MSc Computer Science

Model-Order Reduction and Control of Partial Differential Equations using Denoising Diffusion Probabilistic Models

Agata Sowa

Committee:
Dr. N. Strisciuglio
Prof.Dr. C. Brune
Dr. N. Botteghi
Dr. F. Califano

November, 2024

Faculty of EEMCS
University of Twente

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Dr. Nicolò Botteghi and Dr. Federico Califano for their invaluable guidance, support, and insight throughout the course of this research. Their expertise and encouragement have been crucial to the completion of this thesis, and their constructive feedback has significantly enhanced the quality of my work.

I am also sincerely grateful to Dr. Nicola Strisciuglio and Prof. Dr. Christoph Brune for their insights during the crucial stages of this research. Their contributions provided clarity and direction that were essential for the project's progress.

A heartfelt thank you goes to my parents, whose unwavering support made it possible for me to pursue my studies abroad. Their belief in my potential and encouragement have been the foundation of my academic journey and personal growth.

To my boyfriend, your (emotional) support has been an anchor throughout this journey. Your belief in my abilities helped me push through the challenging times and kept me motivated when I needed it most.

Lastly, I would like to thank all my housemates, past and present, at Calslaan 56 A. You have been with me through every stage of this thesis, from the highs to the lows, providing companionship, laughter, and a place to share stories or take a break. Your friendship made this experience truly memorable and filled with warmth.

Thank you all for being an essential part of this journey.

Abstract

Modelling complex systems requires accurate simulations to predict behaviours under various conditions. This research explores the use of Denoising Diffusion Probabilistic Models (DDPMs) to efficiently generate solutions for parametric Partial Differential Equations (PDEs), providing an alternative to traditional computational methods. The research demonstrates that DDPMs can produce accurate and (temporally-)coherent solutions for the 1D Kuramoto-Sivashinsky equation, showing strong generalisation capabilities to scenarios involving unseen parameter values. Additionally, to enhance computational efficiency, dimensionality-reduction techniques, such as Singular Value Decomposition and Autoencoders, were employed to reduce the data's dimensionality. This approach simplifies the generation task by allowing the DDPM to operate on lower-dimensional data. Eventually, DDPMs show potential for generating controlled solutions, supporting applications where system intervention is necessary. While the models exhibited limitations in long-term prediction accuracy, the results indicate that DDPMs can generalise effectively to different instances of the PDE parameter, opening avenues for advanced simulation and control in complex systems.

Notation

Time and Spatial Variables

- t Time step
- T Total number of time steps, horizon
- L Length of the spatial domain
  - x Spatial variable, where $x \in [0, L]$

States and Trajectories

- s State
- a Action
- $\mathcal{T}$ Set of trajectories
  - τ Trajectory, where $\tau = \{s(t, x) | t \in [0, T], x \in [0, L]\}$
- R Return function
- y Conditional information

Processes and Distributions

- p Probability distribution
- q Forward diffusion process
- ϵ Noise added during the forward diffusion process
- ϵ_θ Learned noise estimate (or gradient) during the reverse diffusion process

Model Parameters and Variables

- θ Model parameters
- λ Hyperparameter
- ω Weight factor
- b Offset (constant)
- v Vector, output of the model
- z Latent variable
- c Class label
- α Scaling factor of the noise
- β Variance schedule

Functions

- ϕ_{dec} Decoder function
- ϕ_{enc} Encoder function
- I Identity matrix
- $\mathcal{R}$ Range
- $\mathcal{D}$ Dataset

Table of Contents

1 Introduction

Understanding and accurately modelling complex systems is essential for critical infrastructures such as the International Space Station, aircraft, power stations, and chemical reactors. These systems involve numerous variables and intricate dynamics that require precise simulations to predict their behaviour under various conditions. The complexity and high stakes of these environments necessitate advanced methods for generating data, especially when real-world experimentation or historical data collection is limited or costly. Effective control strategies also rely on accurate models that simulate a system's behaviour under different control inputs.

For many complex systems, particularly those involving distributed parameters - such as fluid dynamics in aerospace engineering or heat distribution in power stations - partial differential equations (PDEs) provide a mathematical framework for modelling their behaviour accurately. Solving these PDEs allows for simulating system behaviours under various conditions and generating synthetic data that reflects the system's response to different scenarios [62]. However, solving these PDEs using traditional numerical methods can be computationally intensive and time-consuming, particularly for high-dimensional systems [10]. This computational burden can hinder exploring scenarios necessary for robust system understanding and decision-making, including developing effective control strategies.

Given the challenges associated with numerical solutions, there is a growing need for innovative approaches to generate high-fidelity synthetic data efficiently. Denoising diffusion probabilistic models (DDPMs) [53] present a promising alternative by generating data that captures the complex dynamics of systems. Although DDPMs have not yet been widely explicitly applied to solving PDEs, they have the potential to approximate solutions by learning patterns within the data, offering a scalable and efficient alternative to conventional solvers.

While this research primarily focuses on leveraging DDPMs for generating PDE solutions, traditional dimensionality-reduction (DR) techniques, such as Singular Value Decomposition (SVD) [24] and Autoencoders (AEs) [26] will be explored in combination with DDPM to enhance their computational efficiency. By reducing the dimensionality of the synthetic data generated by DDPMs, DR techniques can further streamline simulations without compromising essential system dynamics [47]. This complementary approach provides additional avenues for managing the computational challenges associated with high-fidelity simulations.

In summary, this research explores using DDPMs as an alternative to traditional solvers for generating high-fidelity synthetic data for PDEs. Additionally, combining DDPMs with DR techniques, such as SVD, can further reduce computational complexity. This combined approach could significantly advance state-of-the-art system simulation and data generation, offering a more effective means of modelling complex infrastructures with potential applications in control.

1.1 Outline

This research builds upon the premise of using DDPMs to generate synthetic data within the context of PDEs. The generative approach enables the utilisation of a single PDE to generate multiple solutions, each representing distinct system behaviours under various conditions. Specifically, a single PDE, which describes fundamental system dynamics such as heat distribution, can be solved repeatedly with different initial conditions or parameters to reflect a range of scenarios. This method enables extensive exploration of the system's behaviour without needing multiple distinct PDEs. Instead, a single PDE can be adapted to simulate various possible states by varying its inputs, facilitating the generation of new, previously unseen solutions.

Focusing on generating solutions from a single PDE enhances adaptability and versatility in simulating complex systems. It broadens the spectrum of potential system responses, ensuring the analysis is not constrained to reproducing responses solely based on historical data. These generated solutions can also provide valuable insights for tasks such as control, where understanding diverse system behaviours is crucial for optimising performance across varying conditions.

Furthermore, the research investigates the potential of utilising low-dimensional representations of the PDE solutions to simplify the input to DDPMs, thereby reducing computational complexity. The latent representation captures the essential features of the system's behaviour in a more condensed form compared to the original PDEs, reducing dimensionality. By representing the system dynamics more concisely, the DDPM can efficiently generate trajectories, leading to shorter computational times.

Lastly, this research explores the potential of using DDPMs to generate controlled solutions for parametric PDEs. By integrating control strategies into the DDPM framework, the model can adapt the system's behaviour to a desired outcome. This can be particularly valuable in applications where system dynamics must be actively adjusted to achieve specific objectives, offering a way to manage complex systems.

Therefore, the main research question can be formulated as follows:

- **RQ1** To what extent is it possible to generate solutions of parametric PDEs using DDPMs?

With complimentary sub-research questions:

- **subRQ1** To what extent can the model generalise for unseen values of the parameterised PDE?
- **subRQ2** To what extent does a latent representation of PDEs reduce the computational complexity of the model?
- **subRQ3** To what extent can the model generate controlled solutions for parametric PDEs?

2 Background

The foundations of this research are the denoising diffusion probabilistic models (DDPMs), singular value decomposition (SVD), autoencoder (AE), and Control. This section provides a literature-based explanation of each concept, uniform notation, and the definition of terms used within the context of this paper. The section begins by explaining what a diffusion probabilistic model is. Subsequently, SVD and AEs are presented, followed by an introduction to the latent diffusion model and Control. The order of topics mentioned in this section allows for a logical flow of concepts.

2.1 Diffusion Probabilistic Model

Sohl-Dickstein [53] first introduces the DDPM. The DDPM is a generative model belonging to the same family as generative adversarial networks (GANs) and variational autoencoders (VAE). Generative models are unsupervised machine learning models that capture dependencies and structures within the data without relying on prepared, labelled information. Image generation, completion, and denoising are examples of the tasks of the generative models.

The idea behind the DDPM is to learn how to remove distortion or noise from the input. During training, the model is given an image, which then gets more and more distorted. The goal of the model is to learn how to retrieve an image as close to the original as possible, given only the noisy input. This can be achieved by estimating the diffusion process parameters, which removes the noise from the image. After the training, the model can generate new samples with noisy input.

Since the DDPM is log-likelihood-based, it does not experience mode-collapse contrary to the GAN [49]. Additionally, due to parameter sharing, DDPMs efficiently use a smaller number of parameters to model complex distributions. In comparison, an autoregressive transformer would require a significantly bigger number of parameters for the same task.

2.1.1 Model implementation

The diffusion models are latent variable models characterized by Equation (1) [27], where s represents any input and θ the parameters over which the probability distribution p is parametrised:

$$p_\theta(s_0) := \int p_\theta(s_{0:T}) ds_{1:T} \,, \tag{1}$$

where $p_\theta(s_0)$ represents the probability distribution of the final observed state s_0. $p_\theta(s_{0:T})$ denotes the joint probability distribution of the states from s_0 to s_T. The integral $\int ds_{1:T}$ sums over all possible intermediate states s_1 to s_T, effectively marginalising them to obtain the distribution of s_0.

The underlying structure of the DDPM is a parameterised Markov chain. The parametrisation makes it so that the probabilities of rules for transitioning between the current and next state are not fixed. On the contrary, they can be adjusted based on the parameters introduced to

the chain. The parameters used for the transition between the states are learned following an inverse diffusion process, Equation (2). The Markov chain adds a small amount of Gaussian noise in the direction opposite to the sampling as long as the signal is not destroyed:

$$p_\theta(s_0) := p(s_T) \prod_{t=1}^{T} p_\theta(s_{t-1}|s_t), \qquad p_\theta(s_{t-1}|s_t) := N(s_t; \mu_\theta(s_t,t), \Sigma_\theta(s_t,t)), \tag{2}$$

where $p_\theta(s_0)$ again represents the probability distribution of the final observed state s_0. The term $p(s_T)$ is the prior distribution of the state s_T. The product $\prod_{t=1}^{T} p_\theta(s_{t-1}|s_t)$ represents the chain of conditional probabilities from s_T back to s_0, describing the reverse diffusion process. The conditional probability $p_\theta(s_{t-1}|s_t)$ is parameterised by θ and is defined as a Gaussian distribution $N(s_t; \mu_\theta(s_t,t), \Sigma_\theta(s_t,t))$, with mean $\mu_\theta(s_t,t)$ and covariance $\Sigma_\theta(s_t,t)$. These parameters are functions of s_t and t and are learned during training.

Ho, Jain and Abbeel [27] point out that the diffusion probabilistic model can be distinguished from other latent variables, as it uses the forward process (3), which is fixed to a Markov chain gradually adding Gaussian noise to the data according to a variance schedule $\beta_1, ..., \beta_T$:

$$q(s_{1:T}|s_0) := \prod_{t=1}^{T} q(s_t|s_{t-1}), \qquad q(s_t|s_{t-1}) := N(s_t; \sqrt{1-\beta_t} s_{t-1}, \beta_t I). \tag{3}$$

In this equation, $q(s_{1:T}|s_0)$ represents the forward process, which is a product of conditional probabilities $q(s_t|s_{t-1})$ from $t = 1$ to T. The term $q(s_t|s_{t-1})$ is the conditional probability of state s_t given the previous state s_{t-1}, defined as a Gaussian distribution with mean $\sqrt{1-\beta_t} s_{t-1}$ and variance $\beta_t I$

Consequently, the forward process admits sampling s_t in a closed form at an arbitrary time step t, where α_t is the scaling factor for the noise at time step t and $\bar{\alpha}_t$ is the cumulative scaling factor up to the time step t and I is the identity matrix:

$$\alpha_t := 1 - \beta_t \ \ and \ \ \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s \,, \tag{4a}$$

$$q(s_t|s_0) = N(s_t; \sqrt{\bar{\alpha}_t} s_0, (1-\bar{\alpha}_t) I). \tag{4b}$$

When training the model, the aim is to optimise the variational bound L of negative log-likelihood [27], as demonstrated by Equation (5). In this equation, the left-hand side represents the expected negative log-likelihood of the data. The aim is to minimise the left-hand side as much as possible. The right-hand side is the variational bound, which is decomposed into the log probability of the prior distribution $p(s_T)$ and the sum of the log ratios of the reverse process probabilities $p_\theta(s_{t-1}|s_t)$ to the forward process probabilities $q(s_t|s_{t-1})$.

$$\mathbb{E}\left[-\log p(s_0)\right] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(s_{0:T})}{q(s_{1:T}|s_0)}\right] = \mathbb{E}_q = \left[-\log p(s_T) - \Sigma_{t\geq 1} \log \frac{p_\theta(s_{t-1}|s_t)}{q(s_t|s_{t-1})}\right] =: L. \tag{5}$$

The training objective $L_{Simple}(\theta)$ according to Ho, Jain and Abbeel [27] can be defined as presented in Equation (6), where the ϵ refers to the noise variable and ϵ_θ to the learned

gradient of the data density. The goal of the training is for the model to predict the learned gradient ϵ_θ during the reverse process:

$$L_{simple}(\theta) := \mathbb{E}_{t,x_0,\epsilon}\left[\left\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\right\|^2\right] \tag{6}$$

For the training objective, the model was reweighed so that values corresponding to small amounts of noise weigh less than the ones corresponding to more significant amounts of noise. This way, the network can focus on more challenging tasks. This training approach enhances the model's ability to generate data accurately [27].

2.1.2 Model architecture

The standard underlying structure of DDPM is a U-Net [49]. The U-Net is a convolutional neural network that owes its name to its architecture, which resembles the letter "U". The U-Net consist of two parts: the contracting and expanding path [52]. The first part consists of convolutional layers, followed by the ReLU activation function and max pooling layers. It performs feature extraction while increasing the number of feature channels and reducing the dimensionality [50]. The second path upsamples the feature map and halves the feature channels using convolution. It then concatenates this with the corresponding cropped feature map from the contracting path. This path helps regain the spatial information lost during the contraction path. The final layer is a convolutional layer which maps each feature vector to the different classes or categories, depending on the task. Additionally, the U-Net uses skip connections, which directly link layers from the contracting path to corresponding layers in the expansive path. The skip connection contributes to preserving details and spatial information while upsampling.

2.1.3 Model improvements

Ho, Jain and Abbeel [27] reported that the presented solution performed worse in log-likelihood values than other solutions presented in the literature. Nichol and Dhariwal [42] prove that DDPMs can achieve competitive log-likelihood while maintaining high sample quality.

Ho, Jain and Abbeel [27] observed that the best results for DDPM were observed with static variance σ_t^2 with $\Sigma_\theta(s_t, t)$ expressed by Equation (7).

$$\Sigma_\theta(s_t, t) = \sigma_t^2 I\,, \tag{7a}$$

$$\sigma_t^2 I = \beta_t. \tag{7b}$$

The $\Sigma_\theta(s_t, t)$ is the model's representation of the uncertainty in predicting the state s_t at time t.

The changes introduced by Nichol and Dhariwal [42] involve making the noise variance σ_t^2 not static. The authors argue that with the $\Sigma_\theta(s_t, t)$ defined as above, that range for the neural network would be too small for the model to predict the $\Sigma_\theta(s_t, t)$ directly. Thus, the researchers propose parameterising the log domain variance as an interpolation between β_t

and $\tilde{\beta}_t$. The output of the model proposed by Nichol and Dhariwal [42] is a vector v containing one component per dimension, and the variance can be obtained as shown in Equation (8b):

$$\tilde{\beta}_t = \frac{1-\bar{\alpha_{t-1}}}{1-\bar{\alpha_t}}\beta_t\,, \tag{8a}$$

$$\Sigma_\theta(x_t,t) = \exp(v\log\beta_t + (1-v)\log\tilde{\beta}_t). \tag{8b}$$

That modification also results in changes in the variational bound. Ho, Jain and Abbeel [27] only used L_{simple}, and now the Equation (9) has been updated by variational lower bond L_{vbl}. The hyperparameter λ controls the tradeoff between the two terms:

$$L_{hybrid} = L_{simple} + \lambda L_{vbl}. \tag{9}$$

Subsequently, the linear noise schedule introduced by Ho, Jain and Abbeel [27] is replaced by a different noise schedule, described by Equation (10). The constant offset b prevents the variance schedule β_t from being too small when close to $t=0$, which could prevent the model from accurately predicting the noise ϵ:

$$\bar{\alpha_t} = \frac{f(t)}{f(0)}, \quad f(t) = \cos(\frac{\frac{t}{T}+b}{1+b}\cdot\frac{\pi}{2})^2. \tag{10}$$

Such a construction of the noise schedule guarantees that the $\bar{\alpha_t}$ will remain almost the same near the extremes: $t=0$ and $t=T$ while having a linear drop-off in the middle of the process.

Additionally, Nichol and Dhariwal [42] observed that the gradient of the variational lower bond L_{vlb} was considerably noisier than the one of L_{hybrid}. To reduce the unnecessary noise in l_{vlb}, the importance sampling is used as shown in the Equation (11), where L_t is the log-likelihood at a time step t, the $\propto$ denotes proportionality, p_t is the probability distribution associated with time step t normalised such that Σp_t equals 1.

$$L_{vlb} = E_{t\sim p_t}\left[\frac{L_t}{p_t}\right], where\ p_t \propto \sqrt{E\left[L_t^2\right]}\ and\ \Sigma p_t = 1. \tag{11}$$

Described changes allowed not only for improvements in log-likelihood but also enabled the researchers to increase the sampling speed. By selecting arbitrary sequence $\mathcal{R}$ of t values $(0,1,2,...,T)$ and using the corresponding training noise schedule $\bar{\alpha_t}$ sampling variances β can be calculated as shown in Equation (12):

$$\beta_{\mathcal{R}_t} = 1 - \frac{\bar{\alpha_{\mathcal{R}_t}}}{\bar{\alpha_{\mathcal{R}_{t-1}}}},\ \tilde{\beta_{\mathcal{R}_t}} = \frac{1-\bar{\alpha_{\mathcal{R}_{t-1}}}}{1-\bar{\alpha_{\mathcal{R}_t}}}\beta_{\mathcal{R}_t}. \tag{12}$$

Such a change allows for making the diffusion process shorter as $\Sigma_\theta(s_{\mathcal{R}_t},\mathcal{R}_t)$ can be parametrised using $\beta_{\mathcal{R}_t}$ and $\tilde{\beta_{\mathcal{R}_t}}$. The original model proposed by Ho, Jain and Abbeel [27] needed 4000 sampling steps. The improved model needed only 100 to reach the same Fréchet Inception Distance (FID).

In conclusion, the authors of the paper Nichol and Dhariwal [42] proved that by introducing specific changes to the DDPM, the model can become much faster and achieve better log-likelihood without compromising the quality.

2.1.4 Classifier guided and classifier free model

The improved DDPM model introduced by Nichol and Dhariwal [42] is classifier-guided. The advantage of using another additionally trained classifier next to the model is that it improves the sample quality. The diffusion model's score estimate is mixed with the input gradient of the log-likelihood of the auxiliary classifier. The latter's strength can be adjusted, thus adjusting the trade-off between precision and recall. The modified score can be expressed as shown in Equation (13) [28]:

$$\tilde{\epsilon_\theta}(z_\lambda, c) = \epsilon_\theta(z_\lambda, c) - \omega\sigma_\lambda \nabla_{z\lambda} \log p_\theta(c|z_\lambda) \approx -\sigma_\lambda \nabla_{z\lambda} \left[\log p(z_\lambda|c) + \omega \log p_\theta(c|z_\lambda)\right] , \tag{13}$$

where z is the latent variable, c the class label, $\epsilon_\theta(z_\lambda, c)$ is the diffusion score and $\epsilon_\theta(\tilde{z_\lambda}, c)$ is the modified score to include the gradient of the log-likelihood of an auxiliary classifier model $p_\theta(c|z_\lambda)$. The λ represent the hyperparameters where $z = \{z_\lambda | \lambda \in [\lambda_{min}, \lambda_{max}]\}$. Increasing the weighting factor ω will increase the strength of the log-likelihood gradient in the modified score. The classifier rewards the correctly assigned labels with high log-likelihood probability in such a solution. The model's Inception score increases while the sample diversity decreases.

Nevertheless, there are downsides to using the classifier-guided model. First, training the whole DDPM becomes more complicated as an additional classifier requires training. Moreover, that classifier has to be trained on the noisy data; thus, it is almost impossible to reuse the trained classifier for another model. Secondly, Ho and Salimans [28] suggest that because the model's score also consists of the classifier's gradient, the sampling can be interpreted as a gradient-based adversarial attack on the image classifier. Therefore, the increase in the FID and Inception score of the model might simply be the result of the model being adversarial against the classifier.

Therefore, the paper's authors [28] present a classifier-free model to eliminate the need for the classifier and the related issues described above. In the classifier-free model, the score is a mix between the estimates of a conditional diffusion model $p(z|c)$ and an unconditional diffusion model $p(z)$. In the model the $p_\theta(z)$ is parametrised through $\epsilon_\theta(z_\lambda)$ and $p_\theta(z|c)$ through $\epsilon_\theta(z_\lambda|c)$. A single neural network with a null token $\varnothing$ is used for the parameterisation. Both of those models are jointly trained, and the score estimate $\epsilon_\theta(\tilde{z_\lambda}|c)$ can be described as shown in Equation (14):

$$\epsilon_\theta(\tilde{z_\lambda}|c) = (1+\omega)\epsilon_\theta(z_\lambda, c) - \omega\epsilon_\theta(z_\lambda) , \tag{14}$$

where ω is again the weighting factor. As there is no classifier gradient present, there is no possibility to interpret the step in the $\tilde{\epsilon_\theta}$ direction as an adversarial attack. Nevertheless, the trade-off between the Inception and FID scores is similar for guided and free classifier models.

2.2 Dimensionality reduction

Intuitively, dimensionality reduction is a technique that transforms high-dimensional data into a lower-dimensional form without significant loss of critical information. Reducing the input data size enhances the efficiency and performance of machine learning models, reducing

computational costs and memory requirements and making it feasible to process and analyse extensive datasets [56].

There are two approaches to dimensionality reduction: feature selection and feature extraction. Feature selection involves selecting a subset of the original features based on certain criteria, such as relevance to the target variable, statistical significance, or correlation with other variables. In contrast, feature extraction involves transforming the original features into a new set of features that capture the essential information from the dataset. This process reduces the overall dimensionality of the dataset, as the new features are designed to capture the most important patterns and structures [32]. Techniques such as SVD or AEs are commonly employed for feature extraction, enabling the extraction of meaningful patterns and structures in a lower-dimensional space.

2.2.1 Singular Value Decomposition

SVD is a linear and unsupervised technique for feature extraction. It decomposes a data matrix into three fundamental matrices: U, Σ, and V^T, as shown in Equation (15):

$$X \rightarrow U\Sigma V^T. \tag{15}$$

The data matrix X is typically an $m * n$ matrix, where m represents the number of samples (or observations) and n represents the number of features (or variables). Each row of X corresponds to a single sample, and each column corresponds to a specific feature. For example, in a dataset of images, each row might represent an image, and each column might represent a pixel intensity value. U and V are orthogonal matrices and Σ is a diagonal matrix containing the singular values. These singular values indicate the importance of the corresponding vectors in the original data matrix X.

To perform dimensionality reduction, the matrix X is reconstructed using only k largest singular values and their corresponding singular vectors:

$$X_k \rightarrow U_k\Sigma_k V_k^T, \tag{16}$$

where U_k consists of the first k columns of U, Σ_k is the top-left $k \times k$ submatrix of Σ, and V_k consists of the first k rows of V. This reduced representation X_k retains the most important features of the original data in a lower-dimensional space [5].

2.2.2 Autoencoder

AEs are artificial neural networks used to learn the input data representation. The AE consists of an encoder and a decoder. The role of the encoder is to compress the input into latent space. The latent space typically has a lower dimensionality than the input dimensionality. Therefore, the encoder has to reduce the dimensionality of the input while preserving the meaningful features. Subsequently, the decoder aims to reconstruct, as accurately as possible, the original input based on the output of the encoder. During the training, the AE learns to minimise the difference between the encoder's input and the decoder's output - the reconstruction error.

The Equation (17) describes the representation of the AE:

$$\hat{s} = \phi_{dec}(\phi_{enc}(s;\theta_{enc});\theta_{dec}) = \phi_{dec}(z;\theta_{dec}) \tag{17}$$

In the proposed notation for autoencoders, s is the datapoint, $\hat{s}$ its reconstruction, z the vector in the latent space Z, and θ is the vector of neural network parameters in some other space Θ. The latent space Z and the parameter space Θ are mapped to the original data space S by a function f where $f : Z \times \Theta \to S$.

The encoder ϕ_{enc} and the decoder ϕ_{dec} with parameters θ_{enc} and θ_{dec} respectively. The mapping of input s to latent representation is expressed as $z = \phi_{enc}(s;\Theta_{enc})$, while the mapping back to output as $\hat{s} = \phi_{dec}(z;\Theta_{dec})$. The reconstruction loss can be expressed as shown in Equation (18):

$$\min_{\theta_{enc},\theta_{dec}} L_{AE}(\theta_{enc}\,,\theta_{dec}) \tag{18a}$$

where:

$$\begin{aligned} L_{AE}(\theta_{enc},\theta_{dec}) &= \mathbb{E}_{s\sim\mathcal{D}}\left[\|s-\hat{s}\|^2\right] \\ &= \mathbb{E}_{s\sim\mathcal{D}}\left[\|s-\phi_{dec}(z;\theta_{dec})\|^2\right] \\ &= \mathbb{E}_{s\sim\mathcal{D}}\left[\|s-\phi_{dec}(\phi_{enc}(s;\theta_{enc});\theta_{dec})\|^2\right]. \end{aligned} \tag{18b}$$

In the Equation (18), the reconstruction loss L_{AE} is minimised over the encoder and decoder parameters θ_{enc} and θ_{dec}. The loss function L_{AE} is the expected value of the squared difference between the input s and its reconstruction $\hat{s}$, which can be expressed in terms of the decoder's output $\phi_{dec}(z;\theta_{dec})$ and the encoder's output $\phi_{enc}(x;\theta_{enc})$.

2.2.3 Variational Autoencoder

The variational autoencoder (VAE) is an extension of the traditional AE, designed to generate new data and learn more structured latent space representations. While standard AEs focus on compressing and decompressing data, VAEs introduce a probabilistic framework where the latent space is Gaussian distributed. This probabilistic nature allows VAEs to generate realistic data points by sampling from the learned latent space distribution, described by a normal distribution characterised by its mean and variance [35].

In VAE, the latent space can be described as $\mathcal{N}(\mu,\Sigma)$, where μ and Σ are the mean and covariance matrix, respectively, learned by the encoder for a given input. The VAE can be described as shown in Equation (19) [19]:

$$P(s) = \int P(s|z;\theta)P(z)dz\,, \tag{19}$$

where $P(s)$ represent the probability distribution of data s and the $P(s|z;\theta)$ is the likelihood of the data given the latent variable z and the parameter θ. The term $P(z)$ is the prior distribution of the latent variable z and the integral sums over all possible values of z.

The VAE is trained to minimise reconstruction and regularisation errors. The regularisation error is the Kullbach-Leibler divergence (KL), and it ensures that the learned latent space distribution matches the desired prior distribution [14]. The KL aims to bring the latent space distribution $\mathcal{N}(\mu, \Sigma)$ as close to the normal distribution as possible $\mathcal{N}(0, I)$, as shown in Equation (20):

$$\min_{\theta_{enc}, \theta_{dec}} L_{AE}(\theta_{enc}, \theta_{dec}) + L_{KL}(\theta_{enc}) \,, \tag{20a}$$

$$L_{AE}(\theta_{enc}, \theta_{dec}) = \mathbb{E}_{s\sim\mathcal{D}}[-\log \phi_{dec}(s|z; \theta_{enc}, \theta_{dec})] \,, \tag{20b}$$

$$L_{KL}(\theta_{enc}) = \mathbb{E}_{s\sim D}[\mathrm{KL}(\mathcal{N}(\mu, \Sigma)||\mathcal{N}(0, I))] \tag{20c}$$

In these equations, the total loss function to be minimised consists of two parts: the reconstruction loss L_{AE} and the KL divergence L_{KL}. The reconstruction loss L_{AE} is the expected negative log-likelihood of the data given the latent variable and the encoder and decoder parameters.

The motivation behind using VAEs lies in their ability to generate new data points similar to the training data, making them highly valuable for tasks such as image generation [48], anomaly detection [3], and data augmentation [31]. VAEs provide a more structured and interpretable representation than traditional autoencoders by explicitly modelling probability distributions in the latent space. This probabilistic framework also allows for better handling of uncertainty and variability in the data, leading to more robust and flexible models [48].

In conclusion, VAEs extend the capabilities of traditional AEs by leveraging a probabilistic approach in the latent space, enabling them to generate new data and better capture the underlying structure of the input.

2.3 Latent Diffusion Model

The previously discussed implementations of DDPMs operate in the pixel space [27, 42]. In this context, "pixel space" refers to the original, high-resolution representation of images, where every pixel corresponds to a specific point in the image grid (1:1 ratio). Rombach [49] proposes using a latent space to reduce computational demands, achieving a near-optimal trade-off between complexity and detail preservation.

Any likelihood-based model, such as the latent diffusion model (LDM), works in two stages. In the first stage, perceptual compression, high-frequency details are removed while retaining important semantic features. This reduction helps in lowering the computational burden by compressing the data into a lower-dimensional latent space. In the second stage, semantic compression, the generative model learns the data's semantic and conceptual composition, leveraging the compact latent representation to produce high-quality results.

For LDM, the first stage involves training an AE to compress the image into a lower-dimensional latent representation. This latent space is then used to train the diffusion model, which im-

proves scalability. The universal AE needs to be trained only once and can be reused for different tasks.

Another advantage of using an AE is the flexibility in choosing the compression level, which allows for balancing between perceptual compression and the generative model. Rombach [49] builds on this by proposing a distinct separation between the compressive and generative learning phases.

The first part of perceptual image compression can be mathematically expressed using Equation (21):

$$\hat{s} = \phi_{dec}(z) = \phi_{dec}(\phi_{enc}(s)) \,, \tag{21}$$

where $\hat{s}$ represents the reconstructed image after compression and decompression. The function $\phi_{dec}(z)$ denotes the decoder function ϕ_{dec} applied to the latent representation z. The full process $\phi_{dec}(\phi_{enc})$ shows how the encoder ϕ_{enc} compresses the original image s into the latent representation z, and then the decoder ϕ_{dec} reconstructs $\hat{s}$ from z.

Subsequently, the objective of the latent diffusion model is defined as:

$$L_{LDM} := \mathbb{E}_{\varepsilon(s),\epsilon\sim N(0,1),t}\left[\|\epsilon - \epsilon_\theta(zt,t)\|_2^2\right] \,, \tag{22}$$

where L_{LDM} is the loss function for the latent diffusion model. The term $\mathbb{E}\varepsilon(s), \epsilon \sim N(0,1), t$ denotes the expectation over the noise $\varepsilon(s)$, Gaussian noise ϵ sampled from a standard normal distribution $N(0,1)$, and time step t. The notation $||\cdot||_2^2$ represents the squared L_2 norm, which measures the square difference between the actual noise ϵ and the predicted noise $\epsilon\theta(z_t,t)$, where $\epsilon\theta(z_t,t)$ is the predicted noise by the model at time step t, parameterised by θ.

In summary, the first equation describes compressing an image into a latent representation and then reconstructing it, which helps manage computational complexity by working in a lower-dimensional space. The second equation defines the objective of the LDM, which aims to minimise the discrepancy between the actual noise and the predicted noise in the latent space.

Minimising this discrepancy is crucial because it impacts the quality and fidelity of the generated images. Accurate noise prediction allows the model to reverse the diffusion process effectively, leading to high-quality reconstructions and preserving fine details and overall coherence [45].

The application of LDMs has demonstrated their effectiveness across various domains, including image generation, editing [34], and video synthesis [12]. LDMs enhance model stability and performance by optimising noise prediction, resulting in improved visual outcomes and more efficient training processes. This makes LDMs a powerful tool for generating high-quality data with reduced computational resources [49].

2.4 Control

Control refers to the ability to influence a system's state behaviour. It ensures that the system will behave according to given objectives or instructions. The next part of this section delves deeper into control with conditional generative modelling.

2.4.1 Control using Conditional Generative Modeling

Conditional generative modelling is a machine learning approach where a model generates outputs based on specific conditions or context. Consequently, outputs generated by such a model can be tailored to match specific criteria, allowing for a more targeted generation of data.

Ajay [2] proposes implementing conditional generative modelling, specifically the DDPM, described in Section 2.1 for control. The benefit of this implementation is that the generated trajectory could consider more than one constraint simultaneously. Two datasets of trajectories could be used for a task that involves the robot moving from point A to point B while avoiding an obstacle at the beginning. The first one is where trajectories go from point A to B, and the second one has trajectories avoiding obstacles. Both sets of constraints can be integrated during trajectory generation by employing conditional generative modelling, such as the DDPM. This method allows the robot to learn from and adapt to various scenarios, effectively incorporating multiple constraints into the generated trajectories.

Two distinct approaches exist in the domain of conditional modelling with DDPMs: classifier-guided or classifier-free method, as detailed in Section 2.1.4. The conditional generative modelling for control, introduced by [2], adopts the classifier-free methodology.

2.4.2 Control with classifier-free DDPMs

Ajay [2] proposes using classifier-free DDPMs to perform control. As discussed in Section 2.4.1, this approach allows the generated trajectory to consider several constraints, rules, or desired skills. In the context of conditional generative modelling, the objective is typically to maximise the expectation of the log-likelihood of generating the initial state $s_0(\tau)$, given some information $y(\tau)$, as shown in Equation (23). This expectation is taken over trajectories τ sampled from the dataset $\mathcal{D}$:

$$\max_{\theta} \mathbb{E}_{\tau \sim \mathcal{D}}[\log p_\theta(x_0(\tau)|y(\tau))]. \tag{23}$$

In the Equation (23), θ represents the model parameters, $\mathbb{E}$ denotes the expectation, $\tau \sim \mathcal{D}$ indicates that the trajectories τ are sampled from the dataset $\mathcal{D}$, and p_θ is the probability distribution parameterised by θ.

In the context of diffusion models, the generative process is governed by both forward (3) and reverse (2) diffusion processes, as outlined in Section 2.1.1. In Ajay's implementation, diffusion is applied only to state variables (not actions), ensuring continuous data flow across

states. The trajectory τ consists of states s and actions a, with the state transitions at each step represented as:

$$\tau_k := (s_t, s_{t+1}, ..., s_{t+T-1})_k \,, \tag{24}$$

where τ_k represents the state at step k within the trajectory τ. The sequence of states starting from time t to $t+T-1$ at step k is denoted as $(s_t, s_{t+1}, \ldots, s_{t+T-1})k$. Here, s_t is the state at time t, $st+1$ is the state at the next time step, and so on, up to s_{t+T-1}, which is the state at the $(t+T-1)$-th time step. This sequence captures the transitions between states over a horizon of T steps.

However, only using states to sample trajectories will not be sufficient to obtain a controller, as actions are also necessary to determine the control inputs. While states are continuously distributed, actions can introduce discontinuities that make direct modelling more challenging. Therefore, Ajay [2] proposes to estimate the action by looking at two consecutive states in an inverse dynamics manner, as shown in Equation (25):

$$a_t := f_\phi(s_t, s_{t+1}). \tag{25}$$

In this equation, a_t represents the action at time t, and f_ϕ is a function parameterised by ϕ that estimates the action based on the states s_t and s_{t+1}. By inferring actions in this way, the model avoids issues related to discontinuity in the action space and ensures that control decisions are based on state transitions.

The following step is to use the constructed model for planning. To include the information $y(\tau)$, the author of the paper [2] uses the classifier-free denoising model described in Section 2.1.4 with low-temperature sampling to obtain the trajectories with the highest log-likelihoods. Those trajectories also describe the best, most desirable behaviours from the dataset. The model can be described as shown in Equation (26), where $\hat{\epsilon}$ is the model estimate for planning:

$$\hat{\epsilon} := \epsilon_\theta(\tau_k), \varnothing, k) + \omega(\epsilon_\theta(\tau_k), y(\tau), k) - \epsilon_\theta(\tau_k), \varnothing, k)). \tag{26}$$

Additionally, the term $\epsilon_\theta(\tau_k), \varnothing, k)$ is the model's estimate of the noise at step k for the state τ_k without any conditional information. The term $\epsilon_\theta(\tau_k), y(\tau), k)$ is the model's estimate of the noise with the conditional information $y(\tau)$. The difference between these two terms is scaled by the factor ω, which adjusts the influence of the conditional information $y(\tau)$ on the noise estimation.

The authors of the paper reason that with the presented construction of the model, the sampling from the decision diffuser becomes similar to the planning in reinforcement learning: first observing a state in the environment, sampling the states into the horizon, and finally identifying the best action to take.

Additionally, the conditional variable $y(\tau)$ can be defined in various ways depending on the desirable behaviour of the model. For example, if the goal is to maximise the return function $R(\tau)$, the conditional variable $y(\tau)$ should be set as follows:

$$\epsilon_\theta(\tau_k, y(\tau), k) := \epsilon_\theta(\tau_k, R(\tau), k) \,, \tag{27}$$

where $\epsilon_\theta(\tau_k), R(\tau), k)$ represents the noise estimate conditioned on the return function $R(\tau)$.

