A Multi-Fidelity approach to Deep Kernel Learning of dynamical systems from high-dimensional data sources

---

Author: Federico Lancellotti

Professor: Prof. Luca Formaggia
Teaching assistants: Dr. Alberto Artoni, Dr. Beatrice Crippa
Advisor: Prof. Andrea Manzoni
Co-advisors: Dr. Nicolò Botteghi

Table of contents

Introduction

Introduction

When modelling dynamical systems, data is often **high-dimensional**.

Dynamical systems can be described by a **small number of state variables**.

High-fidelity data is usually well-correlated with their **low-fidelity** counterpart.

Introduction

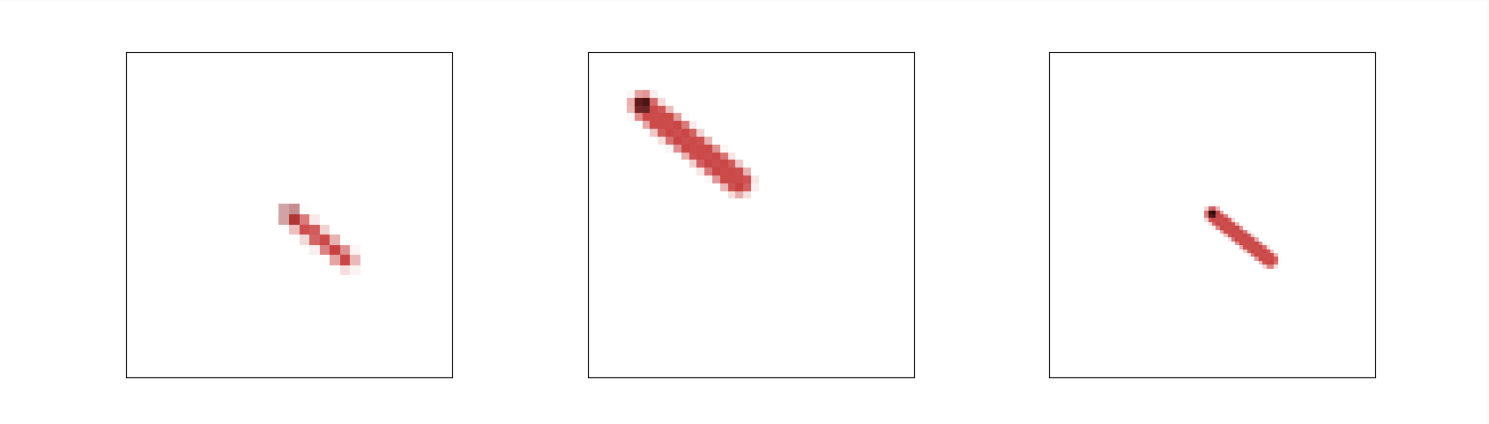
We present a Multi-Fidelity generalization of the framework presented by Botteghi et al., considering:

- both **high** and **low fidelity** measurements;
- a larger variety of test cases.

The goal of our model is to **reduce the dimensionality** of the input data, **learn the latent state and dynamics** of the system, and **quantify the uncertainty**.

Introduction

In particular, we will consider **RGB videos** of the dynamical systems.



(a) Fidelity level 0 **(b)** Fidelity level 1 **(c)** Fidelity level 2

Preliminaries

Gaussian Process Regression

Definition

Gaussian Process Regression (GPR) is a non-parametric regression technique that models the prior knowledge about a computational model $G(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, with a Gaussian process $Z(\mathbf{x})$, with mean function $m(\mathbf{x})$ and kernel function $k(\mathbf{x}, \mathbf{x}')$.

Consider the training data inputs $\mathcal{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}), \mathbf{x}^{(i)} \in \mathbb{R}^n$, and the corresponding training targets $\mathcal{Y} = (G(\mathbf{x}^{(1)}), G(\mathbf{x}^{(2)}), ..., G(\mathbf{x}^{(N)}))$.

The predictive distribution evaluated at N^* test data points $\mathcal{X}^*$ is

$$[Z(\mathcal{X}^*)|Z(\mathcal{X}) = \mathcal{Y}, \sigma^2, \boldsymbol{\theta}] \sim \mathcal{N}(\mathbb{E}[Z(\mathcal{X}^*)], C(Z(\mathcal{X}^*)))$$

Drawback

Fitting the model is expensive for high-dimensional and large datasets.

Deep Kernel Learning

Embed a deep NN into the kernel function:

$$k(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) \rightarrow k(g(\mathbf{x}, \mathbf{w}), g(\mathbf{x}', \mathbf{w}); \boldsymbol{\theta}, \mathbf{w}) \ ,$$

where $g(\mathbf{x}, \mathbf{w})$ is a **non-linear mapping** given by a deep architecture, parametrised by weights $\mathbf{w}$.

Stochastic Variational Deep Kernel Learning

We approximate the posterior distribution with the best fitting Gaussian to a set of inducing data points sampled from the posterior.

Multi-Fidelity methods

The core idea is to leverage

- a large amount of **cheap low-fidelity** (LF) data,
- a limited number of **high-fidelity** (HF) samples, from detailed simulations or accurate sensors,

and **exploit their correlation**.

Linear Model of Coregionalisation

Each level of solution $Z_l(\mathbf{x})$ can be written as a **linear combination of** L **independent Gaussian processes** $u_i \sim \mathcal{GP}(0, k_i(\cdot,\cdot))$:

$$Z_l(\mathbf{x}) = \sum_{i=0}^{L-1} a_{l,i} u_i(\mathbf{x}), \quad l = 0, 1, ..., L-1,$$

Levina-Bickel algorithm

Consider a MLE of the dimension ID from i.i.d. observations $\mathbf{z}_1, ..., \mathbf{z}_n$ in $\mathbb{R}^p$. The observations represent an embedding of a lower-dimensional sample, i.e. $\mathbf{z}_i = g(\mathbf{s}_i)$, where $\mathbf{s}_i$ are sampled from an unknown smooth density f on $\mathbb{R}^{ID}$.

The **global** ID **estimator** is calculated as

$$ID_{L-B} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{k-2} \sum_{j=1}^{k-1} \log \frac{T_k(\mathbf{z_i})}{T_j(\mathbf{z_i})} \quad ,$$

where $T_k(\mathbf{z})$ is the Euclidean distance between $\mathbf{z}$ and the its k-th nearest neighbor in $\mathbf{z}_1, ..., \mathbf{z}_n$.

Methods

Framework

Consider the following nonlinear dynamical system:

$$\frac{d}{dt}\mathbf{s}(t) = \mathcal{F}(\mathbf{s}(t)), \quad \mathbf{s}(t_0) = \mathbf{s}_0, \quad t \in [t_0, t_f]$$

- $\mathbf{s}(t) \in \mathcal{S} \subset \mathbb{R}^d$ is the state vector at time t,
- $\mathcal{F} : \mathcal{S} \to \mathcal{S}$ is a nonlinear function determining the evolution of the system given the current state $\mathbf{s}(t)$

Goal

Identify a meaningful representation of the hidden states and a surrogate model for $\mathcal{F}$, **exploiting all data sources, of any fidelity**.

Framework

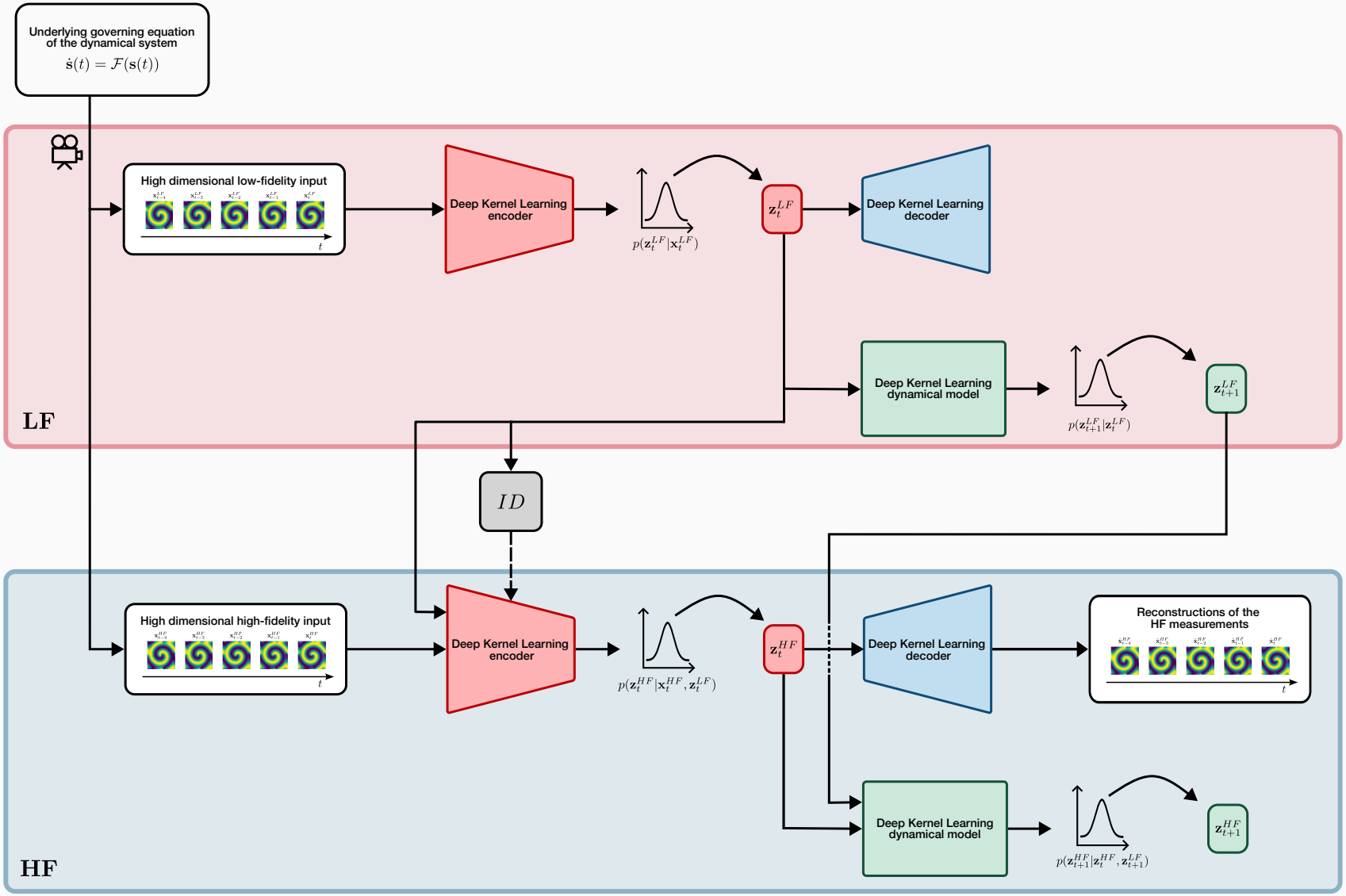
Consider L sets of N_l d_l-dimensional measurements of such system $\mathbf{X}_l = (\mathbf{x}_l^1, ..., \mathbf{x}_l^{N_l}) \in \mathcal{X}_l, \mathbf{x}_l^i \in \mathbb{R}^{d_l}$, with $d_l \gg 1$ and l being the fidelity level, $0 \leq l \leq L$.

