $w_2(t) \sim t^{2/3}$.

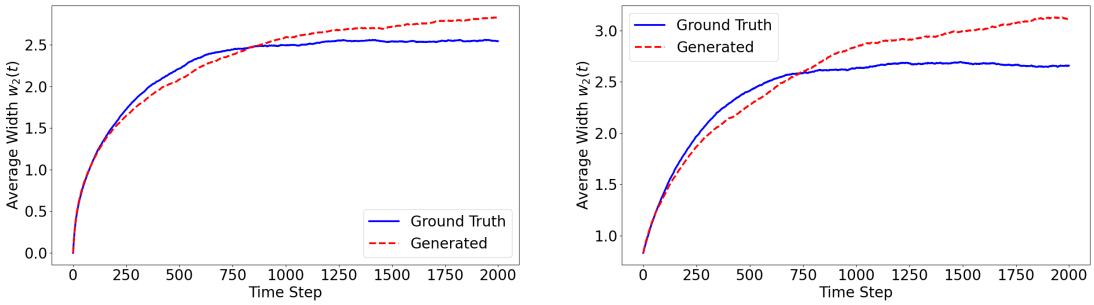
However, a discrepancy emerges in the long-time behavior, where the saturation plateau of the width, w_2^{sat}, tends to be overestimated. The magnitude of this overestimation varies in the range $2\% - 10\%$. Furthermore, the generated saturation plateau is imperfect, sometimes displaying a slight positive slope rather than being completely flat.

We suspect the overestimation of the saturation plateau may be an artifact of the learned coordinate transformation. It is plausible that our current latent dynamics model cannot fully accommodate all the properties of this transformed space. Therefore, we are actively investigating more flexible latent dynamic structures to address this limitation.



(A) Mean interface height $\langle \bar{h}(t) \rangle$. (B) Second uncentered moment $\langle \overline{h^2}(t) \rangle$.

FIGURE 3.8: Comparison of statistical moments from model-generated trajectories (red) and ground truth test data (blue). The plots show the evolution of the first (left) and second (right) uncentered moments. To produce these statistics, the model autonomously evolved a set of initial conditions, sampling new noise at every step and using the dynamics it learnt from training with the Three–Point scheme (2.5). A strong agreement is observed between the statistical moments generated by the model and those from the ground truth data.



(A) $w_2(t)$ for a model trained on a Central Discretization dataset. (B) $w_2(t)$ for a model trained on a Three-Point Discretization dataset.

FIGURE 3.9: Time evolution of the interface width, $w_2(t)$, for models trained on datasets from the (a) Central (2.4) and (b) Three-Point (2.5) discretization schemes. The models encode the initial conditions and generate new trajectories by drawing novel noise at every step. In both cases, the model-generated statistics (red) accurately reproduce the early-time KPZ growth exponent ($w_2 \sim t^{2/3}$) observed in the test data (blue). However, the model consistently overestimates the final saturation width, w_2^{sat}.

Generative Task: Two-Point Correlations To further assess the model's performance, we evaluated its ability to reproduce the spatio-temporal correlations of the process. We computed the spatial two-point connected correlation function,

$$G_{x,t}(\xi) = \langle h(x,t)h(x+\xi,t) \rangle - \langle h(x,t) \rangle \langle h(x+\xi,t) \rangle, \tag{3.12}$$

and the temporal autocorrelation function,

$$F_{x,t}(\tilde{\tau}) = \langle h(x,t)h(x,t+\tau)\rangle - \langle h(x,t)\rangle\langle h(x,t+\tilde{\tau})\rangle \tag{3.13}$$

For our analysis, we used the discretized versions of these functions. The spatial correlation is given by $G^t_{ij} = \langle h^t_i h^t_j\rangle - \langle h^t_i\rangle\langle h^t_j\rangle$, with $j = i + \xi$ for a given integer distance ξ. The temporal correlation is $F^t_i(\tilde{\tau}) = \langle h^t_i h^{t+\tilde{\tau}}_i\rangle - \langle h^t_i\rangle\langle h^{t+\tilde{\tau}}_i\rangle$. Given the translational symmetry of our flat initial condition, we evaluated these correlations at the central point of the lattice, $i = 0$. The results were consistent for models trained on different discretization schemes.