Furthermore, the $y(\tau)$ can also be adjusted to comply with various constraints, each represented by set $\mathcal{C}_i$. To generate trajectories which will satisfy a constrain $\mathcal{C}_i$, for example, avoiding certain parts of the state space,e the $y(\tau)$ is conditioned on a one-hot encoding:

$$\epsilon_\theta(\tau_k, y(\tau), k) := \epsilon_\theta(\tau_k, 1(\tau \in \mathcal{C}_i, k). \tag{28}$$

Finally, the noise model can also be conditioned to show a skill i which can be specified from a set of demonstrations $\mathcal{B}_i$, as follows:

$$\epsilon_\theta(\tau_k, y(\tau), k) := \epsilon_\theta(\tau_k, 1(\tau \in \mathcal{B}_i, k)\,, \tag{29}$$

where $1(\tau \in \mathcal{B}_i)$ is a one-hot encoding indicating whether the trajectory τ demonstrates the skill i from the set $\mathcal{B}_i$

The training is done offline for constraints and skills; therefore, only one skill or constraint can be trained at a time. However, multiple constraints and skills can later be composed at interference to achieve a complex model.

The reverse diffusion process p_θ and inverse dynamic model f_ϕ are then trained, given dataset D of trajectories, where each has a return, constraint or skill. The reverse diffusion process is parametrised through the noise model ϵ_θ using temporal U-net architecture, and the inverse dynamic model f_ϕ has a loss described by Equation (30):

$$\mathcal{L}(\theta, \phi) := \mathbb{E}_{k,\tau,D,\gamma\sim\text{Bern}(\pi)}[||\epsilon - \epsilon_\theta(x_k(\tau), (1-\gamma)y(\tau) + \gamma\varnothing, k)||^2] + \mathbb{E}_{(s,a,s')\in D}[||a - f_\phi(s, s')||^2] \tag{30}$$

The first expectation term in this equation involves the denoising process using the reverse diffusion model ϵ_θ. The variable γ is a binary variable sampled from a Bernoulli distribution with parameter π. The first part of the loss term measures the difference between the true denoising score ϵ and the denoising score predicted by the model. The second part involves the loss for training the inverse dynamic model f_ϕ. The loss is calculated using the squared L_2 norm between the true action a and the predicted action $f_\phi(s, s')$.

3 Related Work

Understanding previous research in the field is essential for progressing further with advancements. Therefore, with the support of relevant papers and publications, this section explains major topics and concepts that are the foundation of this paper.

3.1 Numerical and Analytical PDE solvers

3.1.1 Analytical solvers

Classical analytical methods for PDEs have been used in mathematical analysis for decades. Techniques such as the separation of variables, Fourier and Laplace transform, and Green's functions are commonly employed to find exact solutions to PDEs [21]. For instance, the separation of variables simplifies a PDE into a set of ordinary differential equations (ODEs), which can be more easily solved. Fourier and Laplace transform convert PDEs into algebraic equations in the frequency domain, facilitating their solution through inverse transforms [7].

However, these methods have significant limitations. They often apply only to linear PDEs with specific boundaries and initial conditions and struggle with nonlinearity and complex geometries [43]. The exact solutions provided by analytical methods are important for understanding the fundamental properties of PDEs, but their applicability is restricted to a narrow class of problems.

3.1.2 Numerical solvers

Numerical methods have been developed to overcome the limitations of analytical approaches, providing approximate solutions to a broader range of PDEs. The Finite Difference Method (FDM) discretizes the PDE using a grid of points, approximating derivatives with differences [54]. This method is straightforward and effective for simple geometries, but its accuracy depends on the grid resolution, which can lead to high computational costs.

The Finite Element Method (FEM) offers greater flexibility by dividing the domain into smaller, irregularly shaped elements, making it well-suited for complex geometries and boundary conditions [39]. FEM constructs approximate solutions using basis functions, providing high accuracy with relatively fewer elements compared to FDM.

The Finite Volume Method (FVM) is another powerful numerical approach that integrates the PDE over control volumes, ensuring the conservation of changes across their boundaries [4]. This method is particularly effective for problems involving conservation laws and fluid dynamics.

However, these methods have limitations. FDM can be less accurate on complex geometries, and FVM may require complex reconstruction schemes for higher-order accuracy. FEM, while powerful, can be computationally intensive and time-consuming [44]. Additionally, Götschel

and Weiser [25] argue that the growing disparity between computational power and memory bandwidth poses a challenge for large-scale problems.

3.1.3 Curse of Dimensionality

The term "curse of dimensionality" was first introduced in 1950 by Bellman [9]. In the context of PDEs, it refers to an exponential increase in computational complexity and resource requirements when solving PDEs analytically or numerically as the number of dimensions (spatial or temporal variables) increases. Despite advancements in numerical methods, this phenomenon remains a significant challenge across various PDEs.

High dimensionality leads to large, sparse matrices that are computationally intensive. Moreover, the accuracy of numerical methods diminishes as dimensionality increases, necessitating finer grids or more elements, further worsening the computational burden. These challenges underscore the need for novel approaches that can efficiently handle high-dimensional PDEs without succumbing to the curse of dimensionality.

3.2 Neural Network solvers

In recent years, neural networks have emerged as a tool for solving PDEs, offering a promising alternative to traditional numerical and analytical methods. Unlike conventional approaches that rely on grid-based discretizations or analytical simplifications, neural networks can directly learn and approximate complex functions from data, making them particularly well-suited for high-dimensional and nonlinear problems [1]. Moreover, by training on examples of input-output pairs that represent the PDE's behaviour, neural networks can ideally generalize to new scenarios and provide accurate solutions across various conditions. Additionally, neural networks can be used as model order reduction tools for solving PDEs, providing computational efficiency by solving the PDE with a simple forward pass of the network [33].

The forward pass refers to the process of passing input data, such as initial conditions or parameters of the PDE, through the neural network to produce an output, which is the predicted solution of the PDE. By leveraging the pre-trained weights and biases of the network, the forward pass efficiently maps the inputs to the desired outputs without the need for iterative solvers or complex numerical methods. This allows the PDE to be solved almost instantaneously, making neural networks particularly advantageous for scenarios requiring rapid or real-time computations, such as control and optimization tasks [33].

3.2.1 Deep Neural Networks

Deep Neural Networks (DNNs) represent a significant advancement over traditional neural networks due to their ability to leverage multiple layers of interconnected neurons. This deep architecture allows DNNs to model complex mappings from inputs, such as spatial coordinates and time, to outputs corresponding to PDEs solutions [41]. By exploiting hierarchical repre-

sentations within the solution space, DNNs can capture intricate patterns and dependencies that may not be discernible to shallower models.

However, the depth of DNNs introduces several challenges when applied to PDE solving. Firstly, their increased depth and complexity lead to higher computational demands during training and inference phases [51]. This computational overhead can make DNN-based PDE solvers slower and more resource-intensive than traditional numerical methods. Secondly, the large number of parameters in deep networks makes them prone to overfitting, where the model learns to memorize training data rather than generalize to unseen data [16]. This issue can compromise the robustness and reliability of DNN solutions, especially in complex, real-world applications.

3.2.2 Physics-informed Neural Networks

Physics-Informed Neural Networks (PINNs) directly integrate domain knowledge into training, enhancing solutions' accuracy and physical fidelity. PINNs enforce the underlying physics governing the PDEs as constraints during training, ensuring that solutions respect conservation laws, boundary conditions, and other physical principles [18]. This approach improves the reliability of neural network predictions and reduces the dependency on large amounts of labelled data, which can be scarce or costly.

However, despite these advantages, PINNs encounter significant challenges in practical applications. One notable difficulty arises when approximating solutions for PDEs that exhibit multi-scale, chaotic, or turbulent behaviours, which are prevalent in many physical systems [6]. The complex interactions and rapid changes across different scales pose difficulties for neural networks to effectively capture and predict accurately.

Additionally, Antonion et al. [6] highlight another challenge PINNs face: the computational intensity and training complexity, especially for high-dimensional problems or those requiring fine spatial or temporal resolution. Optimizing complex neural network architectures over large datasets demands substantial computational resources and time, making the training process resource-intensive.

3.2.3 Latent Spectral Models

Latent Spectral Models (LSM) use a special network to simplify complex data by condensing high-dimensional inputs and outputs into a simpler form. Inspired by traditional methods, LSM uses a neural block to break down complicated mappings into simpler parts, ensuring it works well in theory and practice. Wu et al. [61] show that LSM performs exceptionally well in solid and fluid physics tests, offering better accuracy and efficiency than older methods.

Despite its advancements, LSM faces potential pitfalls. Firstly, the design and implementation of LSM require careful tuning of hyperparameters and architectural choices, which can be non-trivial and time-consuming. Moreover, LSM's applicability to extremely high-dimensional

or highly dynamic systems might be limited, as capturing all relevant features and interactions in such scenarios could still pose challenges [61].

3.2.4 Neural Operators

Neural operators are designed to map between function spaces, providing a flexible and efficient framework for solving complex, high-dimensional problems such as those involving PDEs. Unlike traditional neural networks that map individual data points, neural operators handle variable input sizes and geometries by learning the underlying operator that governs the transformation from inputs (e.g., initial and boundary conditions) to outputs (e.g., PDE solutions). This capability allows neural operators to generalize across resolutions and problem setups, making them highly versatile [36].

Kovachki et al. [36] propose a set of neural operators, with the Fourier Neural Operator (FNO) highlighted as the fastest and most competitive approach. By using relative L2 error, as shown in Equation (31), the authors compared the FNO with other available PDE solvers: CN, a Fully Convolution Network [63], PCA+NN, which uses Principal Component Analysis (PCA) combined with a fully connected neural network [11], RBM, the classical Reduced Basis Method with PCA [17] and DeepONet, which employs a Deep Operator Network with standard fully connected layers and a solid approximation theory [38]. The performance of these solvers was assessed using the relative L2 error, calculated as shown in Equation (31):

$$L_2 = \frac{\sqrt{\Sigma(s_t - \hat{s_t})^2}}{\sqrt{\Sigma s_t^2}}\,, \tag{31}$$

where s_t is the original solution at a time step t, and $\hat{s_t}$ is the generated solution at a time step t. The analysis specifically measures the solution at a fixed future time step rather than the entire solution trajectory. For instance, in the context of the one-dimensional viscous Burgers' equation, the goal is to learn the mapping from the initial condition to the solution at this fixed future time. For assessing the FNO on this 1D PDE case, the relative L_2 error was in the range of 10^{-3}, whereas other methods reported error values in the interval of 10^{-1} to 10^{-2}.

3.2.5 Model-Order Reduction

Model order reduction (MOR) techniques aim to simplify the complex systems described by PDEs by reducing the number of degrees of freedom while retaining essential dynamics and accuracy. When used in MOR, neural networks can efficiently capture the system's underlying patterns and key features, enabling faster and more scalable solutions. This approach involves training the network to project the high-dimensional PDE problem onto a lower-dimensional latent space, from which accurate approximations of the original solution can be reconstructed. As a result, neural network-based MOR can significantly reduce computational costs and enhance the feasibility of solving large-scale PDEs [11].

3.3 PDE Control

Control of PDEs involves influencing the behaviour of a system described by PDEs through external inputs or controls. The goal is often to optimise performance criteria or steer the system to a desired state. Traditional control strategies often rely on solving complex optimisation problems to determine the best control inputs. In contrast, modern approaches leverage deep learning and reinforcement learning techniques, which provide innovative methods for managing high-dimensional and nonlinear systems, offering new possibilities for enhancing control and achieving desired outcomes.

3.3.1 Optimal control

Optimal control is a mathematical and computational framework that can be applied to influence the behaviour of systems governed by PDEs [40]. The primary objective is to optimise a given performance criterion, such as minimising costs or achieving desired system states. This process involves solving complex optimisation problems where the control inputs are designed to steer the system towards optimal performance [55].

Like traditional approaches for solving PDEs, as described in Section 3.1, optimal control also suffers from the curse of dimensionality, which can render solutions infeasible for large-scale problems. Other challenges include the computational complexity associated with high-dimensional systems, issues with nonlinearity and stability, and sensitivity to model inaccuracies and boundary conditions. Additionally, ensuring the uniqueness and stability of solutions, along with implementing these strategies in real-time applications, can be difficult. These challenges underscore the need for advanced numerical techniques and robust control approaches.

Modern approaches to optimal control, such as deep learning and reinforcement learning, address some of these challenges by leveraging data-driven techniques and advanced computational methods.

3.3.2 Deep Learning

Long et al. [37] introduced PDE-Net, a deep learning framework designed to achieve two main objectives: accurate prediction of complex system dynamics and identification of the underlying partial differential equations (PDEs) governing these systems. PDE-Net utilises convolutional filters, learned directly from the data, to represent differential operators while simultaneously learning a nonlinear response function that captures the system's inherent nonlinearity.

In the control context, PDE-Net can design control strategies by predicting the system's future states and adjusting control inputs accordingly. By learning the underlying PDEs, PDE-Net enables the development of data-driven controllers that can handle complex, high-dimensional systems. This approach allows real-time adjustments and improved control performance,

even in noisy data. Uncovering hidden PDE models also aids in refining control strategies by providing deeper insights into the system dynamics, thus enhancing the overall effectiveness of the control process.

Another deep learning approach proposed by Holl et al. [29] aims to understand and control complex nonlinear physical systems over time. This method divides the problem into planning and control tasks, featuring a predictor network that plans optimal trajectories and a control network that infers the corresponding control parameters. Both networks are trained end-to-end using a differentiable PDE solver, allowing the system to learn from interactions and refine predictions over time. The authors state that by recursively refining predictions and correcting deviations, this hierarchical predictor-corrector scheme outperforms traditional iterative optimisation methods, providing stable long-term control.

3.3.3 Reinforcement Learning

Farahmand et al. [22] introduced a data-driven method for PDE control using deep reinforcement learning (RL) techniques, specifically through the Deep Fitted Q-Iteration (DFQI) algorithm. DFQI directly manages high-dimensional state representations of PDEs, avoiding the model order reduction step common in traditional methods. DFQI handles complex state spaces and learns from data without manual feature design by utilising deep convolutional networks to approximate the RL value function. Applied to control flow in a time-varying 2D convection-diffusion PDE, the method also explores transfer learning, adapting policies trained on one PDE to other related PDEs.

Building on the advancements in data-driven PDE control, Peitz et al. [46] proposed a novel convolutional framework for distributed control of dynamical systems governed by PDEs, leveraging reinforcement learning (RL). Their approach further reduces the complexity of high-dimensional PDE control problems by transforming them into multi-agent systems with identical, uncoupled agents. This method addresses the curse of dimensionality inherent in deep RL and enhances the scalability and transferability of trained models across different domains. By incorporating distributed control strategies, Peitz et al.'s framework complements the efforts of Farahmand et al. in managing complex PDE control challenges, offering a unified approach that integrates advanced RL techniques for more effective and adaptable solutions.

Additionally, the control of fluids has been explored with Deep Reinforcement learning (DRL). Vignon et al. [58] reviewed the application of DRL to active flow control (AFC), demonstrating its potential in managing high-dimensional and nonlinear fluid dynamics problems. The paper discussed the effectiveness of DRL in two-dimensional and chaotic conditions and provided promising results in increasingly complex flows. Additionally, Vinuesa et al. [59] explored the transformative potential of machine learning, including DRL, for experimental fluid mechanics, emphasising its role in real-time estimation and control of fluid flows.

3.4 Denoising Diffusion Probabilistic Model

The results of the DDPM presented by Ho and Salimans [28] show that the model can perform better in quality and FID scores than most of the other generative models described in the literature. However, the model was underperforming regarding log-likelihood values compared to other results in the literature.

The improvements introduced by Nichol and Dhariwal [42], explained in Section 2.1.3 solved that issue while maintaining the high sample quality. Furthermore, when comparing the model with GANs, the DDPM obtained a much higher recall for similar FID, which suggests that DDPM can be more effective in generating data points that closely match the actual data distribution, capturing a larger portion of its characteristics. Additionally, the original model proposed by Ho and Salimans [28] needed 4000 sampling steps. The improved model needed only 100 to reach the same FID, which greatly reduced the model's training time.

The DDPM presented by Nichol and Dhariwal [42] is a classifier-guided model. Ho and Salimans [28] introduce a classifier-free model that outperforms the classifier-guided one while having a simpler architecture and training pipeline. Additionally, changes to the model introduced by Rombach [49] show that using latent data representations as input to the DDPM model can significantly improve the sampling and training efficiency without loss in quality.

3.4.1 Control using DDPM

DDPMs can also be applied to control nonlinear systems, as demonstrated by Elamvazhuthi et al. [20]. In this approach, the control problem is reframed as a generative modelling task, aiming to design a feedback controller that transitions the system's state distribution from an initial to a desired target configuration. The method ensures that the controlled system adheres to the reverse trajectory of the DDPM's forward process to achieve alignment with the target distribution. The authors used KL divergence to assess the controller's performance, demonstrating its effectiveness across various nonlinear systems without drift; however, they did not compare this approach with other existing methods.

Huang et al. [30] introduced DiffuseLoco, a novel framework that utilizes DDPMs to train multi-skill, diffusion-based policies for dynamic-legged locomotion using offline datasets. Traditionally, robotics research has focused on specialized single-skill models for dynamic motions, leaving the challenge of unifying diverse locomotion skills unsolved. DiffuseLoco addresses this gap by leveraging DDPMs to learn from multimodal offline datasets containing various abilities, such as bipedal walking and quadrupedal locomotion, without requiring manual skill labelling. By effectively capturing the multi-modalities in these datasets, DiffuseLoco integrates a wide range of agile-legged locomotion skills into a single scalable policy, enabling real-time deployment on robots and ensuring robust performance across diverse locomotion tasks.

3.5 Control of PDE using DDPM

Ajay [2] used DDPM for offline control in generating trajectories and showed that such a solution could maximise returns, satisfying more than one constraint and performing different skills together. The research of Botteghi [15] supports this observation, where trajectories generated using DDPM adhered to predefined safety rules, avoiding restricted areas.

Diffusion Generative PDE Control (DiffConPDE), introduced by Wei et al. [60], is a framework designed to optimize control for PDEs. Central to this method is an energy-based model that captures both the dynamics and constraints of the PDE system. This model includes an energy function parameterized to represent the distribution of system states and control inputs, taking into account system constraints. The energy function quantifies the combined cost of system evolution and control objectives, incorporating a learned generative component to ensure that the generated state-control pairs adhere to the system's dynamics and boundary conditions.

DiffConPDE further enhances its optimization capability through a technique called prior reweighting. This approach adjusts the influence of prior knowledge during training, allowing the framework to explore control strategies beyond the scope of the training data. By doing so, DiffConPDE can identify superior control solutions that align effectively with the PDE system's dynamics and the specified control objectives. For 1D Burgers' equation, DiffConPDE achieves the lowest control error across different scenarios, outperforming PID and SAC.

To support the evaluation of PDE control problems and safety-critical industrial settings, Zhang [62] introduced `controlgym`, a library that provides examples and tools for assessing PDE dynamics, implementing controllers, and generating PDE trajectories.

4 Methodology

The overall design and implementation of the model used to generate trajectories from PDEs are divided into three key stages: (i) data collection, (ii) learning the system's dynamics and control strategies with DDPM and optionally (iii) dimensionality reduction. After all those stages, the generated solution is obtained. Figure 1 illustrates the complete process, where the light purple section highlights the initial data collection and the solution generation stages. The blue section corresponds to the DDPM, and the purple box encapsulates control strategies.

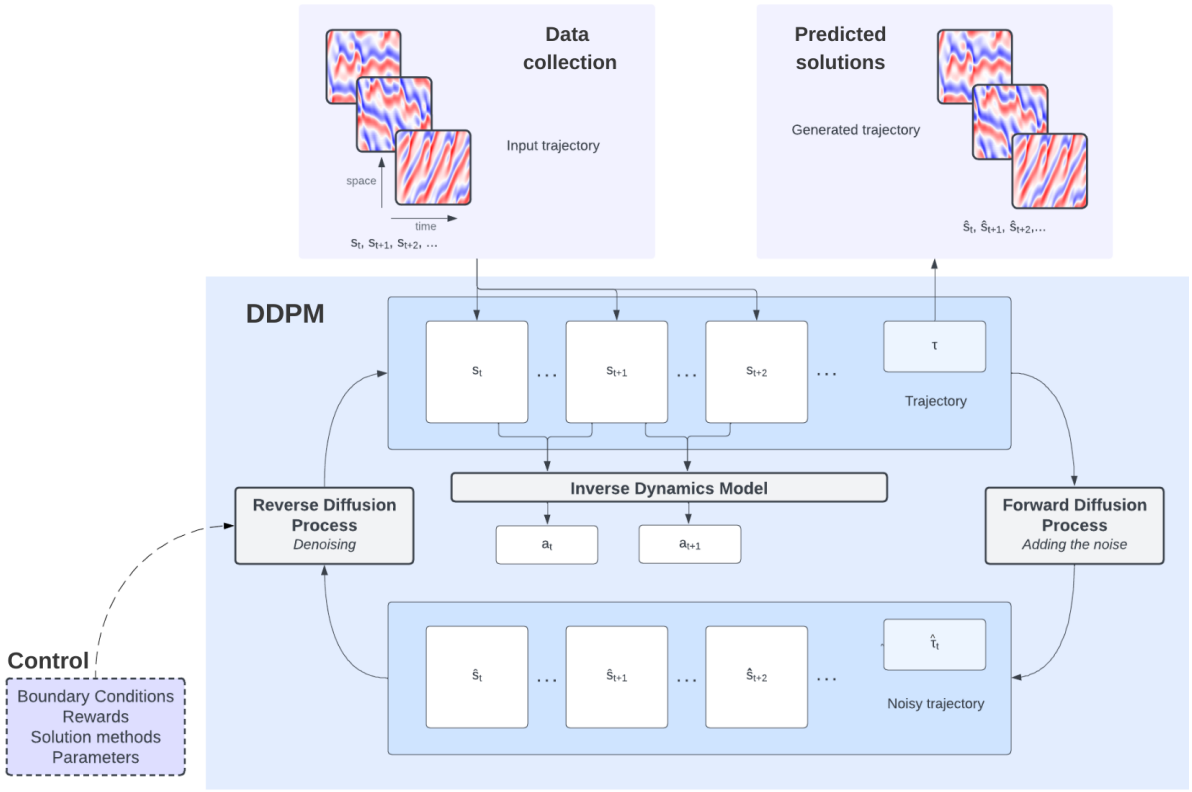

Figure 1: Schematics of the basic model.

4.1 Data collection

The `controlgym` library allows for simulating various PDE-based control problems, both linear and nonlinear equations. This project focuses on using the nonlinear 1D Kuramoto-Sivashinsky (KS) equation as input for the model, with more details to be provided in the results in Section 5.1. In addition to generating standard versions of specified PDE environments, this library facilitates the customization of equation parameters and the implementation of controllers. Model's input and output will be a trajectory $\tau = s_t, s_{t+1}, s_{t+2}, ..., s_{t+T-1}$, where T is the horizon's length. The s_t represents a state of the system at a time t. The state describes various physical quantities or variables that describe the system's behaviour, which vary based on the PDE selected. The state s_t at time t is followed by the state s_{t+1} at time $t+1$. This sequence of states forms a trajectory, which captures the system's evolution over time.

In addition to time discretisation, space discretisation is essential to fully understanding the structure of each state s_t. For a 1D PDE, each state s_t belongs to $\mathbb{R}^L$, where L is the number of spatial steps used to discretise the domain. This discretisation allows the PDE to be represented in a finite-dimensional space, making it suitable for numerical simulations and modelling.

Consequently, the total trajectory τ can be viewed as a matrix of size $L \times T$ for the 1D PDE case, where each column corresponds to the state at a particular time step, and each row corresponds to a spatial position. This matrix encapsulates the complete spatiotemporal evolution of the system over the prediction horizon T.

These assumptions imply that the model learns to produce solutions of PDEs by approximating the solution operator implemented by the discrete numerical solver used. This approach enables a data-driven approximation of PDE solutions for given input conditions. The model, therefore, learns to replicate the discrete numerical solver's behaviour for given input conditions, effectively providing a data-driven approach to approximating the underlying PDE solutions.

4.2 Learning dynamics and Control strategies with DDPMs

The input to the model, the trajectory τ of length T, can be represented as a sequence of states and actions: state-action-state-action, and so forth as described in Section 2.4.2, and more specifically, Equations (24) and (25). In this sequence, each action represents a control input that drives the system from one state to the next.

Action is based on a current state and used to derive the subsequent state. Therefore, knowing only the sequence of states makes it possible to derive the corresponding action $\hat{a}_t$ using an inverse dynamics model $\mathcal{I}$ approximated by a neural network, as shown in the Equation (32):

$$\hat{a}_t = \mathcal{I}(s_t, s_{t+1}) \tag{32}$$

Consequently, only the states are the subjects of the forward diffusion process and, subsequently, the reverse diffusion process.

The reverse process can incorporate control requirements such as boundary conditions, rewards, and parameters, allowing the model to generate trajectories that align with specific control objectives; see the purple box in Figure 1. The reverse process performs conditional sampling by embedding these requirements, producing new trajectories tailored to meet predefined conditions or criteria. For instance, boundary conditions can enforce constraints on the trajectory's behaviour at the system's boundaries, while rewards can incentivize desired behaviours or outcomes. This approach effectively integrates control objectives into the generation process, ensuring that the resulting trajectories adhere to the desired specifications.

4.3 Dimensionality reduction

The PDEs in their raw form are extremely complex due to their high-dimensional nature. Moreover, numerous parameters and boundary conditions can influence their behaviour.

Encoding PDEs into a latent space offers an approach to address these issues. By transforming the high-dimensional input space of the PDEs into a lower-dimensional latent space, complex relationships and underlying patterns within the data can be more effectively captured and represented. This information compression reduces the computational load and facilitates the discovery of meaningful representations and features that may otherwise be obscured in the original high-dimensional space. Given the high dimensional data $s_t, s_{t+1}, ..., s_{t+T-1}$, SVD and AE are utilized to learn low dimensional representation $z_t, z_{t+1}, ..., z_{t+T-1}$. Figure 2 shows the updated schematics with the grey box decided for the dimensionality reduction.

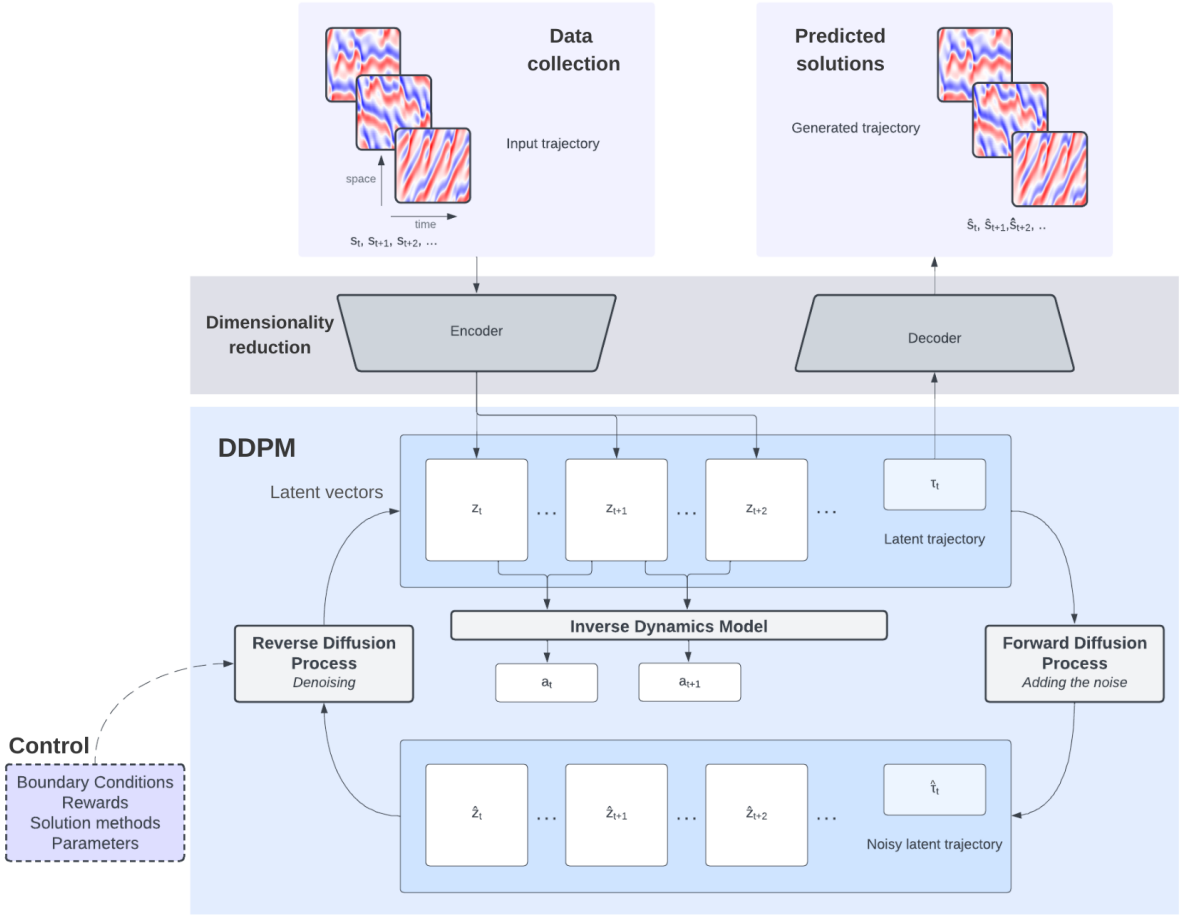

Figure 2: Schematics of the full model, including encoding to the latent space.

4.4 Training

The training process varies depending on the case. For the basic case, where no dimensionality reduction is applied, the training is straightforward. The model is trained directly on the original input trajectories, allowing it to learn the dynamics and control strategies in a single phase.

In contrast, when dimensionality reduction is employed, the training process is divided into two distinct phases. Firstly, the focus is training the AE or applying the SVD to the input

trajectories for dimensionality reduction. During this phase, the objective is to effectively capture the underlying structure of the input trajectories and represent them in a reduced form.

In the second phase, the DDPM is employed using the reduced-dimensionality data obtained from SVD or the AE. This two-stage approach enables the model to focus exclusively on learning the data's latent representations. By eliminating the need to encode and decode these representations back to their original dimensions during this phase, the DDPM can fully learn the dynamics and control strategies in the latent dimension.

5 Results

5.1 Kuramoto-Sivashinsky PDE

The Kuramoto-Sivashinsky (KS) equation is the Partial Differential Equation (PDE) used for the subsequent experiment. The solutions for this PDE, generated by discretising the space and time domains, are used to train the model. The KS equation is a nonlinear PDE used in various contexts, such as fluid dynamics, plasma physics and combustion. It is particularly known for modelling the behaviour of diffusive instabilities in laminar flame fronts [62]. The implementation of the KS equation in this paper follows the method presented by Botteghi in [13], as shown in Equation (33):

$$\begin{aligned} \frac{\partial s}{\partial t} + s\frac{\partial s}{\partial x} + \frac{\partial^2 s}{\partial x^2} + \frac{\partial^4 s}{\partial x^4} + \mu \cos\left(\frac{4\pi x}{L}\right) &= \phi(x,a), \\ \phi(x,a) &= \Sigma_{i=1}^{N_a} a_i \psi(x,c_i), \\ \psi(x,c_i) &= \frac{1}{2}\exp\left(-\frac{(x-c_i)^2}{\sigma^2}\right). \end{aligned} \tag{33}$$

The variable $s = s(t,x)$ represents the state of the system, where x is the spatial variable and t is the time variable. The function $\phi(x,a)$ denotes the control input function, with $a_i \in [-1,1]$ as control variables, while $\psi(x,c_i)$ represents a Gaussian kernel centred at c_i with a standard deviation of $\sigma = 0.8$. The number of control parameters is denoted by N_a, and each a_i scales the corresponding Gaussian kernel $\psi(x,c_i)$. Other parameters, such as the domain size L, the spatial domain discretization, and the main parameter μ, which scales the cosine forcing term, will be varied in the experiments detailed in the following section.

An important property of the KS equation, particularly under periodic boundary conditions, is the conservation of the spatial mean of the solution $s(t,x)$ when the $\mu = 0.0$. This conservation can be shown by integrating the KS equation over the spatial domain, where the integral of the nonlinear, second-order, and fourth-order derivative terms vanishes due to the boundary conditions. As a result, the spatial average of the state $s(t,x)$ remains constant over time.

5.2 Experiments Outline

According to the pipeline architecture described in Section 4, the following experiments were conducted. First, during the inference phase, the model's ability to generate solutions for the KS PDE for different parameter μ values was tested. This evaluation can be distinguished into two cases: an unseen μ value within the training range (parameter interpolation) and an unseen μ value outside the training range (parameter extrapolation).

Furthermore, Singular Value Decomposition (SVD) and autoencoder (AE) were introduced. These modifications reduced the PDE to a smaller dimension within the latent space before passing the data through the diffusion model. This experiment was conducted again for multiple values of parameter μ, with the original dimension of the solution being reduced to $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{8}$.

Lastly, the experiments assessed the control properties of the model. In this setup, a controller was first applied to the PDE to generate controlled solutions, which were then used to train the model. The trained model was used to generate controlled solutions during the inference phase. This approach was tested on a 1-dimensional PDE where the spatial domain is discretised to 64 elements with different parameter μ values. In this experiment, no dimensionality reduction was applied to the input data before it was processed by the model, allowing for the evaluation of its performance directly in the original input space.

The evaluation metric for the model during training is the model loss; see Equation (6). After the training is complete, prediction error, as shown in (34) is used to evaluate the model's performance:

$$\text{Prediction error} = 0.5 \times \sqrt{\frac{(\|\tau - \hat{\tau}\|)^2}{(\|\tau\|)^2}}. \tag{34}$$

The prediction error utilises the Frobenius norm to measure the difference between the original trajectory τ from the test set and the generated trajectory $\hat{\tau}$.

The absolute error, see Equation (35), serves as a means to provide visual information about the error distribution:

$$\text{Absolute Error} = |\tau - \hat{\tau}| \ , \tag{35}$$

In addition, the model is also evaluated using inpainting techniques. Inpainting involves filling in missing parts of a trajectory based on the observed data. Evaluation through inpainting is performed under three distinct scenarios: from the beginning (sweep front), from the end (sweep back), and from both sides of the trajectory (sweep both).

- **Sweep front:** In this scenario the evaluation focuses on how a system progresses forward in time. It assesses the model's capability to generate future states based on the initial part of the trajectory. This setup tests the model's ability to predict time evolution when only the beginning segment of the trajectory is known. It evaluates the model's performance in time extrapolation, determining how well it can infer unseen future states based on limited prior information.
- **Sweep back:** In this scenario, the evaluation starts with the system's final state and aims to infer the preceding states. This helps assess the model's ability to reconstruct the history of the trajectory from its endpoint. While this might seem like a form of time interpolation, it is not strictly so, as the first step in the sequence must be provided for the model to effectively generate prior states (see Figure 54).
- **Sweep both:** This scenario addresses situations where there is a break in data collection, such as a gap in sensor readings. The model needs to inpaint the missing middle part of the trajectory based on the available data from both the beginning and the end.

The prediction error, as defined in Equation (34), is then calculated for the inpainted sections of the trajectory. This prediction error is divided by the number of inpainted steps to obtain

a uniform error measure per step. This provides a consistent metric to compare the model's performance across different inpainting scenarios.

5.3 Experiment - Multiple μ value

The first experiment examines the model's ability to generate a solution for the KS PDE across varying values of μ. Solutions for μ ranging from -0.2 to 0.2 with increments of 0.005 were generated as the training data. This dataset was passed to the model along with the non-discounted value of the parameter μ. Additionally, trajectories for μ outside the -0.2 to 0.2 range were generated for testing purposes. The complete set of test μ values is -0.25, -0.19, -0.12, -0.09, -0.04, -0.03, 0.02, 0.05, 0.08, 0.11, 0.14, 0.15, 0.2, 0.21, 0.24, 0.25; those values were excluded from the training data. For this case, the values of other parameters are spatial domain $L = [0, 22]$, domain discretisation $N_x = 64$, simulation time $T = 20s$, timestep size for integration $dt = 0.1$ resulting in 200 steps. For this experiment, for each μ value, 50 trajectories were generated, and the train and test split is $90 : 10\%$. Each trajectory per μ uses different random initial conditions to avoid duplicate data.

The model was evaluated using prediction error, calculated exclusively for the inpainted steps. The total prediction error for these steps was then divided by the number of inpainted steps to ensure a uniform assessment across all sweep cases. Figure 3 shows the prediction error for sweep front (time extrapolating), back, and both for μ inside the -0.2 to 0.2 range and μ outside the range. The sweep front is characterised by increasing error while the sweep back and both present error decline with an increasing number of the provided steps. This trend holds for the μ value inside and outside the training range: -0.2 to 0.2. This is because the model makes the most significant errors in prediction towards the horizon's end, regardless of the number of provided steps at the horizon's beginning. The Appendix A provides examples and explanations of this phenomenon. Based on the prediction error for all cases, the sweep back method performed the best, followed by the sweep both. Since the model is probabilistic, each generated solution may vary even with the same initial conditions and parameters. For this variability, 20 solutions per trajectory were generated for each μ value. The prediction error reported is the average across these trajectories, providing a more reliable assessment of the model's overall performance.

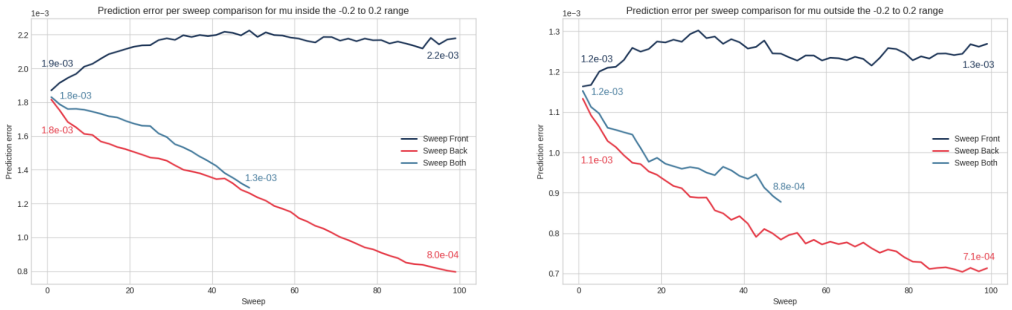

Figure 3: Prediction error for μ inside the range of -0.2 and 0.2 (left) and outside the range (right).