We train a sub-model Z_l on the set of measurements $\mathbf{X}_l$ and leverage the knowledge on the system acquired by the previous less accurate models Z_i, $i < l$.

Each sub-model is composed of:

- **a SVDKL Autoencoder**;
- **a SVDKL dynamical model**.

Scheme of the model



Learning the hidden state

We introduce a **Stochastic Variational DKL encoder** $E_l : \mathcal{X}_l \to \mathcal{L}_l$.

A latent state sample can be obtained as:

$$z_{i,l}^{(t)} = Z_i^{E_l}(\mathbf{x}_l^{(t)}) + \varepsilon_{E_l}\,, \quad \varepsilon_{E_l} \sim \mathcal{N}(0, \sigma_{E_l}^2)$$

$$Z_{l,i}^{E_l}(\mathbf{x}_l^{(t)}) \sim \mathcal{GP}(\mu(g_{E_l}(\mathbf{x}_l^{(t)}; \mathbf{w}_{E_l})), k(g_{E_l}(\mathbf{x}_l^{(t)}; \mathbf{w}_{E_l}), g_{E_l}(\mathbf{x}_l^{(t')}; \mathbf{w}_{E_l}); \boldsymbol{\theta}_{E_l,i}))\,,$$
$$1 \leq i \leq |\mathbf{z}_l|$$

The parameters $(\mathbf{w}_{E_l}, \boldsymbol{\theta}_{E_l}, \sigma_{E_l}^2)$ are optimised with a **decoder** $D_l : \mathcal{L}_l \times \mathcal{L}_{l-1} \to \mathcal{X}_l$ that reconstructs the measurements given

$$\tilde{\mathbf{z}}_l = \mathbf{z}_l + \rho \mathbf{z}_{l-1}\,.$$

Learning the latent dynamics

We introduce a **SVDKL surrogate model** $F_l : \mathcal{L}_l \times \mathcal{L}_{l-1} \to \mathcal{L}_l$.

The next latent states, at fidelity level l, $\mathbf{z}_l^{(t+1)}$ can be sampled with F_l:

$$z_{i,l}^{(t+1)} = Z_i^{F_l}(\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)}) + \varepsilon_{F_l}, \quad \varepsilon_{F_l} \sim \mathcal{N}(0, \sigma_{F_l}^2)$$

$$Z_i^{F_l}(\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)}) \sim$$

$$\mathcal{GP}(\mu(g_{F_l}(\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)}; \mathbf{w}_{F_l})), k(g_{F_l}(\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)}; \mathbf{w}_{F_l}), g_{F_l}(\mathbf{z}_l^{(t')}, \mathbf{z}_{l-1}^{(t'+1)}; \mathbf{w}_{F_l}); \theta_{F_l,i})), \quad 1 \leq i \leq |\mathbf{z}_l|$$

where $g_{F_l}(\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)}; \mathbf{w}^{F_l})$ is the feature vector output of the NN of F_l.

Training the model

The two components E_l and F_l are jointly trained.

- E_l is trained by minimising the error between the reconstruction $\hat{\mathbf{x}}_l^{(t)}$ and the measurement $\mathbf{x}_l^{(t)}$;
- F_l is trained by minimising the **Kullback-Leibler divergence** between the distributions $p(\mathbf{z}_l^{(t+1)}|\mathbf{x}_l^{(t+1)})$ and $p(\mathbf{z}_l^{(t+1)}|\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)})$.

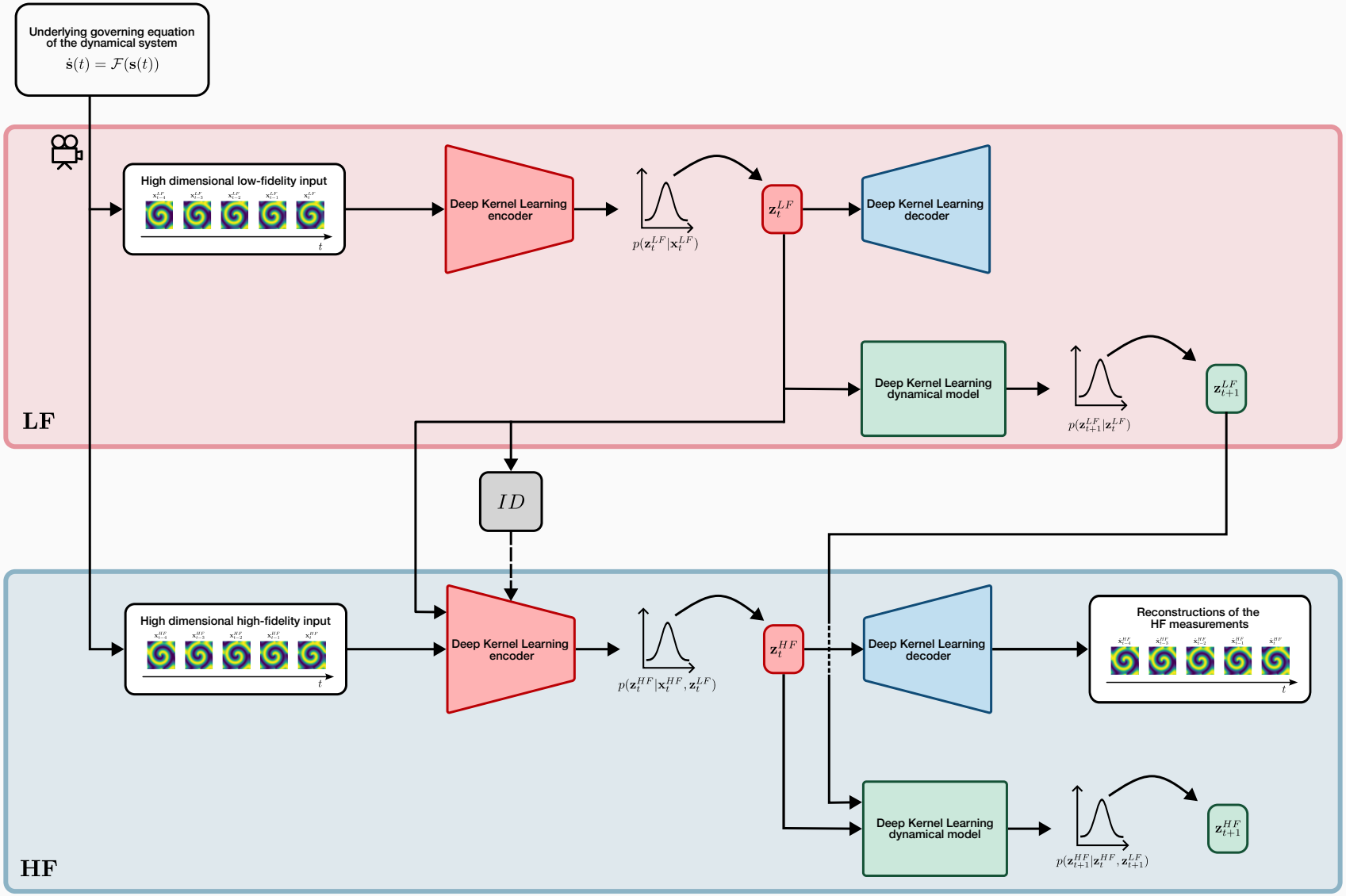
The loss function is given by:

$$\text{loss}_l(\mathbf{w}_{E_l}, \boldsymbol{\theta}_{E_l}, \sigma_{E_l}^2, \boldsymbol{\theta}_{D_l}, \mathbf{w}_{F_l}, \boldsymbol{\theta}_{F_l}, \sigma_{F_l}^2) =$$

$$\mathbb{E}_{\mathbf{x}_l^{(t)}, \mathbf{x}_l^{(t+1)} \sim \mathbf{x}_l}[-\log p(\hat{\mathbf{x}}_l^{(t)}|\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t)}) + \beta \text{KL}[p(\mathbf{z}_l^{(t+1)}|\mathbf{x}_l^{(t+1)})||p(\mathbf{z}_l^{(t+1)}|\mathbf{z}_l^{(t)}, \mathbf{z}_{l-1}^{(t+1)}] +$$
$$- \log p(\hat{\mathbf{x}}_l^{(t+1)}|\mathbf{z}_l^{(t+1)}, \mathbf{z}_{l-1}^{(t+1)})]$$

in which β is used to scale the contribution of the two loss terms.

Scheme of the model



Implementation

Module structure

The module is implemented in **Python**.

It is composed of three main parts:

1. **the dataset generation**, to produce the data for the training and testing of the model;
2. **the training of the model**, to fit the model exploiting a dedicated training dataset;
3. **the testing of the model**, to test the goodness of fit of the trained model and produce some reconstructions from the dedicated testing dataset.

Repository

The source code can be found **here**.

Folder structure

The Git repository contains the following files and folders:

- `README.md`, with instructions on how to run the code;
- `config.yaml`, to set the parameters;
- `generate_dataset.py`, to produce the train and test data;
- `main-gym.py`, `main-pde.py`, to train the model;
- `test-gym.py`, `test-pde.py`, `test_utils.py`, to test the model;
- `requirements.txt`, that lists the required libraries;
- `/src` (folder), containing the implementation of the Python module;
- `/Data` (folder), to store the train and test datasets;
- `/Results` (folder), to store weights and plots.

`GenerateDataset` class

The `GenerateDataset` class provides an interface to **produce the training and testing datasets**, at multiple levels of fidelity and considering the desired test case.

Two major dataset environments are available:

- **Gym**, producing the Pendulum, Acrobot and MountainCarContinuous datasets;
- **PDE**, producing the reaction-diffusion and diffusion-advection datasets.