As shown in Figure 3.10, the spatial correlations $G^t_{0\xi}$ generated by our model closely match the dataset during the initial phase of the dynamics (t < 100). However, as time progresses, the model's predictions for the correlations become progressively smaller than the target values. This is an interesting outcome, especially considering that the interface width w_2 (variance) tends to be larger in the later stages of the dynamics.

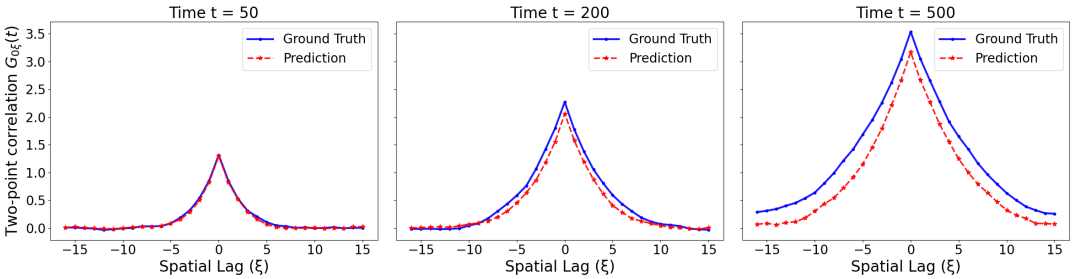


FIGURE 3.10: The spatial two-point correlation function $G^t_{0\xi}$ at three different times (t = 50, 200, 500). The x-axis represents the distance ξ from the central point. The model's predictions (red) are compared against the dataset (blue). While the initial correlations are well-matched, the model increasingly underestimates the correlations at later times. The model was trained on the Central approximation scheme (2.4).

In contrast, the temporal autocorrelation $F_0(\tau)$, shown in Figure 3.11, are well-reproduced by the model throughout the simulation, with only minor deviations at the very end.

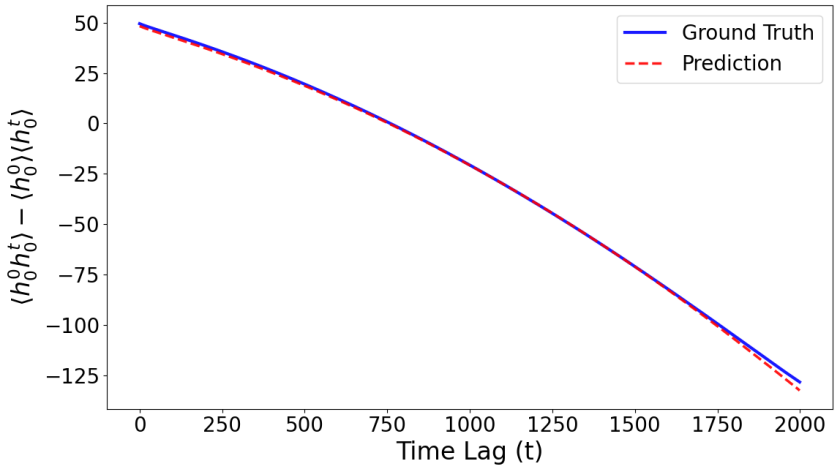
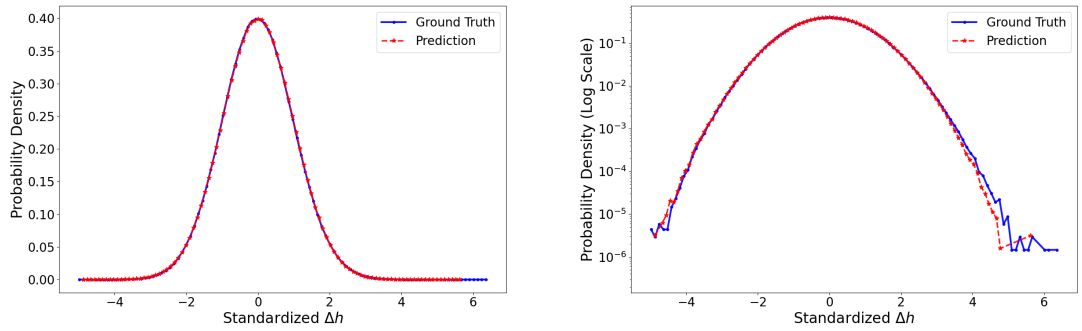


FIGURE 3.11: The temporal two-point correlation function $F_0(\tau)$ evaluated at position $i = 0$. The model's predictions (red) show excellent agreement with the ground truth (blue), accurately capturing the decay of temporal correlations. Th model was trained on the Central approximation scheme (2.4).