A notable factor is that for all the sweeps, the prediction error is smaller for the μ values outside the training range than for the μ values inside the range. This can be attributed to the behaviour of the PDE for larger values of μ. As μ moves further from zero, the cosine term in the equation begins to dominate the dynamics, leading to a less chaotic system. In this case, the influence of the other terms diminishes, and the solution exhibits a more regular pattern, thus making it easier for the model to predict the solution. Figure 4 and 5 show how the prediction error for each μ value in the test set for sweep front and back, respectively. For those cases, the best error was chosen from all the generated trajectories instead of the average error. In the Figures, the solid line represents the in-range values of μ, and the dash-dotted lines are the μ values outside the range. The complete results are in the tables in Appendix F together with the graph and description for sweep both, which is similar in results to sweep-back.

Figure 4 shows the time extrapolation (sweep-front) prediction error for various μ. Of 16 μ values, 10 had the lowest prediction error for only one provided step. This ties in with the general trend for sweep-front, where the smallest error happened for the smallest number of steps provided. However, for the other 6 μ values, the smallest prediction error for more than 50 provided steps. The μ value with the smallest prediction error is $\mu = -0.25$ with prediction error equal to $8.38 * 10^{-4}$, followed by $\mu = -0.19$ and $\mu = -0.12$ with error of $1.16 * 10^{-3}$. Notably, the μ values outside the range exhibit lower prediction errors than the inclusive μ values. Additionally, most errors either increase or remain constant as the number of provided steps increases. Again, this follows the general trend for the sweep front.

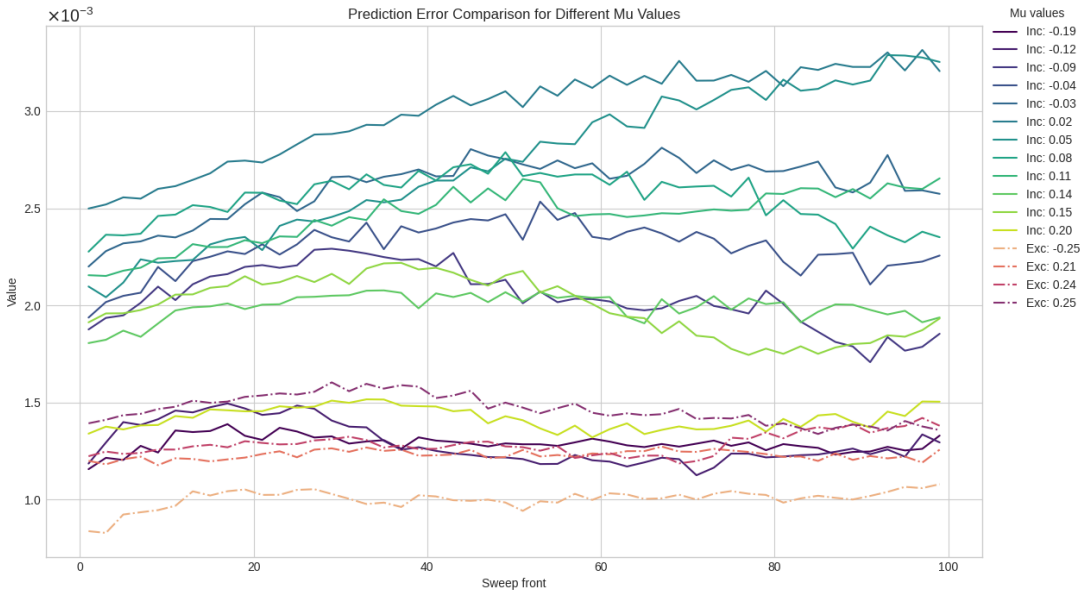

Figure 4: Prediction error for sweep-front for various μ.

Figure 5 shows the prediction error for sweep-back. Besides the lowest prediction error for $\mu = -0.25$ for 19 provided steps, for all the other values, the lowest error occurs toward

the higher number of provided steps, with the vast majority of error happening in the final provided steps. The smallest prediction error with the value of $5.83 * 10^{-4}$ is for $\mu = 0.14$ and the second smallest for $\mu = 0.21$ with value for $5.9 * 10^{-4}$.

Unlike the previous case, there is no clear distinction between the μ values inside and outside the range. While the prediction error for μ outside the range generally tends to be lower than for the μ inside the range when fewer steps are provided, this trend does not hold for more provided steps. Additionally, for most μ values, the prediction error decreases as the number of provided steps increases, aligning with the overall behaviour observed for the sweep-back scenario.

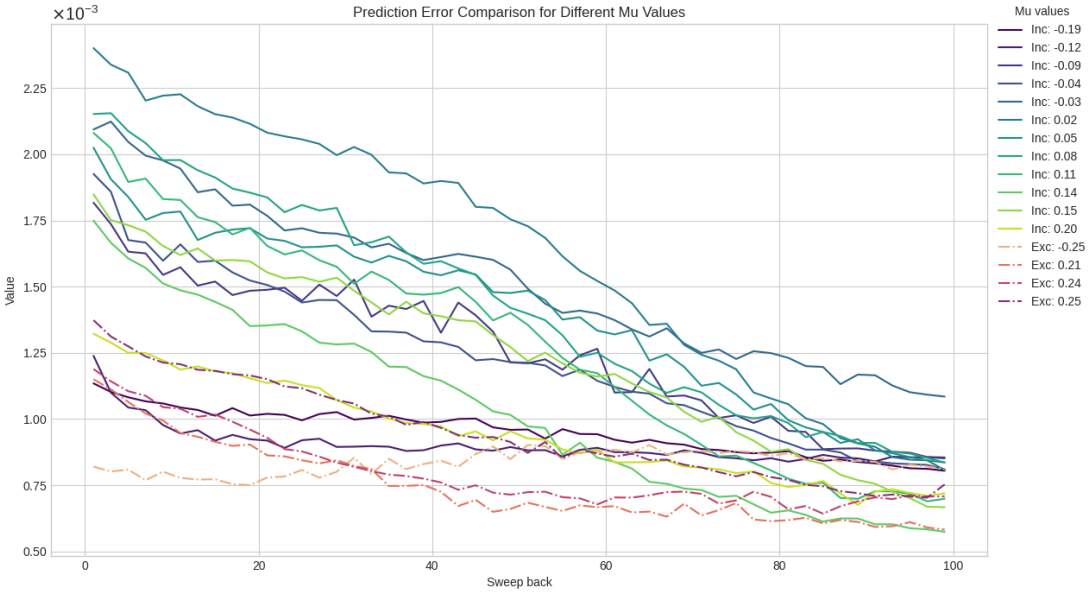

Figure 5: Prediction error for sweep-back for various μ.

Figure 6, 7 and 8 below showcase some of the solutions generated by the model for various sweeps and the number of steps provided. For each sweep, there is a solution for μ value inside and outside the range.

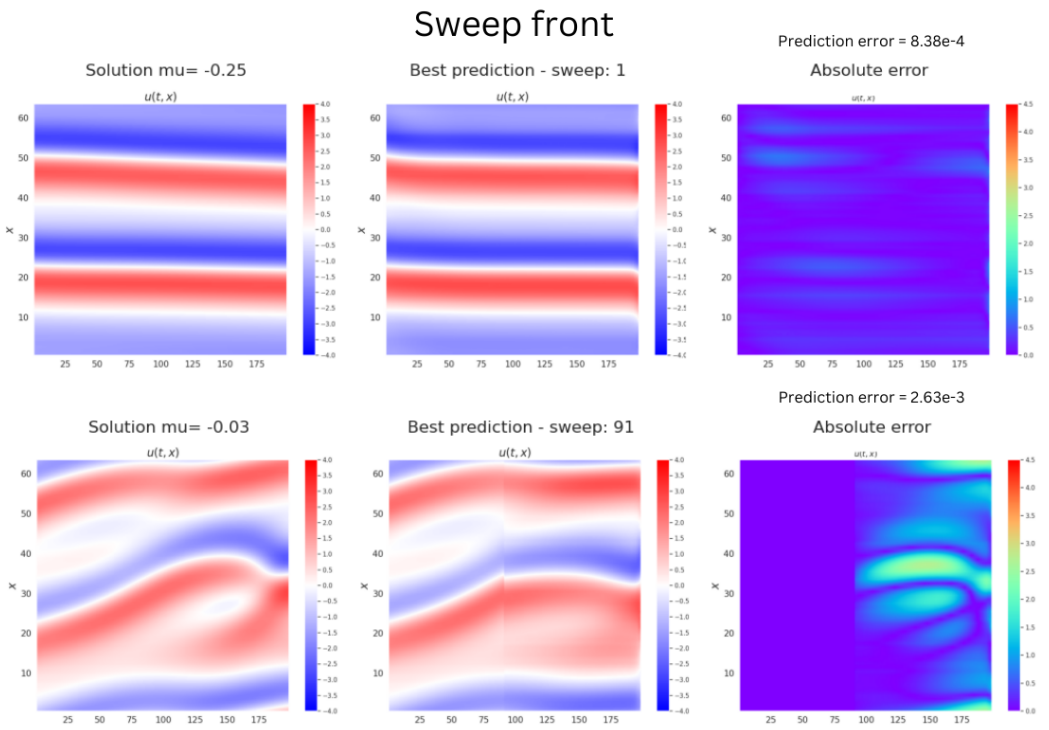

Figure 6: Sweep front generated solution visualisation.

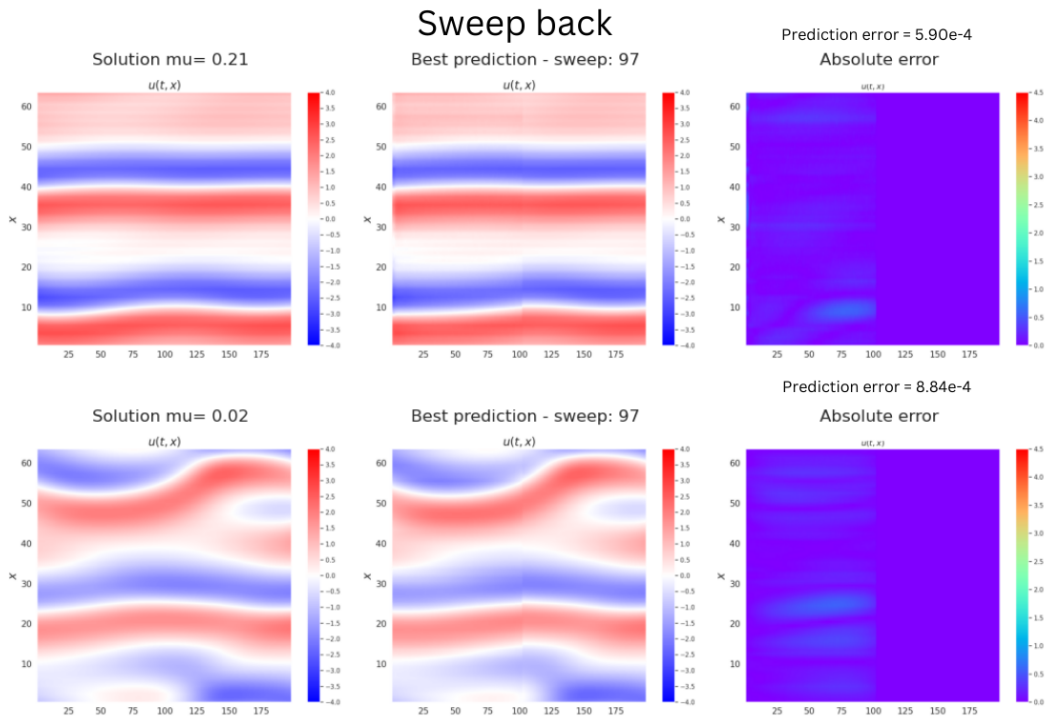

Figure 7: Sweep back generated solution visualisation.

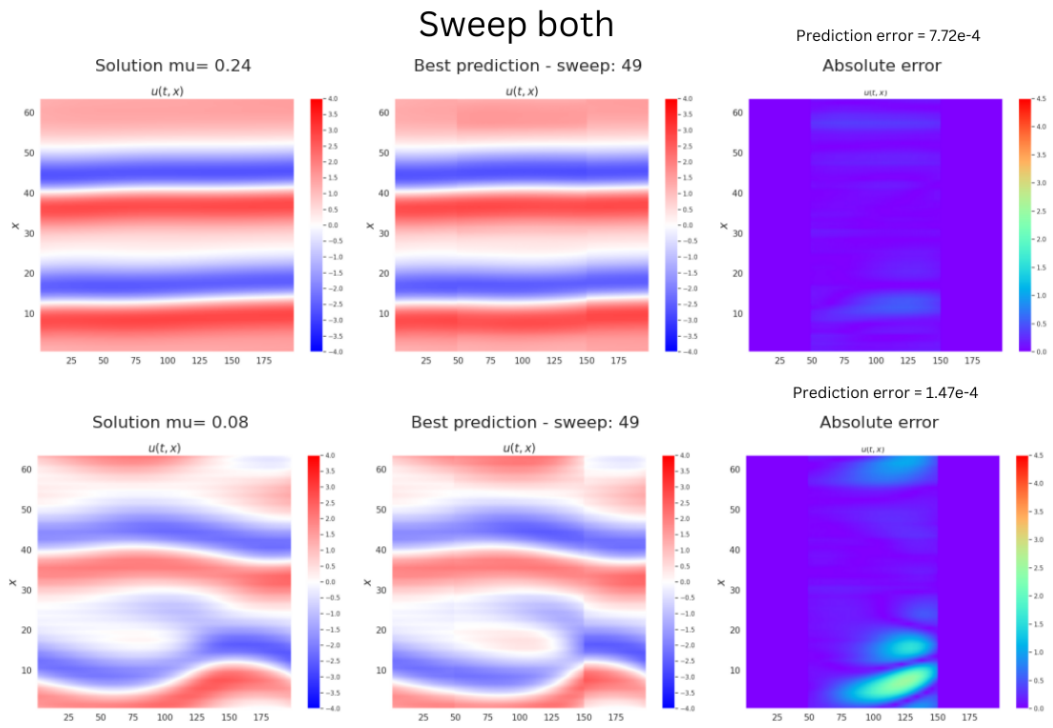

Figure 8: Sweep both generated solution visualisation.

In all the figures presenting the best-generated trajectories by the model, the sweep-front consistently performs the worst in absolute and prediction error compared to the other methods. Specifically, for the sweep-front with 91 provided steps for $\mu = -0.03$, as shown in Figure 6, there is a noticeable difference between the provided and inpainted steps. This disparity is further highlighted by the high absolute error values at the initial inpainted steps, indicating a significant mismatch between the generated and actual trajectories at the transition points.

In contrast, for the sweep-back and sweep-both approaches, it is still possible to discern the boundary between the inpainted and provided steps, although the difference is less pronounced than in the sweep-front. The sweep-back performs best among these methods, achieving the lowest absolute and prediction errors. This is evident in Figure 7 where the occurrence of light blue colour, indicative of higher errors, is noticeably reduced compared to Figure 6 and 8. This suggests that the sweep-back method enables the model to generate solutions that more closely approximate the actual trajectories, outperforming the sweep-both approach.

Despite these differences, all generated solutions broadly resemble the actual ones, preserving the general behaviour of the PDE. This indicates that while the choice of the sweep method significantly affects the accuracy of the generated trajectory, the model is generally capable of maintaining the essential dynamics of the PDE across different configurations. The results underscore the importance of method selection in optimising model performance, with the sweep-back method emerging as the most effective strategy for minimising errors and achieving solutions that closely align with the system's actual behaviour.

5.3.1 Comparison with FNO

The Fourier Neural Operator introduced in Section 3.2.4 is the state-of-the-art for generating solutions to PDEs and was used for comparison. The primary distinction between the FNO and DDPM models is that FNO predicts the solution at a single time step [36], while DDPM generates an entire trajectory. Two approaches to trajectory generation using FNO were examined: training a separate FNO for each time step and training a single FNO using consecutive trajectory steps as input data. The first approach yielded better results regarding prediction error and was selected as the benchmark. An analysis of both approaches is provided in Appendix E.

Figure 9 presents the prediction error per trajectory per step for FNO (red), DDPM (light blue), and DDPM with sweep back for 50 provided steps (dark blue). As shown in the figure, FNO generally yields a lower prediction error than the other two methods, reflected in the mean error values: 0.009 for FNO, 0.328 for DDPM, and 0.144 for DDPM with sweep back for 50 provided steps.

However, when focusing on the μ values at the extremes of the x-axis, specifically $\mu = -0.25, 0.2, 0.21, 0.24, 0.25$, which are outside the training range, the DDPM with sweep back outperforms FNO. For the negative values of μ outside the training range, the mean of the prediction errors for FNO is 0.146, compared to 0.118 for DDPM with sweep back. Similarly, FNO shows a mean error of 0.154 for the positive values, while DDPM with sweep back achieves a lower error of 0.113.This is also highlighted by the lower figure where the mean prediction error per μ is shown. It supports the conclusion that the DDPM with sweep back does better than FNO for the μ values located furthest from 0.

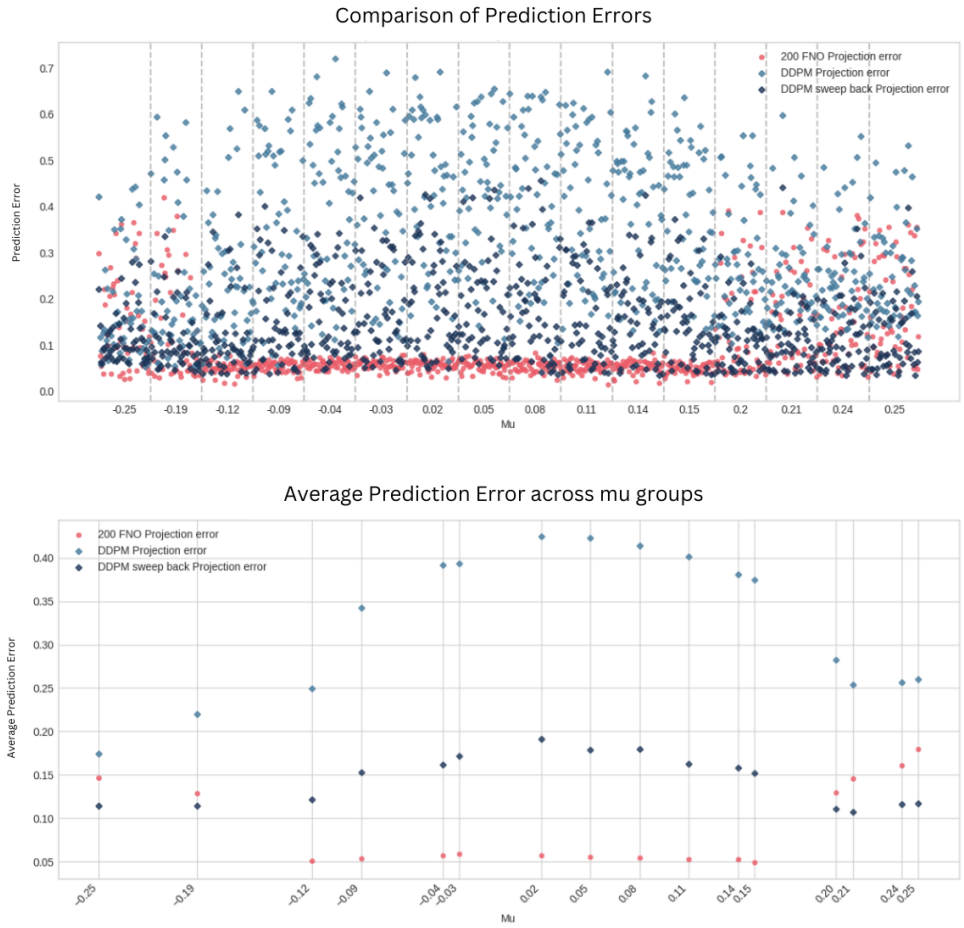

Figure 9: Prediction error per each trajectory in the train set (upper) and the average error per mu (lower) for FNO, DDPM and DDPM with sweep back.

Figure 10 illustrates an example of the trajectory generated by the FNO and DDPM. Neither method fully recreated the true trajectory; however, the results produced by DDPM show smoother transitions compared to those of the FNO in terms of temporal consistency. The smoother results from DDPM can be attributed to its ability to generate entire trajectories, whereas the FNO approach—trained using 200 separate models for each time step—lacks temporal continuity. Since FNO predicts each time step independently, no temporal information is carried over when training or predicting the solution at any given step, leading to a more disjointed result.

Additionally, the error percentage shown in the figure represents the portion of the trajectory where the absolute error exceeds a threshold of 0.4, which was selected as approximately 5% of the maximal error range. For the DDPM interpolation method, the error is calculated excluding the 50 provided steps, focusing only on the inpainted portions of the trajectory.

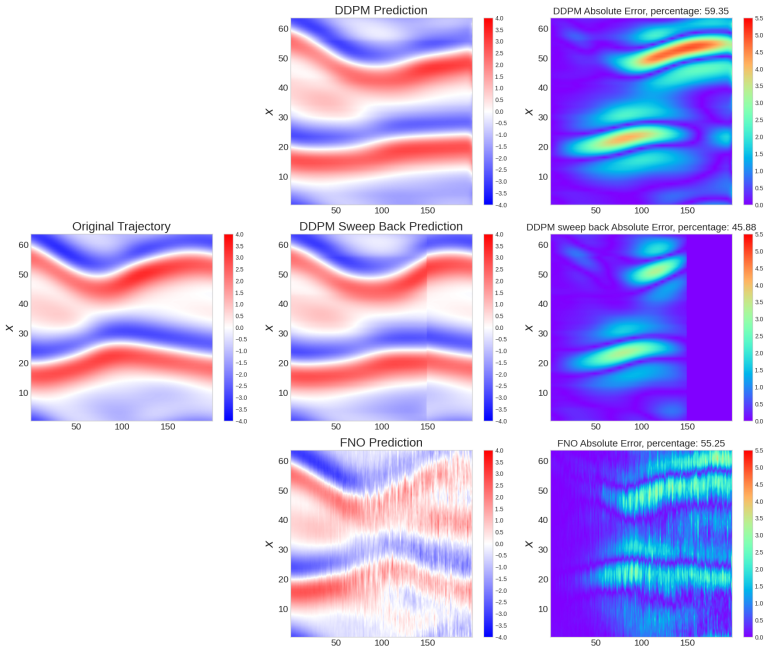

Figure 10: Example of solutions generated by DDPM, DDPM with sweep back and FNO, together with absolute error.

Figure 11 shows a more stable trajectory where the FNO and DDPM approaches perform well. Regarding the percentage error, the DDPM sweep back method still performs better. While the factors contributing to the temporal discontinuity in FNO are still present, they are less pronounced than the results shown in the previous figure. This indicates that, although FNO's lack of temporal information can affect performance, its impact is less severe in this case.

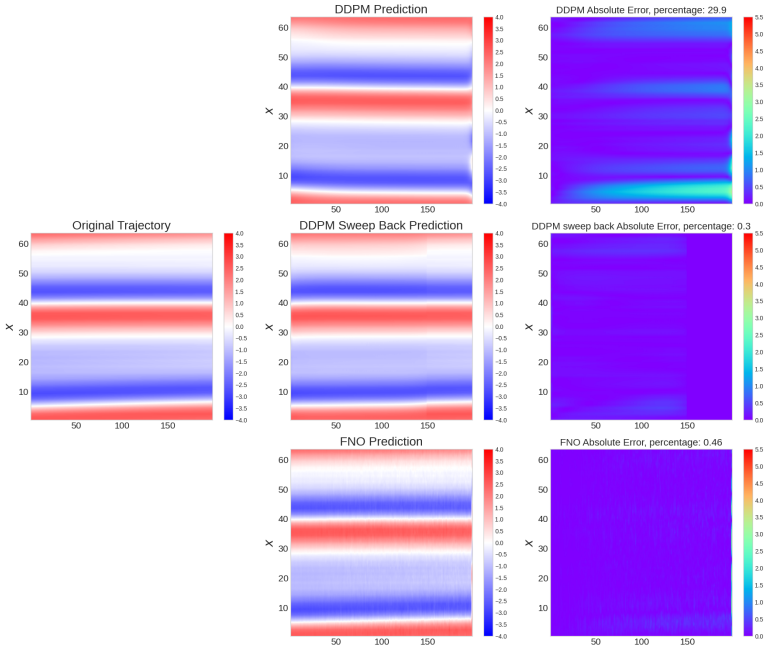

Figure 11: Example of solutions generated by DDPM, DDPM with sweep back and FNO, together with absolute error for stable trajectory.

Figure 12 compares the FNO, DDPM, and DDPM sweep back using two smoothness metrics: Finite Difference [8] and Lipschitz Continuity [57]. The Finite Difference measures the average change between consecutive points in the trajectory, as indicated in Equation (36):

$$\text{Finite Difference} = \frac{1}{T}\Sigma_{i=1}^{T-1} \left| s_t - s_{t-1} \right|. \tag{36}$$

In contrast, Lipschitz Continuity assesses the maximal rate at which the values in the trajectory change, as shown in Equation (37):

$$\text{Lipschitz Continuity} = \max_{1 \leq t \leq T} \left| s_t - s_{t-1} \right|. \tag{37}$$

Both variations of the DDPM outperform the FNO regarding the Lipschitz Continuity metric. While the difference in performance between the FNO and DDPM under the Finite Difference metric is less pronounced for μ values within the training range, it significantly widens for μ values outside this range. In both cases, the DDPM consistently performs better than the FNO.

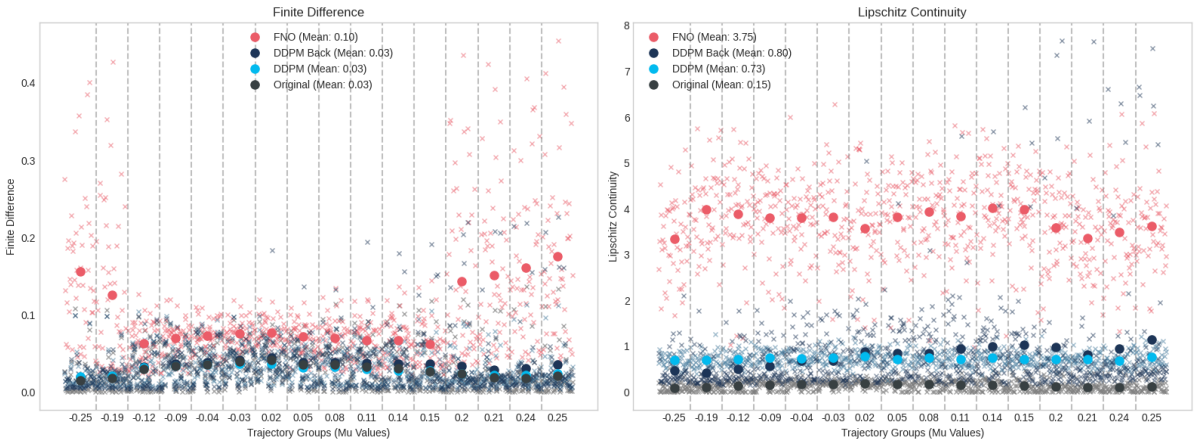

Figure 12: Comparison of FNO, DDPM and DDPM with sweep back using smoothness metrics: Finite Difference (left) and Lipschitz Continuity (right).

In conclusion, while the DDPM model generally performs worse than the FNO across all μ values in the test set, there are specific sweep configurations—such as the sweep back with 50 provided steps—where DDPM outperforms FNO, particularly for μ values outside the training range. Notably, as indicated by Figure 3, with an increase in the number of provided steps, the performance of DDPM sweep back improves, suggesting that the gap between DDPM and FNO could widen in favour of DDPM with more steps added.

Although FNO results in a smaller overall prediction error, the temporal discontinuity inherent in its design—due to predicting each step independently—affects the continuity of the generated solution. This temporal inconsistency is present throughout the solution, making FNO less smooth overall (see the absolute error in Figures 10 and 11. In contrast, DDPM, despite showing higher error on average, generates smoother solutions, with errors concentrated in specific parts of the trajectory rather than spread uniformly. This is evident from the percentage error, highlighting that DDPM errors are localised rather than pervasive.

Thus, while FNO effectively minimises prediction error, DDPM can produce smoother, more temporally consistent solutions—despite occasional localised errors - making it a compelling alternative for generating entire trajectories in PDE solutions. DDPM's performance could be further enhanced with additional provided steps, making it increasingly competitive with FNO.

5.3.2 Conclusion - Multiple μ

This experiment assessed the model's ability to generate solutions for the KS equation with previously unseen parameter values μ. The findings indicate that the model can effectively generalise for these unseen μ values, producing coherent solutions even when specific initial conditions were not part of the training dataset. This demonstrates the model's capacity for generalisation for parameterised PDEs.

Although DDPMs initially show higher per-step prediction errors than state-of-the-art method, their ability to generate smooth and temporally consistent entire trajectories is a significant strength. As the number of provided steps increases, the model's performance improves, particularly in sweep back and both tasks, confirming its potential for robust generalisation in solving PDEs.

5.4 Experiment - Latent representation

The original data was generated using the KS PDE, with the same properties as the multiple μ experiment (see Section 5.3). The latent representation of the data was reduced to 32, 16 and 8 observations per step using SVD truncation and AE. Figure 13 shows the reconstruction error the SVD and AE made for each latent dimension. While the AE's reconstruction error drops more rapidly for the first dimensions, it remains almost constant from latent dimension 8 to 64. In contrast, the SVD's reconstruction error decreases gradually at first but continues to improve steadily, resulting in lower projection errors than AE for latent dimensions 16, 32, and 64. The split between the test and train set was identical to the one for the previous experiment, described in Section 5.3.

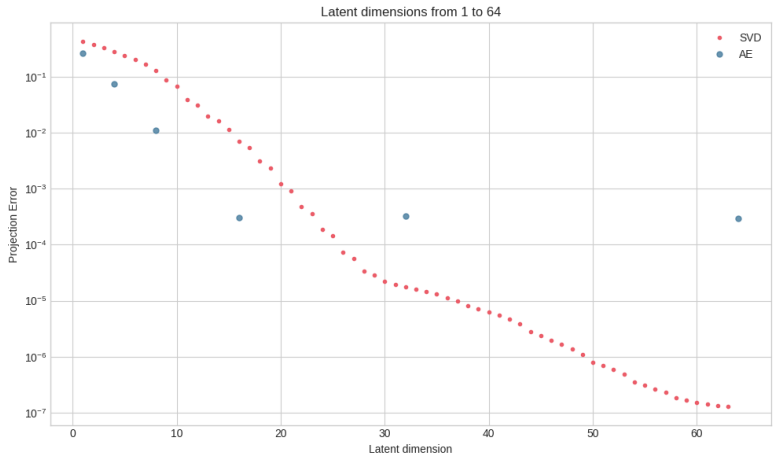

Figure 13: Projection error for each latent dimension after SVD and AE truncation and reconstruction.

Figure 14 shows how applying the SVD and then reconstructing the image affect the error - this is done without the DDPM. Naturally, the error increases as the latent dimension decreases; however, the reconstructed solution appears very close to the original one for dimensions 64 (original size), 32, and 16.

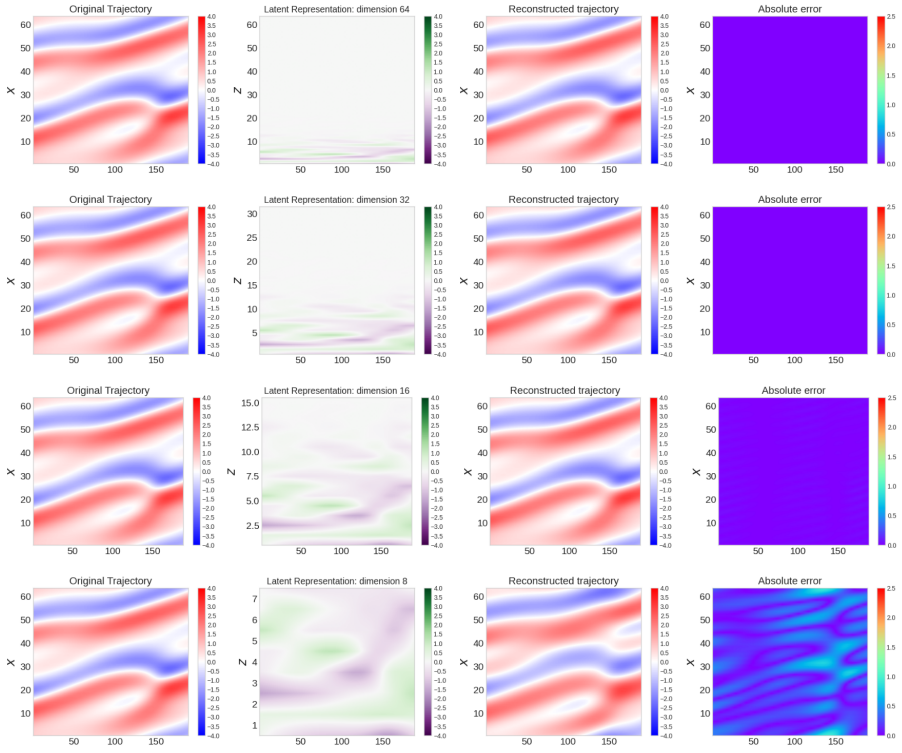

Figure 14: Latent representation and reconstruction for each of the examined latent dimensions (32, 16, 8) and dimension 64 using SVD.

Figure 15 shows the same comparison for the AE. For both SVD (Figure 13) and AE, a lower latent dimension leads to more prominent values in the latent representation of the observations. When comparing latent dimensions 8 and 32, for both cases, the plots for dimension 8

are filled with colour, indicating a dense representation. In contrast, for dimension 32, more white spots appear, signalling regions of very low values in the representation. This suggests that fewer features dominate the representation as the latent dimension increases. Following the graphs in Figure 13, the AE performs better in latent dimension 8 than the SVD, as barely any reconstruction error is visible.

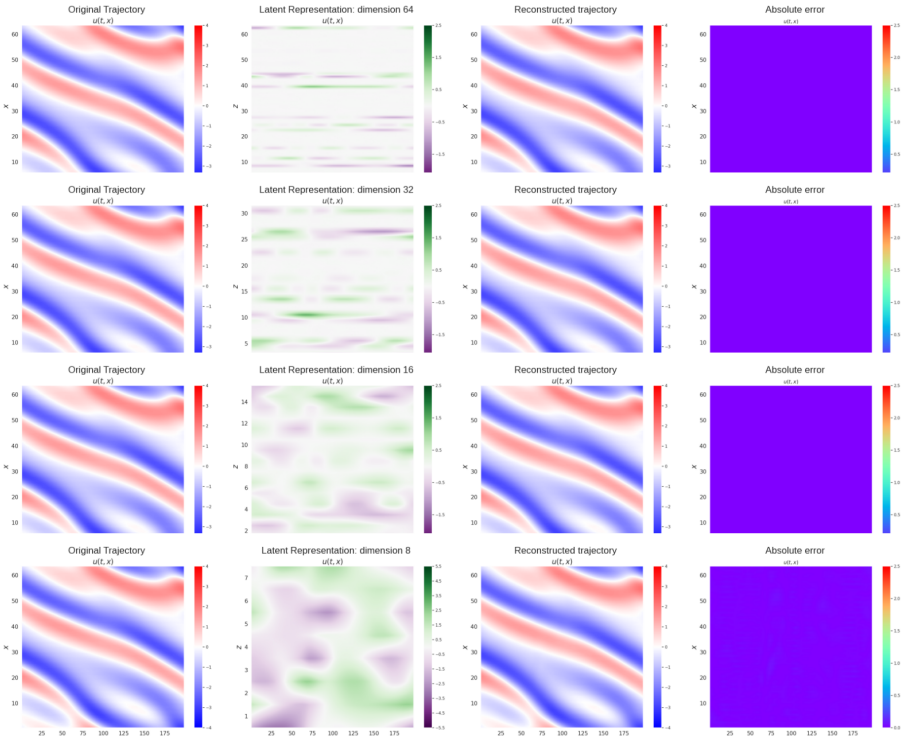

Figure 15: Latent representation and reconstruction for each of the latent dimensions of 8, 16, 32, and 64 using AE.

One variable that can be adjusted when training the diffusion model is the number of diffusion steps. Denoising steps, often called diffusion steps in the context of DDPMs, should not be confused with the computation of spatial derivatives for trajectories generated by the KS equation. Denoising steps refer to the iterative refinements made by the model as they progressively add and reduce noise in the generated solutions, ultimately impacting the overall quality and accuracy of the PDE solutions.

In theory, increasing the number of denoising steps in DDPMs should improve the quality of generated solutions by progressively refining them and reducing noise, leading to more accurate representations of the PDE solutions. More diffusion steps would allow the model to capture finer granularity in its predictions, resulting in smoother and more precise trajectories for the KS equation. However, in the case of a single μ, as discussed in Appendix A, this was not observed: a model with 200 diffusion steps yielded the best results compared to models with 500 and 1000 steps. This suggests that an optimal number of steps exists for capturing the KS solution dynamics without introducing over-refinement.

In the case of the latent representation, however, a higher number of diffusion steps consistently improves both prediction error (see Figure 16) and smoothness metrics (see Figure 17). Here, the SVD models benefited from increased diffusion steps, with each additional step helping the model better approximate the smooth and complex behaviour of the KS equation in its reduced form. Interestingly, for the AE, the two analysed models performed similarly, showing that additional diffusion steps may not significantly impact model performance beyond a certain threshold.

