THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire d'Informatique de Grenoble

Entrainement de réseaux de neurones profonds en ligne à partir de données synthétiques avec apprentissage actif

High Performance Online Deep Neural Network Training from Synthetic Data with Active Learning

Présentée par :

Sofya DYMCHENKO

Direction de thèse :

Bruno RAFFIN DIRECTEUR DE RECHERCHE, CENTRE INRIA UNIVERSITE GRENOBLE ALPES	Directeur de thèse

Rapporteurs :

MATHIAS NIEPERT
FULL PROFESSOR, UNIVERSITÄT STUTTGART

RONAN FABLET
PROFESSEUR, IMT ATLANTIQUE BREST

Thèse soutenue publiquement le **21 avril 2026**, devant le jury composé de :

ERIC BLAYO, PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Président
BRUNO RAFFIN, DIRECTEUR DE RECHERCHE, CENTRE INRIA UNIVERSITE GRENOBLE ALPES	Directeur de thèse
MATHIAS NIEPERT, FULL PROFESSOR, UNIVERSITÄT STUTTGART	Rapporteur
RONAN FABLET, PROFESSEUR, IMT ATLANTIQUE BREST	Rapporteur
KYLE CRANMER, FULL PROFESSOR, UNIVERSITY OF WISCONSIN, MADISON	Examinateur

High Performance Online Deep Neural Network Training from Synthetic Data with Active Learning for Scientific Computing

Sofya Dymchenko

July 21, 2026

All intervening steps, scribbles, sketches, drawings, failed work models, studies, thoughts, conversations are of interest. Those that show the thought process of the artist are sometimes more interesting than the final product.

Paragraphs on Conceptual Art, 1967
Sol LeWitt


simulation
training
surrogate
method
data
model
PDE
sample
time
solver
Breed
Active Learning
data generation
sampling
online training
parameter
input parameter
provide
improve
space
prediction
solution
value
chapter
surrogate training
distribution
PINN
state
design
update
online
NN
experiment
dataset
architecture
iteration efficiency
run
show
learn
framework
often
trajectory
training data
function
require
performance
higher
lower
process
three
width
several
R3
signal
via
initial
limited
larger
limit
IC parameter
increase
remain
overhead
benchmark
may
NS
Melissa framework
sampling mechanism
available
case
specifically
active sampling
complexity
time step
uncertainty
multiple
indicate
full
within
Nx
high
knowledge
context
study
equation
directly
OGAS
parameter space
approaches
cost
correspond
physics
field
implementation
adaptive
training sample
take
generalisation
work
line
colour
setup
training iteration
Nt
gradient
convergence
achieve
given
trajectories
computed
Buffer
large scale
based
system
discretisation
perform
overall
dimension
range
difference
ST
complex
better
set
surrogate model
GPU
pool based
component
offline
operator
behaviour
compute
corresponding
trained
capture
dynamic
challenge
include
contribution
Deep Learning
loss value
Xj
computational
domain
task
differential equation
per
scale
step
Server
information
thesis
approach
optimisation
sampled
current
APEBench
DL
region
must
timestep
mean
many
ensemble
selection
large
PDE case
Melissa
batch
resource
perspective
improvement
output
metric
compared
residual
required
Client
generator
data driven
next
storage
average
significant
well
learning
problem
setting
configuration
change
regime
thus
enable
memory
different
generated
comparison
present
efficient
total
create
goal
term
Neural Operator
bottleneck
challenging
training set
new
without
hyperparameter
specific
small
reduce
size
delta loss
numerical solver
number
grid
uniform
defined
resolution
train
random
Machine Learning
f
ensure
fixed
concept
collocation point
spatial
uniform sampling
introduce
previous
solver input
algorithm
result
parameter
across
strategy
error
single
resampling
designed
accuracy
neural network
application
limitation
deep surrogate

Synopsis

Numerical simulations of partial differential equations (PDEs) are fundamental tools in scientific computing, enabling the study of complex physical systems from climate modelling to drug discovery. However, their computational expense often limits applicability when large ensembles of simulations are required. Deep learning-based surrogate models have recently shown orders of magnitude faster inference whilst maintaining acceptable accuracy, yet training such surrogates faces significant efficiency challenges arising from the synthetic data paradigm, wherein training data must be generated with expensive numerical solvers.