The `GenerateDataset` class is implemented as an **abstract class**, with `GenerateGym` and `GeneratePDE` inheriting from it and implementing the abstract method `generate_dataset()`.

`GenerateGym` and `GeneratePDE` classes

`GenerateGym` class

The method `generate_dataset()` runs episodes in the chosen Gymnasium environment, and it produces a pickle file for each fidelity level.

`GeneratePDE` class

The method `generate_dataset()` employs a dedicated solver from the class `PDESolver` to solve the equation at each level of fidelity and for each parameter `mu`. Solutions are saved as RGB heatmaps.

The `Logger` class is implemented and used to log and save the generated datasets.

`PDESolver` class

The abstract class `PDESolver` has two implemented child classes.

`ReactionDiffusionSolver` class

Solves the lambda-omega reaction-diffusion system. The class leverages on the the SciPy library, for the explicit Runge-Kutta method of order 5 to solve the system, to compute the laplacians.

`DiffusionAdvectionSolver` class

Solves an diffusion-advection problem describing a fluid motion in the shallow water limit. The class leverages on the Numpy library for the gradient and the laplacian, and on the SciPy function `odeint()` to solve the system.

Both classes implement a `_rhs()` method to compute the discretised right hand side of the equation.

`BuildModel` class

The training of the model is performed for each fidelity level **sequentially**.

The `BuildModel` class initializes, trains and tests the model, exploiting the following methods:

- `add_level()`, that initializes the l-th sub-model;
- `train_level()`, that trains the previously initialized sub-model;
- `get_latent_representations()`, that evaluates the trained sub-model and returns the latent representation;
- `eval_level()`, that evaluates the trained sub-model and return both the latent representations and the relative reconstructions.

`DataLoader` classes, `train()` function, *ID* estimation

`DataLoader` classes

A parent `BaseDataLoader` class and two specialized classes, `GymDataLoader` and `PDEDataLoader`, are implemented, to load and manipulate the dataset.

`train()` function

Loops over a number of random batches: the gradients are initialized, a forward pass of the model is performed and the loss is updated. Then, the gradients are backward propagated and the parameters updated.

The loss terms are implemented in `losses.py` and `variational_inference.py`.

ID estimation

The `eval_id()` function exploits the Levina-Bickel algorithm to estimate the ID of a dataset.

models

The final model is implemented by the `SVDKL_AE_latent_dyn()` class, which exploits two layers: `SVDKL_AE` and `Forward_DKLModel`.

`SVDKL_AE` class

A forward pass encodes the input and corrects the resulting latent representation with the corresponding LF one. The features are then passed through the `GaussianProcessLayer`, which produces a predictive latent state distribution $p(\mathbf{z}_t|\mathbf{x}_t)$. Finally, the latent state vector $\mathbf{z}_t$ is sampled and decoded.

`Forward_DKLModel` class

It is composed of a neural network and a `GaussianProcessLayer`. The output features of the neural network are fed to independent GPs to produce a next state distribution $p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{z}_{t+1}^{LF})$. Again, we sample the next latent states $\mathbf{z}_{t+1}$.

Testing the model

The model can **tested on an unseen dataset**.

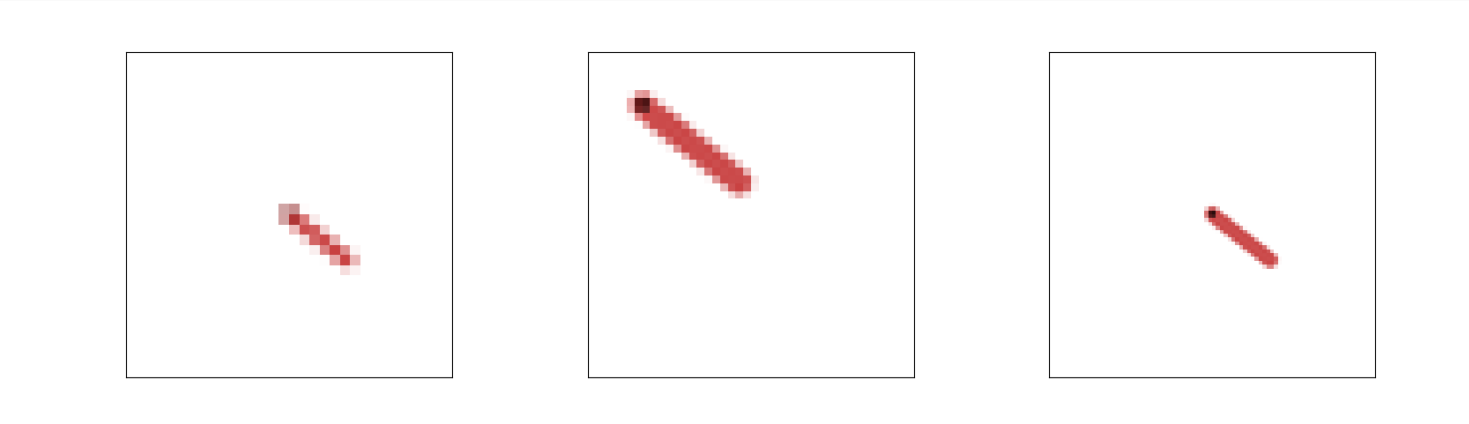
Again, we instantiate the model as a `BuildModel` object, and **load the trained weights**. The methods `add_level()`, `test_level()` and `eval_level()` are used in this case.

In a typical setup, we

- evaluate the HF level on new test data;
- plot the latent variables, using the function `plot_latent_dims()`;
- reconstruct some frames and compare them with the true measurements;
- predict one-step forward, exploiting the model method `predict_dynamics_mean()`;
- extrapolate in time, by iterating on the one-step prediction.

Numerical results

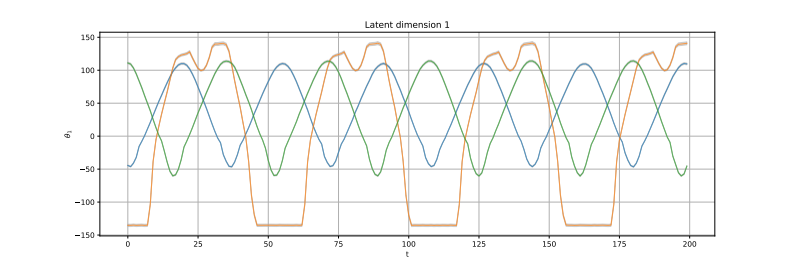
Pendulum: input data



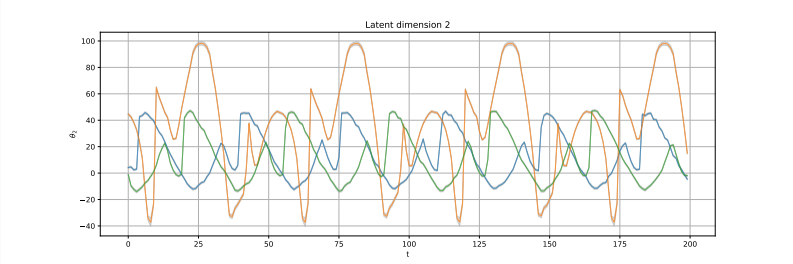
(a) Fidelity level 0 **(b)** Fidelity level 1 **(c)** Fidelity level 2

Figure 2: Pendulum case.

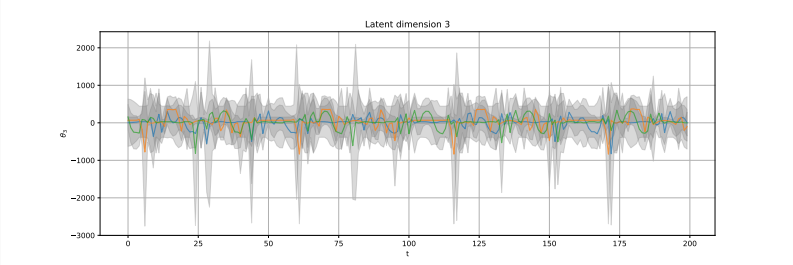
Pendulum: ID and latent variables



(a) $\theta_1^2(t)$



(b) $\theta_2^2(t)$



(c) $\theta_3^2(t)$

Figure 3: Pendulum case.