Mapping and Latent Dynamics. Results are less consistent then the deterministic regime. for what concerns the transformations Φ and Φ^{-1}, with results often diverging from what we would expect from theory. The output of the encoder Φ is more consistent with a decaying exponential than an increasing one. Nevertheless, the found mapping is self-consistent; each loss component approaches the theoretical lower bound meaning that Itô's Lemma is approximately satisfied, and the generated trajectories have close statistics to those of the original dataset. The found dynamical matrices have a close resemblance to the structure found in the training regime with given noise (Fig. 3.5). As a consequence, they are consistent with the expected theoretical predictions.

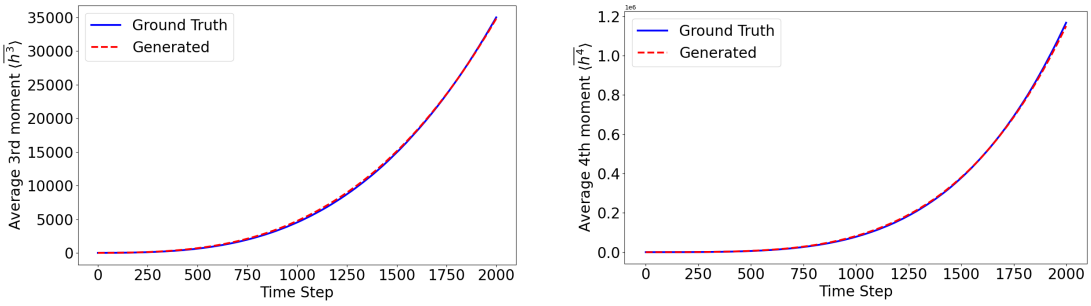
Tracy-Widom distribution We compared the distributions of the variables $\Delta h_i = h_i^t - h_i^{t'}$ to check wether the trajectories generated by our model exhibit the characteristic asymmetric tails of the Tracy-Widom distribution (Sect.2.2). The generated trajectories reproduce bot the left and right tail behaviours.



(A) Probability density. (B) Probability density in semi-log scale.

FIGURE 3.12: Probability density of the variable $\Delta h = h_i^t - h_i^{t'}$ sampled using $t - t' = 10\Delta t$. Samples are gathered once the width has already plateaued. Variables were rescaled to have 0 mean and unitary variance. The decays of both tails are consistent with the expected analytical form of $\approx \exp\left(-c_1|\Delta h|^3\right)$ for the left tail and $\approx \exp\left(-c_2|\Delta h|^{\frac{3}{2}}\right)$ for the right tail.

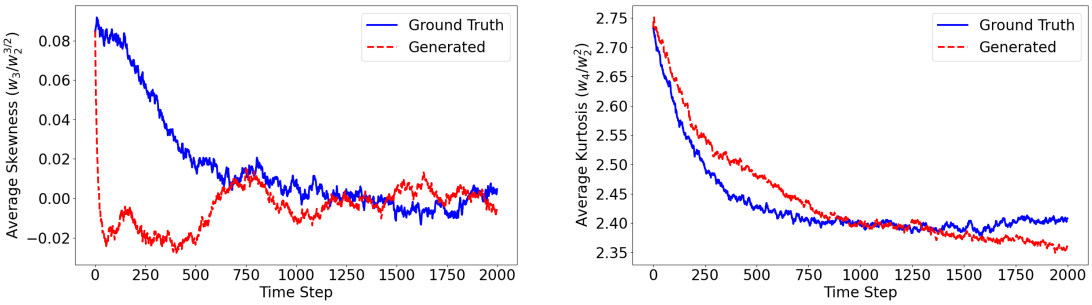
Higher-Order Moments and Cumulants We also analyzed the 3rd and 4th order moments and cumulants of the generated trajectories. The uncentered moments show excellent agreement with the test data (Fig. 3.13).