This thesis addresses the training efficiency problem from a data-centric perspective, focusing on two interconnected bottlenecks. First, the large scale of simulation data imposes severe storage and input-output constraints in traditional offline training workflows, where datasets must be generated, written to disk, and subsequently loaded for training. Second, the complex, high-dimensional solver input parameter spaces suffer from the curse of dimensionality, leading to often non-representative training datasets, whose size is limited by available storage.

To address the storage bottleneck, this work leverages an online training approach through the Melissa framework, which couples data generation and neural network (NN) training in a streaming fashion, transforming the training process from storage-bound to compute-bound. This approach eliminates the need for storage of large datasets, their writing and reading, and enables steering of data generation adaptive to the evolving state of the surrogate model during training. With storage no longer a bottleneck, the limiting factor becomes how to spend the simulation budget wisely.

The thesis makes two primary contributions. First, an active learning (AL) method called Breed is developed for physics-informed NNs in the context of collocation point selection, employing loss-driven importance sampling to concentrate training data in regions of high prediction error whilst maintaining exploration through a scheduled balance. This approach addresses the exploration-exploitation dilemma and mitigates catastrophic forgetting through an epsilon-greedy-inspired strategy with curriculum learning principles. Experiments on benchmark problems including the Allen-Cahn equation demonstrate superior convergence and stability compared to existing residual-based approaches.

Second, building upon this foundation, the online variant of Breed is developed specifically for data-driven surrogates within the online training setting. Breed is adapted to accommodate concurrent data generation constraints and autoregressive temporal predictions. As the method uses per-sample loss calculated during NN optimisation steps, it is agnostic to the surrogate model and light-weight. In turn, the Melissa framework is advanced with an active sampling mechanism that enables efficient data creation steering along the training process. The mechanism operates as a feedback loop: an AL algorithm adapts a sampling distribution over the input parameter space in response to the surrogate's training progress, and new solver input parameters are drawn from this distribution

to generate the next simulation data.

Extensive experiments demonstrate effectiveness across multiple PDEs including the heat transfer, Kuramoto-Sivashinsky, and Korteweg-de Vries equations, with varying degrees of nonlinearity and chaos. Results consistently show that Breed significantly reduces the variance and maximum values of test errors compared to uniform baseline sampling, improving reliability and robustness of surrogate predictions at minimal cost.

Extending the previous contribution, the thesis introduces OGAS, a generative active learning method that replaces the Breed's direct importance sampling with a neural generative model for efficient approximation of the evolving distribution over informative solver parameters. Larger scale experiments show that OGAS yields more reliable surrogates across diverse PDEs and architectures than Breed and adapted pool-based AL baselines.

On the whole, the thesis contributes to the challenge of efficient training of deep surrogates for PDE simulations by addressing it from a data-centric perspective. This work aspires to pave the way to the use of deep learning for scientifically challenging and societally significant applications at large scale without ever-growing compute resource budgets and negative environmental impact.

Thesis Organisation

The thesis is structured in two parts: Part I for background, which consists of four chapters that progressively narrow from fundamental concepts to the specific efficiency questions that the thesis addresses, and Part II for contributions, which consists of three chapters presenting the contributions of the thesis. We illustrate the logic of the thesis organisation in Figure 1.

Remark on LLM Assistance

I would like to acknowledge the use of large language models (LLMs) and code assistants as support tools during the preparation of this thesis. They helped me express and refine ideas, especially in English writing and in managing some ADHD-related organisational challenges, but did not generate new scientific ideas, research questions, or conclusions. All research contributions, methodological choices, and interpretations are my own, and I used these tools selectively and mindfully, recognising the environmental and societal costs of large-scale AI systems. In this sense, LLMs and code assistants functioned as powerful "horses" that can enhance a researcher's capabilities, while direction and responsibility remain with the human rider.