Pendulum: reconstruction and one-step forward prediction

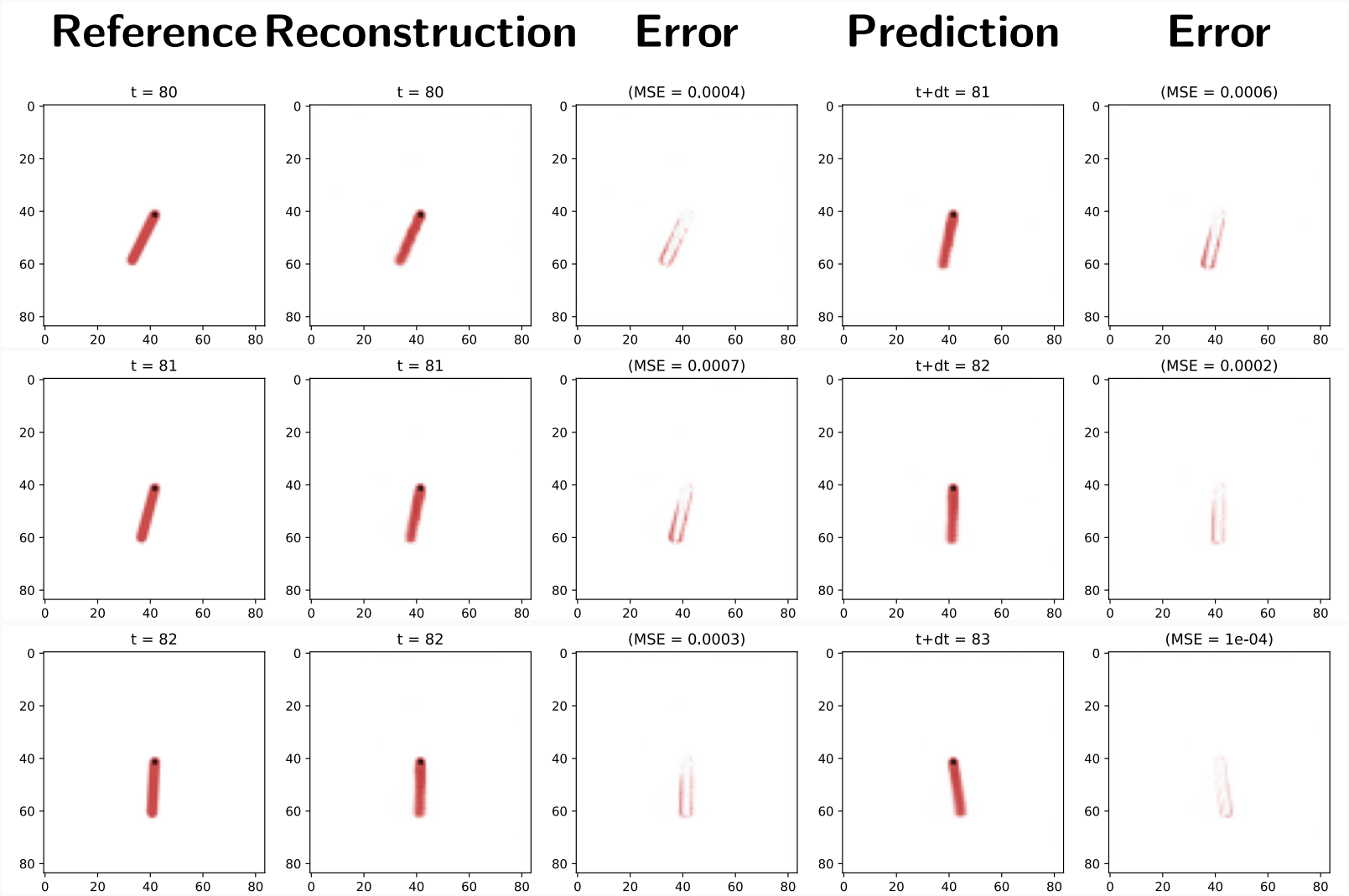


Figure 4: Pendulum case.

Pendulum: extrapolation in time

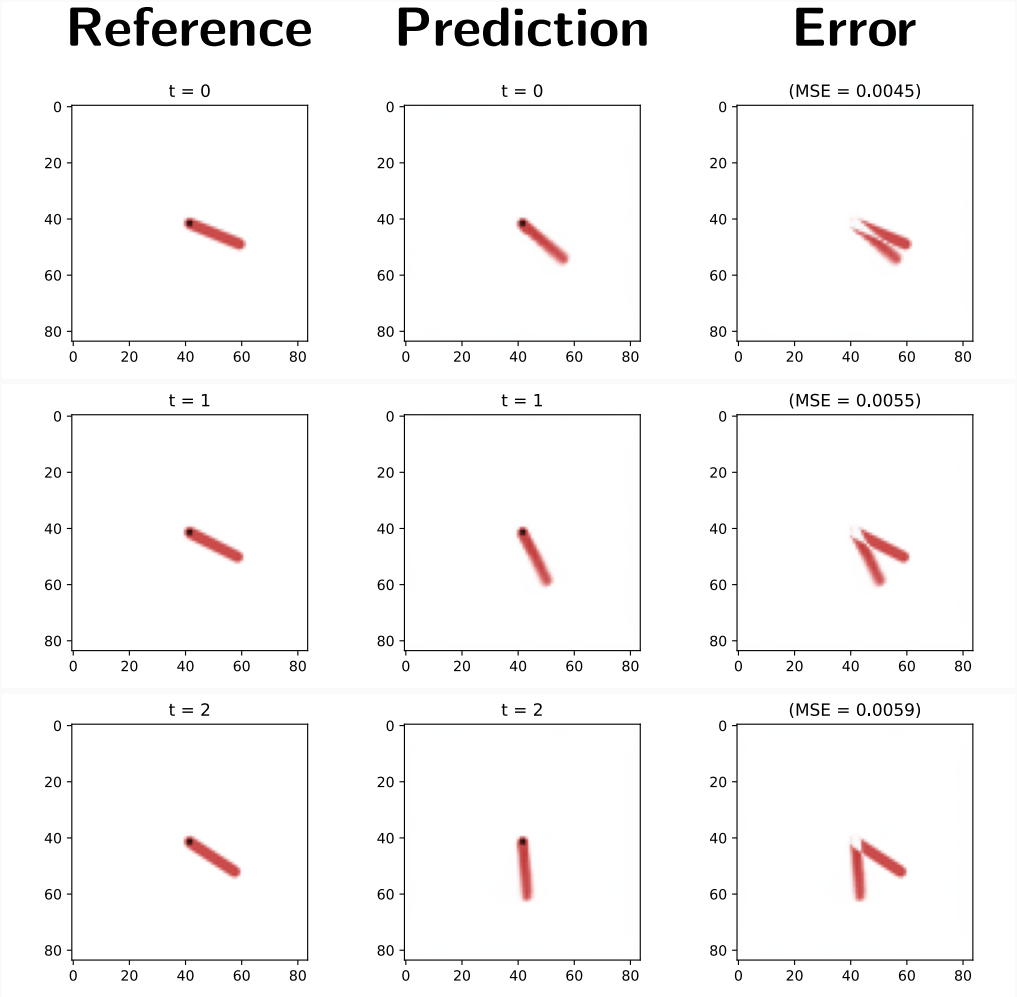
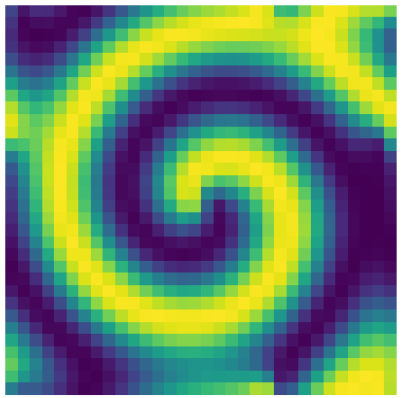
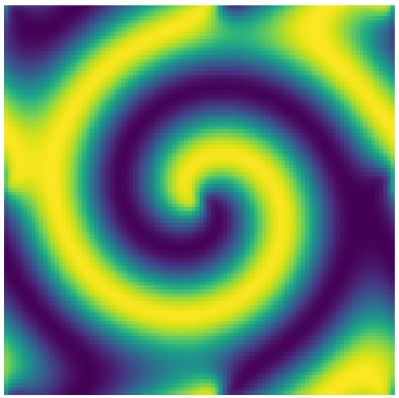


Figure 5: Pendulum case.

PDE: input data



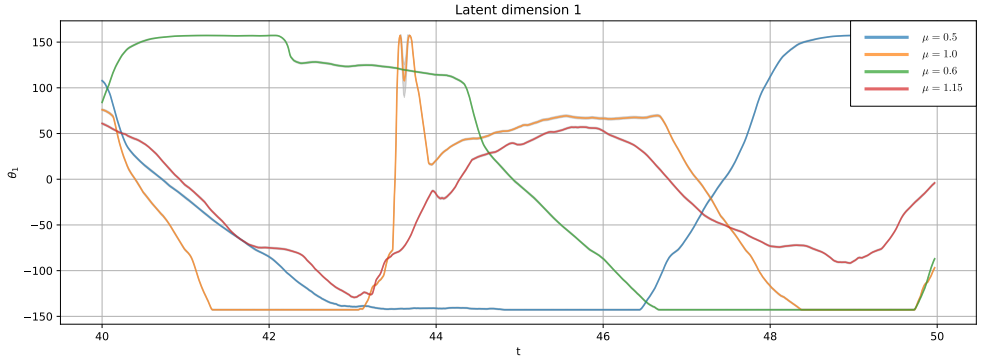
(a) Low fidelity sample



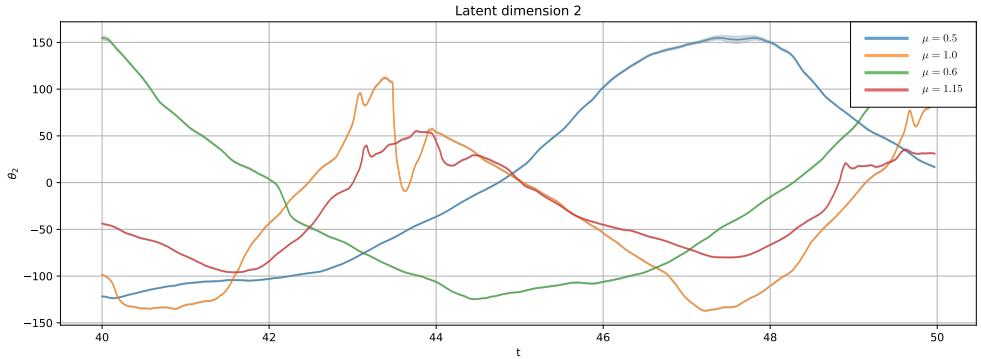
(b) High fidelity sample

Figure 6: Samples from the reaction-diffusion dataset.

PDE: ID and latent variables



(a) $\theta_1^{HF}(t)$



(b) $\theta_2^{HF}(t)$

Figure 7: Reaction-diffusion case.