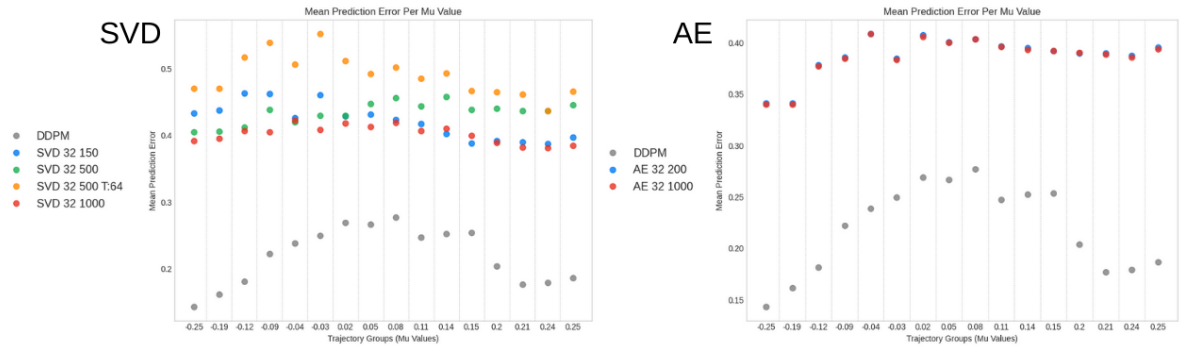

Figure 16: Average Prediction error per μ for different number of diffusion steps for SVD and AE of dimension 32.

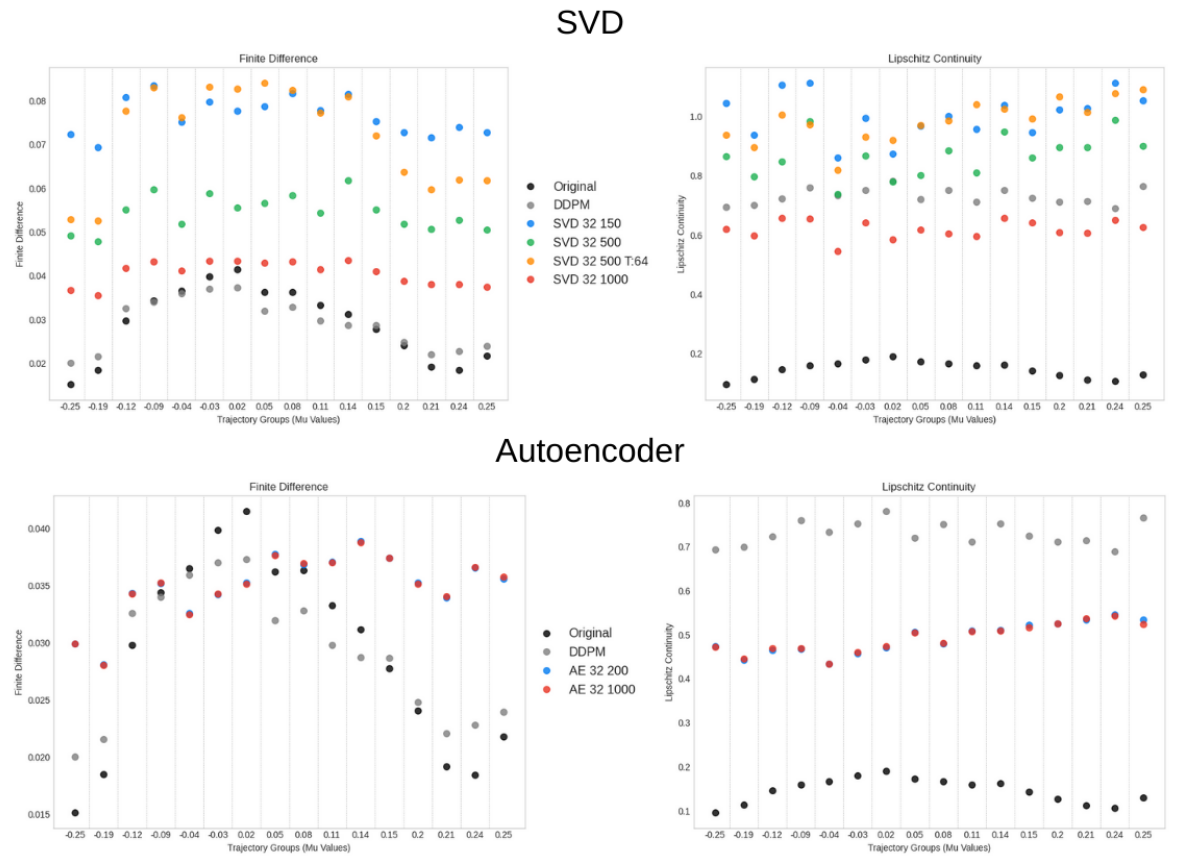

Figure 17: Finite Difference (left) and Lipschitz Continuity (right) per μ for different numbers of diffusion steps for SVD and AE of dimension 32.

The top-performing models for latent dimensions 8, 16 and 32 were selected to evaluate their performance across prediction error and smoothness metrics, as shown in Figure 18. Regarding prediction error, all SVD-based models, along with the AE 32 1000, exhibited similar performance, with errors closest to those observed in the DDPM without dimensionality reduction. However, the DDPM model without dimensionality reduction achieved a notably lower error, with at least a 0.1 advantage over the lowest error attained by any model using reduced-dimensional data. The prediction error is also smallest for the μ values outside the -0.2 to 0.2 range. However, the differences between the errors for μ inside and outside the range are not as pronounced as for the DDPM without dimensionality reduction.

The SVD 8 and AE 32 models excelled in Finite Differences and Lipschitz Continuity metrics regarding smoothness. Notably, all models trained in the latent space achieved higher consistency with the ground truth regarding Lipschitz Continuity than the DDPM without dimensionality reduction. This suggests that while dimensionality reduction introduces some compromise in prediction accuracy, it can still achieve smooth and continuous solutions that align well with the underlying structure of the KS dynamics, especially when using appropriately configured latent dimensions.

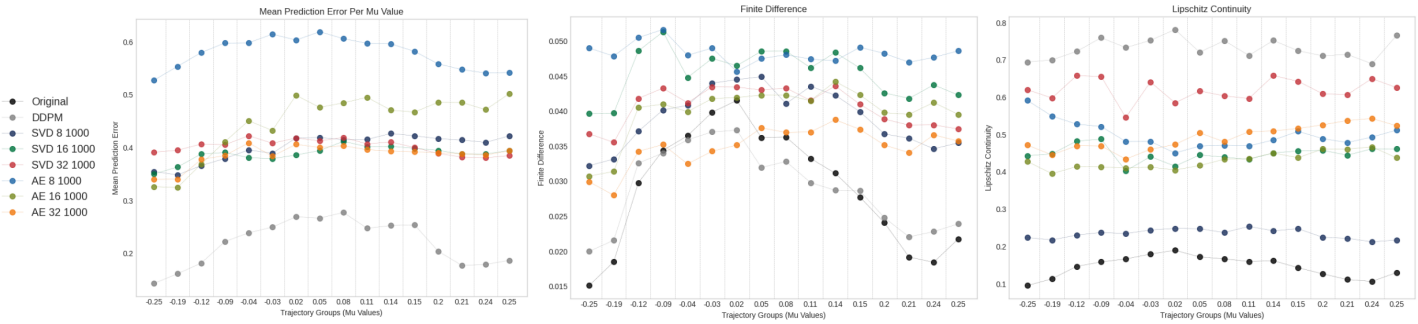

Figure 18: Prediction error and smoothness metrics for selected SVD and AE models (the lines interpolating the data points are introduced for visibility purposes. The data points are not connected).

Figure 19 shows the model's performance under different sweep strategies, dimensionality reduction approaches and number of diffusion steps for μ inside and outside the range from -0.2 and 0.2. The general trend is that the models using SVD as a dimensionality reduction method have a lower prediction error than models using AE. Additionally, in contrast to the case for multiple μ, the errors grow as the number of the provided steps grows, while in the multiple μ case for sweep both and back, the error was going down as seen in Figure 3. Additionally, the prediction error for the latent models was in the range from 10^{-3} to 10^{-2} for sweeping both, while for the normal model case, the error was in the range of 10^{-4} to 10^{-3}.

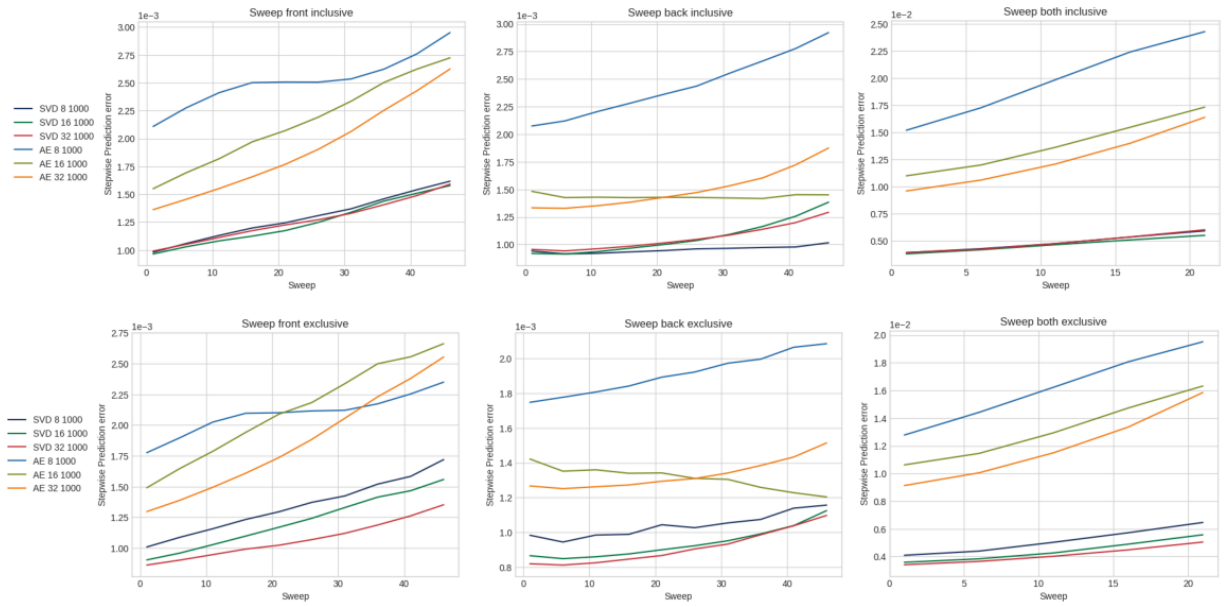

Figure 19: Comparison of prediction error for different sweeps by different models.

When reducing dimensionality using SVD, the reconstruction error associated with the reduction can be precisely quantified. As illustrated in Figure 13, a clear trend emerges: larger reduced dimensions correspond to lower reconstruction errors. Figure 20 illustrates the average stepwise prediction error across all values of μ for SVD with latent dimensions of 8, 16, and 32 and 1000 diffusion steps. The difference in error for SVD 16 — both with and without the projection error subtracted—is minimal, while for SVD 32, the difference is barely noticeable. However, even though the stepwise prediction error for SVD 8 1000 is the highest among the models, this error becomes the smallest after subtracting the SVD projection error, suggesting that the model's performance is better than initially indicated for the lower latent dimensions. The high initial prediction error may reflect the SVD 8 model's struggle to reconstruct the latent data adequately. Therefore, reducing the projection error could enhance the quality of the generated solutions when using the SVD 8 configuration. This suggests that strategies aimed at minimising projection error may significantly improve the performance of the diffusion model for lower dimensionality representations in this context. Additionally, these results suggest that the DDPM performs better with a latent representation where the entire latent space is populated with non-zero values, as seen in the SVD 8 case (see Figure 14). A more comprehensive discussion on the underlying reasons for this observation can be found in Section 6.2.

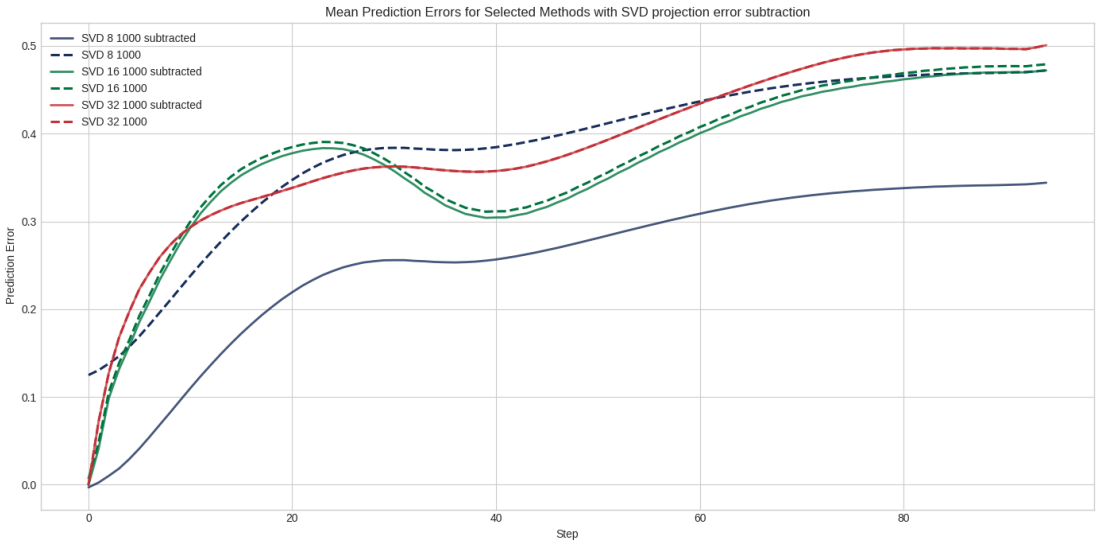

Figure 20: Average Prediction error per model with the SVD projection error subtracted.

Figure 21 presents example solutions generated by the models using SVD and AE for dimensionality reduction. In both cases, reducing the dimensionality to 8 yields the best results in terms of the overall fidelity of the solutions. However, it is notable that the latent representations significantly deviate from the true latent representation of the solutions. This discrepancy indicates that the error emerges primarily during the DDPM's attempt to generate the latent representation rather than in the subsequent translation from latent space back to the full-scale solution.

Interestingly, across all examples, the parts of the solution that align most closely with the true solution occur within the initial steps of the prediction horizon. This is best visible for dimensions 16 and 32. For this model, the training horizon was set to 32 timesteps, suggesting that the model performs well as long as the predictions remain within this timeframe. Beyond this horizon, the model struggles to pinpoint the solutions accurately. These observations highlight the importance of the training horizon in combination with the chosen dimensionality reduction technique and diffusion step size. These factors could yield the optimal combination for accurately generating solutions, ensuring that the model retains fidelity over short-term and long-term predictions.

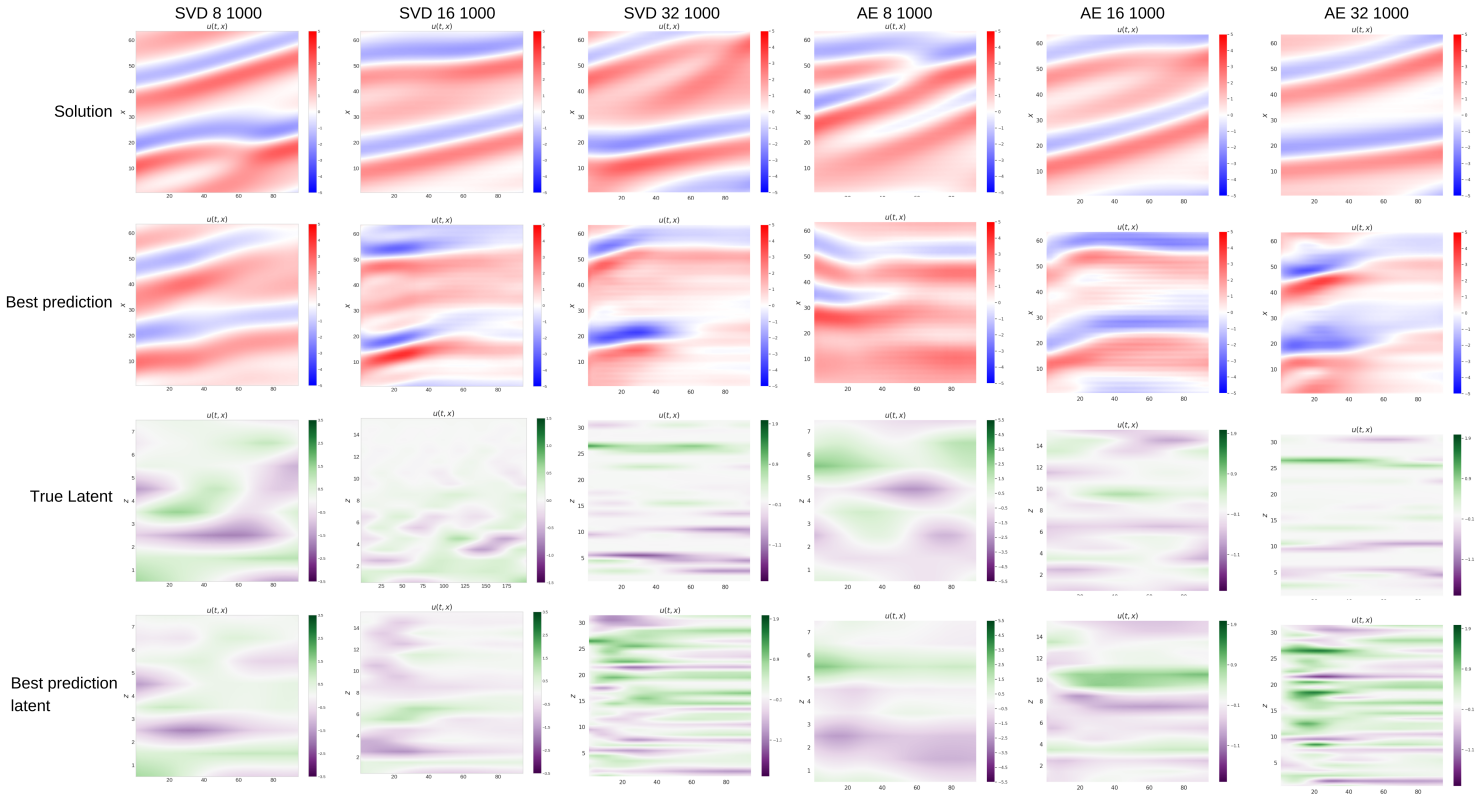

Figure 21: Example solutions for models using SVD and AE.

The varying number of diffusion steps directly affects the time required to generate solutions, as shown in Table 1, which compares the average time of generating one solution across all μ values in the test set using an NVIDIA GeForce RTX 2080 Ti GPU. The generation time is influenced by the method used for latent dimensionality reduction, whether AE or SVD, the size of the latent dimension, and the number of diffusion steps. The most evident conclusion is that more diffusion steps result in longer generation times. Models requiring 1000 diffusion steps took the longest overall. The dimension to which it is reduced also played a role since the generation of normal solution (64-dimensional observation) comes last from all the models needing 200 diffusion steps or less. Interestingly, models reduced to 16 and 8 dimensions took similar amounts of time, while models reduced to 32 dimensions generally took longer. Furthermore, there is no significant difference in generation time between models using SVD and AE for dimensionality reduction.

Table 1: Average time to generate one solution over a horizon of 96 steps for selected models (GPU: NVIDIA GeForce RTX 2080 Ti).

Model	Time [s]
SVD 32 50	1.84
SVD 32 100	3.56
SVD 32 150	4.88
SVD 16 200	4.97
AE 32 200	7.03
Normal 200	**10.84**
AE 16 1000	18.45
SVD 8 1000	18.64
AE 8 1000	18.75
AE 32 1000	33.75
SVD 32 1000	34.72

5.4.1 Conclusion - Latent representation

In conclusion, the DDPM can use its latent representation to generate solutions to the KS PDE. Still, the results indicate that the model's performance is somewhat limited in this setup. While it can generate reasonable approximations of the solutions, the multiple μ case demonstrates significantly better performance in capturing the dynamics of the PDE. Additionally, the analysis indicates that prediction error alone may not fully capture how well the generated solutions resemble the original ones and metrics like smoothness should also be considered when evaluating model performance, as low prediction error does not necessarily mean a good quality solution. Furthermore, the results show that finding the right balance between the horizon length, the number of input steps, and the number of diffusion steps is crucial to achieving high-quality solutions. While more diffusion steps often lead to better refinement, they also increase the generation time, highlighting the importance of optimising these parameters for accuracy and efficiency.

Moreover, while this section demonstrates the potential to achieve faster results than in the multiple μ case, the models here were tested with relatively small latent and original dimensions. It would be valuable to explore the model's performance with higher-dimensional representations to determine what time savings can still be realised and whether the model remains robust in handling more complex data. Overall, the findings suggest that further research is needed to optimise the generation process, particularly about the trade-off between computational cost and the quality of generated solutions.

5.5 Experiment - Control

The multiple μ experiment (see Section 5.3) demonstrated that the model is capable of generating solutions for the KS PDE. Building on these findings, the subsequent control experiment will focus specifically on the control properties of the model, examining how a controller is applied to the PDE equation. For this, a twin-delayed deep deterministic policy gradient (TD3) [23, 13] controller is employed. The controller was trained on both test and train instances of the parameter μ to ensure that the test data provided to the DDPM model was of the highest quality, consequently creating a benchmark for comparison. In addition, another TD3 controller was trained only on μ values from the train set, referred to as the "Test TD3". This controller will also be used to evaluate the DDPM's performance.

The data in this experiment consists of solutions to the PDE discretized to 64 dimensions, with varying values of μ. This dataset is split into training and testing sets, with the training range for μ defined between -0.2 and 0.2. The set of test μ values consists of μ = -0.25, -0.19, -0.12, -0.09, -0.04, -0.03, 0.02, 0.05, 0.08, 0.11, 0.14, 0.15, 0.2, 0.21, 0.24, 0.25; those values were excluded from the training data. The split between the test and train set is identical to the one described for the experiment for multiple values of the parameter μ; the only difference is that in the test set, the number of trajectories per μ has been reduced to 5. Each solution has a horizon of 32 time steps, where the first time step is the only instance at which the controller does not exert influence. Thus, the observations encompass 32 steps, and the controller begins to act from step 1 onward.

Figure 22 illustrates a sample of a controlled PDE with a horizon of 200 steps, where the controller is activated at step 50, providing a clearer view of the controller's effects on the solution. However, it is important to note that this experiment focuses on the first 31 steps influenced by the controller.

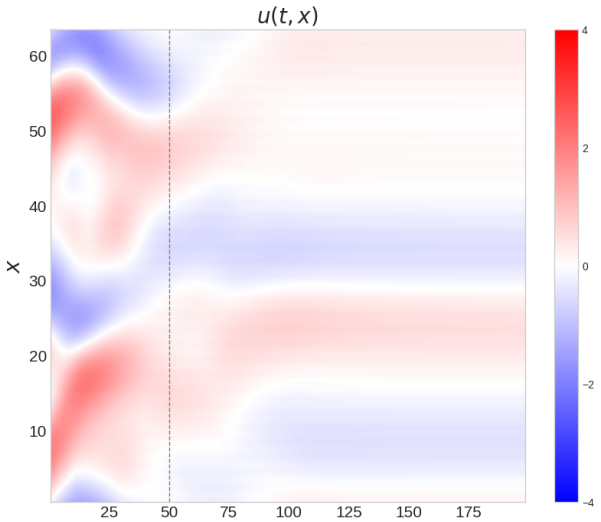

Figure 22: Example of a controlled trajectory using the TD3 controller

Three different scenarios were tested in this experiment to determine which approach yielded the best results. The first scenario involved solely using the DDPM model to inpaint the entire controlled trajectory segment without employing the inverse model. The second scenario implemented open-loop control, where the observations predicted by the DDPM were processed through the inverse model to determine actions, which were subsequently applied to the KS environment to get new observations. The third scenario involved closed-loop control, in which the model predicted over a short horizon, generating actions based on these predictions. These actions were then used to obtain the next observations, which was given back to the model for subsequent inpainting, and this process continued iteratively. Moreover, early stopping was implemented to mitigate overfitting, which stopped the training of the DDPM if the validation error did not decrease for five consecutive epochs.

Cost functions associated with actions, observations, and total rewards are used to further evaluate the model's control properties. Actions and observation costs quantify the penalty associated with actions or observation at a given system state. They are both mean square errors, which means that the larger the action or observation, the bigger the penalty. Consequently, smaller, more controlled actions and observations are favoured. The action cost is expressed as shown in Equation (38):

$$\text{Action cost} = \frac{1}{T}\Sigma_{t=0}^{T-1}(\frac{1}{N_a}\Sigma_{i=0}^{N_a}(a_{t,i})^2). \tag{38}$$

The observation cost defined in Equation (39):

$$\text{Observation cost} = \frac{1}{T}\Sigma_{t=0}^{T-1}(\frac{1}{N_s}\Sigma_{i=0}^{N_s}(s_{t,i})^2)\,, \tag{39}$$

where N_a and N_s are the numbers of actions and observations, respectively. The costs are averaged over T, yielding the average cost per trajectory, and the smaller the cost, the better. Equation (40) shows the total reward, which is the weighted sum of the action and observation costs:

$$\text{Total reward} = -(\text{Action cost} * 0.1 + \text{Observation cost}) \tag{40}$$

Figure 23 displays the mean actions and observation cost per μ across the test set, and Figure 24 the total reward. For all metrics, the closed-loop early-stopped model reached the highest values. However, that configuration also reached the lowest values for the observation cost and total reward, particularly at the ends of the μ range. Despite these localized variations, the closed-loop early-stopped model achieved the highest overall mean for the total reward. Notably, all the closed-loop models achieved a higher mean in the total reward than the Test TD3. The fully-trained open-loop model also reached identical total reward distribution across μ values as the TD3 model. While the closed-loop case benefited from early-stoping, the open-loop performed better when the model was fully trained. Only for action cost did both early-stopped models outperform their fully trained counterparts, suggesting that early-stopping can be a reliable way of generating trajectories where the primary focus is on actions.

An opposite trend was observed compared to the experiment with multiple μ values (see Section 5.3), where performance improved as the μ value moved further from 0 due to higher

μ values contributing to more stable solutions. In this control-focused experiment, however, that stability factor no longer seemed to play a significant role. The highest total rewards were instead obtained for positive μ values close to 0, suggesting that the model's control performance depends on factors beyond just the inherent stability provided by higher μ values.

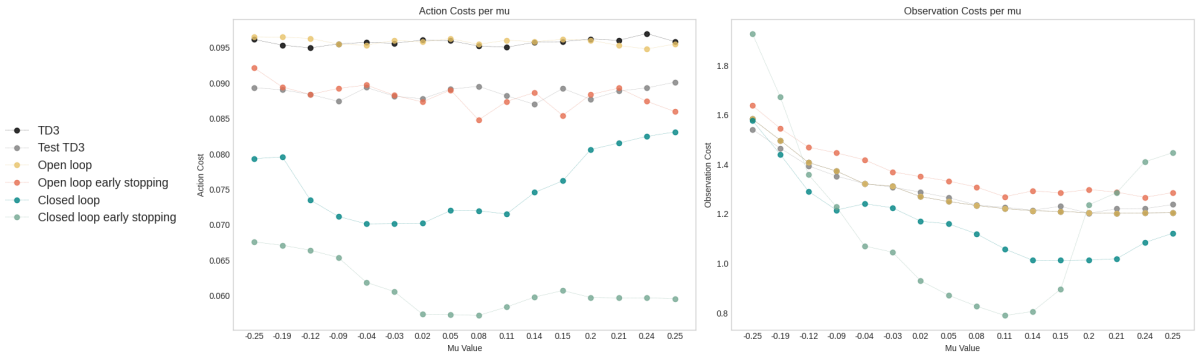

Figure 23: Actions and observation costs for various control strategies per μ (the points are not connected, lines added for visibility).

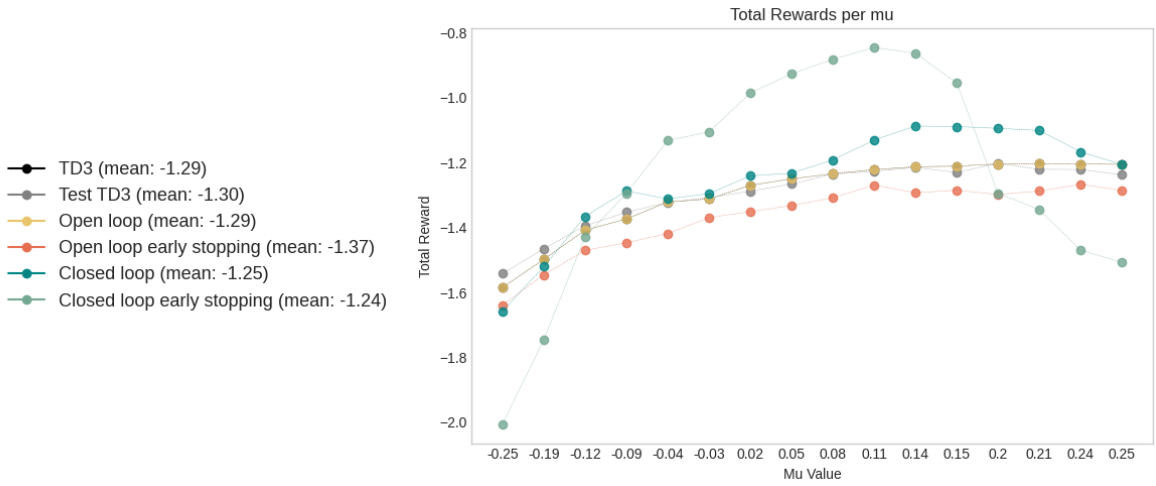

Figure 24: Total reward for various control strategies per μ (the points are not connected, lines added for visibility).

Figure 25 evaluates the performance under smoothness metrics. Both closed-loop and open-loop control methods produced the most temporally consistent results, as measured by the Finite Difference method, thus outperforming TD3. This suggests that the TD3 controller can change the PDE more drastically than the DDPM. The TD3 controller's ability to exert a more significant influence on the PDE could lead to less smooth transitions, as it may prioritize achieving immediate control objectives over maintaining continuity in the solution. Additionally, the early-stopped DDPM model demonstrated notably lower values regarding Lipschitz continuity than a fully trained one.

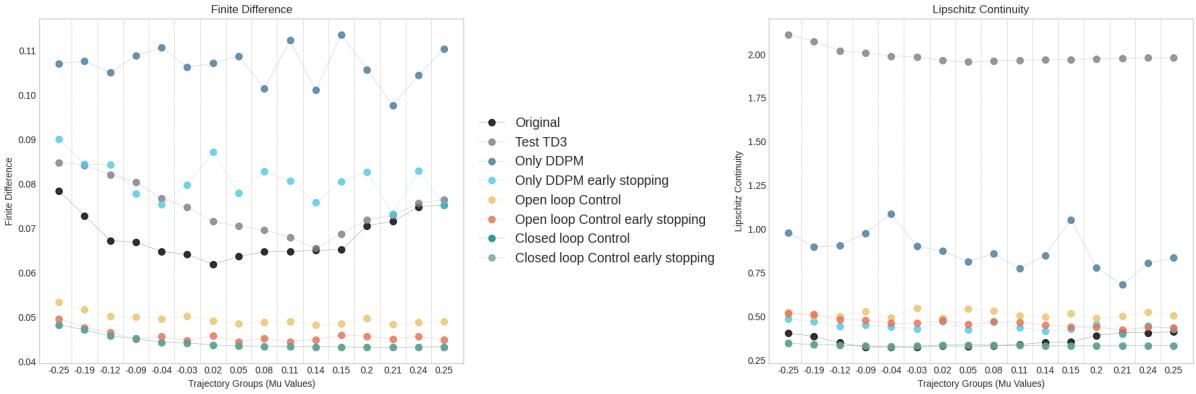

Figure 25: Prediction error for various Control strategies (the points are not connected, lines added for visibility).

Figures 26, 27 and 28 show example trajectories generated by all the models discussed in this Section, with the total reward for the trajectory. For the DDPM-only models, it was impossible to obtain the total reward, as the DDPM model does not generate actions by itself. For this specific trajectory, the closed-loop models achieved the highest total reward, followed by the open-loop model, with the early stopped one performing better than the fully trained model.

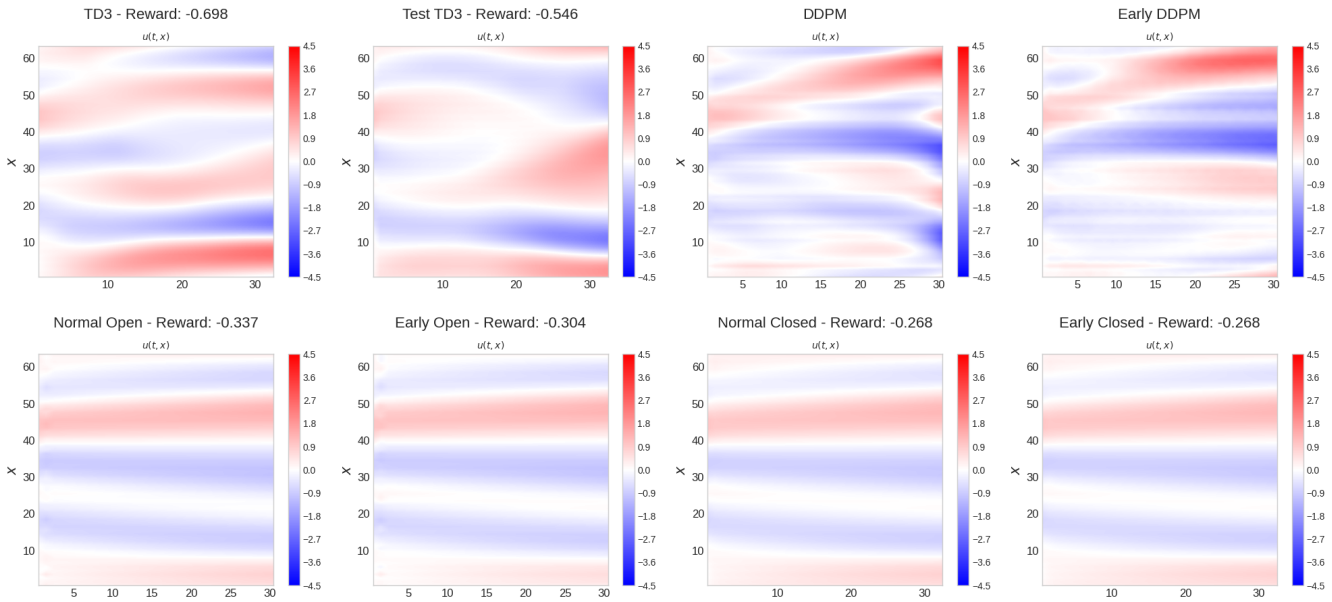

Figure 26: Predicted solution for $\mu = 0.15$ by various models with total reward specified (where possible).

Figure 27 presents an example of a more straightforward trajectory that resulted in a high total reward across all models. Notably, all models achieved closely aligned total reward values, in contrast to the more varied performance seen in Figure 26. It suggests that the complexity of the control problem influences the differences in performance; for more straightforward control scenarios, the effectiveness of the models is comparable.

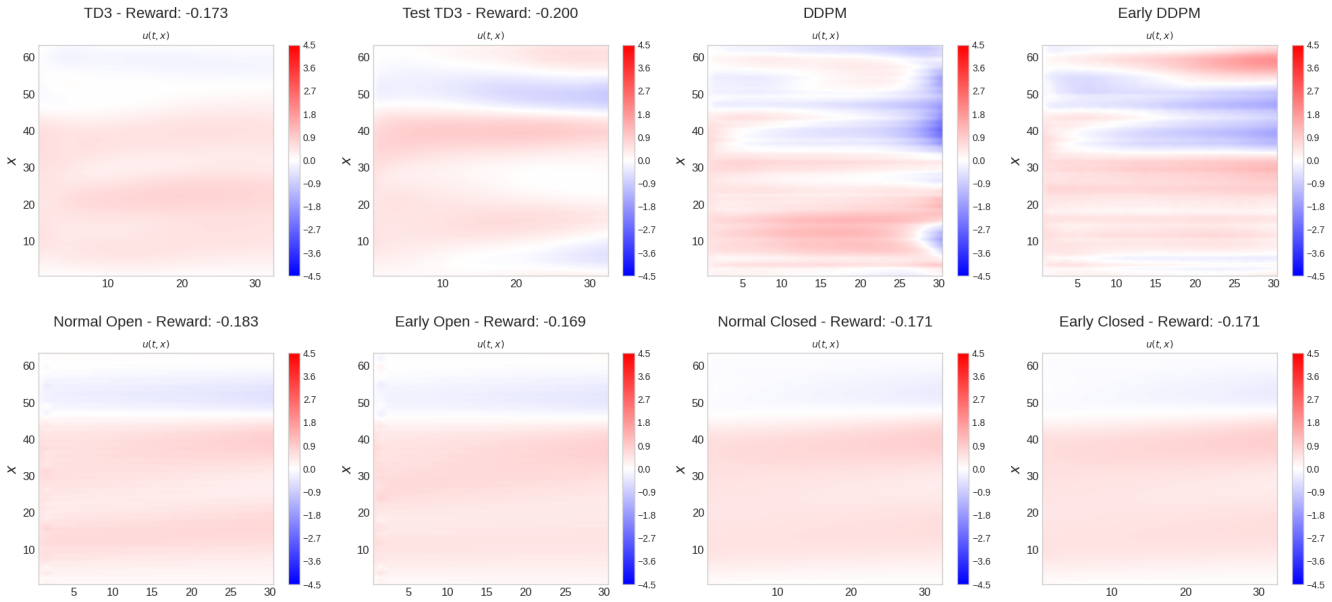

Figure 27: Predicted solution for $\mu = 0.05$ by various models with total reward specified (where possible).