(A) Third uncentered moment $\langle \overline{h^3}(t) \rangle$. (B) Fourth uncentered moment $\langle \overline{h^4}(t) \rangle$.

FIGURE 3.13: Comparison of statistical moments from model-generated trajectories (red) and ground truth test data (blue). The plots show the evolution of the third (left) and fourth (right) uncentered moments. To produce these statistics, the model autonomously evolved a set of initial conditions, sampling new noise at every step and using the dynamics it learnt from training with the Three–Point scheme (2.5). A strong agreement is observed between the statistical moments generated by the model and those from the ground truth data.

An analysis of higher-order cumulants reveals a more nuanced performance. The model successfully captures the general qualitative trends for skewness ($S = w_3/w_2^{3/2}$) and kurtosis ($K = w_4/w_2^2$), as depicted in Figure 3.14. Nevertheless, the model struggles to replicate their behaviours quantitatively, especially for the early-time dynamics of the skewness.



(A) Skewness evolution.

(B) Kurtosis evolution.

FIGURE 3.14: Time evolution of skewness (left) and kurtosis (right) for a model trained with the Three-Point scheme (2.5). While the model-generated statistics (red) reproduce the correct qualitative trends seen in the test data (blue), they deviate quantitatively. The discrepancy is most pronounced in the early-time evolution of skewness.

Conclusion

We propose a Neural Network model that linearizes the dynamics of non-linear stochastic processes by mapping them onto a latent space. This enables the efficient creation of a novel dataset which is statistically similar to the original. We defined a new training strategy focused on matching the statistical properties of the predictions to those of the original dataset. We validated our approach on datasets generated via the KPZ equation, where an exact linearization is known to exist. The model's performance converged to a high level of accuracy, though we note that guaranteed convergence is not yet established and is a subject of ongoing research. Training stability could potentially be enhanced through a more informed approach similar to Physics-Informed Neural Networks (PINNs) [19], where prior knowledge of the underlying equations governing the dataset generation process is incorporated into the model architecture. Furthermore, a significant direction for future work would be to extend this framework to discover non-local transformations; this could be done with a similar framework with a more complex architecture and by adapting the Itô consistency terms in the loss design.

Bibliography

[1] Mehran Kardar, Giorgio Parisi, and Yi-Cheng Zhang. "Dynamic Scaling of Growing Interfaces". In: *Physical Review Letters* 56.9 (1986), pp. 889–892. DOI: 10.1103/PhysRevLett.56.889.

[2] Fernando Ojeda et al. "Dynamics of rough interfaces in chemical vapor deposition: experiments and a model for silica films". In: *Physical review letters* 84.14 (2000), p. 3125.

[3] Fredric Lam, XiaoCheng Mi, and Andrew J Higgins. "Front roughening of flames in discrete media". In: *Physical Review E* 96.1 (2017), p. 013107.

[4] María Ana Cristina Huergo et al. "Growth dynamics of cancer cell colonies and their comparison with noncancerous cells". In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 85.1 (2012), p. 011918.

[5] Guillaume Barraquand, Pierre Le Doussal, and Alberto Rosso. "Stochastic growth in time-dependent environments". In: *Physical Review E* 101.4 (2020), p. 040101.

[6] Massimiliano Gubinelli and Nicolas Perkowski. "KPZ reloaded". In: *Communications in Mathematical Physics* 349 (2017), pp. 165–269.

[7] Martin Hairer. "Solving the KPZ Equation". In: *Annals of Mathematics* 178.2 (2013), pp. 559–664. DOI: 10.4007/annals.2013.178.2.4. URL: https://annals.math.princeton.edu/wp-content/uploads/annals-v178-n2-p04-p.pdf.