PDE: reconstruction and one-step forward prediction

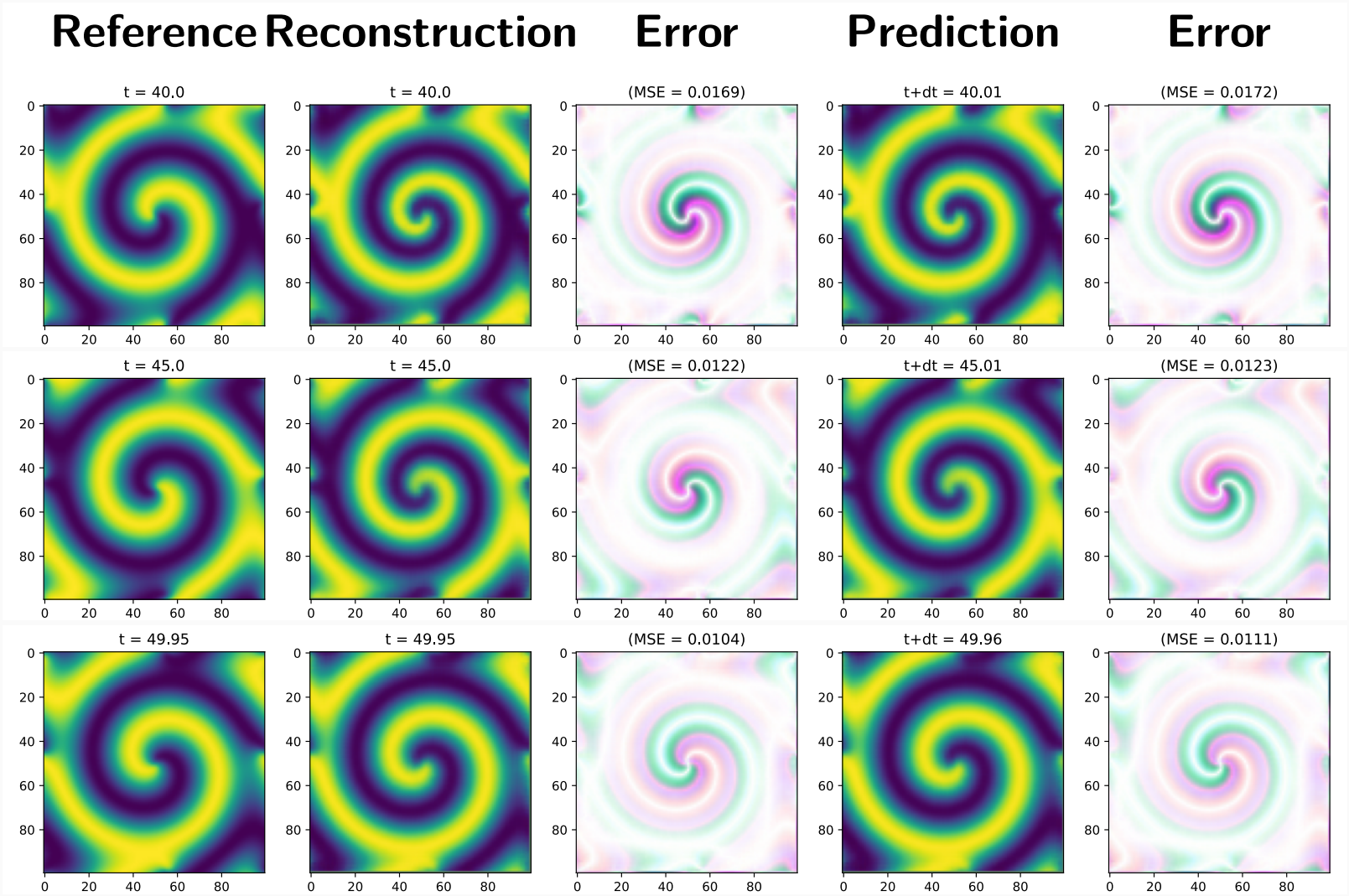


Figure 8: Reaction diffusion case.

PDE: extrapolation in time

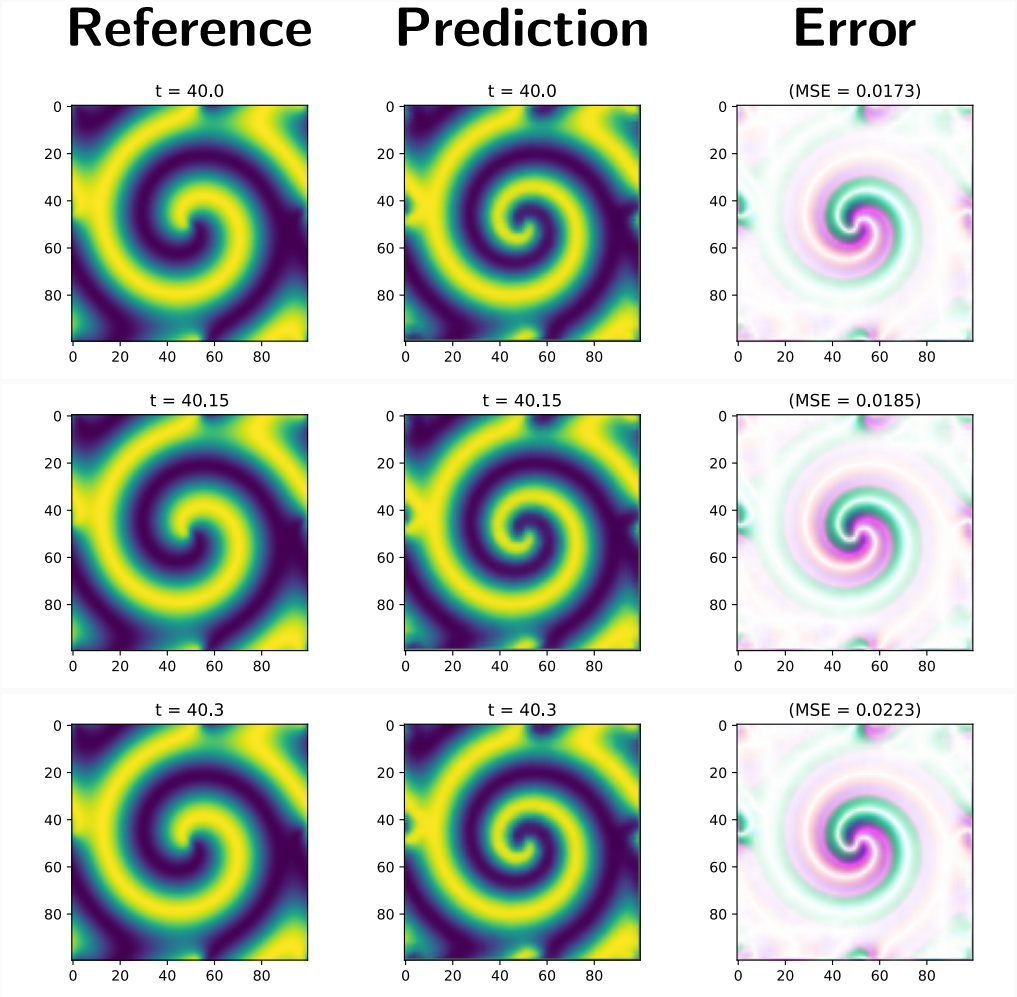


Figure 9: Reaction-diffusion case.

Conclusions

Summary

Conclusions

- ✓ The implementation expands the work from Botteghi et al., **generalising to the MF context**, testing on new cases and introducing a new user interface.
- ✓ It is capable of considering a **wide range of configurations and test cases**, it supports the dataset generation and the initialization of **complex model structures**.
- ✓ The model proves **effective at learning unknown system dynamics**, with respect to the considered test cases.

Further developments

- Expand to a wider range of data sources.
- Leverage both low-dimensional and high-dimensional measurements.
- Consider different time steps of discretisation among fidelity levels.

Thank you for the attention!

Questions?

Appendix: Preliminaries

Gaussian process

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Let $\mathbf{x} \in \mathbb{R}^n$, we define the mean function $m(\mathbf{x})$ and the kernel function $k(\mathbf{x}, \mathbf{x}')$ of a real process $Z(\mathbf{x})$ as

$$\begin{aligned} m(\mathbf{x}) &= E[Z(\mathbf{x})]\,, \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(Z(\mathbf{x}) - m(\mathbf{x}))(Z(\mathbf{x}') - m(\mathbf{x}'))]\,. \end{aligned} \tag{1}$$

We write the Gaussian process as

$$Z(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))\,. \tag{2}$$

Kernel function

The kernel function can be parametrised as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_k^2 r(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) \tag{3}$$

where σ^2 and $\boldsymbol{\theta}$ have to be estimated.

To build up a GP model, the class of the correlation kernel $r(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta})$ needs to be set. A popular choice is the automatic relevance determination (ARD) squared exponential (SE) kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_k^2 \exp\left(-\frac{1}{2} \sum_{j-1}^{d} \frac{(x_j - x_j')^2}{l_j^2} \right) \tag{4}$$

where $\{l_j\}_{(1 \leq j \leq d)}$ are the length-scales along each individual input direction.

Predictive distribution

Assuming additive Gaussian noise $G(\mathbf{x})|Z(\mathbf{x}) \sim \mathcal{N}(0, \sigma_\epsilon^2)$, the predictive distribution of the GP evaluated at N^* test data points $\mathcal{X}^*$ is

$$[Z(\mathcal{X}^*)|Z(\mathcal{X}) = \mathcal{Y}, \sigma^2, \boldsymbol{\theta}] \sim \mathcal{N}(\mathbb{E}[Z(\mathcal{X}^*)], C(Z(\mathcal{X}^*))) \tag{5}$$

where

$$\begin{aligned} \mathbb{E}[Z(\mathcal{X}^*)] &= \boldsymbol{\mu}_{\mathcal{X}^*} + K_{\mathcal{X}^*,\mathcal{X}}(K_{\mathcal{X},\mathcal{X}} + \sigma I)^{-1}\mathcal{Y} \\ C(Z(\mathcal{X}^*)) &= K_{\mathcal{X}^*,\mathcal{X}^*} + K_{\mathcal{X}^*,\mathcal{X}}(K_{\mathcal{X},\mathcal{X}} + \sigma I)^{-1}K_{\mathcal{X},\mathcal{X}^*} \end{aligned} \tag{6}$$

$K_{\mathcal{X}^*,\mathcal{X}}$ is an $N^* \times N$ matrix of covariances between the GP evaluated at $\mathcal{X}^*$ and $\mathcal{X}$. $\boldsymbol{\mu}_{\mathcal{X}^*}$ is the $N^* \times 1$ mean vector, and $K_{\mathcal{X},\mathcal{X}}$ is the $N \times N$ covariance matrix evaluated at training inputs $\mathcal{X}$. All covariance matrices implicitly depend on the kernel hyperparameters $\boldsymbol{\theta}$.

Parameter optimisation

By maximising the marginal likelihood given the training targets $\mathcal{Y}$, the optimal values of $(\boldsymbol{\theta}, \sigma_\epsilon)$ can be estimated:

$$
\begin{aligned}
(\hat{\boldsymbol{\theta}}, \hat{\sigma}_\epsilon) &= \arg\max_{\boldsymbol{\theta}, \sigma_\epsilon} \log p(\mathcal{Y}|\mathcal{X}; \boldsymbol{\theta}) = \\
&= \arg\max_{\boldsymbol{\theta}, \sigma_\epsilon} \Big\{ -\mathcal{Y}^\top (K_{\mathcal{X},\mathcal{X}|\boldsymbol{\theta}} + \sigma_\epsilon^2 I)^{-1} \mathcal{Y} - \log |K_{\mathcal{X},\mathcal{X}|\boldsymbol{\theta}} + \sigma_\epsilon^2 I| \Big\}
\end{aligned} \tag{7}
$$

Levina-Bickel algorithm 1

The Levina-Bickel algorithm for ID estimation provides a maximum likelihood estimator (MLE) of the dimension m from i.i.d. observations $\mathbf{z}_1, ..., \mathbf{z}_n$ in $\mathbb{R}^p$.