Lastly, Figure 28 shows a trajectory for a μ outside the -0.2 to 0.2 range. For this case, the TD3 controller performed the best, followed by Test TD3. The third best-performing model was the open-loop fully trained one, which was among the worst-performing models for the previous two cases. Figures 26, 27 and 28 show that while the closed-loop early model achieved the highest total reward, the performance of each of the evaluated models varies on the KS solution they try to control.

Additionally, the DDPM-only models produced solutions typically closer to those generated by the TD3 solutions than the open and closed-loop models. This is expected, as the DDPM model was trained on data derived from the TD3 controller. Additionally, the DDPM-only solutions performed the worst in terms of smoothness, as also evidenced in the figures above. This indicates that when controlling PDEs, employing a method that interacts with the PDE's environment during prediction is more advantageous, as demonstrated in the open and closed-loop cases. This interaction allows for more adaptable and refined control, leading to smoother and more stable outcomes.

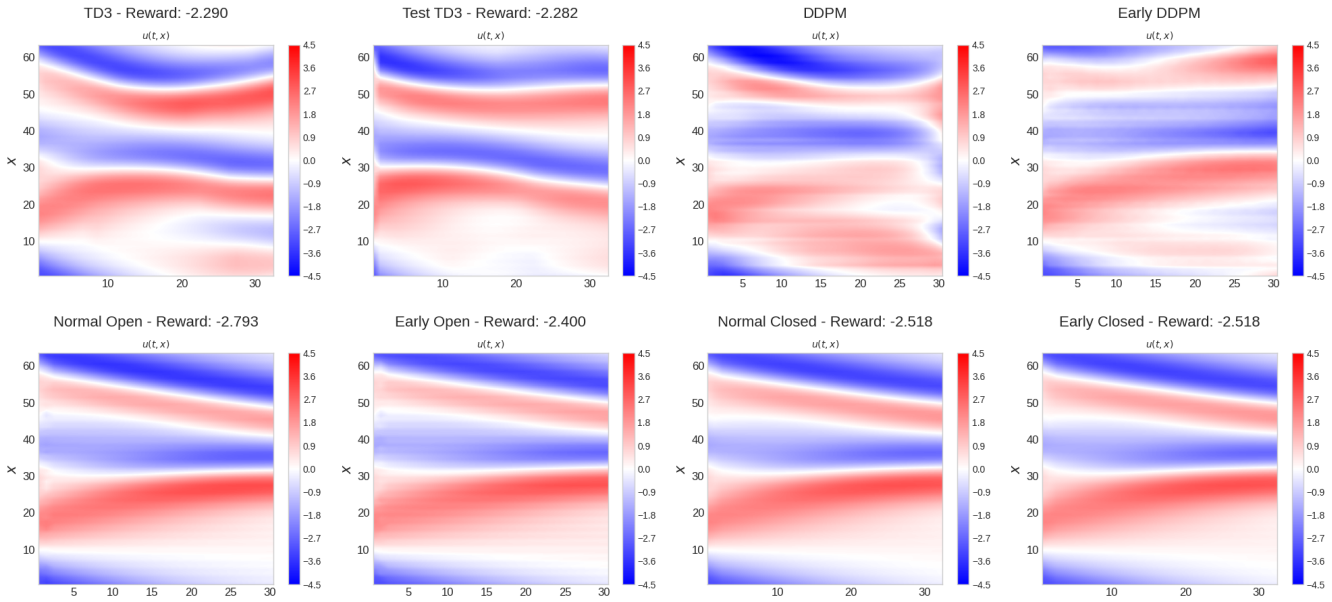

Figure 28: Predicted solution $\mu = -0.25$ by various models with total reward specified (where possible).

5.5.1 Conclusion - Control

This experiment assessed the model's ability to generate controlled solutions for the KS PDE. The results show that the closed-loop control approach consistently outperformed both the open-loop and DDPM-only methods, showcasing its ability to maintain predictive accuracy. The closed-loop approach achieved the highest total reward when paired with early stopping. Furthermore, based on Finite Difference measures, both open and closed-loop strategies demonstrated better temporal consistency than DDPM-only approaches and the TD3 controller. Early stopping effectively prevented overfitting and improved action costs, though its effectiveness and that of the other models varied depending on the specific solution being controlled.

In conclusion, the model can generate controlled solutions for parametric PDEs, especially with the closed-loop approach. While early stopping can enhance performance, the choice between early stopping and a fully trained model should be based on the specific control scenario.

6 Analysis and Discussion

This section provides deeper insights into the findings, examining the underlying factors contributing to the observed results. It highlights the strengths and limitations, offering a broader perspective on the model's performance in PDE generation and control.

6.1 Experiment - Multiple μ values

In this section, the general behaviour of the model is analysed. Figure 29 presents the model's average solution without any conditions or parameters. The average solution oscillates more towards the PDE's negative values. The training data in the lower row of Figure 29 illustrates that these values were balanced, with the average solution oscillating slightly above zero.

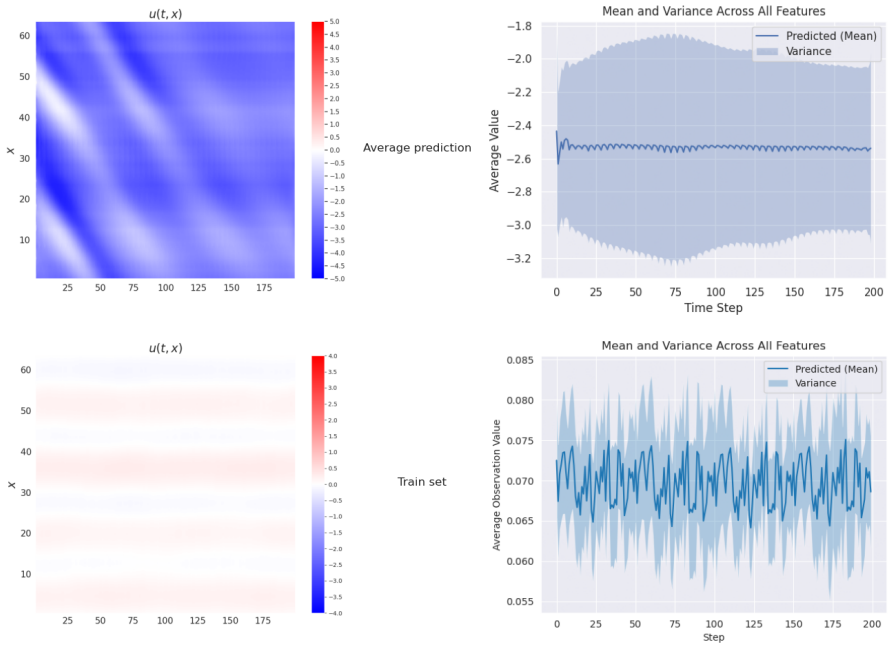

Figure 29: Comparison of a general predicted solution (top row) with a general solution in the train set (bottom row) for varying μ.

In the next experiment, with the introduction of the value of the parameter μ, the model's interpretation of the parameter embeddings is analysed before and after training. Figure 30 shows how the parameter μ influences the PDE solution - all the results in this Figure were obtained for the same starting conditions. To assess this, Principal Component Analysis (PCA), SVD, and t-distributed Stochastic Neighbor Embedding (t-SNE) were utilised, as shown in Figures 31 and 32. For the PCA and SVD, Figure 31 before training, the embeddings displayed some degree of linear separation. After training, this separation became even more pronounced, especially for the SVD, indicating that the model effectively learned to distinguish between different μ values.

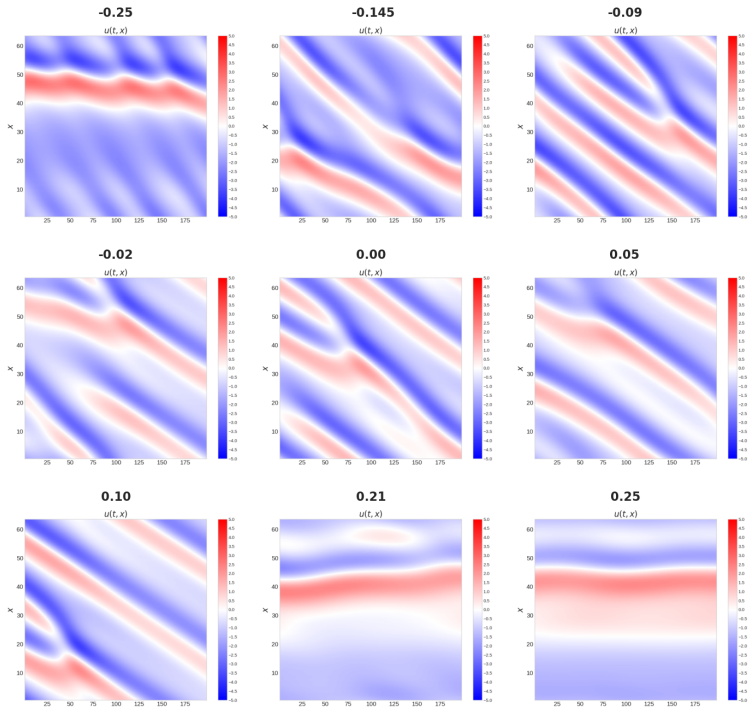

Figure 30: Actual solutions for various μ and the same starting conditions.

Multiple embeddings are generated for the same parameter value to explore their consistency for each parameter. In the PCA and SVD analyses, all embeddings aligned closely with the average parameter value, marked with a black "x". However, in the t-SNE analysis, this was not the case; embeddings showed greater variability, with some embeddings deviating significantly from the average cluster.

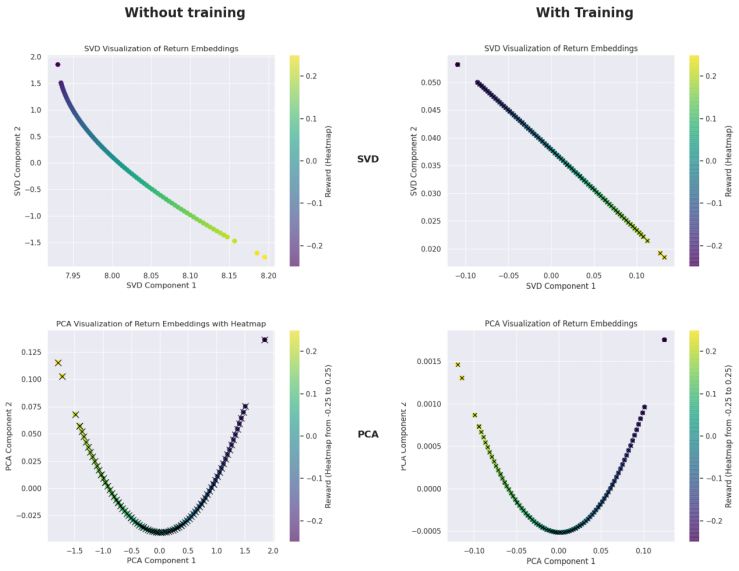

Figure 31: Parameters embedding reduced to two dimensions using SVD and PCA before (left) and after training (right).

Additionally, t-SNE visualisation, Figure 32, reveals that, after training, the number of elements in clusters corresponding to μ values that differ from the average value (marked with a black "x") is reduced. This suggests that the model has improved in clustering similar μ values more tightly, enhancing its ability to generalise across different parameter conditions. The results indicate that the model can distinguish between different parameter values.

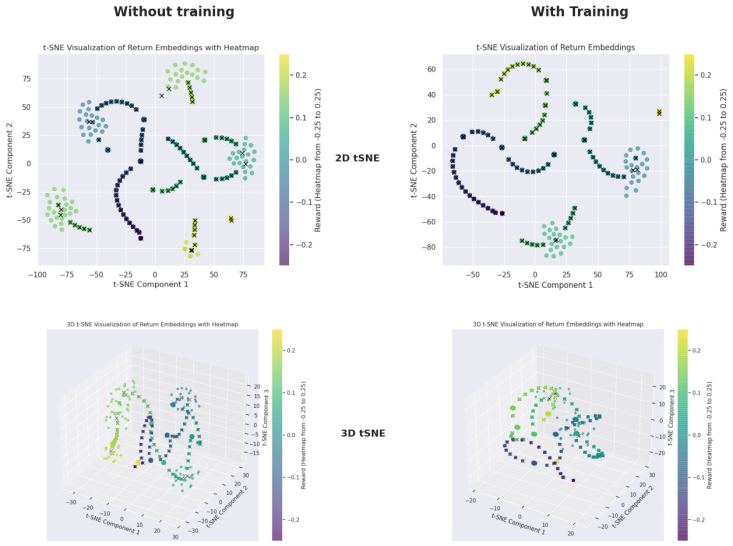

Figure 32: Parameters embedding reduced to two and three dimensions using t-SNE before (left) and after training (right).

The subsequent analysis investigates how an average solution appears when the model is provided only with the parameter value without the initial starting condition. The left side of Figure 33 demonstrates that regardless of the μ value passed as a parameter, the model's average solution consistently oscillates within the negative values that the PDE can assume. Furthermore, there are no distinct differences between the various μ values. Although the generated solutions differ from one another, the influence of the parameter value (μ) is not as pronounced as observed on the right-hand side in Figure 33, which shows the generated solution by the model when also the first step was provided. This suggests that while the model can conceptually distinguish parameter values, the parameter alone cannot significantly influence the average predicted solution.

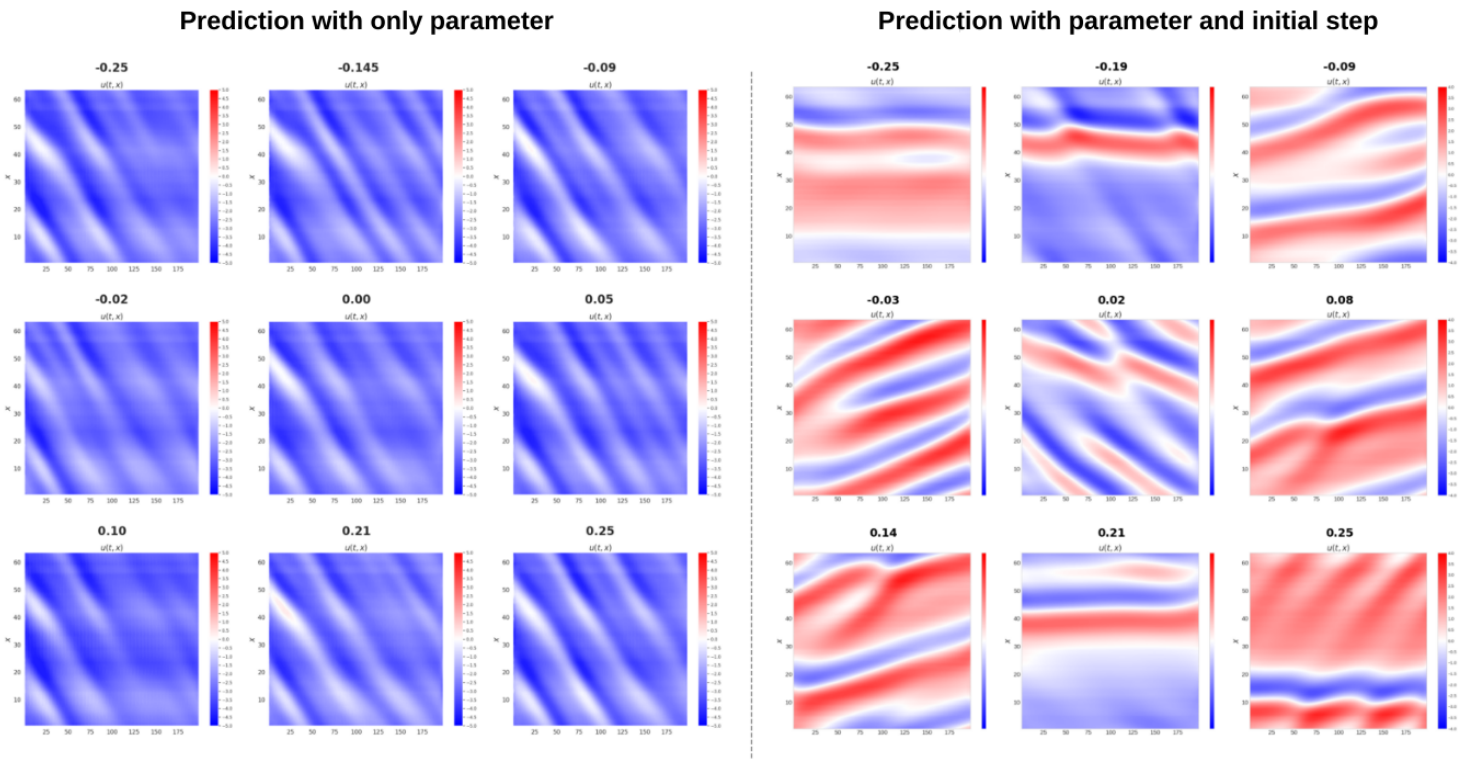

Figure 33: Comparison of solutions generated by the model by only using μ conditioning (left) or only the initial conditions (right).

The model's performance is subsequently evaluated under two conditions: first, by generating solutions using only the starting conditions, and second, by generating solutions using both the starting conditions and conditioning the model on the parameter value μ. Figure 34 compares the best solutions from both scenarios without and with the parameter. The average prediction error between the two cases does not differ significantly; however, a consistent trend is observed where the error with the parameter shows less variance between subsequent solutions, best visible in the case of $\mu = -0.12$. Additionally, the differences between the best solutions from both cases are minimal. The prediction error tends to increase towards the final steps of the horizon, indicating that the accuracy of the best solution deteriorates more noticeably near the end of the prediction window.

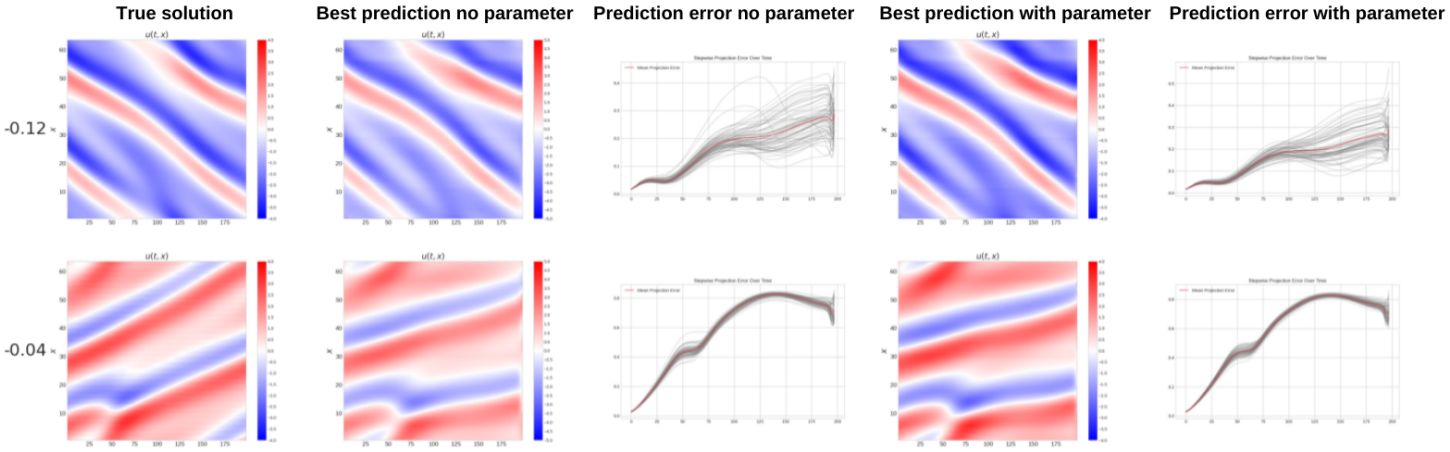

Figure 34: Best solutions of the model without and with conditioning on the parameter and prediction error with the average error marked.

These observations suggest that while including the μ value as a parameter does not significantly improve the accuracy of the model's predictions, it contributes to stabilising the predicted solutions, resulting in less variability between subsequent outputs. This stabilisation could be beneficial in applications where consistency of predictions is critical, even if it does not directly enhance the accuracy compared to the true solution. Therefore, incorporating μ as a parameter can be seen as a strategy to achieve smoother and more consistent outputs from the model.

Figure 35 presents the experiment results, where the true solution is compared with the predicted average solution, the mean of all predicted solutions, norm, and PCA. The general solution predicted by the model oscillates within the negative values, as shown in Figure 29, resulting in an observed mean that is consistently lower than the actual mean. Additionally, the model struggles to capture the linear trend of the actual mean. However, for negative μ values, the predicted mean aligns more closely with the actual mean than for positive μ values. As μ approaches the negative range, the predicted mean shows a declining trend that mirrors the actual mean. In contrast, for higher positive μ values, the predicted mean oscillates around a fixed point.

In terms of PCA, the model shows better alignment with the actual solution when the solution contains more negative values, indicated by the blue colour. For instance, in the cases of $\mu = -0.25$ and $\mu = -0.12$, where negative values are predominant, the predicted PCA trajectory closely matches the actual PCA. This behaviour is less evident for $\mu = -0.04$ and $\mu = 0.21$, where the model's alignment with the actual trajectory is weaker. Despite these challenges, comparing the norm of the observations vector indicates that the model can approximate the system's true magnitude and behaviour accurately, even when PCA alignment is not consistent, suggesting that the model can capture the overall dynamics of the PDE.

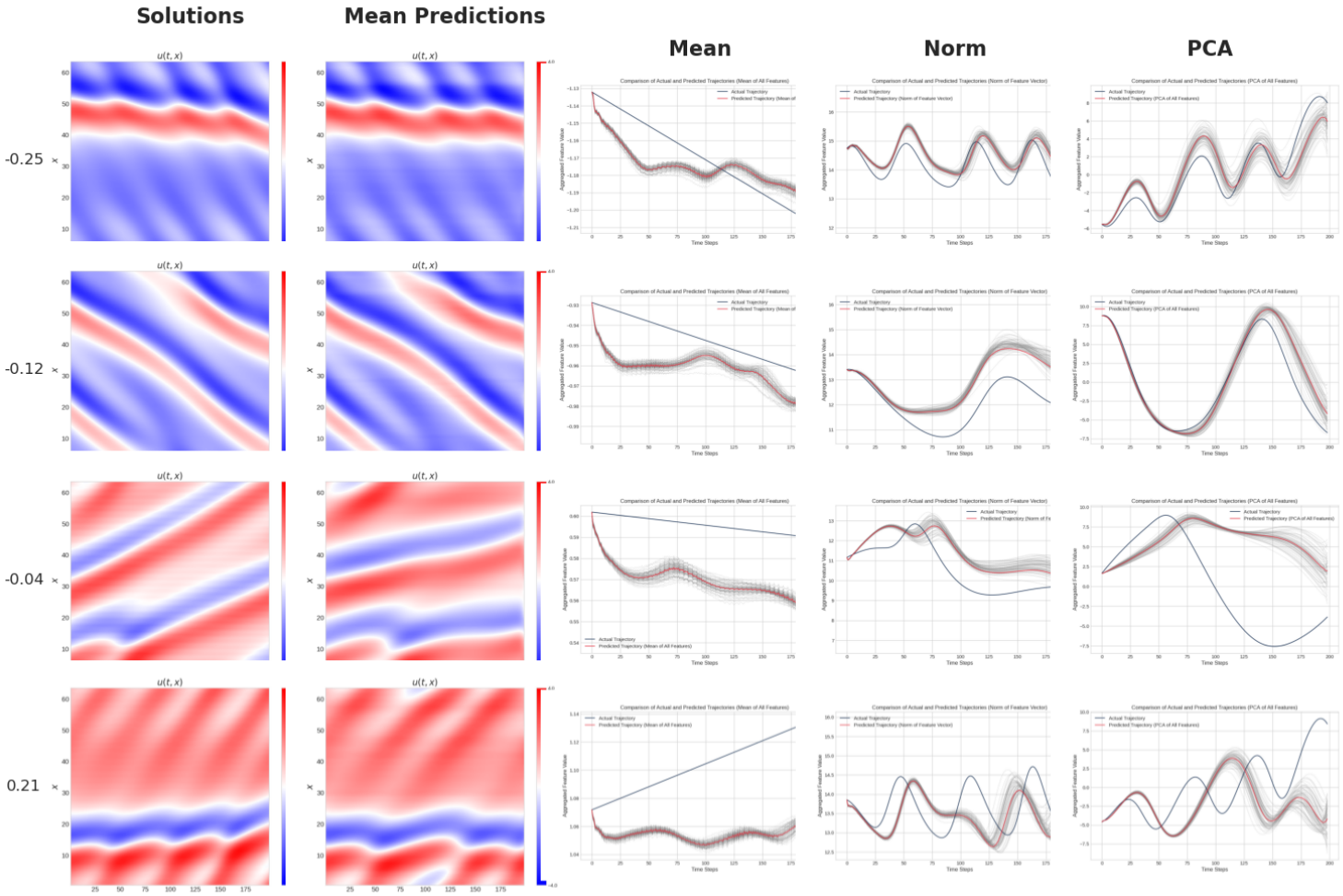

Figure 35: Comparison of model performance across different criteria for varying starting conditions and varying μ with the μ value as a parameter.

Figure 36 and Figure 37 show the prediction errors per step for all the μ values in the test set. The first Figure 36 presents the prediction error for 50 trajectories per μ, with the left Figure corresponding to μ values inside the -0.2 to 0.2 range and the right figure to μ values outside the range. From the first figure 36, it can be concluded that the prediction error for the exclusive μ values is generally smaller than for the inclusive μ values. However, this difference is likely because the behaviour of the actual solutions is smoother for μ values further from 0, as shown in Figure 30. This trend is also observed when examining the smallest prediction error per step in Figure 37.

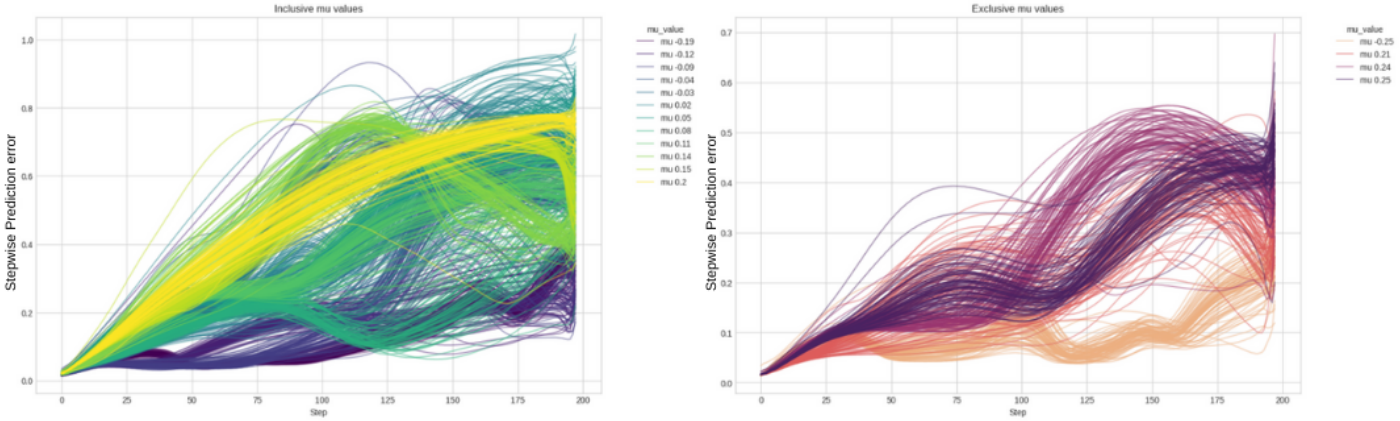

Figure 36: Prediction error per step for trajectories for various μ.

When comparing the smallest stepwise prediction error by selecting the trajectory with the lowest cumulative prediction error over all 200 steps, it is again evident that the error for μ values further from zero is smaller, as indicated by the yellow and purple lines on Figure 37. Additionally, all the smallest prediction errors exhibit a notable characteristic: towards the last step, the error grows significantly. Moreover, none of the errors exhibits linear behaviour regarding steady growth. An interesting observation for $\mu = 0.14$, $\mu = 0.15$, and $\mu = 0.20$ is that around the 125th step, the error reaches its highest value but then decreases towards the last steps. This pattern suggests that there may be complex dynamics within the PDE at this step that the model struggles to predict accurately, particularly at certain points in the trajectory.

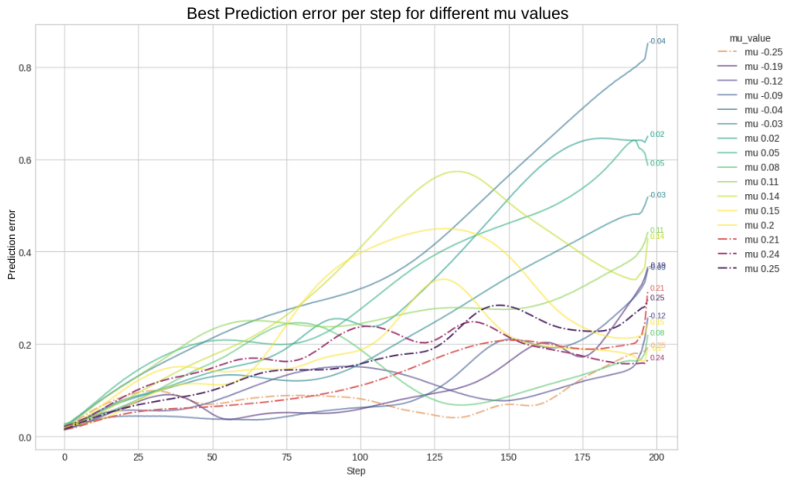

Figure 37: Lowest prediction errors per step for various μ.

Figure 38 shows the trajectories for $\mu = 0.14$, $\mu = 0.15$, and $\mu = 0.20$, where the cumulative stepwise prediction error was the smallest. The black dotted line marks where the stepwise prediction error was the largest. For $\mu = 0.14$ and $\mu = 0.15$, there is a sharp and unexpected change in the PDE behaviour at these points. Specifically, for $\mu = 0.14$, this change is seen as an abrupt end of a line formed by positive PDE values, while for $\mu = 0.15$, it is characterised by the appearance of a blue spot near the lower border of the graph.

These observations suggest that the largest errors correspond to regions where the PDE exhibits complex or abrupt changes in behaviour, which the model finds challenging to predict accurately. Such patterns might indicate areas of higher sensitivity in the PDE's dynamics, where small variations in model predictions can lead to significant deviations in the trajectory.

For both $\mu = 0.15$ and $\mu = 0.20$, the best solutions share the same starting conditions, and while the error at the critical step for $\mu = 0.20$ is lower than for $\mu = 0.15$, the largest error still occurs at the same point. This pattern suggests that the prediction challenges may be tied more to the starting conditions than the specific μ value. The fact that the model struggles similarly across different μ values, despite their differences in the parameter, indicates that there might be underlying complexities in the dynamics of the KS PDE that persist regardless of the exact value of μ.

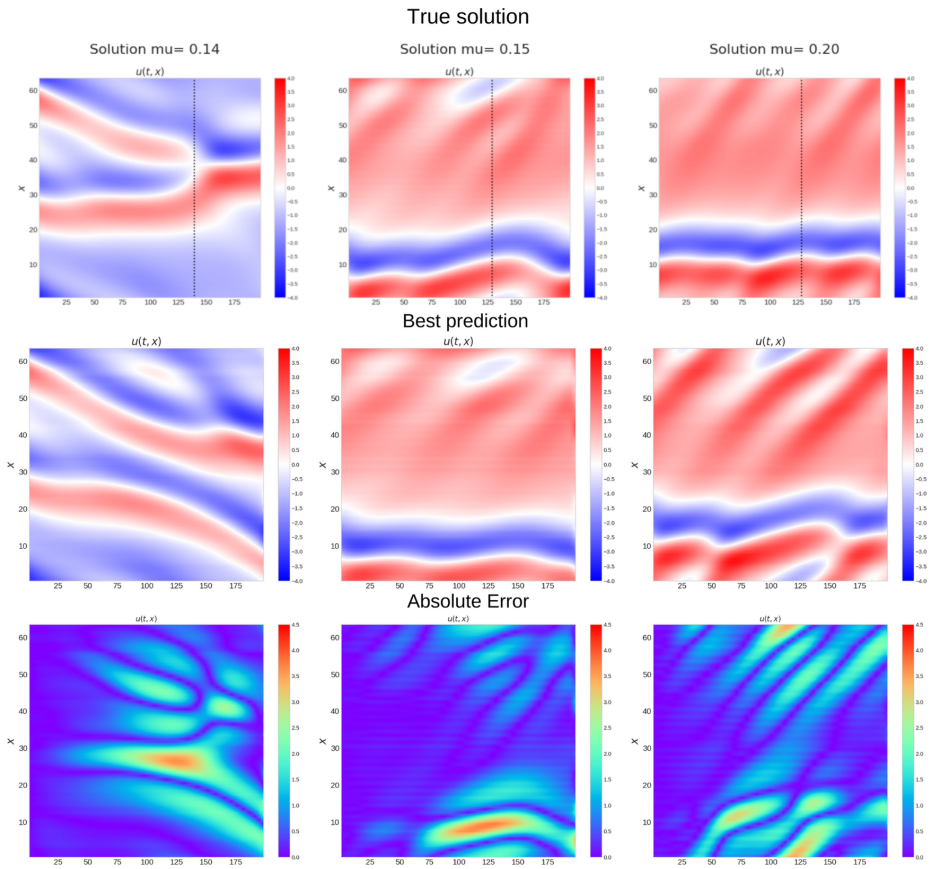

Figure 38: Solutions for μ 0.14, 0.15, 0.20, which exhibited the lowest protection error with the highest stepwise prediction error marked by the black dotted line, the best-generated solution and absolute error.

The findings demonstrate, once again, that incorporating μ as a parameter helps stabilise predictions, leading to more consistent outputs. However, the influence of the starting conditions remains a crucial factor in the model's performance. Accurately capturing the true PDE solution, especially during dynamic transitions, remains a challenge. Therefore, future work could explore more complex conditioning and investigate whether refined training strategies can improve the model's ability to learn and predict these problematic points in the trajectory.

6.2 Experiment - Latent representation

The experiments with dimensionality reduction methods highlight key differences in how each approach (SVD or AE) structures the latent space and ultimately influences the accuracy and temporal consistency of the generated solutions. The spatial organisation of observations within the latent space differs significantly between the two approaches, especially when examining representations at higher dimensions such as 16 or 32. SVD compresses the original data with substantial white space in the latent representation, where observations tend to cluster near the bottom, leaving large regions with negligible or zero values, as seen in Figure 14. This sparsity suggests that SVD effectively preserves global spatial structures in fewer dimensions by concentrating information in compact regions, leading to a lower prediction error when reconstructing solutions. However, this sparsity also introduces potential challenges for the DDPM.

Adding noise to these sparse regions during the DDPM training can interfere with the model's ability to learn meaningful patterns, as noise in low-value areas has limited relevance to the solution's structure. This effect may explain why SVD models benefit from using fewer diffusion steps: with lower noise added to sparse regions; the DDPM focuses more directly on reconstructing the primary features of the KS solution without over-emphasising areas of minimal or negligible information. While fewer steps capture the general form of the equation, additional diffusion steps remain advantageous in smoothing the solution, particularly for denser latent representations where such steps can better enhance finer detail without overwhelming sparse areas with noise. Therefore, while the SVD approach performs well with fewer diffusion steps, it is still less consistent when preserving finer temporal transitions in the KS solution.

In contrast, the AE provides a denser and more distributed latent representation, with fewer white spaces and more consistently occupied regions in higher dimensions, as shown in Figure 15. The AE's latent representation enables a more uniform application of noise across the entire latent space, resulting in smoother generated solutions. Therefore, the AE solutions have a slightly better temporal consistency, as the denser representation allows smoother transitions across time steps, suggesting that the non-linear encoding of an AE is advantageous for maintaining the dynamics of time-evolving processes, such as the KS equation.

In summary, while SVD achieves lower prediction errors by preserving spatial structures compactly in latent space, this approach's sparsity can impact how effectively noise is added and removed during diffusion. The AE, by contrast, provides a more temporally consistent solution due to its denser representation, although this comes at the cost of slightly higher prediction errors. These findings indicate that selecting between SVD and AE for latent space representation should balance spatial accuracy, temporal consistency, and noise management, which are critical factors in generating high-quality solutions for the KS PDE.

Figure 39 provides further insight into why the models with lower latent dimensions produced better results. In the standard case, the normalised data used for training in the multiple μ

experiment is distributed broadly, indicating a balanced spread of values across the range. However, when dimensionality is reduced through methods like SVD and AE, the distribution changes notably as the representation becomes more compressed. Specifically, for both SVD and AE, the lower the dimensionality, the more the data values cluster around zero, resulting in a sharper concentration in the centre of the histogram. This observation suggests that the DDPM performs best when trained on data with a more diversified range, as seen in the latent dimension 8 and the standard case. Conversely, the model's performance reduces as the data distribution becomes dominated by a group of values.