[8] Tadahisa Funaki and Jeremy Quastel. "KPZ Equation, Its Renormalization and Invariant Measures". In: *Stochastic Partial Differential Equations: Analysis and Computations* 3.2 (2015), pp. 159–220. DOI: 10.1007/s40072-015-0052-7. arXiv: https://arxiv.org/abs/1407.7310 [math.PR].

[9] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *Machine learning for data science handbook: data mining and knowledge discovery handbook* (2023), pp. 353–374.

[10] Craig Gin et al. "Deep learning models for global coordinate transformations that linearise PDEs". In: *European Journal of Applied Mathematics* 32.3 (2021), pp. 515–539.

[11] Simone Cuonzo. "Using deep neural networks to find a coordinate transformation that linearizes KPZ equation". Master Thesis. 2024.

[12] Bernt K. Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. 6th. Berlin: Springer, 2003.

[13] Lu Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature machine intelligence* 3.3 (2021), pp. 218–229.

[14] Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.

[15] Barry J Wythoff. "Backpropagation neural networks: a tutorial". In: *Chemometrics and Intelligent Laboratory Systems* 18.2 (1993), pp. 115–155.

[16] Tao Lin et al. "Don't use large mini-batches, use local SGD". In: *arXiv preprint arXiv:1808.07217* (2018).

[17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[18] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[19] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: A review". In: *Acta Mechanica Sinica* 37.12 (2021), pp. 1727–1738.

[20] Jeremy Quastel. "Introduction to KPZ". In: *Current developments in mathematics* 2011.1 (2011).

[21] Yi-Kuo Yu, Ning-Ning Pang, and Timothy Halpin-Healy. "Concise calculation of the scaling function, exponents, and probability functional of the Edwards-Wilkinson equation with correlated noise". In: *Physical Review E* 50.6 (1994), p. 5111.

[22] Joachim Krug, Paul Meakin, and Timothy Halpin-Healy. "Amplitude universality for driven interfaces and directed polymers in random media". In: *Physical Review A* 45.2 (1992), p. 638.

[23] Craig A. Tracy and Harold Widom. "Level-spacing distributions and the Airy kernel". In: *Communications in Mathematical Physics* 159.1 (1994), pp. 151–174.

[24] Tomohiro Sasamoto and Herbert Spohn. "One-dimensional Kardar-Parisi-Zhang equation: An exact solution and its universality". In: *Physical Review Letters* 104.23 (2010), p. 230602.

[25] G. Amir, I. Corwin, and J. Quastel. "Probability distribution of the free energy of the continuum directed random polymer in 1+1 dimensions". In: *Communications on Pure and Applied Mathematics* 64.4 (2011), pp. 466–537.

[26] Jinho Baik, Robert Buckingham, and Jeffery DiFranco. "Asymptotics of Tracy-Widom distributions and the total integral of a Painlevé II function". In: *Communications in Mathematical Physics* 280 (2008), pp. 463–497.

[27] Ivan Corwin. "The Kardar-Parisi-Zhang equation and universality class". In: *Random Matrices: Theory and Applications* 1.1 (2012), p. 1130001.

[28] Kazumasa A Takeuchi. "Crossover from growing to stationary interfaces in the Kardar-Parisi-Zhang class". In: *Physical review letters* 110.21 (2013), p. 210604.

[29] Bastien Marguet et al. "Supersymmetries in nonequilibrium Langevin dynamics". In: *Physical Review E* 104.4 (2021), p. 044120.

[30] H Lamba, Jonathan C Mattingly, and Andrew M Stuart. "An adaptive Euler–Maruyama scheme for SDEs: convergence and stability". In: *IMA journal of numerical analysis* 27.3 (2007), pp. 479–506.

[31] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. "Introduction to multi-layer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[32] Murat H Sazlı. "A brief review of feed-forward neural networks". In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[33] Chigozie Nwankpa et al. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018).

[34] PyTorch. *torch.optim.lr*$_s$*cheduler.ReduceLROnPlateau*. PyTorch Documentation. Accessed: 2025-06-12. URL: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html.

[35] Kaiming He and Jian Sun. "Image completion approaches using the statistics of similar patches". In: *IEEE transactions on pattern analysis and machine intelligence* 36.12 (2014), pp. 2423–2435.