The observations $\mathbf{z}_i = g(\mathbf{s}_i)$, where $\mathbf{s}_i$ are sampled from an unknown smooth density f on $\mathbb{R}^{ID}$, with unknown $ID \leq p$, and g is a continuous and sufficiently smooth mapping.

The key idea is to fix a point $\mathbf{z}$, assume $f(\mathbf{z}) \approx const$ in a small sphere $S_{\mathbf{z}}(R)$ of radius R around $\mathbf{z}$, and treat the observations as a homogeneous Poisson process in $S_{\mathbf{z}}(R)$.

Levina-Bickel algorithm 2

The Levina–Bickel algorithm derives the local *ID* estimator near $\mathbf{z}$ as

$$ID_{L-B}(\mathbf{z}) = \frac{1}{k-2} \sum_{j=1}^{k-1} \log \frac{T_k(\mathbf{z})}{T_j(\mathbf{z})} \quad , \tag{8}$$

where we are fixing the number of neighbors k rather than the radius of the sphere R, and $T_k(\mathbf{z})$ is the Euclidean distance between $\mathbf{z}$ and the its k-th nearest neighbor in $\mathbf{z}_1, ..., \mathbf{z}_n$.

By dividing by $k-1$, rather than $k-2$, we obtain an asymptotically unbiased estimator.

Appendix: Methods

Variational inference

Since the two SVDKL components exploit variational inference to approximate the respective posterior distributions, we add two extra terms to the loss function, of the form

$$\text{loss}_{var}(\mathbf{w}, \boldsymbol{\theta}) = \text{KL}[p(\mathbf{v})||q(\mathbf{v})]\,, \tag{9}$$

where $p(\mathbf{v})$ is the posterior to be approximated over the inducing points $\mathbf{v}$, and $q(\mathbf{v})$ represents an approximating candidate distribution. The inducing points $\mathbf{v}$ are placed on a grid, following the intuition in the original SVDKL work.

Kullback–Leibler divergence

Definition

The Kullback–Leibler (KL) divergence is a measure of how one probability distribution P diverges from a second, reference probability distribution Q.

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{10}$$

Properties:

- KL divergence is non-negative: $D_{\mathrm{KL}}(P \parallel Q) \geq 0$.
- $D_{\mathrm{KL}}(P \parallel Q) = 0$ if and only if $P = Q$ almost everywhere.
- It is not symmetric: $D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P)$.
- Often interpreted as the "information loss" when Q is used to approximate P.

Algorithm i

for level = LF, HF **do**
  if level == LF **then**
    Initialise $\mathbf{z}_{LF}, \mathbf{z}_{LF}^{next}, \mathbf{z}_{LF}^{fwd} = \mathbf{0}$;
    Set latentDimension to an arbitrary value;
  else
    Initialise $\mathbf{z}_{LF} = \mathbf{z}$, $\mathbf{z}_{LF}^{next} = \mathbf{z}^{next}$, $\mathbf{z}_{LF}^{fwd} = \mathbf{z}^{fwd}$;
    Set latentDimension = ID;
  end if

  for epoch = 1, 2, ... **do**
    Encode $\mathbf{x} \rightarrow \mathbf{z}$, $\mathbf{x}^{next} \rightarrow \mathbf{z}^{next}$;
    Decode $\hat{\mathbf{x}} \leftarrow \mathbf{z} + \rho\mathbf{z}_{LF}$, $\hat{\mathbf{x}}^{next} \leftarrow \mathbf{z}^{next} + \rho\mathbf{z}_{LF}^{next}$;
    Estimate $\mathbf{z} + \rho\mathbf{z}_{LF}^{fwd} \rightarrow \mathbf{z}^{fwd}$;
    Decode $\hat{\mathbf{x}}^{fwd} \leftarrow \mathbf{z}^{fwd}$;
    Compute the $loss(\mathbf{x}, \hat{\mathbf{x}}, \hat{\mathbf{x}}^{next}, \hat{\mathbf{x}}^{fwd}, \mathbf{z}^{next}, \mathbf{z}^{fwd})$;

Algorithm ii

16: Backward propagate the gradients;
17: **end for**
18:
19: **if** level == LF **then**
20: Compute ID = $\lceil \frac{1}{3} LB(\mathbf{z}) + \frac{1}{3} LB(\mathbf{z}^{next})) + \frac{1}{3} LB(\mathbf{z}^{fwd}))\rceil$;
21: **end if**
22: **end for**

Appendix: Implementation

Generate the dataset: `utils.py`

- `save_pickle()`
- `stack_frames()`
- `crop_frame()`
- `add_occlusion()`
- `len_of_episode()`
- `heatmap_to_image()`

Training and testing: `utils.py`

- `load_pickle()`
- `align_pde()`
- `check_indices()`
- `plot_frame()`
- `plot_latent_dims()`
- `get_length()`
- `generate_gif()`
- `plot_error()`

Appendix: Results

Pendulum

We consider the simple pendulum, described by the following equation:

$$\ddot{\phi}(t) = -\frac{g}{l} \sin \phi(t) \,, \tag{11}$$

where ϕ is the angle of the pendulum, $\ddot{\phi}$ is the angular acceleration, l is the length, g is the gravitational acceleration. The starting state is a random angle and a random angular velocity, in particular

$$\phi(0) \sim \mathcal{U}([-\pi, \pi]) \,, \tag{12a}$$

$$\dot{\phi}(0) \sim \mathcal{U}([-1, 1]) \,. \tag{12b}$$

Our goal is to approximate the time-dependent state of the pendulum $x_1 = \phi, x_2 = \dot{\phi}$, considering a test set of new episodes, unseen during training.

Pendulum: dataset structure

We adopt 3 fidelity levels that differ in the resolution of the frame:

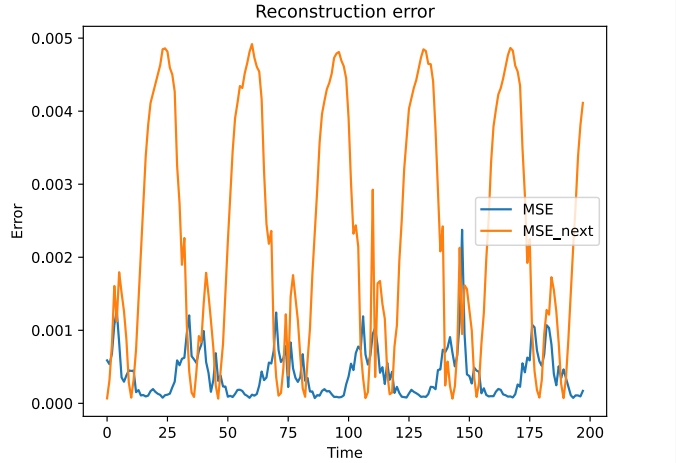
- **level 0**, composed of a number of (full) RGB images of size $l_0 \times l_0 \times 3$;
- **level 1**, composed of cropped RGB images of size $l_1 \times l_1 \times 3$, considering the bottom-right 60% portion of the image;
- **level 2**, that collects full RGB images of high resolution size $l_2 \times l_2 \times 3$, where $l_2 > l_0$.

The initial conditions are randomly initialised for each episode.

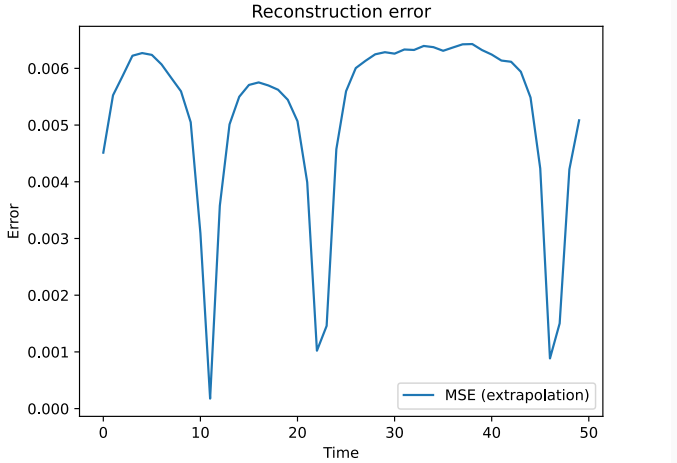
Parameters	**level 0**	**level 1**	**level 2**
N	150	80	50
l	32	42	84
portion	1	0.6	1
N_{test}	3	3	3

Table 1: List of configuration parameters for the pendulum dataset.

Pendulum: errors



(a) MSEs of the reconstruction and forward prediction



(b) MSE of the extrapolation in time

Figure 10: MSEs of the pendulum case.

Reaction-diffusion

We consider a lambda-omega reaction-diffusion system defined by the following equations

$$\dot{u} = \left(1 - \left(u^2 + v^2\right)\right) u + \mu \left(u^2 + v^2\right) v + d\left(u_{xx} + u_{yy}\right), \tag{13a}$$

$$\dot{v} = -\mu \left(u^2 + v^2\right) u + \left(1 - \left(u^2 + v^2\right)\right) v + d\left(v_{xx} + v_{yy}\right) \tag{13b}$$

defined over a spatial domain $(x, y) \in [-L, L]^2$ and a time span $t \in [0, T]$, where μ and d are the reaction and diffusion parameters, respectively. The initial condition is defined as

$$u(x, y, 0) = v(x, y, 0) = \tanh\left(\sqrt{x^2 + y^2} \cos\left((x + iy) - \sqrt{x^2 + y^2}\right)\right). \tag{14}$$

We are interested in approximating the solution components u and v as functions of the varying reaction parameter $\mu \in \mathcal{P} = [0.5, 1.5]$, with a fixed diffusion coefficient d.