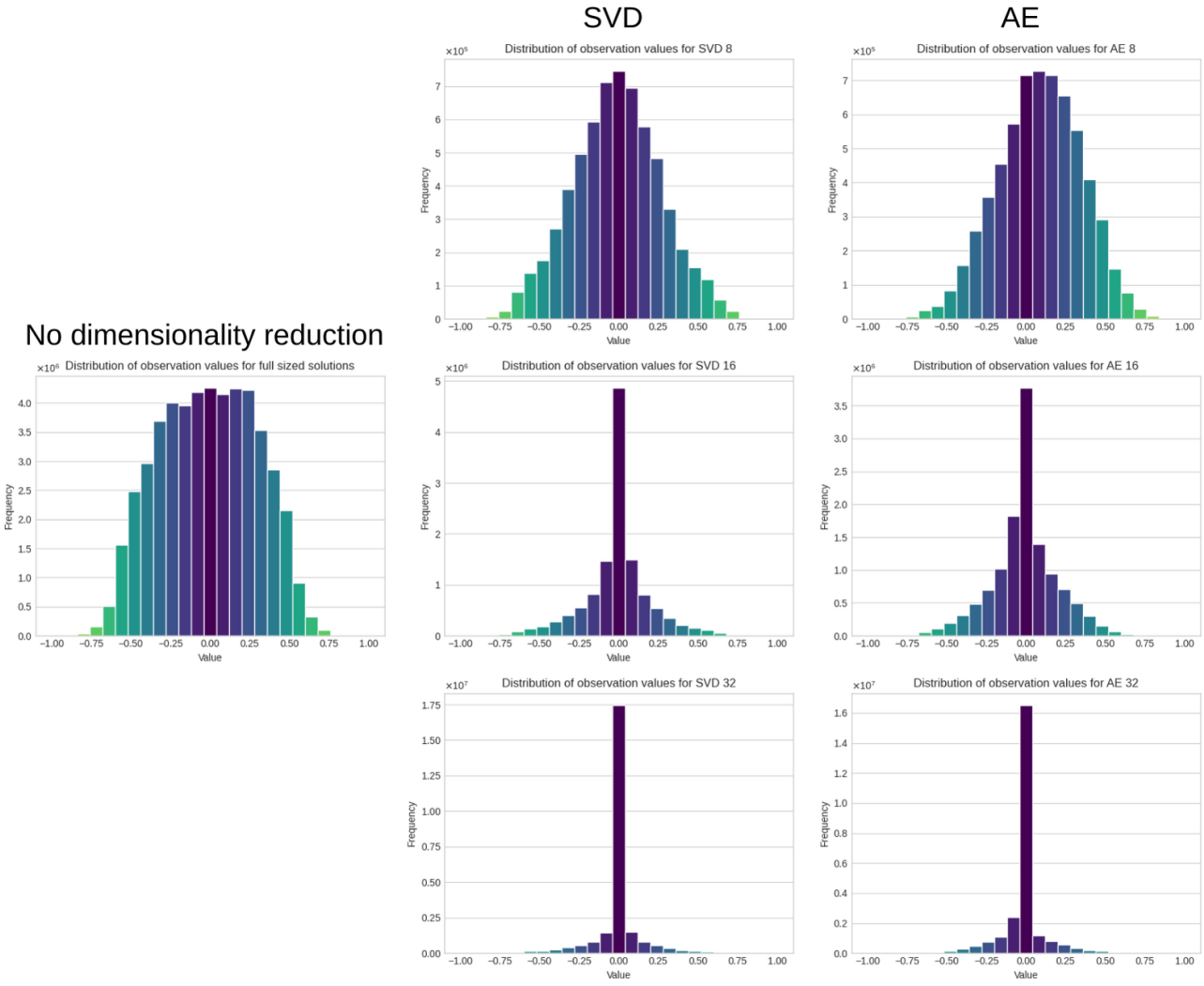

Figure 39: Distribution of observation values across train set for normal data and SVD and AE reduced data.

In addition, inpainting with different sweep strategies reveals significant differences between the standard and latent representations. In the multiple μ experiment without latent-space reduction, sweeps back and both successfully reduce prediction errors by using continuity within the observed space, allowing the model to reconstruct missing values effectively from

known steps. Here, the availability of partial observations improves accuracy by leveraging the underlying structure of the solution.

However, once solutions are transformed into the latent space through dimensionality reduction methods like SVD or AE, the effectiveness of sweep strategies like sweep back and both diminishes, resulting in increased error rather than error reduction. This discrepancy indicates that the latent representations, while useful for dimensional compression and overall reconstruction, lack the inherent structural continuity necessary for interpolating across gaps in the solution. The compressed latent space tends to introduce sparsity or concentrate information in specific regions, complicating the model's ability to apply sweeping in a way that respects the continuous dynamics of the original PDE. This fragmentation in the latent space leads to greater error growth when sweep-based inpainting is applied, suggesting that sweep strategies rely on a consistent and continuous underlying structure to minimise prediction error.

Figure 40 displays the average stepwise prediction error per μ value alongside the overall prediction error across the test set. The thin lines represent individual trajectories. In contrast to the multiple μ case, where the error curves exhibited distinct patterns for each μ value (see Figure 34), the error trends for the latent representation all follow a similar trajectory. Interestingly, although the error lines for individual trajectories do not align with the mean trend, the average behaviour remains consistent across all cases. This observation suggests that, while the model could generate solutions with sufficient accuracy in the multiple μ scenario, where the influence of μ on the error was apparent, the model struggles to recreate trajectories in the latent representation. Consequently, the impact of the parameter μ becomes less detectable due to the more significant errors incurred during trajectory generation.

Furthermore, all the models depicted in Figure 40 demonstrate a rapid increase in stepwise prediction error during the initial steps of the trajectory, followed by a stabilisation of the error in subsequent steps. The most pronounced exponential growth is observed for the SVD 32, SVD 16 1000, and AE 8 1000 models. This observation aligns with the earlier results shown in Figure 21, where the models produced the highest quality solutions at the beginning of the prediction horizon, close to the training horizon. As the number of inpainted steps increased, the quality of the predictions deteriorated. This trend is further reflected in the stepwise error, which continues to grow throughout the values near the training horizon before stabilising and propagating to the end of the solution.

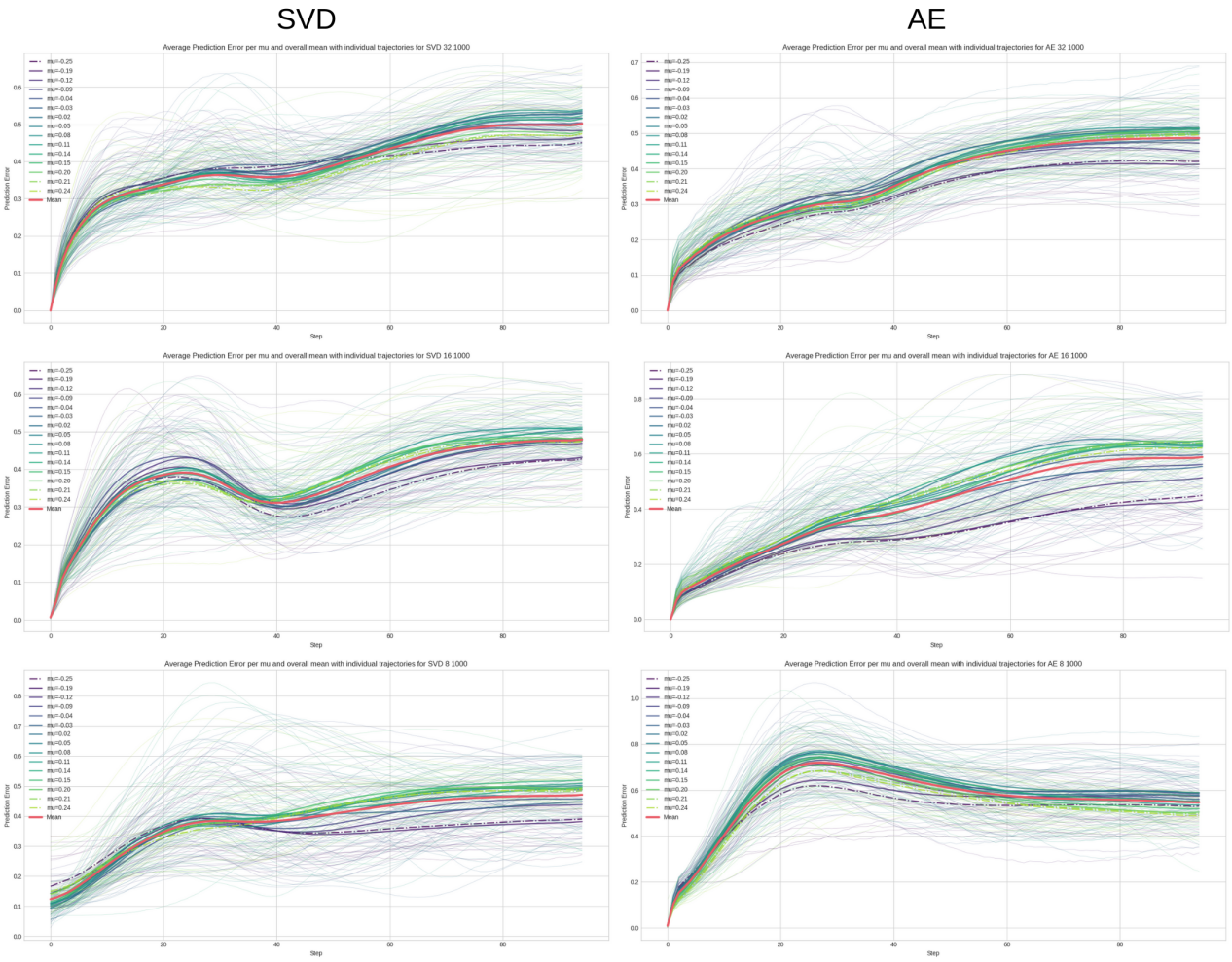

Figure 40: Average stepwise prediction error for SVD [left] and AE [right] for dimensions 32, 16, 8 and 1000 diffusion steps.

In summary, there are significant distinctions in how SVD and AE shape the latent space and impact the DDPM's performance in generating solutions for the KS equation. While SVD effectively preserves spatial structures and achieves lower prediction errors, its sparsity can hinder noise management and the model's ability to maintain temporal consistency. In contrast, the AE provides a denser and more uniformly occupied latent representation, resulting in smoother and more temporally consistent solutions, but at a slight cost to prediction accuracy. These findings suggest that the choice between SVD and AE should be guided by the application's specific requirements, including the desired balance of accuracy and consistency.

This underscores the importance of carefully balancing spatial accuracy, temporal consistency, and noise management when selecting dimensionality reduction methods for optimal performance in generating high-quality PDE solutions. Investigating different dimensionality reduction techniques or alternative AE architectures would be beneficial in obtaining latent representations with a more uniform distribution of values. This could enhance the model's performance by maintaining a broader range of informative data throughout the training and inference processes.

6.3 Experiment - Control

From the previous subsection 6.2, it is clear that the data, particularly its distribution, plays a crucial role in achieving high-quality results when training the model. Initially, the model was trained and evaluated on trajectories of length 200, with the controller activating at step 50 and a prediction horizon of 100 steps. As illustrated in Figure 41, the model consistently predicted the same noisy solution regardless of the starting conditions. In this case, the first 67 steps were provided, so 50 were uncontrolled, and 17 were where the controller worked, but that did not help.

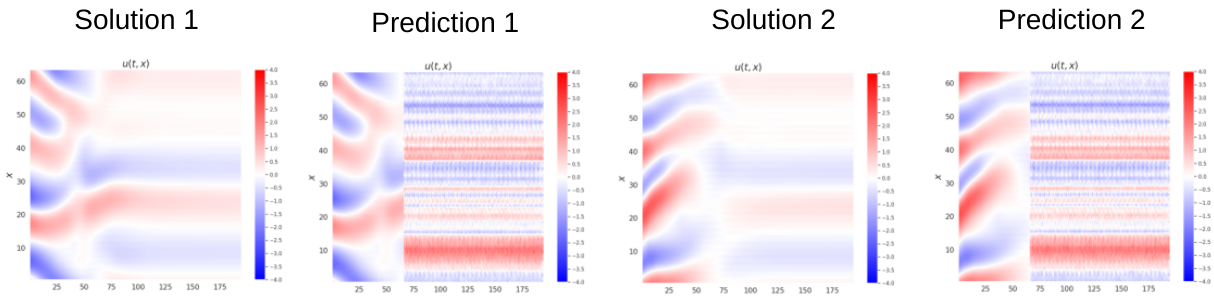

Figure 41: Examples of the solution generated by a model trained and evaluated on trajectories of length 200, with the controller activating at step 50 and a prediction horizon of 100 steps.

In a subsequent attempt, the input data was reduced to only 100 steps, resulting in an equal split between controlled and uncontrolled data, where the controlled segment was actively being inpainted. This adjustment yielded improved results, as shown in Figure 42. However, given that the uncontrolled model can generate good PDE solutions, the focus of the control experiment is exclusively on the controlled part.

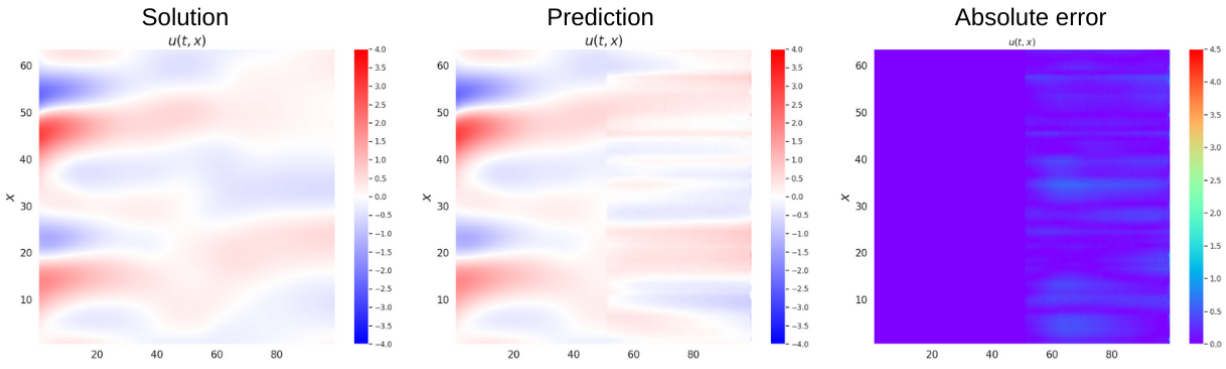

Figure 42: Examples of the solution generated by a model trained and evaluated on trajectories of length 100, with the controller activating at step 50 and a prediction horizon of 100 steps.

It is evident that insufficient variation in the data - such as in the case with 200 steps, where the bigger portion of the solution was controlled — can hinder the model's performance. This

observation highlights the importance of diverse training data, even with the implementation of early stopping to mitigate overfitting.

Besides evaluating the cost function, the controlled models (DDPM, open-loop and closed-loop with and without early stopping) were also evaluated using prediction error with respect to the TD3 controller, see Figure 43. The closed-loop control performed best among the tested approaches, regardless of whether early stopping was applied. This was followed by the open-loop control, which benefited from early stopping. In contrast, when solely utilising the DDPM, the fully trained model outperformed its early-stopped counterpart. These results suggest that while early stopping effectively prevents overfitting, the complete training of the DDPM can lead to better performance in specific contexts.

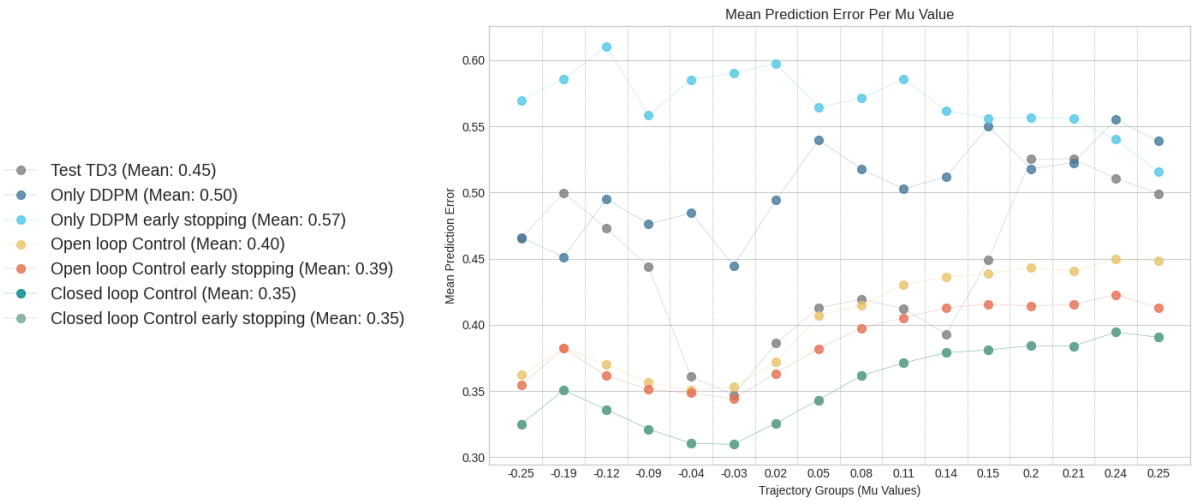

Figure 43: Prediction error for various Control strategies (the points are not connected, lines added for visibility).

However, a different behaviour emerges when examining the actions derived from the inverse model and comparing them to the true actions. Figure 44 illustrates the prediction error associated with the various approaches in terms of actions. The open-loop control exhibited the lowest overall error without early stopping, contrasting with the behaviour observed for state predictions. Additionally, the closed-loop approach showed consistent performance under this metric, regardless of whether early stopping was applied. However, the higher prediction error observed in the closed-loop case may be attributed to the model's iterative nature, as it continuously adjusts its actions with each loop iteration to optimise the trajectory regarding control performance. Such a behaviour can cause greater variability between states, resulting in higher prediction errors.

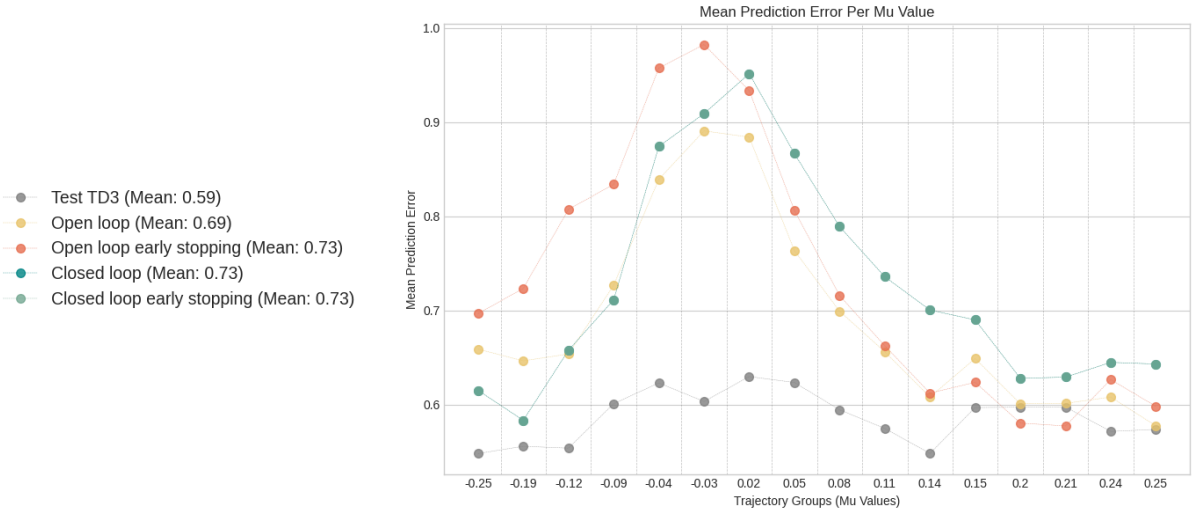

Figure 44: Prediction error for various Control strategies for actions (the points are not connected, lines added for visibility).

Nevertheless, as indicated by the cost function analysis in Section 5.5, the DDPM-based controller performed better regarding observation cost and total reward than the TD3 controller. This suggests that while prediction error is a useful metric for generation tasks, it may not be as effective for control tasks. The prediction error reflects how well the model learns the distribution of actions generated by the TD3 controller. However, this does not necessarily provide meaningful information regarding the effectiveness of the controller itself. In the case of PDE solvers, the prediction error is compared to a well-defined ground truth, but the TD3 controller is not an ideal reference but rather a learned policy. As a result, using projection error to evaluate the DDPM-based controller means comparing it to a non-idea benchmark. This can lead to confusion and a less clear understanding of how well the model performs as a controller, making it difficult to assess its actual control capabilities based on prediction error alone.

Moreover, the example controlled solutions shown in Section 5.5 suggest that the performance of all models varies depending on the system's initial state. Since the DDPM-based model was only provided with the initial step for generating the trajectory, the initial conditions play a significant role in the model's effectiveness. As observed in Figure 25, where rather straightforward solutions were generated, all models achieved relatively low Total Reward. In contrast, more complex solutions, like those shown in Figure 26, yielded higher rewards. This variation raises the question of whether there is a correlation between initial conditions and model performance. Investigating this relationship could provide valuable insights into selecting the most suitable model for controlling a given system based on its initial state.

Another aspect worth considering is the value on which the model is conditioned. In the case of multiple μ 5.3 and latent representations 5.4 experiment, the conditioning parameter was the value of μ. However, in the control experiment 5.5, the model was conditioned on the discounted reward value generated by the TD3 controller during training. For evaluation, the

discounted reward value from the test trajectory was also used as the conditioning parameter for the model. In real-life scenarios, it is often impossible to know the reward value beforehand; therefore, the influence of different values of the conditioning parameter was evaluated to determine which one results in the smallest prediction error. It would be worth investigating how varying the conditioning parameter further impacts the model's performance in future work.

In summary, the model's effectiveness in generating high-quality controlled solutions is significantly influenced by both the variability in the training data and initial conditions. The findings underscore the necessity of providing diverse data during training to enhance model performance and mitigate overfitting. Additionally, it is worth noting that the prediction error and action metrics may not be the best evaluation indicators, as they are inherently tied to the original solutions generated by the TD3 controller. Instead, the cost functions associated with actions and observations offer more meaningful insights into the model's performance and control capabilities.

7 Conclusion

This study investigated the potential of Denoising Diffusion Probabilistic Models (DDPMs) to generate solutions for the parameterised Kuramoto–Sivashinsky (KS) equation, focusing on generalisation capabilities and dimensionality reduction. The research questions are addressed as follows:

Regarding the primary research question — whether it is possible to generate solutions from partial differential equations (PDEs) using DDPMs - the findings demonstrate that DDPMs can generate coherent and smooth solutions for the KS equation. The model produced temporally-consistent results across both (parameter and time) interpolation and extrapolation tasks. Notably, the DDPM can effectively handle unseen parameter values of μ, showing that DDPMs are a robust method for solving parameterised PDEs. However, limitations were observed in the model's long-term predictions, where projection errors accumulated, mainly when predicting further into the future.

The results indicate strong generalisation capabilities in response to the complementary sub-question concerning the model's ability to generalise for unseen parameterised values. The model successfully generated accurate solutions for μ values outside the training range, even in challenging parameter extrapolation cases. Overall, the model's accuracy improved when more solution steps were provided, showing that DDPMs can generalise across unseen PDE parameter values when partial information is available.

Concerning the sub-research question regarding the reducing computational complexity of the model through latent representation, it was observed that reducing the dimensionality of the observations allowed the model to make predictions faster, indicating a reduction in computational complexity. However, it was also noted that other factors, such as the data distribution, the number of diffusion steps and the prediction horizon length, significantly influenced the quality of the generated solutions. Adjustments to these parameters may be necessary to achieve better fidelity, which could increase the model's complexity. Thus, while the results indicate the potential for dimensionality reduction to enhance performance, further research is essential to explore how these parameters are interrelated to optimise fidelity at the lowest computational cost.

Lastly, for the third research question regarding the control abilities of the model, the results highlight the model's strong capacity to generate controlled solutions for parametric PDEs, particularly through the application of closed-loop control strategies. These insights into the control capabilities of the model complement the findings related to generalisation and dimensionality reduction, further underscoring the versatility and potential of DDPMs in addressing parametric PDEs.

8 Future Work

Several promising directions for future research have emerged from the findings of this project. These areas aim to improve model performance further, optimize dimensionality reduction strategies, and extend the approach to more complex and realistic PDE systems.

One key area of improvement is the more in-depth analysis of the conditioning process. While incorporating μ has demonstrated a stabilizing effect on predictions, see Section 6.1, further exploring how additional information could refine the conditioning could result in improved model performance in both controlled and non-controlled solution generation. For example, boundary conditions or time-varying parameters may offer richer conditioning inputs, leading to more accurate reconstructions of the true PDE solution.

Another promising research direction is examining the impact of higher-dimensional inputs on the quality of generated solutions. This study tested input dimensions up to 64 observations for cases involving multiple values of μ. Extending this analysis to even higher input dimensions before reducing them back to manageable sizes (such as 64 or lower) using SVD or AEs could offer insights into how well the model handles more complex input structures while maintaining high-quality predictions.

A preliminary exploration of this approach has been carried out for the single μ case (see Appendix D), showing that the model can manage increased dimensionality while maintaining accuracy. This would involve increasing the input dimensions and adjusting the model architecture to accommodate the expanded complexity. Tuning the AE and modifying the DDPM's handling of these inputs will ensure the model remains efficient and accurate when dealing with larger input spaces. Additionally, investigating control in latent space could offer insights into enhancing the model's adaptability and efficiency.

Investigating higher-dimensional inputs also connects to another promising direction — applying the model to more complex PDEs, such as those involving two or three spatial dimensions. This expansion would require revisiting and tuning the model architecture to capture the increased complexity inherent in higher-dimensional systems. The successful application of the model to more complex PDEs would significantly broaden its usability in real-world scenarios, where systems often exhibit multidimensional behaviour and more intricate dynamics.

Lastly, training the model across different PDE types could broaden its versatility. This would involve creating a dataset that includes various PDEs and conditioning the model on the type of PDE alongside other parameters. This strategy could enable the model to generalize across different dynamical systems, making it capable of generating trajectories for varied PDE types based on initial conditions and conditioning parameters. Achieving this would position the model as a versatile tool capable of tackling a wide range of PDE problems within a unified framework.

9 References

[1] Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. "High-dimensional dynamics of generalization error in neural networks". In: *Neural Networks* 132 (2020), pp. 428–446.

[2] Anurag Ajay et al. "Is conditional generative modeling all you need for decision-making?" In: *arXiv preprint arXiv:2211.15657* (2022).

[3] Jinwon An and Sungzoon Cho. "Variational autoencoder based anomaly detection using reconstruction probability". In: *Special lecture on IE* 2.1 (2015), pp. 1–18.

[4] Eyaya Fekadie Anley. "Numerical solutions of elliptic partial differential equations by using finite volume method". In: *Pure and Applied Mathematics Journal* 5.4 (2015), pp. 120–129.

[5] Farzana Anowar, Samira Sadaoui, and Bassant Selim. "Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne)". In: *Computer Science Review* 40 (2021), p. 100378.

[6] Klapa Antonion et al. "Machine Learning Through Physics–Informed Neural Networks: Progress and Challenges". In: *Academic Journal of Science and Technology* 9.1 (2024), pp. 46–49.

[7] Nakhlé H Asmar. *Partial differential equations with Fourier series and boundary value problems*. Courier Dover Publications, 2016.

[8] J Thomas Beale. "Smoothing properties of implicit finite difference methods for a diffusion equation in maximum norm". In: *SIAM Journal on Numerical Analysis* 47.4 (2009), pp. 2476–2495.

[9] Richard Bellman. "Dynamic programming". In: *science* 153.3731 (1966), pp. 34–37.

[10] Peter Benner, Serkan Gugercin, and Karen Willcox. "A survey of projection-based model reduction methods for parametric dynamical systems". In: *SIAM review* 57.4 (2015), pp. 483–531.

[11] Kaushik Bhattacharya et al. "Model reduction and neural networks for parametric PDEs". In: *The SMAI journal of computational mathematics* 7 (2021), pp. 121–157.

[12] Andreas Blattmann et al. "Align your latents: High-resolution video synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 22563–22575.

[13] Nicolò Botteghi and Urban Fasel. "Parametric PDE Control with Deep Reinforcement Learning and Differentiable L0-Sparse Polynomial Policies". In: *arXiv preprint arXiv:2403.15267* (2024).

[14] Nicolò Botteghi, Mannes Poel, and Christoph Brune. "Unsupervised representation learning in deep reinforcement learning: A review". In: *arXiv preprint arXiv:2208.14226* (2022).

[15] Nicolò Botteghi et al. "Trajectory Generation, Control, and Safety with Denoising Diffusion Probabilistic Models". In: *arXiv preprint arXiv:2306.15512* (2023).

[16] Michael Cogswell et al. "Reducing overfitting in deep networks by decorrelating representations". In: *arXiv preprint arXiv:1511.06068* (2015).

[17] Albert Cohen and Ronald DeVore. “Approximation of high-dimensional parametric PDEs”. In: *Acta Numerica* 24 (2015), pp. 1–159.

[18] Salvatore Cuomo et al. “Scientific machine learning through physics–informed neural networks: Where we are and what’s next”. In: *Journal of Scientific Computing* 92.3 (2022), p. 88.

[19] Carl Doersch. “Tutorial on variational autoencoders”. In: *arXiv preprint arXiv:1606.05908* (2016).

[20] Karthik Elamvazhuthi, Darshan Gadginmath, and Fabio Pasqualetti. “Denoising Diffusion-Based Control of Nonlinear Systems”. In: *arXiv preprint arXiv:2402.02297* (2024).

[21] Gwynne Evans, J Blackledge, and Peter Yardley. *Analytic methods for partial differential equations*. Springer Science & Business Media, 2012.

[22] Amir-massoud Farahmand, Saleh Nabi, and Daniel N Nikovski. “Deep reinforcement learning for partial differential equation control”. In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 3120–3127.

[23] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[24] Gene Golub and William Kahan. “Calculating the singular values and pseudo-inverse of a matrix”. In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2 (1965), pp. 205–224.

[25] Sebastian Götschel and Martin Weiser. “Compression challenges in large scale partial differential equation solvers”. In: *Algorithms* 12.9 (2019), p. 197.

[26] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[27] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[28] Jonathan Ho and Tim Salimans. “Classifier-free diffusion guidance”. In: *arXiv preprint arXiv:2207.12598* (2022).

[29] Philipp Holl, Vladlen Koltun, and Nils Thuerey. “Learning to control pdes with differentiable physics”. In: *arXiv preprint arXiv:2001.07457* (2020).

[30] Xiaoyu Huang et al. “DiffuseLoco: Real-Time Legged Locomotion Control with Diffusion from Offline Datasets”. In: *arXiv preprint arXiv:2404.19264* (2024).

[31] Zubayer Islam et al. “Crash data augmentation using variational autoencoder”. In: *Accident Analysis & Prevention* 151 (2021), p. 105950.

[32] Weikuan Jia et al. “Feature dimensionality reduction: a review”. In: *Complex & Intelligent Systems* 8.3 (2022), pp. 2663–2693.

[33] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. “Solving parametric PDE problems with artificial neural networks”. In: *European Journal of Applied Mathematics* 32.3 (2021), pp. 421–435.

[34] Seung Wook Kim et al. “NeuralField-LDM: Scene Generation With Hierarchical Latent Diffusion Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 8496–8506.

[35] Diederik P Kingma, Max Welling, et al. “An introduction to variational autoencoders”. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[36] Nikola Kovachki et al. “Neural operator: Learning maps between function spaces with applications to pdes”. In: *Journal of Machine Learning Research* 24.89 (2023), pp. 1–97.

[37] Zichao Long et al. “Pde-net: Learning pdes from data”. In: *International conference on machine learning*. PMLR. 2018, pp. 3208–3216.

[38] Lu Lu, Pengzhan Jin, and George Em Karniadakis. “Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators”. In: *arXiv preprint arXiv:1910.03193* (2019).

[39] M Mano et al. “Finite element method”. In: (2003).

[40] Andrea Manzoni, Alfio Quarteroni, and Sandro Salsa. *Optimal control of partial differential equations*. Springer, 2021.

[41] Craig Michoski et al. “Solving differential equations using deep neural networks”. In: *Neurocomputing* 399 (2020), pp. 193–212.

[42] Alexander Quinn Nichol and Prafulla Dhariwal. “Improved denoising diffusion probabilistic models”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8162–8171.

[43] John Noye. “Finite difference techniques for partial differential equations”. In: *North-Holland mathematics studies*. Vol. 83. Elsevier, 1984, pp. 95–354.

[44] George Papanikos, Maria Ch Gousidou-Koutita, et al. “A computational study with finite element method and finite difference method for 2D elliptic partial differential equations”. In: *Applied Mathematics* 6.12 (2015), p. 2104.

[45] Zakaria Patel and Kirill Serkh. “Enhancing Image Layout Control with Loss-Guided Diffusion Models”. In: *arXiv preprint arXiv:2405.14101* (2024).

[46] Sebastian Peitz et al. “Distributed control of partial differential equations using convolutional reinforcement learning”. In: *Physica D: Nonlinear Phenomena* 461 (2024), p. 134096.

[47] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced basis methods for partial differential equations: an introduction*. Vol. 92. Springer, 2015.

[48] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. “Generating diverse high-fidelity images with vq-vae-2”. In: *Advances in neural information processing systems* 32 (2019).

[49] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[51] Lars Ruthotto and Eldad Haber. “Deep neural networks motivated by partial differential equations”. In: *Journal of Mathematical Imaging and Vision* 62.3 (2020), pp. 352–364.

[52] Nahian Siddique et al. “U-net and its variants for medical image segmentation: A review of theory and applications”. In: *Ieee Access* 9 (2021), pp. 82031–82057.

[53] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265.

[54] James William Thomas. *Numerical partial differential equations: finite difference methods*. Vol. 22. Springer Science & Business Media, 2013.

[55] Fredi Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*. Vol. 112. American Mathematical Soc., 2010.

[56] Laurens Van Der Maaten, Eric O Postma, H Jaap Van Den Herik, et al. “Dimensionality reduction: A comparative review”. In: *Journal of machine learning research* 10.66-71 (2009), p. 13.

[57] VM Veliov. “Lipschitz continuity of the value function in optimal control”. In: *Journal of optimization theory and applications* 94.2 (1997), pp. 335–363.

[58] Colin Vignon, Jean Rabault, and Ricardo Vinuesa. “Recent advances in applying deep reinforcement learning for flow control: Perspectives and future directions”. In: *Physics of fluids* 35.3 (2023).

[59] Ricardo Vinuesa, Steven L Brunton, and Beverley J McKeon. “The transformative potential of machine learning for experiments in fluid mechanics”. In: *Nature Reviews Physics* 5.9 (2023), pp. 536–545.

[60] Long Wei et al. “Generative PDE Control”. In: *ICLR 2024 Workshop on AI4DifferentialEquations In Science*.

[61] Haixu Wu et al. “Solving high-dimensional pdes with latent spectral models”. In: *arXiv preprint arXiv:2301.12664* (2023).

[62] Xiangyuan Zhang et al. “Controlgym: Large-Scale Safety-Critical Control Environments for Benchmarking Reinforcement Learning Algorithms”. In: *arXiv preprint arXiv:2311.18736* (2023).

[63] Yinhao Zhu and Nicholas Zabaras. “Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification”. In: *Journal of Computational Physics* 366 (2018), pp. 415–447.

A Experiment and Analysis for single μ

A.1 Experiment - Single μ value

This experiment evaluates the Kuramoto-Sivashinsky (KS) PDE for a single parameter $\mu = 0.0$, representing the most basic case. By focusing on a fixed value of μ, this study aims to establish a foundational understanding of the equation's dynamics before exploring more intricate scenarios. For this case, the values of other parameters are spatial domain $\mathcal{L} = [0, 22]$, domain discretisation $N_x = 64$, simulation time $T = 120s$, time step size for integration $dt = 0.1$. Each trajectory uses different random initial conditions to avoid duplicate data.

For this experiment, a total of 2000 trajectories are generated, each with a duration of 200 steps and different initial conditions. The train and test split proportion is 90/10. Figure 45 depicts examples of trajectories used to train the model.

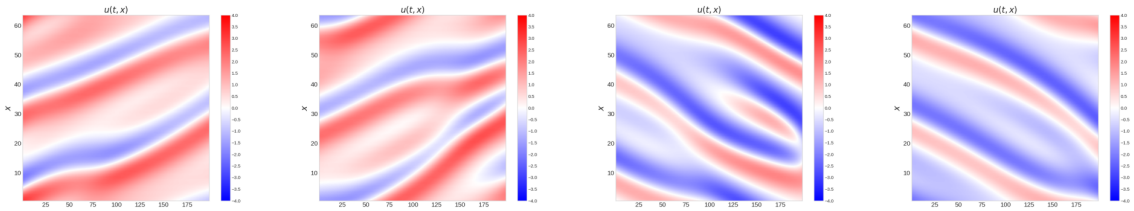

Figure 45: Sample trajectories of the Kuramoto-Sivashinsky equation with $\mu = 0.0$.

This experiment evaluated three cases regarding the number of denoising steps used in the DDPM. The number of denoising steps taken into account was 200 (in line with the original DDPM implementation by [2]), 500 and 1000. For all cases, the model's horizon length is 100. Additionally, since the model is probabilistic, the first solution obtained by inpainting might not be the best. Therefore, for all subsequent evaluations, the model generated ten trajectories for the same initial conditions and then selected the best solution for the lowest prediction error. Figure 46 shows the average prediction error over the ten trajectories compared to the best for the number of denoising steps 200, 500, and 1000. The findings indicate that employing 200 denoising steps results in the lowest prediction error when inpainting the trajectory. This is particularly evident from the dark red line in Figures 46 and 47, which denotes the optimal solution for 200 denoising steps. Additionally, Figure 48 shows how the number of diffusion steps influences the absolute error made by the model, where the model with 200 diffusion steps performs the best. Consequently, 200 denoising steps were identified as the most appropriate value for subsequent experiments.

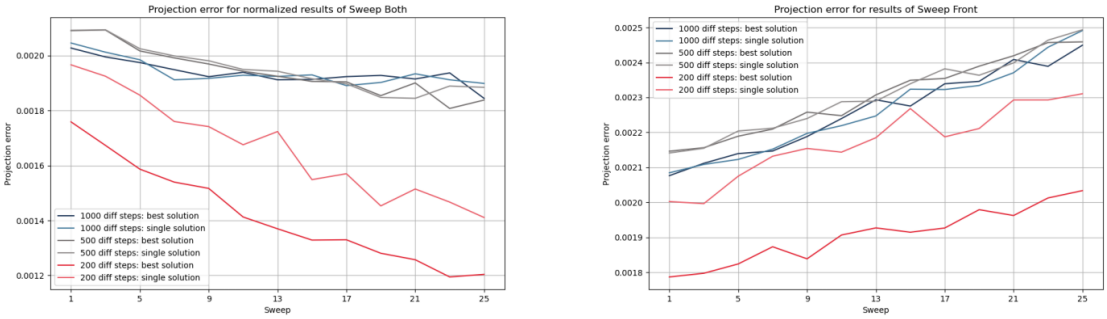

Figure 46: Average prediction error normalised for sweep-both (left) and unnormalised for sweep-front (right) for $\mu = 0.0$.