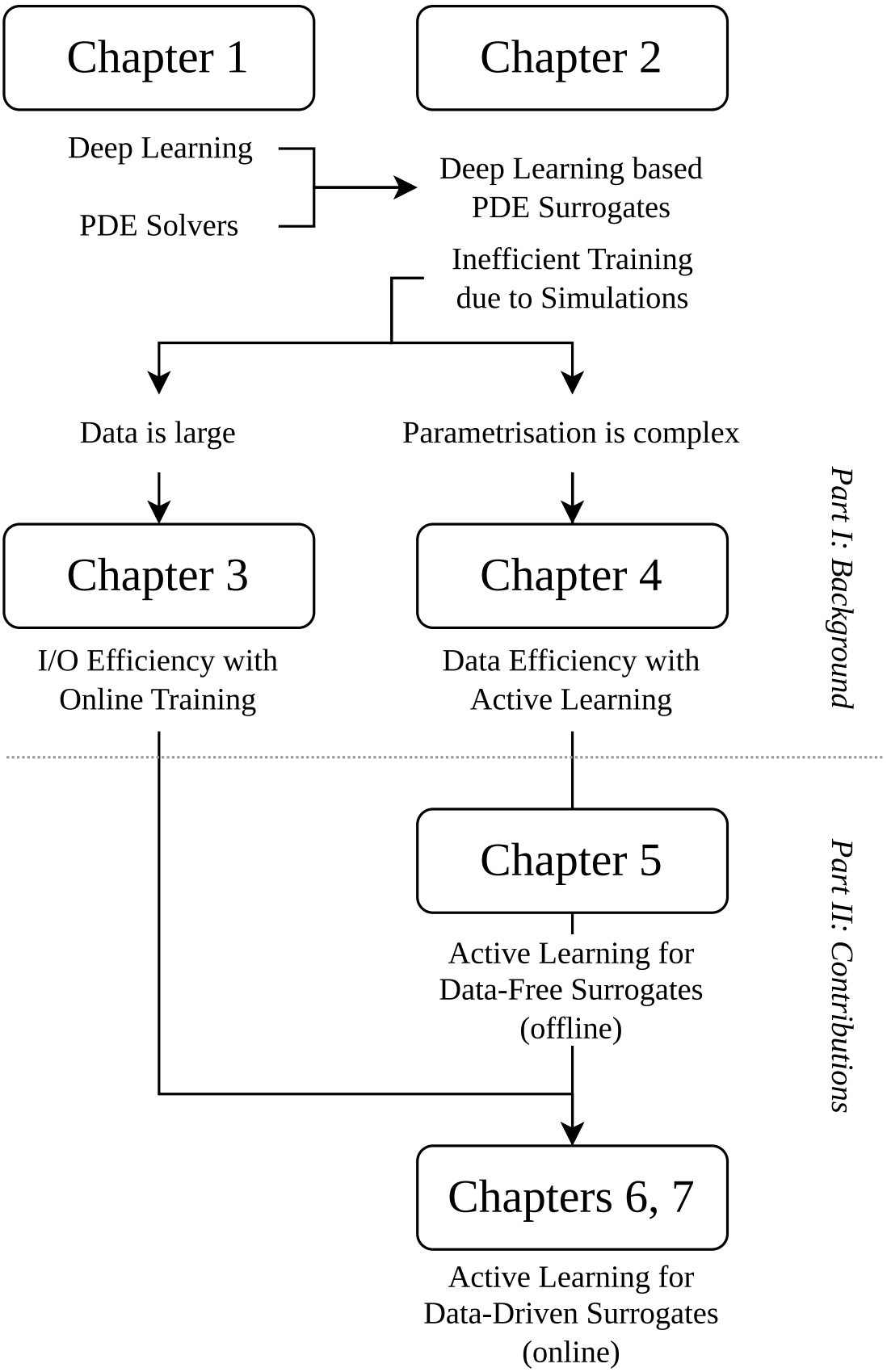


Figure 1: Thesis organisation logic.