Reaction-diffusion: dataset structure

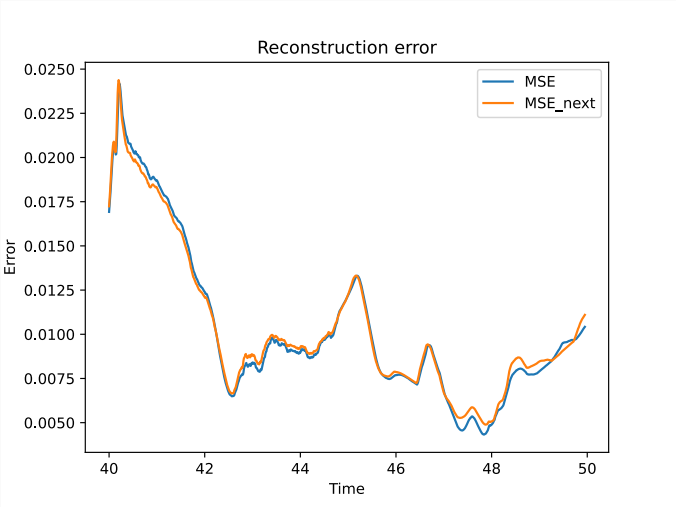
LF data are associated with a coarser grid of discretisation ($n^{LF} < n^{HF}$), but they are observed for a longer time window, $T^{LF}_{train} > T^{HF}_{train}$.

The model is tested on new measurements for $t \in [T^{HF}_{train}, T^{HF}_{train} + T_{test}]$, extrapolating over time and $\mathcal{P}$.

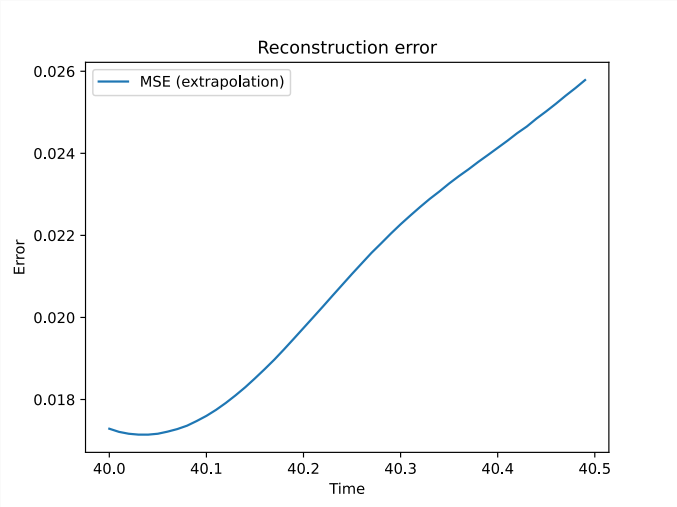
Parameters	**LF**	**HF**
n	32	100
d	0.05	0.05
μ_{train}	$\{0.5, 0.75, 1, 0.6, 1.15, 1.375\}$	$\{0.5, 0.75, 1\}$
T_{train}	80	40
L	10	
dt	0.01	
μ_{test}	$\{0.5, 1, 0.6, 1.15\}$	
T_{test}	10	

Table 2: List of configuration parameters for the reaction-diffusion dataset.

Reaction-diffusion: errors



(a) MSEs of the reconstruction and forward prediction



(b) MSE of the extrapolation in time

Figure 11: MSEs of the reaction-diffusion case.

Diffusion-advection

We consider a diffusion-advection problem describing a fluid motion in the shallow water limit given by

$$\frac{\partial \omega}{\partial t} + \mu \left(\frac{\partial \psi}{\partial x} \frac{\partial \omega}{\partial y} - \frac{\partial \psi}{\partial y} \frac{\partial \omega}{\partial x} \right) = d \nabla^2 \omega, \tag{15a}$$

$$\nabla^2 \psi = \omega \tag{15b}$$

defined over a spatial domain $(x, y) \in [-L, L]^2$ and a time span $t \in [0, T]$. Here $\omega(x, y, t)$ and $\psi(x, y, t)$ represent the vorticity and stream function, respectively, $\nabla^2 = \partial x^2 + \partial y^2$ is the two-dimensional Laplacian, and d is the diffusion coefficient. We consider the following initial condition of vorticity:

$$\omega(x, y, 0) = \exp\left(-2x^2 - \frac{y^2}{20} \right), \quad (x, y) \in [-L, L]^2. \tag{16}$$

Our goal is to approximate the time-dependent vorticity field ω as the parameter μ varies over $\mathcal{P} = [1, 5]$.

Diffusion-advection: dataset structure

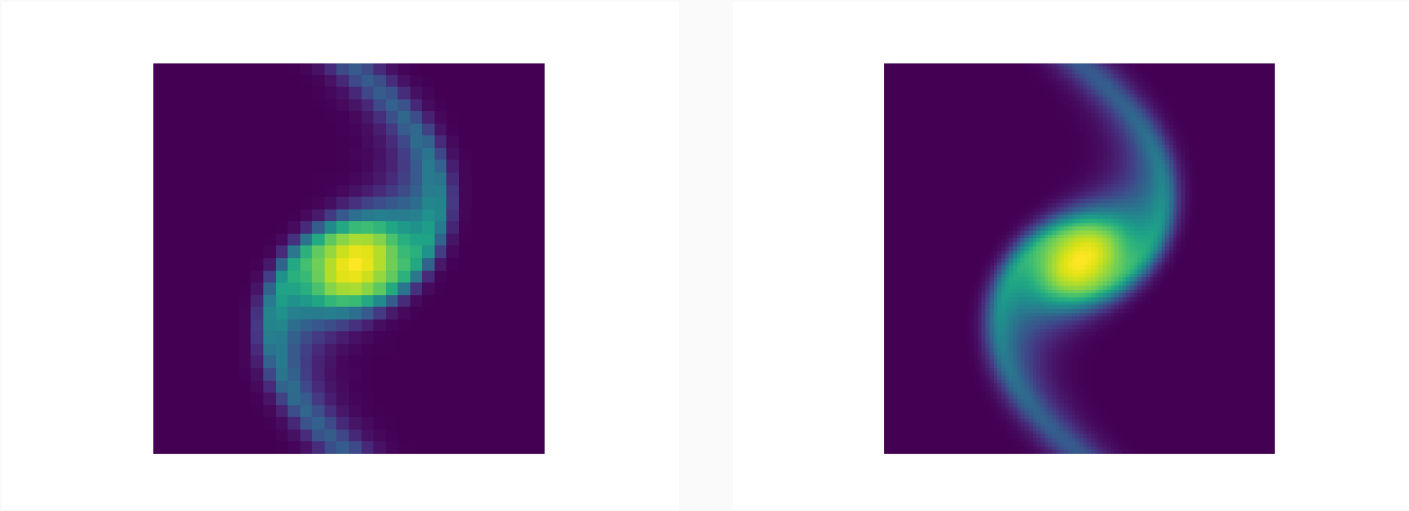
LF data are associated with a coarser grid of discretisation ($n^{LF} < n^{HF}$), but they are observed for a longer time window, $T^{LF}_{train} > T^{HF}_{train}$.

The model is tested on new measurements for $t \in [T^{HF}_{train}, T^{HF}_{train} + T_{test}]$, extrapolating over time and $\mathcal{P}$.

Parameters	**LF**	**HF**
n	32	100
d	0.001	0.001
μ_{train}	$\{1.5, 2.5, 3.5, 4.5\}$	$\{1.5, 2.5\}$
T_{train}	30	10
L	5	
dt	0.01	
μ_{test}	$\{1.5, 2.5, 3.5\}$	
T_{test}	5	

Table 3: List of configuration parameters for the diffusion-advection dataset.

PDE: input-data

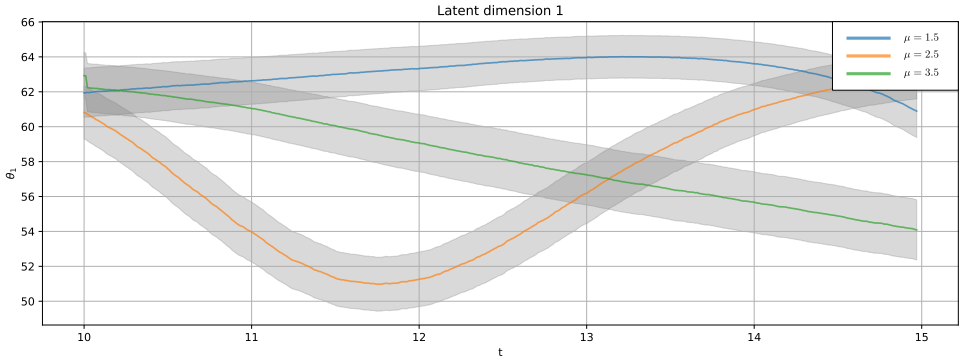


(a) Low fidelity sample

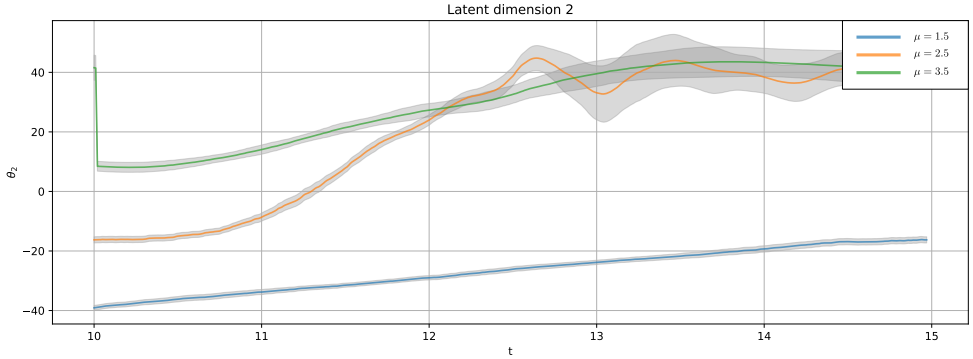
(b) High fidelity sample

Figure 12: Samples from the diffusion-advection dataset.

PDE: ID and latent variables



(a) $\theta_1^{HF}(t)$



(b) $\theta_2^{HF}(t)$

Figure 13: Diffusion-advection case.