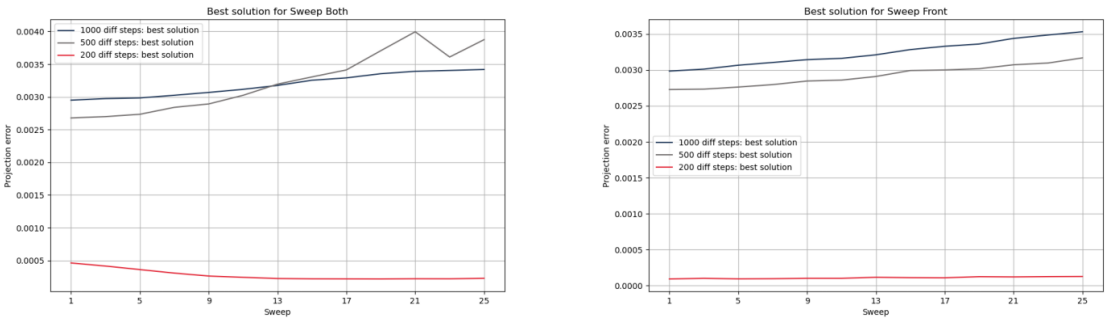

Figure 47: Prediction error for sweep-both (left) and sweep-front (right) for best solution for $\mu = 0.0$.

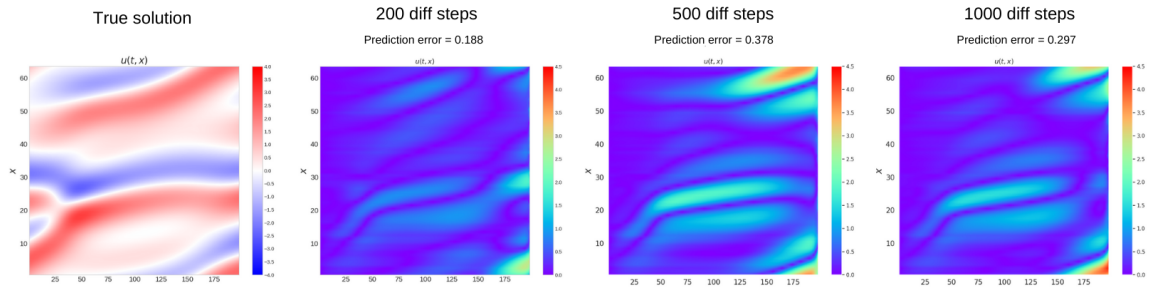

Figure 48: Generated solution's absolute error under a different number of diffusion steps.

The input data to the model is pre-normalised to a range between -1 and 1, and the predictions are subsequently re-normalised to the original scale. As shown in Figure 49, the final prediction error remains consistent whether the data is normalised. This indicates that the normalisation process does not contribute to the model's overall error.

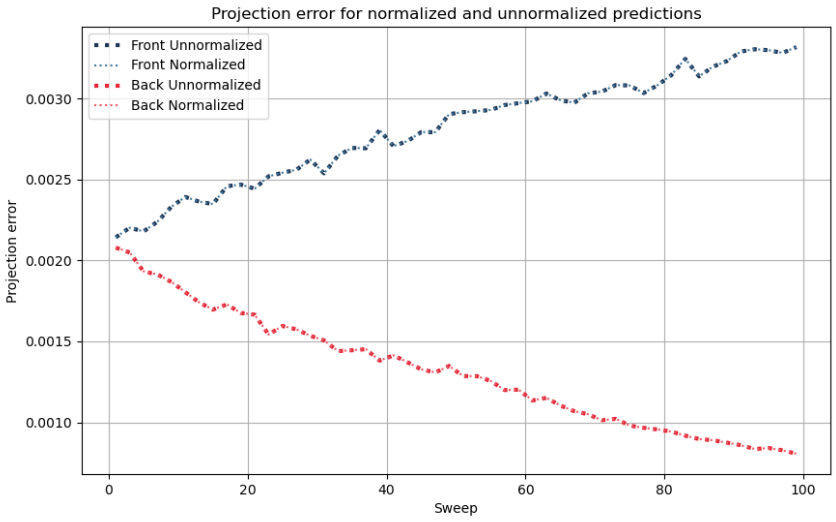

Figure 49: Comparison of prediction error for unnormalised and normalised predictions for sweep-front and back for $\mu = 0.0$.

With the number of diffusion steps equal to 200, the general behaviour of the model is analysed first. The average output of the model, when no initial step is provided, is shown in Figure 50. As can be seen both by the visualisation of the general solution and the plot of the mean and variance, the general predicted solution oscillates around higher values the PDE can assume. This phenomenon is not explained by the training set, where the values are balanced and oscillate around zero, as demonstrated by visualising the general solution and the mean and variance plot.

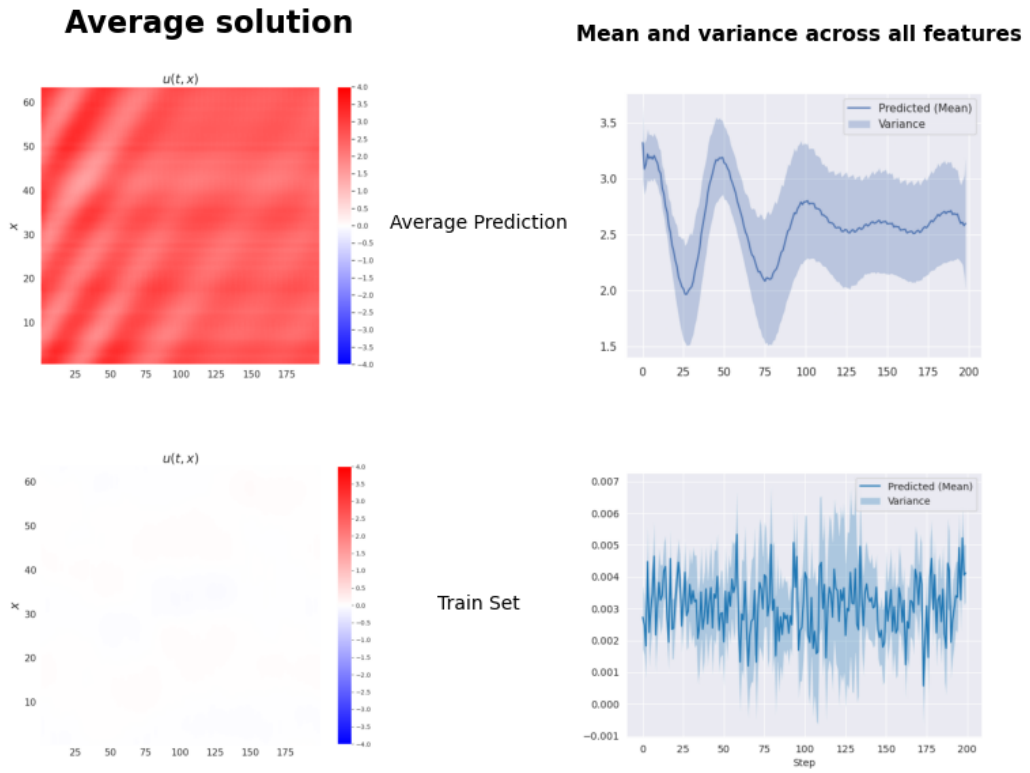

Figure 50: Comparison of a general predicted solution (top row) with a general solution in the train set (bottom row) for $\mu = 0.0$.

When analysing solutions generated by the model, where the model was provided with the initial step of the trajectory to predict subsequent steps, it becomes evident that the model can approximate or closely match the actual solutions (both positive and negative). Figure 51 illustrates the model's performance across various criteria. Specifically:

- **The first column** displays the actual solution from the test set.
- **The second column** shows the mean prediction of the model.
- **The third column** depicts the comparison of means.
- **The fourth column** provides a comparison based on norms.
- **The fifth column** shows the results of Principal Component Analysis (PCA).

Due to the probabilistic nature of the model, 100 trajectories were generated for each starting condition. In the figure, the grey lines represent individual trajectories, while the red line indicates the average solution. Although the model's predictions do not perfectly match the original solutions, they generally follow the trajectory behaviour of the actual solutions.

However, there are noticeable limitations in the model's predictions. Analysis of the means across all depicted cases reveals that the actual mean is consistently lower than the mean of the model-generated solutions. This discrepancy may be attributed to the fact that the

unconditioned model generally produces solutions with higher mean values, as shown in Figure 50. Furthermore, the model fails to maintain the KS PDE property that requires the spatial mean to be constant throughout the trajectory. The most significant observed difference in the predicted spatial mean is a gap of 0.04 between the predicted maximum value and the predicted mean value.

Similarly, the norm analysis shows that the actual norm is smaller than the predicted norm. Lastly, the PCA analysis indicates that the model's predictions align with the true PCA results. Discrepancies between the predicted and actual PCA values are most pronounced after around 120 steps.

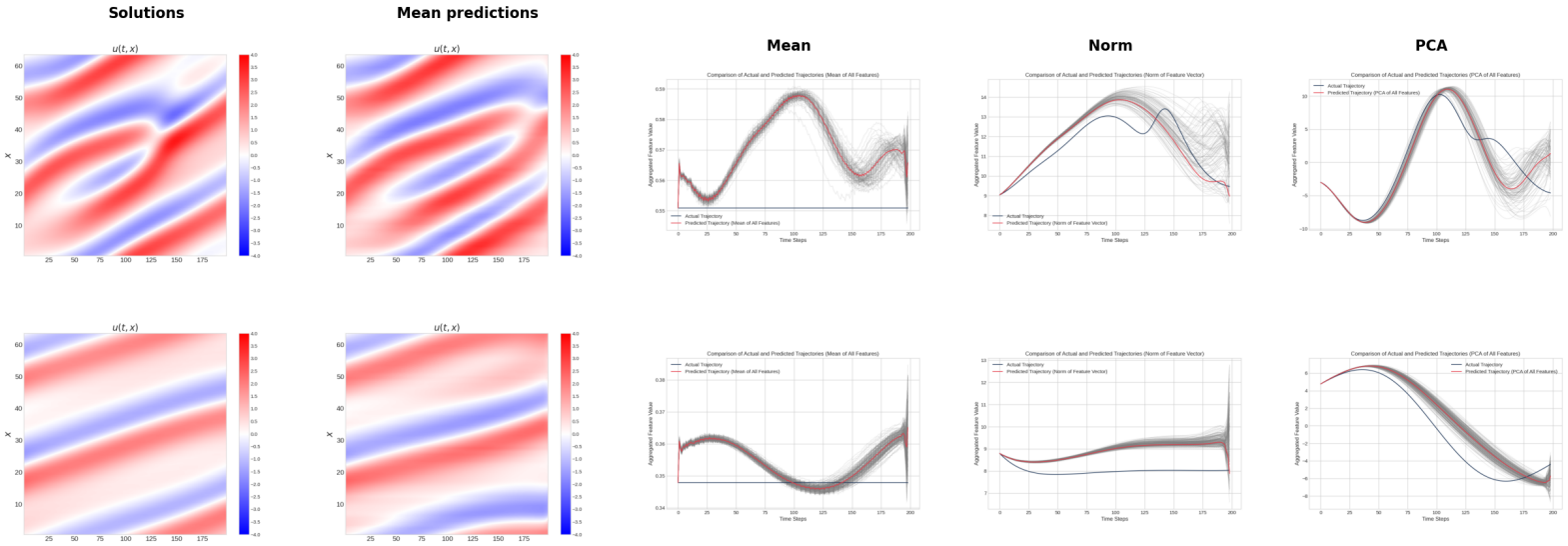

Figure 51: Comparison of model performance across different criteria for varying starting conditions for $\mu = 0.0$.

Figure 51 indicates that the model can approximate its behaviour while not perfectly predicting the solution. The model demonstrates a capability to follow the dynamics of the actual solution, capturing its key features even if it does not match exactly. Although the predicted mean and norm values are generally higher than the actual values, the model still successfully identifies the most significant values at each step (up to a certain point in time). This suggests that despite some discrepancies, the model can capture the essential characteristics of the solution and follows the trajectory reasonably well, as evidenced by the close correspondence in PCA.

Figure 52 compares the actual, the predicted mean, and the best-predicted solution and the stepwise prediction error used to select the best solution. As Figure 52 shows, the mean and best solutions closely track the actual solution up to approximately 100 steps. Beyond this point, the prediction error increases more rapidly.

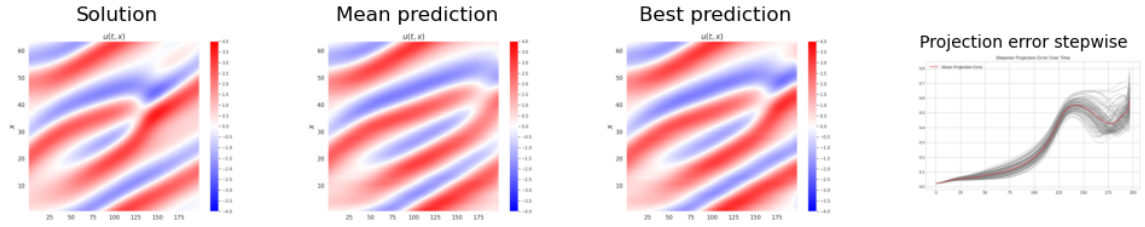

Figure 52: Comparison of actual solution with mean and best solution and stepwise prediction error for $\mu = 0.0$.

The best solution is generally closer to the actual solution compared to the mean solution, which is expected. Notably, several key differences bring the best solution closer to the actual solution. For instance, the sharp tooth visible in the actual solution at coordinates (40, 125), characterised by negative (blue) values, is absent in the mean solution but is partially preserved in the best solution at coordinates (42, 160). Additionally, a prominent line of high positive values extending from coordinates (10, 60) is present in both the actual and best solutions but is missing from the mean solution

Furthermore, while negative values appear in the bottom right corner of the actual solution, they are not present in the mean solution. However, negative values are observed in the best solution, showing an improvement over the mean solution but still not perfectly matching the actual solution.

In conclusion, the best solution is not a perfect match to the actual solution. Still, it provides a more accurate representation by preserving more of the distinctive characteristics present in the original data compared to the mean solution. This indicates that selecting the best solution based on the prediction error offers a meaningful advantage in approximating the true solution compared to using the mean solution. Other examples of behaviour under different starting conditions are shown and described in the Appendix B

Given that the prediction error can significantly increase at the final steps, an analysis was performed to determine whether the prediction horizon length influenced this behaviour. As illustrated in Figure 53, this phenomenon occurs irrespective of the horizon length. Consequently, it is assumed that this behaviour is an inherent characteristic of the model.

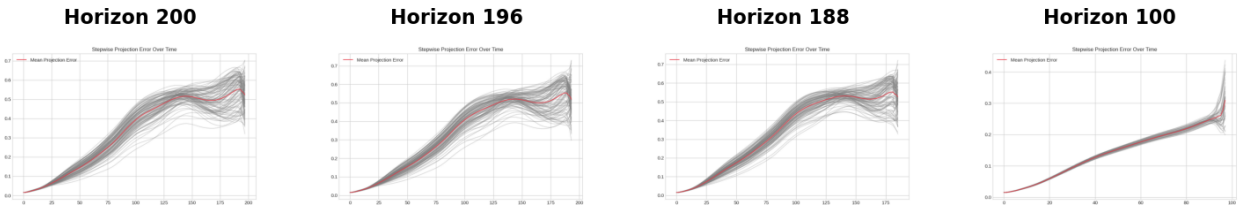

Figure 53: Stepwise prediction error for the horizon of 200, 196, 188 and 100 steps for $\mu = 0.0$.

Another variable examined before evaluating the model with different sweeps was the effect of inpainting the first step of the trajectory s_t when performing inpainting from the final step s_{t+T-1}. The results, illustrated in Figure 54, indicate that providing the first step of the trajectory yields smoother results when inpainting from the last to the initial step. While the trajectory is somewhat preserved when the first step is not provided, it is significantly better preserved and notably smoother when the first step is present. The former case exhibits visible grid-like artefacts, whereas the latter presents a much smoother trajectory.

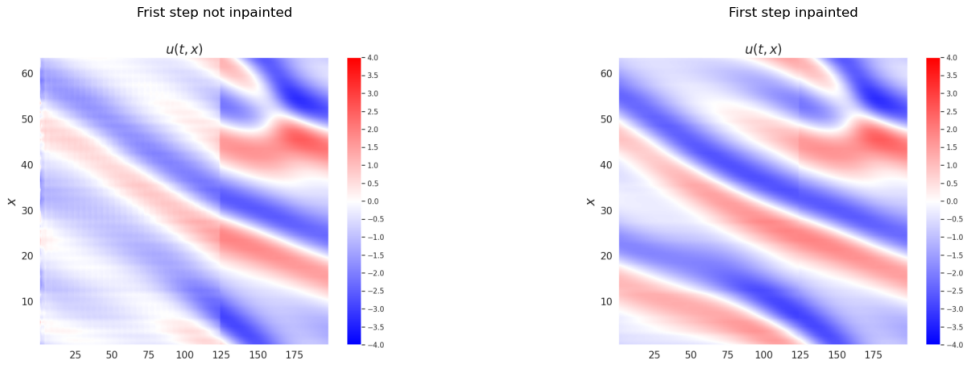

Figure 54: Difference between inpainting and not the first step in the sweep-back.

Figure 55 presents the prediction error for both scenarios. The prediction error is on the order of 10^{-4}. The case where the first step is provided exhibits a lower error than the scenario without it for both the average and the best solution. Additionally, the variation in prediction error is smoother when the first step is provided. Consequently, the first step will be provided during the sweep-back for the remainder of the evaluation.

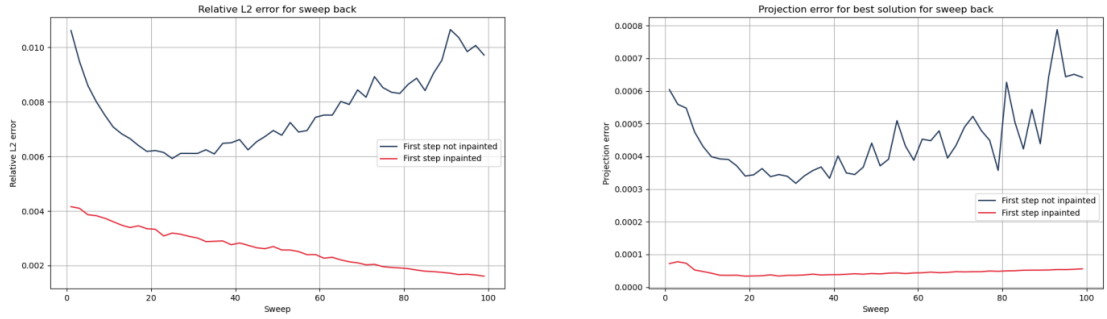

Figure 55: Relative L2 error(left) and best prediction error (right) for sweep-back for $\mu = 0.0$.

Firstly, only inpainting from the first step was evaluated as shown in Figures 56, 57, 58. The leftmost image in each set shows the original solution to the PDE for $\mu = 0$ with varying initial conditions. The second image depicts the model's output with specific steps provided (sweep back, front, or both). The third image represents the prediction error, while the fourth image illustrates the absolute error. In these cases, the generated trajectory closely resembles the

actual one, indicating that the model can capture significant trajectory features. However, notable discrepancies arise, particularly in the later steps. The errors, especially the absolute error, are rather prominent, revealing that precision diminishes over time while the model can replicate the general trajectory shape.

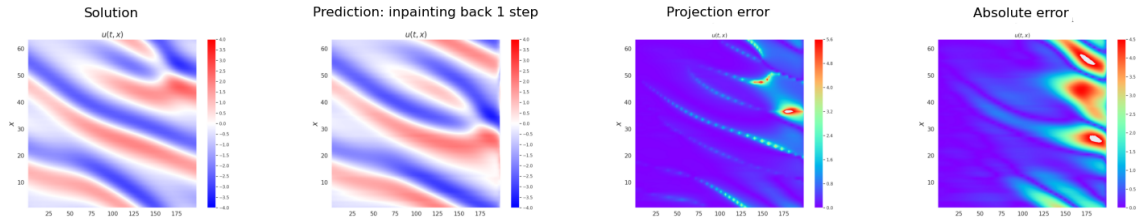

Figure 56: Sweep-back for 1 step and first step provided.

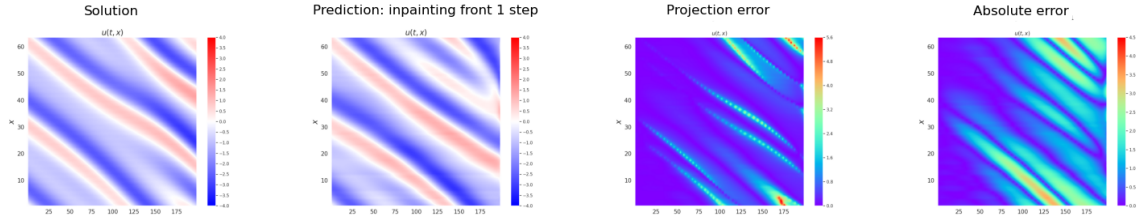

Figure 57: Sweep-front for 1 step.

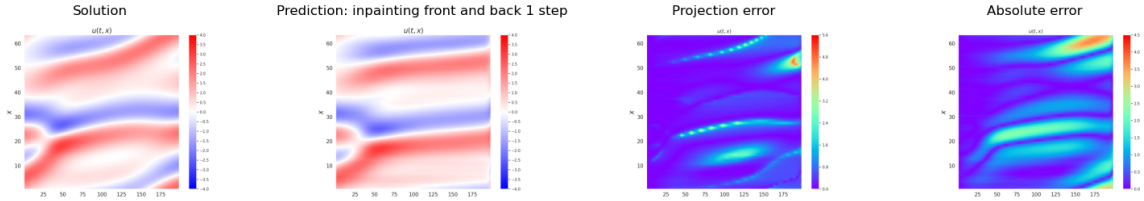

Figure 58: Sweep-both for 1 step.

Secondly, the same analysis was concluded but for varying numbers of the inpainted steps as shown by Figures 59, 60, 61 and 62. In these cases, the generated trajectories show improvement compared to the single-step inpainting approach. Although the trajectories do not perfectly match the original solutions—evidenced by the light blue regions on the error plots—the overall errors are notably reduced.

The sweep-front approach (with 89 steps provided) exhibits the largest errors among the different inpainting strategies. In comparison, the sweep-back method (with 75 steps provided) and the sweep-both method (with a total of 98 steps provided) show the smallest errors. This finding is consistent with the analysis of relative errors, which indicates that the sweep-front approach results in the highest errors. In contrast, the sweep-back approach yields the lowest errors, as illustrated in Figure 63.

Notably, performance improves with multiple provided steps compared to single-step inpainting. For the sweep-back and sweep-both approaches, the largest absolute errors are now situated more centrally within the unpainted trajectory rather than towards the end, suggesting a more balanced error distribution across the trajectory. This indicates that lowering the number of inpainted steps from the back can mitigate the error concentration towards the end, leading to more accurate trajectory predictions.

This improvement aligns with the observation about the prediction error noted earlier in this section. Specifically, as the number of inpainted steps increases, the prediction error per step grows larger, particularly for the sweep-front approach. Therefore, providing additional inpainted steps where larger errors are observed can enhance the overall solution quality. Addressing these error-prone regions through additional inpainting makes the model's performance more robust, and the final trajectory predictions are more accurate.

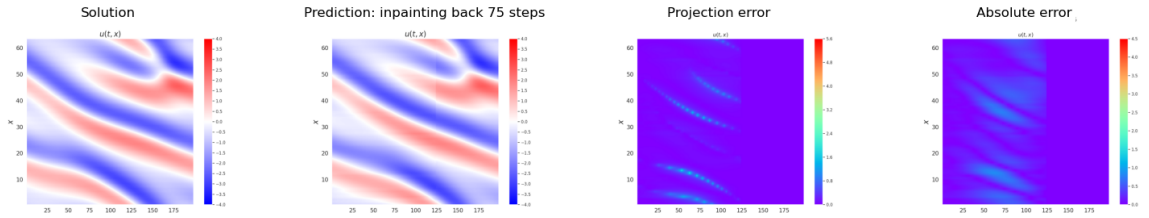

Figure 59: Sweep-back for 75 steps and first step provided.

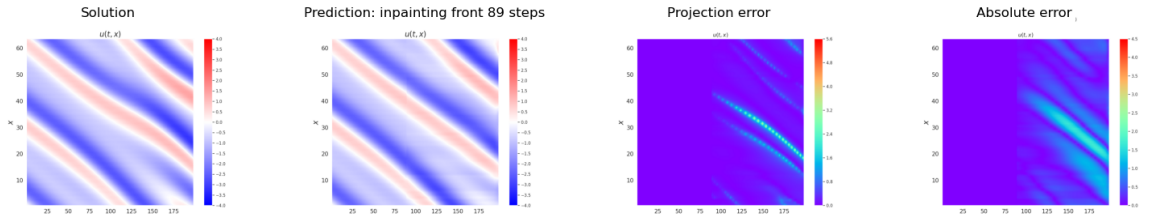

Figure 60: Sweep-front for 89 steps.

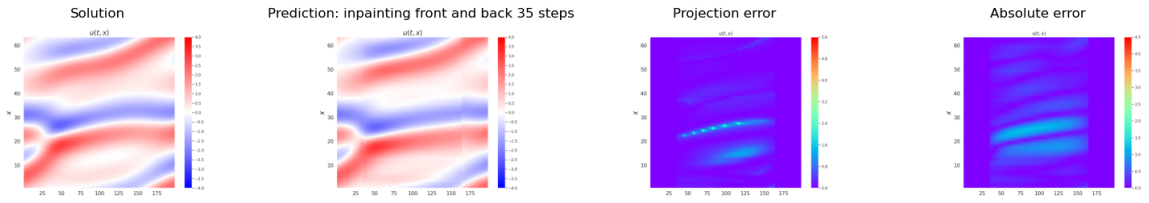

Figure 61: Sweep-both for 35 steps.

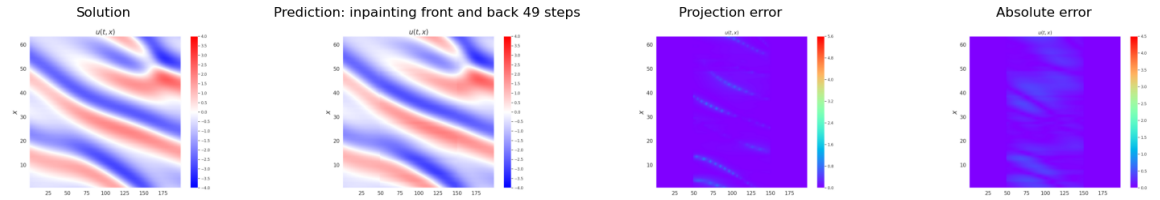

Figure 62: Sweep-both for 49 steps.

The prediction error for each sweep is shown in Figure 63 with the average prediction error on the right-hand side and the best results for the error on the right. The corresponding Table for this visualisation can be found in Appendix C. For the best solution, the error is not always concentrated at the end for the sweep-both and sweep-back methods. In fact, the smallest error for these approaches is found at the beginning of the horizon, suggesting that the model can achieve a more optimal solution earlier when considering the number of steps provided.

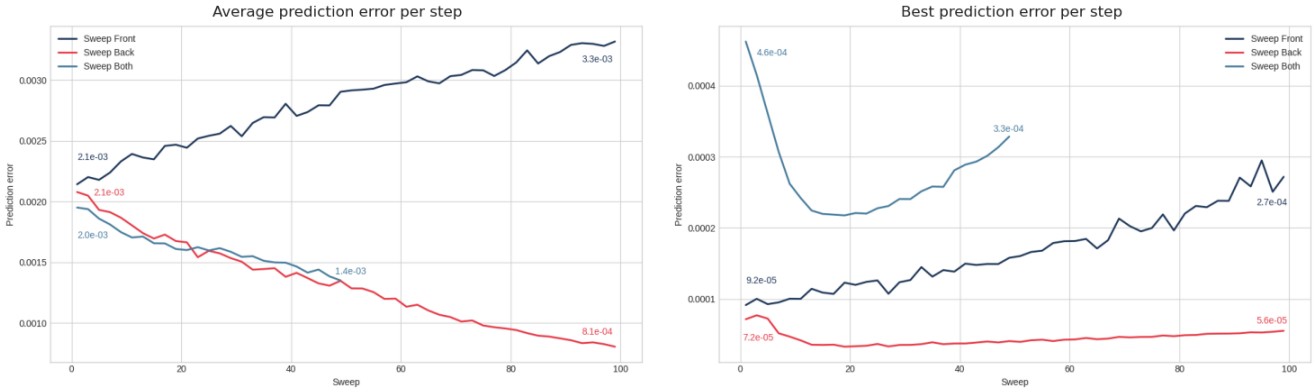

Figure 63: Average (left) and best (right) prediction error for sweep front, back and both for $\mu = 0.0$.

Interestingly, while the average error generally decreases with more steps inpainted, the trend for the sweep-front approach remains consistent, with the smallest error still occurring at the beginning. This observation ties back to increased error at the final steps; when these steps contribute a larger portion of the total trajectory, their higher error significantly impacts the overall error metric. Thus, the error grows more pronounced when the proportion of high-error final steps is larger. The average errors exhibit a steady decline for the sweep-back and sweep-both approaches as the number of provided steps increases. In contrast, the error for the sweep-front approach steadily increases.

The trend differs notably for the best error. For the sweep-both method, the error decreases until it reaches its minimum value at sweep 17 and then starts to rise again. The sweep-back

method initially has a steeper decrease in error, reaching its lowest value at sweep 21 before beginning to increase, but this increase is less prominent than the sweep-both method. On the other hand, the sweep-front method consistently shows an increase in the best error with more steps.

Interestingly, the best error for the sweep-front approach is smaller than the best error for the sweep-both method, which contrasts with the average results where the sweep-both approach performs better. This discrepancy highlights that while the sweep-back and sweep-both methods generally improve with more inpainted steps, the sweep-front method consistently shows higher average errors. However, it achieves better results in specific cases.

A.1.1 Comparison

The performance of the model was compared with the Fourier Neural Operator (FNO), which is a state-of-the-art method for generating solutions to PDEs [36]. In terms of prediction error, both the DDPM and FNO demonstrated comparable results, with prediction errors ranging from 0.1 to 0.7, see Figure 64. This suggests that DDPMs are capable of matching the strong baseline established by the FNO when it comes to accuracy in reproducing PDE solutions.

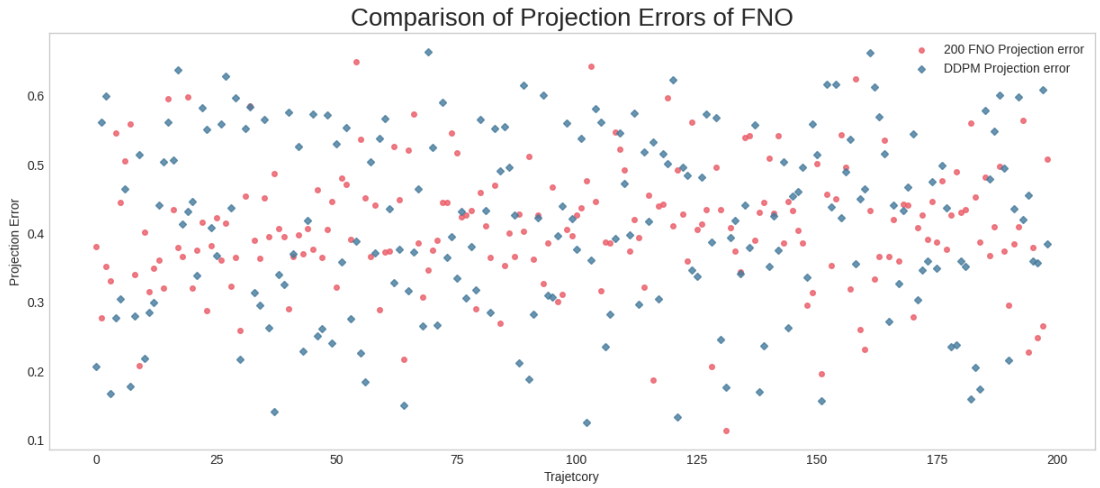

Figure 64: Prediction error comparison between FNO and DDPM.

However, further analysis revealed notable differences when evaluating the continuity metrics, which assesses how smoothly the generated solutions transition over time. The DDPM outperformed the FNO in this area, even without employing any additional inpainting. This advantage highlights the capability of DDPMs to generate solutions that are not only accurate but also maintain higher temporal coherence, resulting in more realistic simulations of dynamic systems.

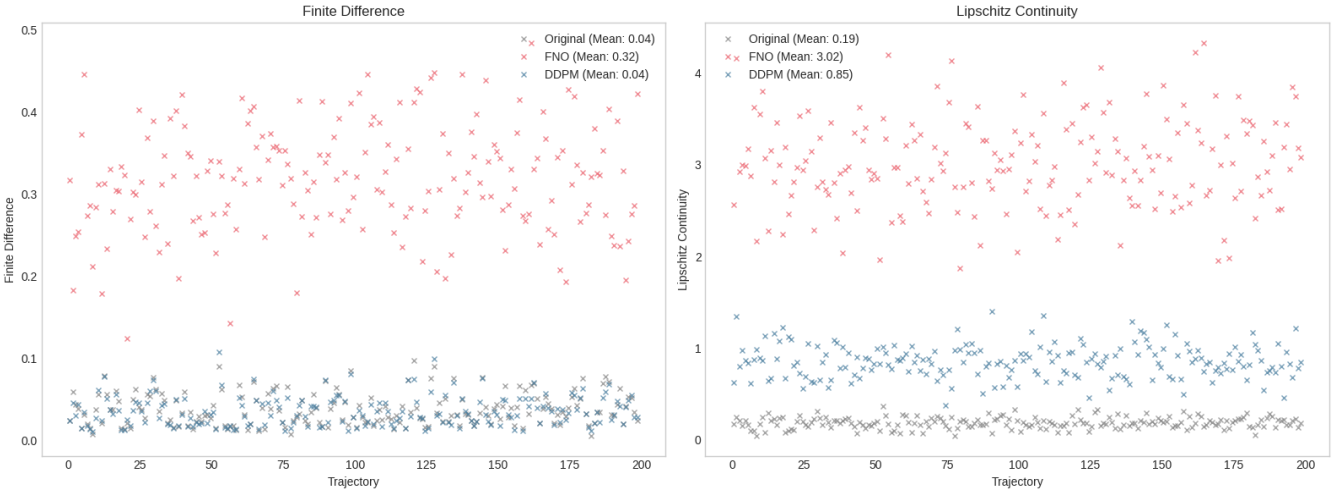

Figure 65: Average (left) and best (right) prediction error for sweep front, back and both for $\mu = 0.0$.

A.1.2 Conclusion - Single μ

This experiment evaluated the model's ability to generate and paint solutions for the KS equation with $\mu = 0.0$ and unseen initial condition. The analysis revealed that the model can capture the general behaviour of the KS equation, particularly when well-chosen configurations — such as the number of denoising steps and inpainting strategy - are applied. The sweep-back method consistently yielded the lowest prediction error and the analysis confirmed that providing more steps improves model accuracy. However, while the model can generate reasonably accurate solutions, the errors increase over time, especially towards the final steps of the trajectory. This suggests that although the model approximates the solution well in the short term, its predictive accuracy diminishes as the time horizon extends, requiring further refinement for long-term predictions and prediction at the end of the horizon.

Regarding generalisation to unseen values of the parameterised PDE, the model demonstrated the ability to handle unseen initial conditions reasonably well. It can generate coherent solutions even when the specific initial states have not been encountered during training. This suggests that DDPMs possess a degree of generalizability, but the accuracy of these predictions still decreases as the system evolves. Additionally, DDPMs offer a promising approach not only for accurate PDE solution generation but also for creating simulations with enhanced continuity, which is vital for applications requiring stable and realistic time-dependent behaviour.

B Different starting conditions

The comparison across the other four distinct starting conditions is presented in Figure 66. A consistent observation across all five instances is the increasing discrepancy between the minimal and maximal values of the projection error as the number of provided steps progresses. For some solutions, such as in row 2, these differences are quite pronounced, while for others, like in row 4, the discrepancy grows slower. Nonetheless, in all cases, the most significant difference between the minimal and maximal values of the projection error occurs at the final step.