Contents

Synopsis iii

Lists of Contents ix
- Figures . . . ix
- Tables . . . xi
- Algorithms . . . xii
- Notations . . . xiii

I Background 1

1 Introduction to Deep Learning and Simulations 5
- 1.1 Deep Learning . . . 5
  - 1.1.1 Supervised Learning Setting . . . 6
  - 1.1.2 Neural Networks Basics . . . 8
  - 1.1.3 Successes of Deep Learning from Compute Perspective . . . 9
- 1.2 Differential Equations and Simulations . . . 9
  - 1.2.1 Example of Differential Equation and Numerical Solving Method . . . 10
  - 1.2.2 General PDE Notation and Terminology . . . 14
  - 1.2.3 General Numerical Solving Method Notation and Terminology . . . 15
  - 1.2.4 Scientific Computing Applications . . . 17
- 1.3 Conclusion . . . 20

2 Deep Learning for Simulations 21
- 2.1 Roles of Deep Learning in Surrogate Modelling . . . 22
  - 2.1.1 Solver Surrogates: Faster Forward Inference . . . 22
  - 2.1.2 Extended Surrogates: Beyond Forward Inference . . . 23
- 2.2 Deep Surrogates Methodology . . . 24
  - 2.2.1 Methods Categorisation . . . 25
  - 2.2.2 Handling Temporal Dimension . . . 25
- 2.3 State-of-the-Art Deep Surrogates . . . 27
  - 2.3.1 Physics-Informed NNs . . . 28
  - 2.3.2 Neural Operators . . . 29
  - 2.3.3 Recent Advances . . . 30
- 2.4 Synthetic Data Bottleneck . . . 32
  - 2.4.1 Large Data Scale . . . 32
  - 2.4.2 Complex Input Parameters Space . . . 33
  - 2.4.3 Mitigating the Bottleneck . . . 34
- 2.5 Conclusion . . . 34

3 I/O Efficiency of Deep Surrogates Training 35
3.1 Offline Training . . . 35
3.1.1 Storage and Compute Budget Limits . . . 35
3.1.2 Compute Budget: Dataset Creation . . . 36
3.1.3 Compute Budget: Surrogate Training . . . 36
3.1.4 Competing for Compute . . . 37
3.2 Online Training . . . 38
3.2.1 Defining Online . . . 38
3.2.2 Comparison to Offline Training . . . 39
3.2.3 Challenges . . . 40
3.2.4 Related Work . . . 41
3.3 Melissa Framework . . . 42
3.3.1 Melissa Components . . . 42
3.3.2 Training Buffer . . . 46
3.3.3 Empirical Results . . . 47
3.3.4 Discussion . . . 47
3.4 Conclusion . . . 49

4 Compute Efficiency with Active Learning (AL) 51
4.1 Improving Data Efficiency with AL . . . 52
4.2 Related Concepts . . . 53
4.3 Classical AL Methods . . . 55
4.4 AL for PINN Collocation Points Selection . . . 56
4.5 AL for Surrogates Data Generation . . . 57
4.6 Conclusion . . . 60

II Contributions 61

5 Adaptive Sampling of Collocation Points for PINNs 65
5.1 Background . . . 66
5.1.1 Problem Setup . . . 66
5.1.2 Related Work . . . 66
5.2 Methodology of Breed . . . 67
5.2.1 Sampling Algorithm . . . 68
5.2.2 Discussion . . . 70
5.3 Experiments . . . 72
5.3.1 Benchmarks . . . 72
5.3.2 Results . . . 75
5.3.3 Discussion . . . 76
5.4 Conclusion . . . 77

6 Online Active Learning for Data-Driven Surrogates 83
6.1 Online Training with AL Design . . . 83
6.1.1 Pillars of Online AL Method . . . 84
6.1.2 Requirements to Active Sampling Mechanism . . . 85
6.2 Melissa Active Sampling with Breed . . . 86
6.2.1 Necessary Modifications to Breed . . . 86
6.2.2 Melissa with Active Sampling Mechanism . . . 87