PDE: reconstruction and one-step forward prediction

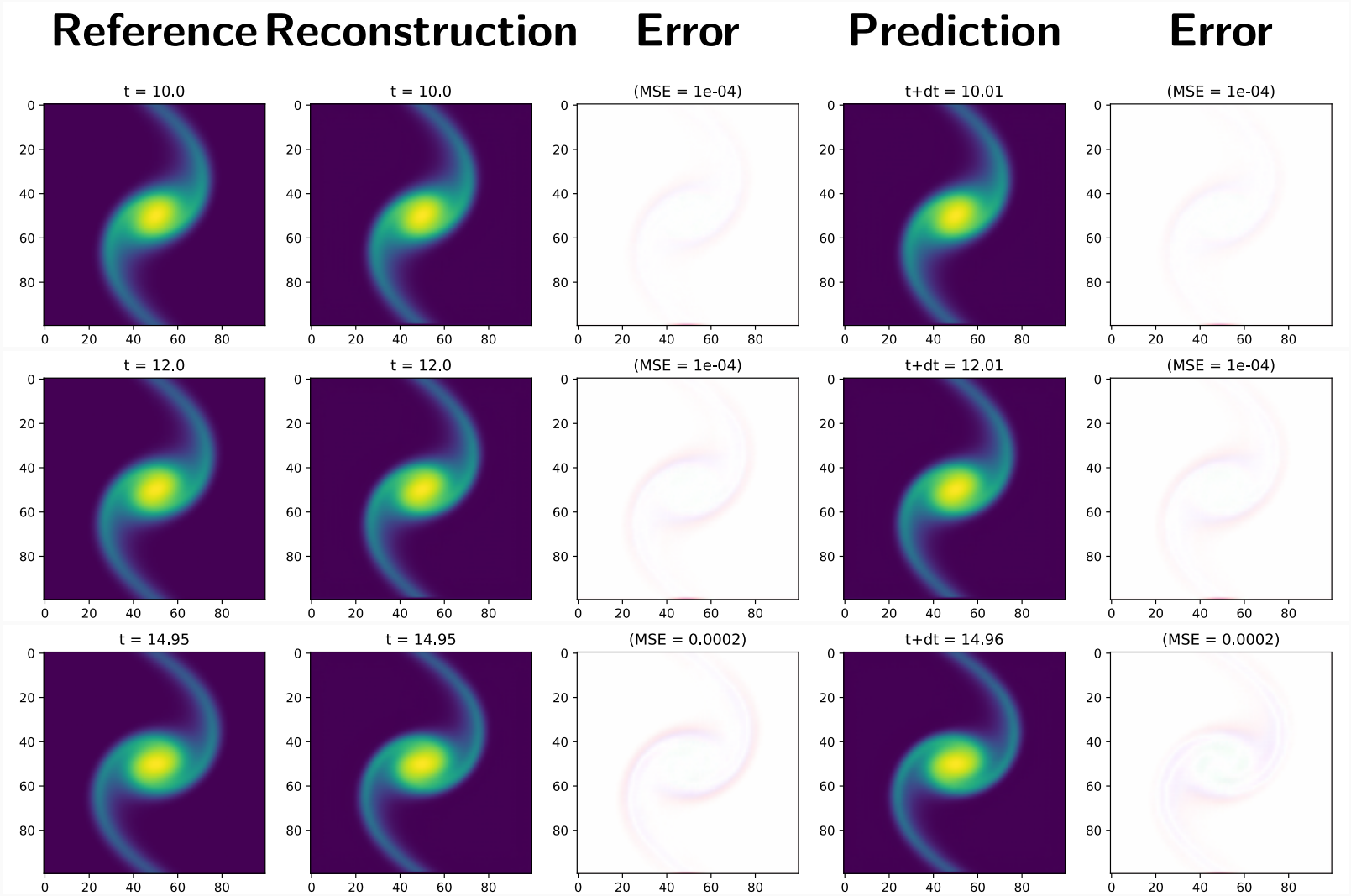


Figure 14: Diffusion-advection case.

PDE: extrapolation in time

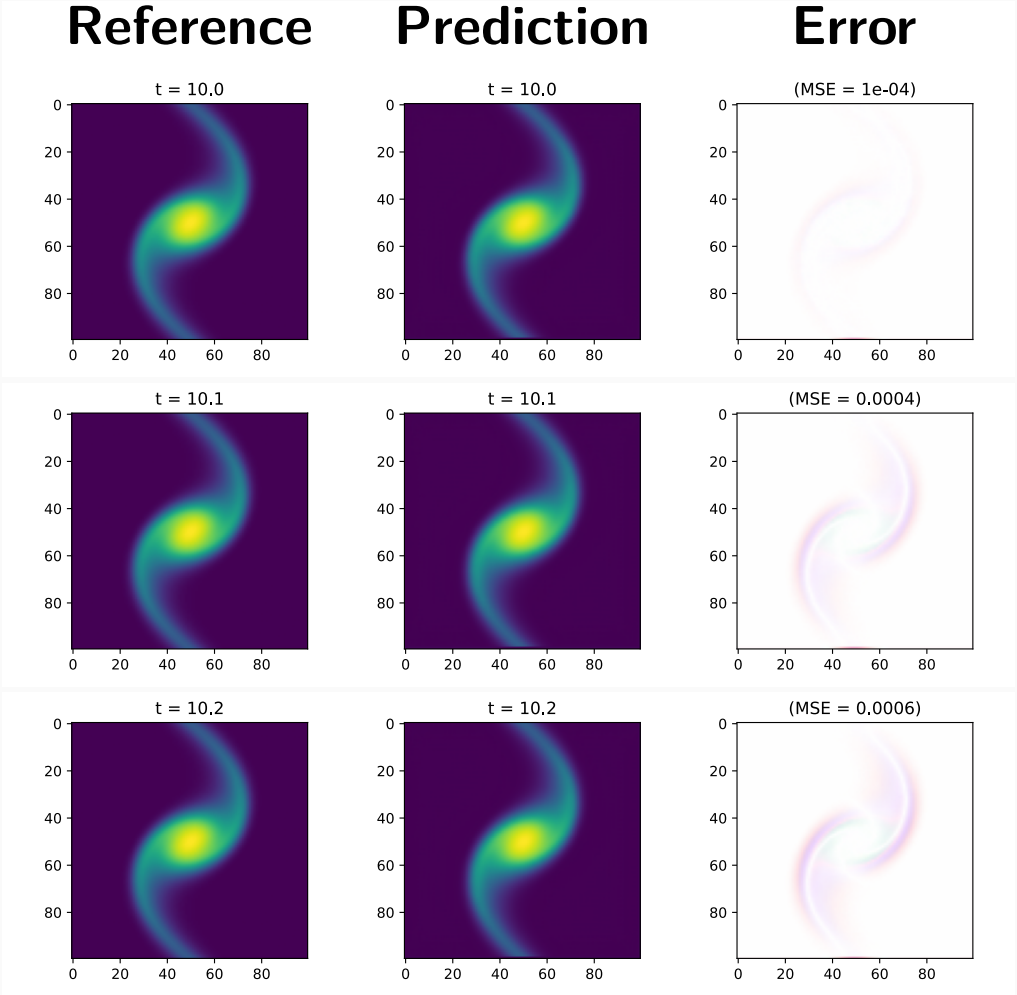
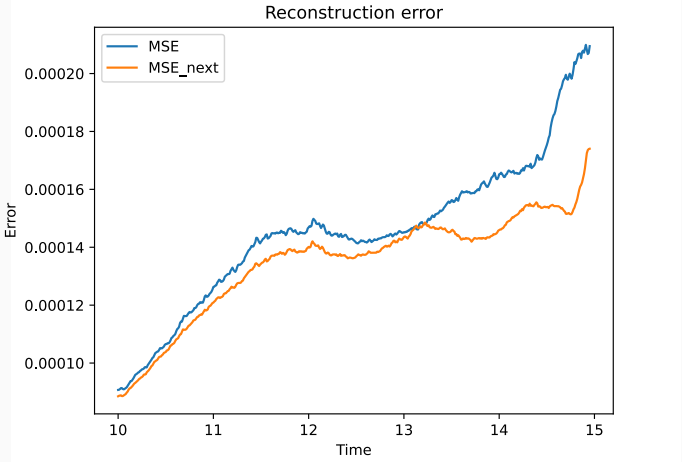
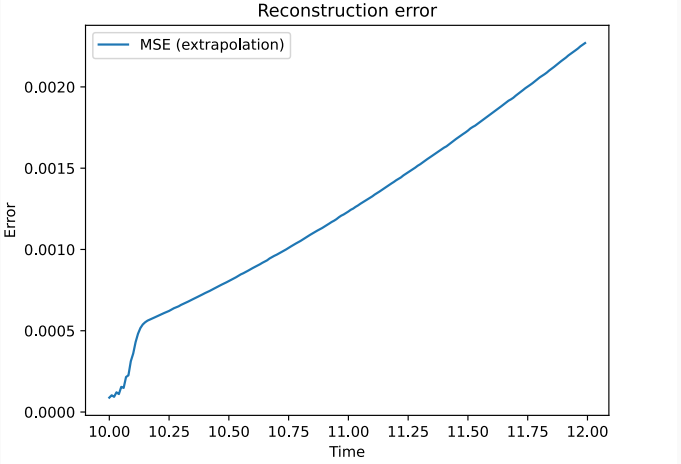


Figure 15: Diffusion-advection case.

Diffusion-advection: errors



(a) MSEs of the reconstruction and forward prediction



(b) MSE of the extrapolation in time

Figure 16: MSEs of the diffusion-advection case.

References i

Nicolò Botteghi, Mengwu Guo, and Christoph Brune.
Deep Kernel Learning of Dynamical Models from High-Dimensional Noisy Data.
Scientific Reports, 12(1):21530, December 2022.
arXiv:2208.12975 [cs, stat].

Boyuan Chen, Kuang Huang, Sunand Raghupathi, Ishaan Chandratreya, Qiang Du, and Hod Lipson.
Discovering State Variables Hidden in Experimental Data, December 2021.
arXiv:2112.10755 [physics].

References ii

Paolo Conti, Mengwu Guo, Andrea Manzoni, Attilio Frangi, Steven L. Brunton, and J. Nathan Kutz.
Multi-fidelity reduced-order surrogate modeling, September 2023.
arXiv:2309.00325 [cs, math].

Mengwu Guo, Andrea Manzoni, Maurice Amendt, Paolo Conti, and Jan S. Hesthaven.
Multi-fidelity regression using artificial neural networks: efficient approximation of parameter-dependent output quantities.
Computer Methods in Applied Mechanics and Engineering, 389:114378, February 2022.
arXiv:2102.13403 [cs, math].

References iii


Elizaveta Levina and Peter Bickel.
Maximum Likelihood Estimation of Intrinsic Dimension.
In *Advances in Neural Information Processing Systems*, volume 17.
MIT Press, 2004.


GPyTorch Developers.
***GPyTorch Documentation*, 2024.**
Accessed: 2024-06-19.


PyTorch Developers.
***PyTorch Documentation*, 2024.**
Accessed: 2024-06-19.


Gymnasium Developers.
***Gymnasium Documentation*, 2024.**
Accessed: 2024-06-24.