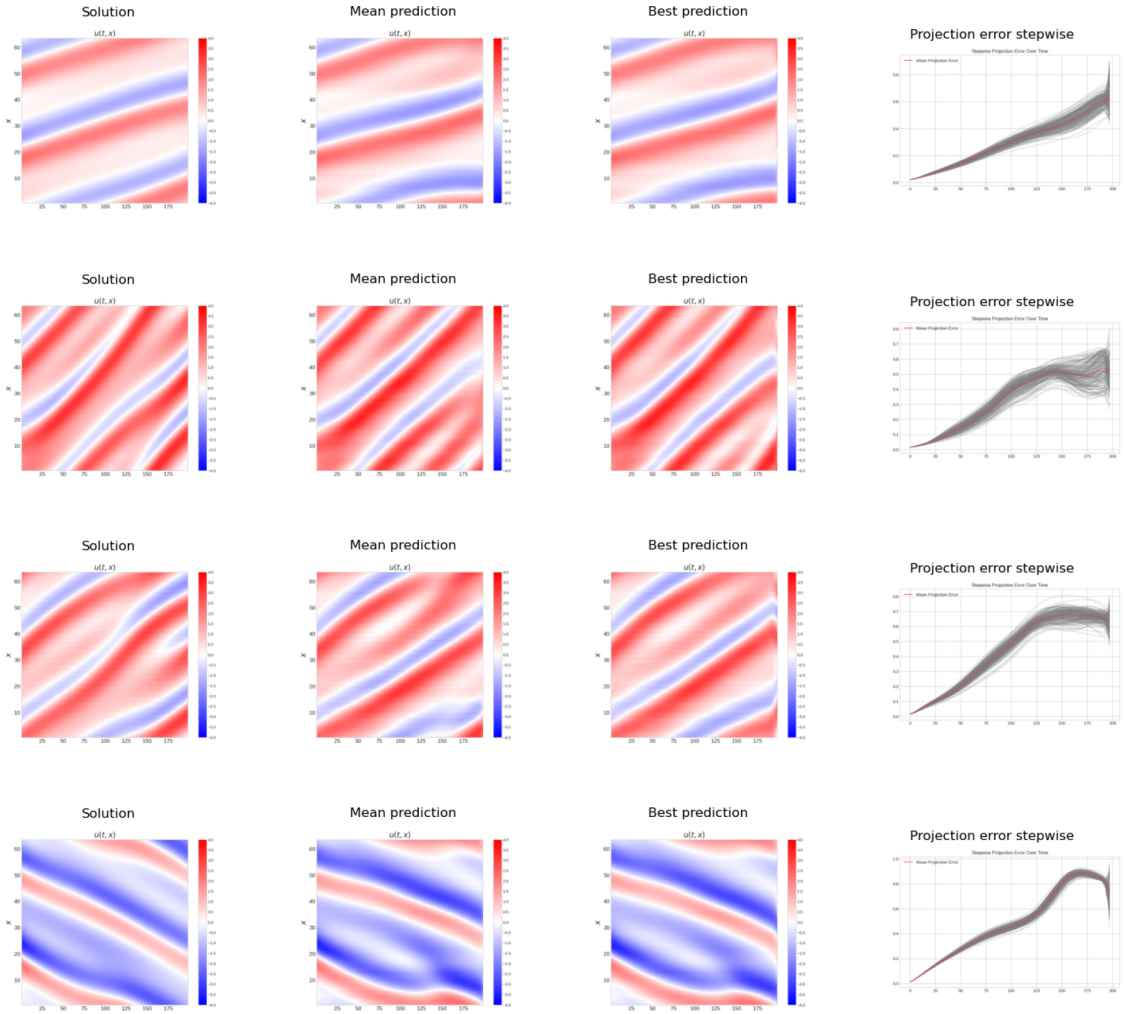

Figure 66: Comparison of actual solution with mean and best solution and stepwise projection error for $\mu = 0.0$ *for distinct starting conditions.*

C Single μ errors

Table 2: Projection error for sweep front, back and both for a selected number of provided steps, with the lowest errors marked red.

	Projection error					
sweep	front	front best	back	back best	both	both best
1	2.14e-03	9.17e-05	2.08e-03	7.18e-05	1.95e-03	4.62e-04
5	2.18e-03	9.30e-05	1.93e-03	7.27e-05	1.86e-03	3.61e-04
9	2.33e-03	1.01e-04	1.87e-03	4.74e-05	1.75e-03	2.62e-04
13	2.36e-03	1.15e-04	1.74e-03	3.60e-05	1.71e-03	2.24e-04
17	2.46e-03	1.07e-04	1.73e-03	3.61e-05	1.66e-03	2.19e-04
21	2.44e-03	1.20e-04	1.67e-03	3.39e-05	1.60e-03	2.21e-04
25	2.54e-03	1.26e-04	1.60e-03	3.71e-05	1.60e-03	2.28e-04
29	2.62e-03	1.24e-04	1.53e-03	3.57e-05	1.59e-03	2.41e-04
33	2.65e-03	1.45e-04	1.44e-03	3.68e-05	1.55e-03	2.51e-04
37	2.69e-03	1.41e-04	1.45e-03	3.68e-05	1.50e-03	2.58e-04
41	2.71e-03	1.50e-04	1.41e-03	3.78e-05	1.47e-03	2.89e-04
45	2.79e-03	1.49e-04	1.33e-03	4.05e-05	1.44e-03	3.01e-04
49	2.90e-03	1.58e-04	1.35e-03	4.11e-05	1.35e-03	3.28e-04
53	2.92e-03	1.66e-04	1.28e-03	4.23e-05		
57	2.96e-03	1.79e-04	1.20e-03	4.11e-05		
61	2.98e-03	1.82e-04	1.13e-03	4.35e-05		
65	2.99e-03	1.71e-04	1.10e-03	4.37e-05		
69	3.03e-03	2.13e-04	1.05e-03	4.70e-05		
73	3.08e-03	1.95e-04	1.02e-03	4.69e-05		
77	3.03e-03	2.19e-04	9.66e-04	4.90e-05		
81	3.15e-03	2.20e-04	9.43e-04	4.94e-05		
85	3.14e-03	2.29e-04	8.97e-04	5.14e-05		
89	3.23e-03	2.38e-04	8.74e-04	5.17e-05		
93	3.31e-03	2.58e-04	8.35e-04	5.34e-05		
97	3.28e-03	2.51e-04	8.27e-04	5.42e-05		

D Single μ latent representation

This appendix shows the preliminary results of an experiment with dimensionality reduction using SVD for the KS PDE with $N_x = 500$ and $L = 200$.The parameter μ was consistently set to 0.0, with variations only in the initial conditions. Figure 67 shows the latent representation for reduced dimensions 240, 100, 64 and 32, the reconstructed solutions and the absolute error between the reconstructions and true solutions. A similar pattern was observed in the SVD analysis of the multiple μ case (Figure 14), where many values close to zero emerged in higher dimensions, shown by the white regions.

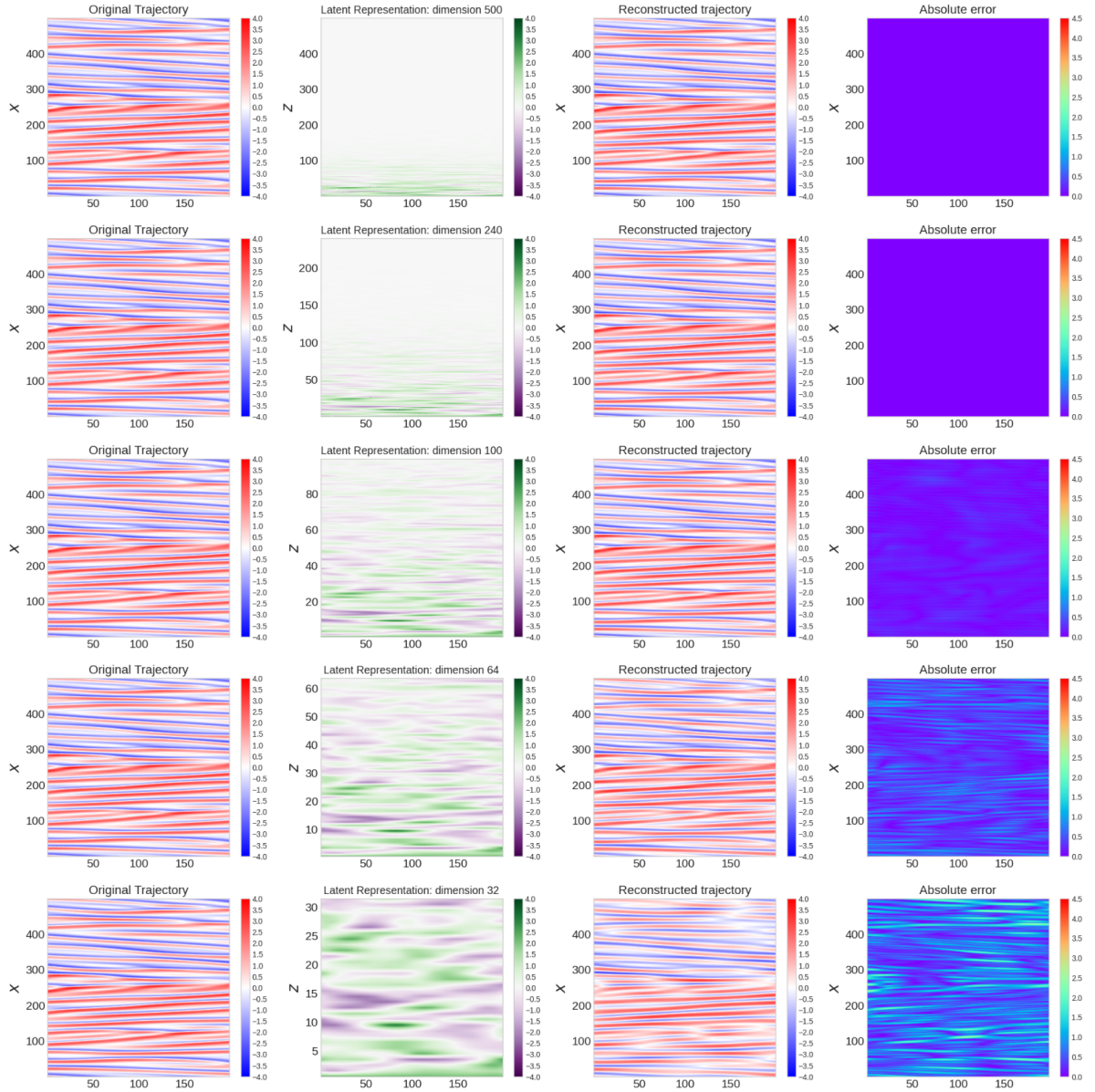

Figure 67: Latent representation and reconstruction for each of the examined latent dimensions (240, 100, 32 and 16) using SVD.

Figures 68 and 69 show example trajectories generated by the model from the data reduced to 64 and 32 observations using the SVD. In higher dimensions (240 and 100), the model predominantly produced noise, likely due to the high proportion of near-zero values, resulting in a sharply peaked data distribution—a problem similar to the one observed in the latent representation experiment for multiple μ experiment, see Figure 39. Notably, in Figure 68, a sweep front method appears to contribute to generating solutions with reduced error noise. These preliminary findings point to the potential for using latent representations in PDE prediction, underscoring the importance of a comprehensive evaluation of dimensionality reduction techniques to achieve more uniform data distributions.

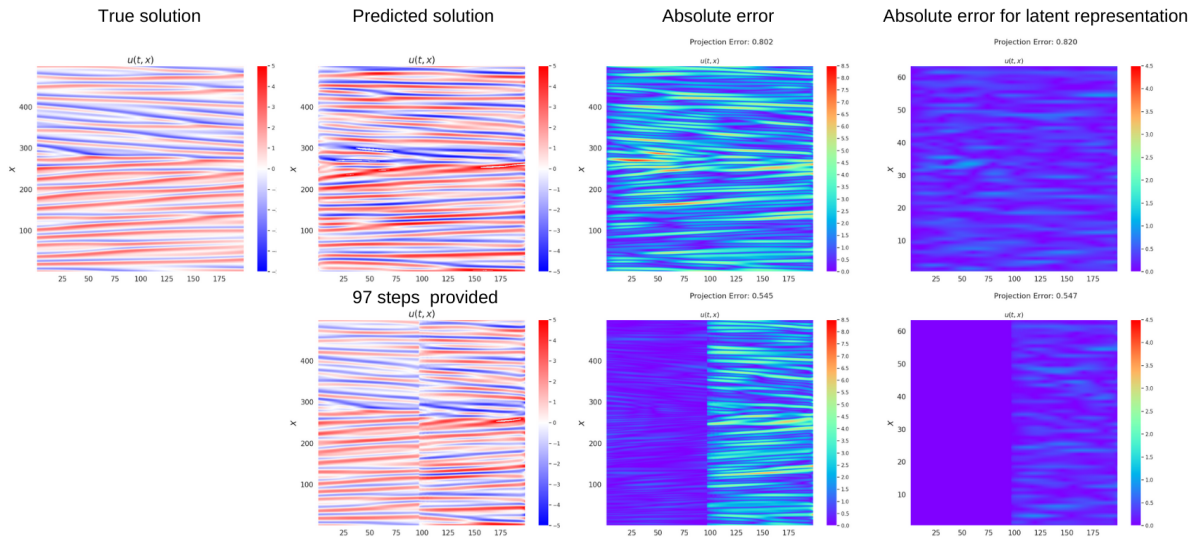

Figure 68: Example predicted solution with no provided steps (upper) and 97 provided steps (lower) for data reduced to 64 dimensions.

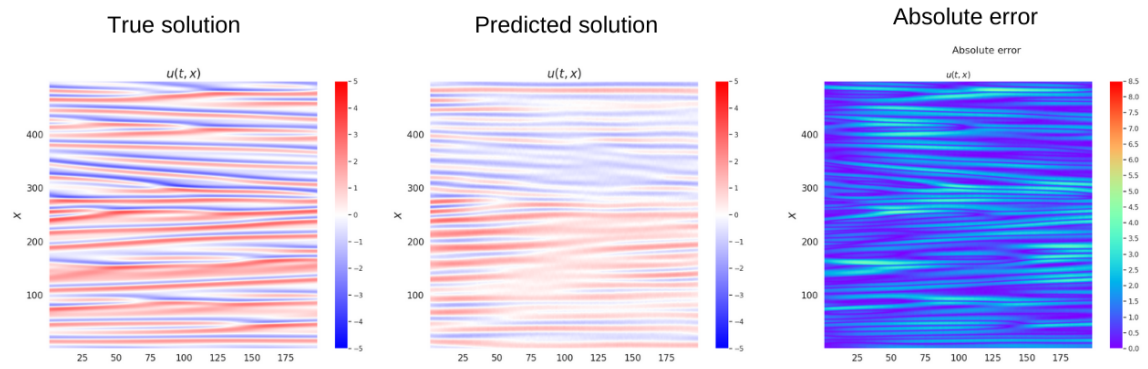

Figure 69: Example predicted solution for data reduced to 32 dimensions.

E FNO

The Fourier Neural Operator (FNO), as proposed by Kovachki et al. [36], is limited to predicting one trajectory step at a time. Two approaches were analyzed to determine the better-performing version of the FNO to address this. The first approach involved training separate FNOs for each trajectory step, resulting in 200 distinct FNOs, which were then combined to generate a complete trajectory prediction. The second approach trained a single FNO using pairs of steps $(s_0, s_1), (s_1, s_2), ..., (s_{T-1}, s_T)$ and the trajectory were generated iteratively by using the last predicted step to generate the next one.

The second approach produced smoother trajectories but showed a bias toward generating specific trajectories, particularly for the model trained exclusively on the μ value of 0.0. This bias persisted despite experimenting with varying training data, early stopping, and different model parameters, as shown in Figure 70.

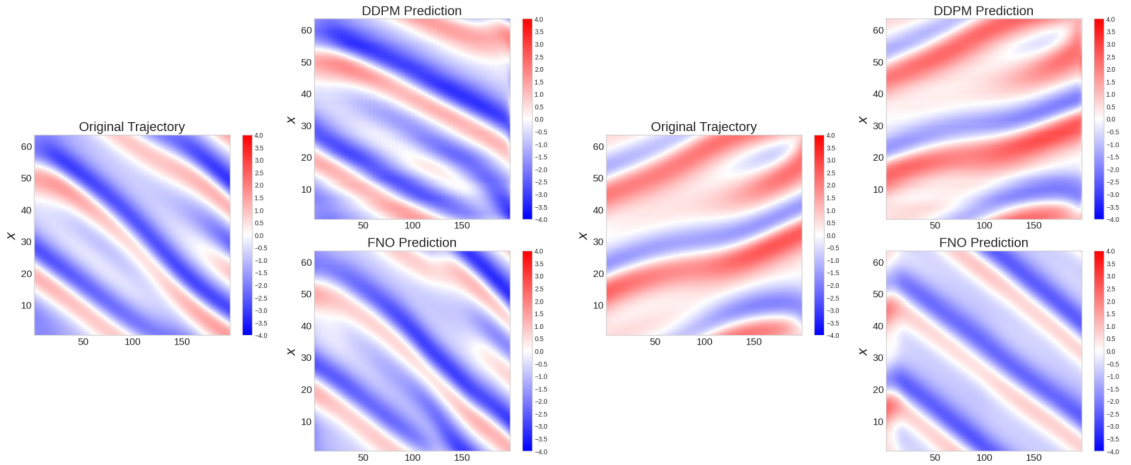

Figure 70: Comparison of trajectories generated by FNO and DDPM indicating a bias towards producing a specific solution from FNO.

While the bias was less pronounced when training the FNO on multiple μ values, the projection error still indicated that training 200 separate FNOs was the superior option under this metric. This discrepancy is illustrated in Figure **??**, which compares the projection error per trajectory between the 200 FNOs and the single FNO, alongside the mean error per μ. The mean projection error for the 200 FNOs was 0.089, while for the single FNO, it was 0.469.

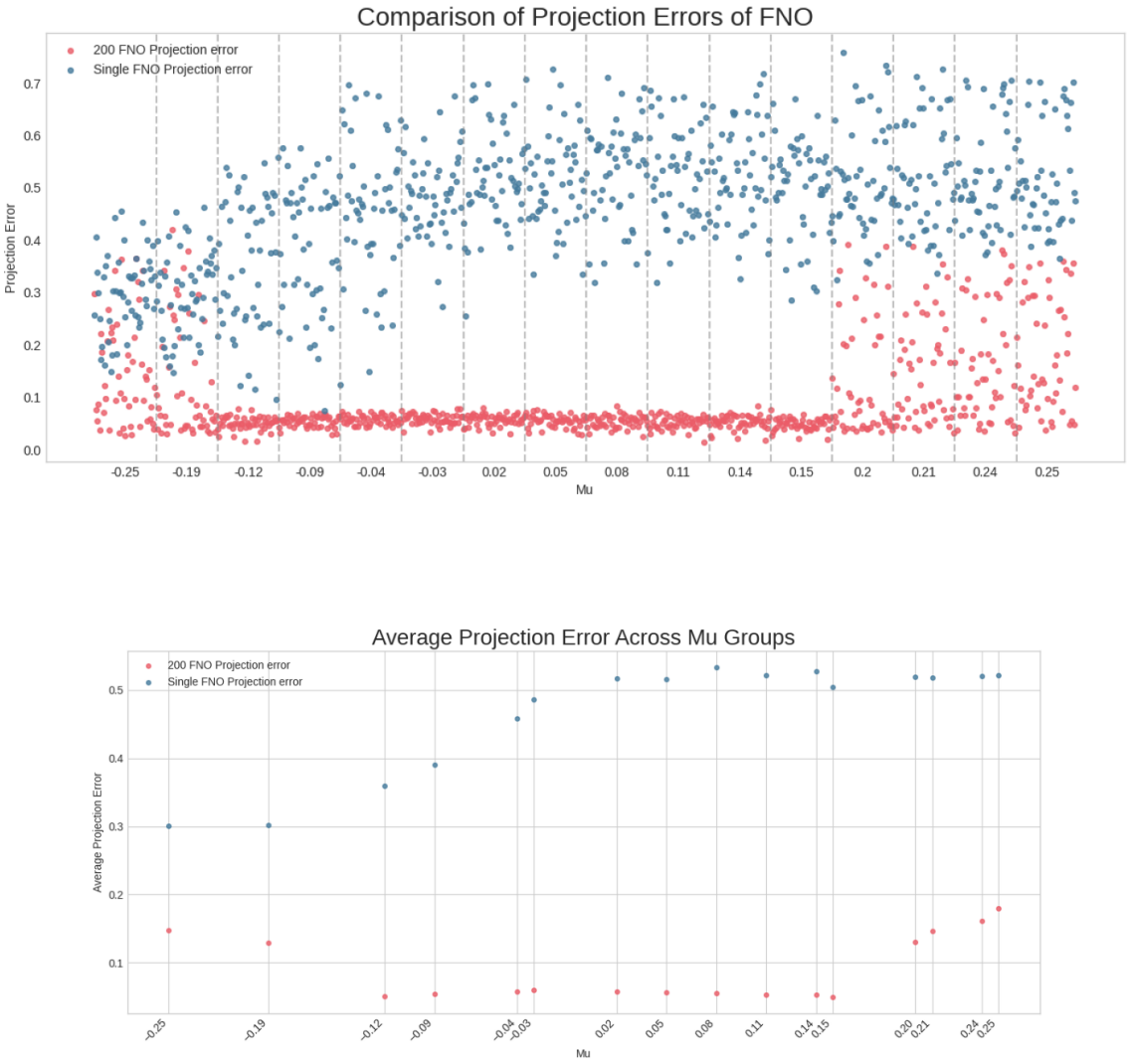

Figure 71: Projection error comparison between 200 FNO and single FNO.

F Multiple μ - Prediction errors

Figure 72 shows the prediction error for sweep-both. The trend is similar to the sweep-back shown in Figure 5. For all the μ values besides $\mu = -0.25$, the smallest terror once again for the larger number of provided steps. The smallest error is observed for $\mu = 0.24$ with value of $7.72 * 10^{-4}$ followed by $8.15 * 10^{-3}$ for $\mu = 0.2$.

Again, the exclusive μ values start with lower prediction error than inclusive ones, but with more steps provided, this trend is no longer evident. Additionally, following the general trend for most values, the error decreases with the number of provided steps increasing. The exception is the $\mu = -0.25$

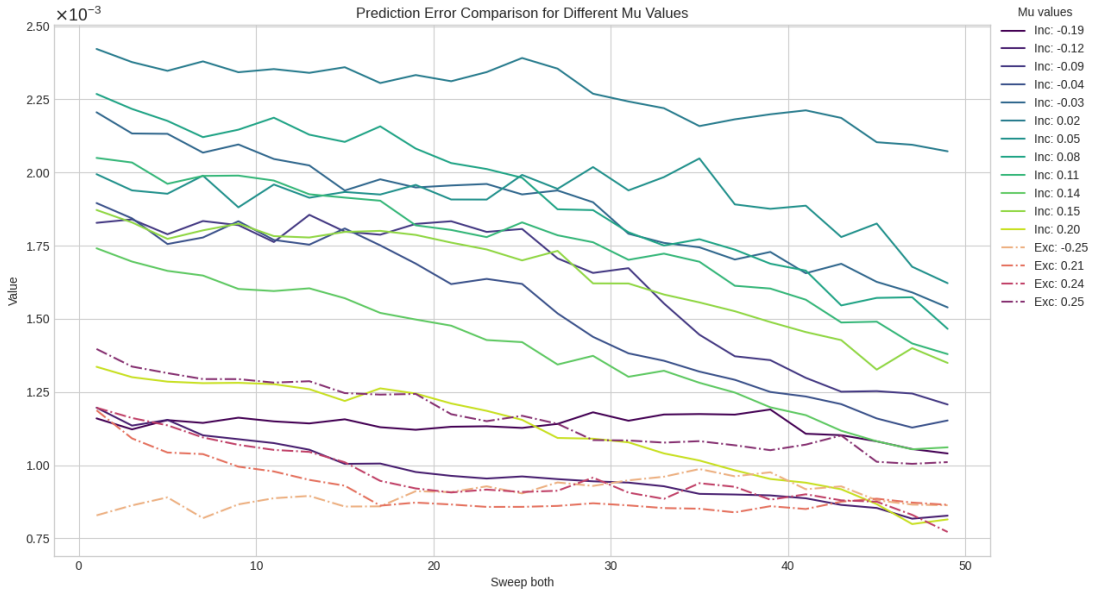

Figure 72: Prediction error for sweep-both for various mu.

The detailed results for prediction errors for various μ are presented in the tables below:

sweep	-0.25	-0.19	-0.12	-0.09	-0.04	-0.03	0.02	0.05	0.08	0.11	0.14	0.15	0.20	0.21	0.24	0.25
1	8.38e-04	1.16e-03	1.19e-03	1.88e-03	1.94e-03	2.20e-03	2.50e-03	2.10e-03	2.28e-03	2.16e-03	1.81e-03	1.91e-03	1.34e-03	1.20e-03	1.22e-03	1.39e-03
7	9.35e-04	1.28e-03	1.38e-03	2.01e-03	2.07e-03	2.33e-03	2.55e-03	2.24e-03	2.37e-03	2.19e-03	1.84e-03	1.98e-03	1.38e-03	1.22e-03	1.24e-03	1.44e-03
13	1.04e-03	1.35e-03	1.45e-03	2.11e-03	2.23e-03	2.39e-03	2.65e-03	2.23e-03	2.52e-03	2.32e-03	1.99e-03	2.06e-03	1.42e-03	1.21e-03	1.27e-03	1.51e-03
19	1.05e-03	1.33e-03	1.47e-03	2.20e-03	2.26e-03	2.52e-03	2.75e-03	2.35e-03	2.58e-03	2.34e-03	1.98e-03	2.15e-03	1.45e-03	1.22e-03	1.30e-03	1.53e-03
25	1.05e-03	1.35e-03	1.48e-03	2.21e-03	2.31e-03	2.49e-03	2.83e-03	2.44e-03	2.52e-03	2.35e-03	2.04e-03	2.15e-03	1.47e-03	1.22e-03	1.29e-03	1.54e-03
31	1.00e-03	1.29e-03	1.38e-03	2.28e-03	2.33e-03	2.66e-03	2.90e-03	2.49e-03	2.60e-03	2.45e-03	2.05e-03	2.11e-03	1.50e-03	1.25e-03	1.32e-03	1.56e-03
37	9.62e-04	1.26e-03	1.26e-03	2.23e-03	2.41e-03	2.68e-03	2.98e-03	2.54e-03	2.61e-03	2.49e-03	2.06e-03	2.22e-03	1.48e-03	1.26e-03	1.28e-03	1.59e-03
43	9.97e-04	1.30e-03	1.24e-03	2.27e-03	2.43e-03	2.67e-03	3.08e-03	2.64e-03	2.71e-03	2.61e-03	2.04e-03	2.17e-03	1.46e-03	1.23e-03	1.28e-03	1.54e-03
49	9.85e-04	1.29e-03	1.22e-03	2.13e-03	2.47e-03	2.75e-03	3.10e-03	2.76e-03	2.79e-03	2.54e-03	2.07e-03	2.15e-03	1.43e-03	1.22e-03	1.27e-03	1.50e-03
55	9.85e-04	1.28e-03	1.18e-03	2.02e-03	2.44e-03	2.75e-03	3.08e-03	2.83e-03	2.66e-03	2.50e-03	2.04e-03	2.10e-03	1.33e-03	1.23e-03	1.27e-03	1.47e-03
61	1.03e-03	1.30e-03	1.20e-03	2.02e-03	2.34e-03	2.65e-03	3.18e-03	2.98e-03	2.62e-03	2.47e-03	2.04e-03	1.96e-03	1.36e-03	1.23e-03	1.24e-03	1.43e-03
67	1.01e-03	1.29e-03	1.22e-03	1.98e-03	2.37e-03	2.81e-03	3.14e-03	3.08e-03	2.64e-03	2.47e-03	2.03e-03	1.86e-03	1.36e-03	1.27e-03	1.23e-03	1.44e-03
73	1.03e-03	1.30e-03	1.16e-03	2.00e-03	2.34e-03	2.75e-03	3.16e-03	3.06e-03	2.62e-03	2.49e-03	2.05e-03	1.84e-03	1.36e-03	1.26e-03	1.22e-03	1.42e-03
79	1.02e-03	1.25e-03	1.22e-03	2.08e-03	2.33e-03	2.69e-03	3.21e-03	3.06e-03	2.46e-03	2.58e-03	2.01e-03	1.78e-03	1.35e-03	1.23e-03	1.35e-03	1.38e-03
85	1.02e-03	1.27e-03	1.23e-03	1.86e-03	2.26e-03	2.74e-03	3.21e-03	3.11e-03	2.47e-03	2.60e-03	1.97e-03	1.75e-03	1.43e-03	1.20e-03	1.37e-03	1.34e-03
91	1.02e-03	1.25e-03	1.23e-03	1.71e-03	2.11e-03	2.63e-03	3.23e-03	3.16e-03	2.41e-03	2.55e-03	1.98e-03	1.81e-03	1.37e-03	1.22e-03	1.34e-03	1.38e-03
97	1.06e-03	1.26e-03	1.34e-03	1.79e-03	2.23e-03	2.59e-03	3.32e-03	3.28e-03	2.38e-03	2.60e-03	1.91e-03	1.87e-03	1.51e-03	1.19e-03	1.42e-03	1.38e-03

Table 3: Projection error for best prediction for sweep-front for various μ, with the lowest error marked red.

sweep	-0.25	-0.19	-0.12	-0.09	-0.04	-0.03	0.02	0.05	0.08	0.11	0.14	0.15	0.20	0.21	0.24	0.25
1	8.20e-04	1.13e-03	1.24e-03	1.82e-03	1.93e-03	2.09e-03	2.40e-03	2.02e-03	2.15e-03	2.08e-03	1.75e-03	1.85e-03	1.32e-03	1.15e-03	1.19e-03	1.37e-03
7	7.69e-04	1.07e-03	1.03e-03	1.63e-03	1.67e-03	1.99e-03	2.20e-03	1.75e-03	2.04e-03	1.91e-03	1.57e-03	1.71e-03	1.25e-03	1.02e-03	1.09e-03	1.24e-03
13	7.71e-04	1.03e-03	9.58e-04	1.50e-03	1.59e-03	1.86e-03	2.18e-03	1.68e-03	1.94e-03	1.76e-03	1.47e-03	1.64e-03	1.20e-03	9.33e-04	1.01e-03	1.19e-03
19	7.51e-04	1.01e-03	9.24e-04	1.48e-03	1.52e-03	1.81e-03	2.12e-03	1.72e-03	1.86e-03	1.72e-03	1.35e-03	1.60e-03	1.15e-03	9.04e-04	9.60e-04	1.16e-03
25	8.07e-04	9.95e-04	9.20e-04	1.45e-03	1.44e-03	1.72e-03	2.06e-03	1.65e-03	1.81e-03	1.64e-03	1.33e-03	1.54e-03	1.13e-03	8.44e-04	8.78e-04	1.12e-03
31	8.54e-04	9.98e-04	8.96e-04	1.53e-03	1.39e-03	1.69e-03	2.03e-03	1.61e-03	1.66e-03	1.51e-03	1.28e-03	1.48e-03	1.04e-03	8.22e-04	8.20e-04	1.06e-03
37	8.11e-04	9.98e-04	8.79e-04	1.42e-03	1.33e-03	1.63e-03	1.93e-03	1.60e-03	1.63e-03	1.47e-03	1.20e-03	1.44e-03	9.85e-04	7.47e-04	7.85e-04	9.78e-04
43	8.20e-04	1.00e-03	9.08e-04	1.44e-03	1.27e-03	1.62e-03	1.89e-03	1.56e-03	1.57e-03	1.50e-03	1.11e-03	1.37e-03	9.35e-04	6.71e-04	7.33e-04	9.39e-04
49	8.48e-04	9.59e-04	8.94e-04	1.21e-03	1.21e-03	1.56e-03	1.75e-03	1.48e-03	1.42e-03	1.40e-03	1.02e-03	1.27e-03	9.54e-04	6.60e-04	7.14e-04	9.13e-04
55	8.49e-04	9.62e-04	8.58e-04	1.19e-03	1.16e-03	1.40e-03	1.61e-03	1.38e-03	1.32e-03	1.23e-03	8.65e-04	1.21e-03	8.85e-04	6.53e-04	7.05e-04	8.52e-04
61	8.84e-04	9.22e-04	8.75e-04	1.10e-03	1.12e-03	1.37e-03	1.49e-03	1.32e-03	1.21e-03	1.12e-03	8.38e-04	1.17e-03	8.36e-04	6.71e-04	7.04e-04	8.58e-04
67	8.66e-04	9.08e-04	8.64e-04	1.08e-03	1.06e-03	1.34e-03	1.36e-03	1.24e-03	1.10e-03	9.76e-04	7.55e-04	1.08e-03	8.43e-04	6.31e-04	7.24e-04	8.47e-04
73	8.71e-04	8.83e-04	8.56e-04	1.00e-03	1.00e-03	1.26e-03	1.22e-03	1.14e-03	1.05e-03	8.59e-04	7.06e-04	1.01e-03	8.10e-04	6.56e-04	6.80e-04	7.99e-04
79	8.60e-04	8.73e-04	8.52e-04	1.01e-03	9.28e-04	1.25e-03	1.08e-03	1.06e-03	1.01e-03	8.06e-04	6.46e-04	8.77e-04	7.59e-04	6.14e-04	7.06e-04	7.81e-04
85	8.52e-04	8.42e-04	8.65e-04	8.86e-04	8.84e-04	1.20e-03	9.81e-04	9.51e-04	9.51e-04	7.59e-04	6.13e-04	8.31e-04	7.66e-04	6.06e-04	6.42e-04	7.46e-04
91	8.38e-04	8.33e-04	8.40e-04	8.80e-04	8.39e-04	1.17e-03	8.95e-04	8.82e-04	9.10e-04	7.27e-04	6.02e-04	7.55e-04	7.26e-04	5.92e-04	7.04e-04	7.09e-04
97	8.21e-04	8.12e-04	8.56e-04	8.56e-04	8.26e-04	1.09e-03	8.44e-04	8.44e-04	8.53e-04	6.89e-04	5.83e-04	6.69e-04	7.09e-04	5.90e-04	7.07e-04	7.03e-04

Table 4: Projection error for best prediction for sweep-back for various μ, with the lowest error marked red.

sweep	-0.25	-0.19	-0.12	-0.09	-0.04	-0.03	0.02	0.05	0.08	0.11	0.14	0.15	0.20	0.21	0.24	0.25
1	8.20e-04	1.13e-03	1.24e-03	1.82e-03	1.93e-03	2.09e-03	2.40e-03	2.02e-03	2.15e-03	2.08e-03	1.75e-03	1.85e-03	1.32e-03	1.15e-03	1.19e-03	1.37e-03
7	7.69e-04	1.07e-03	1.03e-03	1.63e-03	1.67e-03	1.99e-03	2.20e-03	1.75e-03	2.04e-03	1.91e-03	1.57e-03	1.71e-03	1.25e-03	1.02e-03	1.09e-03	1.24e-03
13	7.71e-04	1.03e-03	9.58e-04	1.50e-03	1.59e-03	1.86e-03	2.18e-03	1.68e-03	1.94e-03	1.76e-03	1.47e-03	1.64e-03	1.20e-03	9.33e-04	1.01e-03	1.19e-03
19	7.51e-04	1.01e-03	9.24e-04	1.48e-03	1.52e-03	1.81e-03	2.12e-03	1.72e-03	1.86e-03	1.72e-03	1.35e-03	1.60e-03	1.15e-03	9.04e-04	9.60e-04	1.16e-03
25	8.07e-04	9.95e-04	9.20e-04	1.45e-03	1.44e-03	1.72e-03	2.06e-03	1.65e-03	1.81e-03	1.64e-03	1.33e-03	1.54e-03	1.13e-03	8.44e-04	8.78e-04	1.12e-03
31	8.54e-04	9.98e-04	8.96e-04	1.53e-03	1.39e-03	1.69e-03	2.03e-03	1.61e-03	1.66e-03	1.51e-03	1.28e-03	1.48e-03	1.04e-03	8.22e-04	8.20e-04	1.06e-03
37	8.11e-04	9.98e-04	8.79e-04	1.42e-03	1.33e-03	1.63e-03	1.93e-03	1.60e-03	1.63e-03	1.47e-03	1.20e-03	1.44e-03	9.85e-04	7.47e-04	7.85e-04	9.78e-04
43	8.20e-04	1.00e-03	9.08e-04	1.44e-03	1.27e-03	1.62e-03	1.89e-03	1.56e-03	1.57e-03	1.50e-03	1.11e-03	1.37e-03	9.35e-04	6.71e-04	7.33e-04	9.39e-04
49	8.48e-04	9.59e-04	8.94e-04	1.21e-03	1.21e-03	1.56e-03	1.75e-03	1.48e-03	1.42e-03	1.40e-03	1.02e-03	1.27e-03	9.54e-04	6.60e-04	7.14e-04	9.13e-04
55	8.49e-04	9.62e-04	8.58e-04	1.19e-03	1.16e-03	1.40e-03	1.61e-03	1.38e-03	1.32e-03	1.23e-03	8.65e-04	1.21e-03	8.85e-04	6.53e-04	7.05e-04	8.52e-04
61	8.84e-04	9.22e-04	8.75e-04	1.10e-03	1.12e-03	1.37e-03	1.49e-03	1.32e-03	1.21e-03	1.12e-03	8.38e-04	1.17e-03	8.36e-04	6.71e-04	7.04e-04	8.58e-04
67	8.66e-04	9.08e-04	8.64e-04	1.08e-03	1.06e-03	1.34e-03	1.36e-03	1.24e-03	1.10e-03	9.76e-04	7.55e-04	1.08e-03	8.43e-04	6.31e-04	7.24e-04	8.47e-04
73	8.71e-04	8.83e-04	8.56e-04	1.00e-03	1.00e-03	1.26e-03	1.22e-03	1.14e-03	1.05e-03	8.59e-04	7.06e-04	1.01e-03	8.10e-04	6.56e-04	6.80e-04	7.99e-04
79	8.60e-04	8.73e-04	8.52e-04	1.01e-03	9.28e-04	1.25e-03	1.08e-03	1.06e-03	1.01e-03	8.06e-04	6.46e-04	8.77e-04	7.59e-04	6.14e-04	7.06e-04	7.81e-04
85	8.52e-04	8.42e-04	8.65e-04	8.86e-04	8.84e-04	1.20e-03	9.81e-04	9.51e-04	9.51e-04	7.59e-04	6.13e-04	8.31e-04	7.66e-04	6.06e-04	6.42e-04	7.46e-04
91	8.38e-04	8.33e-04	8.40e-04	8.80e-04	8.39e-04	1.17e-03	8.95e-04	8.82e-04	9.10e-04	7.27e-04	6.02e-04	7.55e-04	7.26e-04	5.92e-04	7.04e-04	7.09e-04
97	8.21e-04	8.12e-04	8.56e-04	8.56e-04	8.26e-04	1.09e-03	8.44e-04	8.44e-04	8.53e-04	6.89e-04	5.83e-04	6.69e-04	7.09e-04	5.90e-04	7.07e-04	7.03e-04

Table 5: Projection error for best prediction for sweep-back for various μ, with the lowest error marked red.