6.2.3 Online Breed Method within Melissa . . . 90
6.2.3.1 Delta-loss statistic for acquisition scores . . . 90
6.2.3.2 Sampling new solver input parameters . . . 93
6.2.3.3 Out-of-bounds handling . . . 93
6.2.4 Conclusion . . . 94
6.3 Experiments for Direct Linear PDE Surrogate . . . 95
6.3.1 Experimental Setup . . . 95
6.3.2 Experimental Results . . . 96
6.3.3 Sampled Parameters Distribution Analysis . . . 99
6.3.4 Conclusion and Limitations . . . 100
6.4 Advanced Experiments for Autoregressive Nonlinear PDE Surrogates . . . 102
6.4.1 Improved Active Sampling Mechanism . . . 102
6.4.2 Breed Hyperparameters Regimes . . . 105
6.4.3 Experimental Setup . . . 106
6.4.3.1 Suite of PDE cases . . . 106
6.4.3.2 Surrogate training details . . . 110
6.4.3.3 Evaluation details . . . 118
6.4.4 Experimental Results and Analysis . . . 120
6.4.4.1 Overall performance from distributional perspective . . . 120
6.4.4.2 Dynamics across training iterations . . . 124
6.4.4.3 Regions of interest in the parameter space . . . 128
6.4.5 Extended Experiments . . . 128
6.4.6 Discussion and Limitations . . . 129

7 Generative-based Active Learning 135
7.1 Online Generative Active Sampling . . . 136
7.2 Problem Setting . . . 137
7.3 Methodology . . . 137
7.3.1 Training Signal . . . 138
7.3.2 Signal-conditional Generator . . . 139
7.3.3 Parameters Sampling Distribution . . . 140
7.3.4 History-bias Correction . . . 140
7.4 Experimental Setup . . . 141
7.4.1 Data Generation . . . 141
7.4.2 Compared Methods . . . 142
7.4.3 Surrogates Training and Validation . . . 144
7.5 Experimental Results . . . 145
7.5.1 Additional Experiments . . . 148
7.6 Conclusion . . . 149

Conclusion 159

A Carbon Footprint 167

Bibliography 173

List of Figures

1 Thesis organisation logic. v

1.1 Heat transfer in a rod example: Discretisation of the 1D spatial domain. . 12
1.2 2D discretisation grid example and notations. 16
1.3 Ensemble forecasting capturing uncertainty in Melissa hurricane trajectory predictions. 18

2.1 Burgers equation solution and example of collocation and data points for PINN training. 28
2.2 Walrus surrogate model predictions vs ground truth for various 3D simulations. 31

3.1 Offline training workflow: data are generated upfront and written to disk, then data are read from disk during training; residual I/O overhead stalls both generation and training compute. 40
3.2 Online training workflow: training is overlapped with data generation, data from solvers are sent as soon as produced directly to the training process, avoiding storage and eliminating I/O overhead. 41
3.3 The Melissa framework architecture for online training of data-driven surrogates. Thanks to Abhishek Purandare for the original figure. 43
3.4 Client code example with Melissa API. 44
3.5 Comparison between offline and online training for a 2D heat transfer surrogate. 48

5.1 Breed sampling algorithm illustration. 69
5.2 Illustration of the IPGD surface and two toy-simulator trajectories starting from different points. 73
5.3 Illustration of the sensitive region on the IPGD surface. 73
5.4 Example of the 1D Allen Cahn equation solution. 74
5.5 IPGD benchmark: Validation loss over training iterations for Breed, R3 and uniform sampling. 78
5.6 Allen Cahn benchmark: Validation error over training iterations for Breed, R3 and uniform sampling. 79
5.7 IPGD benchmark, Breed: Loss distribution over the input domain. 80
5.8 IPGD benchmark, uniform resampling: Loss distribution over the input domain. 81

6.1 Illustration: active sampling mechanism within Melissa. 88
6.2 Intermediate experiments, Exp1: training and validation loss curves for different model configurations. 97

6.3 Intermediate experiments, Exp2 & Exp3: training and validation loss curves for different Breed hyperparameter values. 98
6.4 Intermediate experiments: example of trajectories with low or high difference in input temperature parameters. 100
6.5 Intermediate experiments: comparison of solver input parameters distribution across different sample categories (Uniform vs. Proposal samples for Breed, and Breed vs. Random runs). 101
6.6 Illustration: improved active sampling mechanism within Melissa. 104
6.7 Example trajectories of KdV with different difficulty parameters. 111
6.8 Example trajectories of KS (2 waves) with different difficulty parameters. . 112
6.9 Example trajectories of KS (2 waves, normalisation) with different difficulty parameters. 113
6.10 Example trajectories of KS (3 waves) with different difficulty parameters. . 114
6.11 Example trajectories of KS (3 waves, normalisation) with different difficulty parameters. 115
6.12 Example trajectories of conservative KS with different difficulty parameters. 116
6.13 Advanced experiments: Assets organisation diagram. 118
6.14 Advanced experiments: distribution of the test set losses over the initial condition parameters. 120
6.15 Advanced experiments: test MSE loss ECDF comparison across methods and PDE cases. 123
6.16 Advanced experiments: comparison of test loss evolution during training with statistical significance for a subset of experiments. 125
6.17 Advanced experiments: test metrics evolution across training iterations for different methods (5 PDE cases). 126
6.18 Advanced experiments: test metrics evolution across training iterations for different methods (continued, 5 PDE cases). 127
6.19 Advanced experiments: sampled parameters distribution across resampling iterations with broad sampling. 130
6.20 Advanced experiments: sampled parameters distribution across resampling iterations with precise sampling. 131
6.21 Extended advanced experiments: comparison of test metrics evolution during training with statistical significance for uniform baseline and Breed. . . 132
6.22 Extended advanced experiments: comparison of test loss evolution during training for uniform baseline and Breed. 133

7.1 OGAS within active sampling of Melissa. 136
7.2 OGAS experiments: examples of states for three distinct dynamics PDEs. . 141
7.3 OGAS experiments: example of sampled parameter distribution evolution. 145
7.4 OGAS experiments visual summary: distributional statistics of RMSE errors for each PDE, surrogate architecture, and method. 146
7.5 OGAS vs. Breed vs. SBAL overheads during online training. 148
7.6 OGAS experiments: trajectory prediction examples for Navier-Stokes. . . . 155
7.7 OGAS experiments: trajectory prediction examples for Kuramoto-Sivashinsky. 156
7.8 OGAS experiments: trajectory prediction examples for Gray-Scott. 157

List of Tables

6.1 Intermediate experiments: fixed and varied hyperparameters for each experiment. . . . 96
6.2 Breed regimes hyperparameters. . . . 106
6.3 Advanced experiments: PDE difficulty coefficients. . . . 109
6.4 Advanced experiments: comparison of test metrics across PDE cases and compared methods 122

7.1 OGAS experiments summary: relative improvement of test metrics compared to uniform baseline. . . . 147
7.2 OGAS experiments summary (single-model): relative improvement of test metrics compared to uniform baseline. . . . 149
7.3 OGAS experiments: Navier-Stokes results. . . . 151
7.4 OGAS experiments: Kuramoto-Sivashinsky results. . . . 152
7.5 OGAS experiments: Gray-Scott results. . . . 153
7.6 OGAS additional 1D experiments: across PDEs and ablation on bias correction. . . . 154
7.7 OGAS additional 1D experiments: ablation on buffer size and resampling frequency. . . . 154

List of Algorithms

1 Offline Breed AL method: Training points sampling algorithm. 68
2 The Melissa framework with active sampling mechanism. 89
3 Online Breed AL method: Solver input parameters sampling. 91
4 The Melissa server with improved active sampling mechanism. 103
5 OGAS: Online Generative Active Sampling 138
6 SBAL adaptation for online training. 143

List of Notations

$\boldsymbol{x}$	Vector
$\boldsymbol{X}$	Matrix/tensor
$\mathcal{B}$	Set
$f(\cdot)$	Function
$F[\cdot]$	Operator

Scientific Computing

$u(\boldsymbol{x}, t)$	Solution function of a PDE
$\mathcal{U}$	Spatio-temporal domain $\mathcal{U}_x \times [0, T] \subset \mathbb{R}^{d+1}$
$\mathcal{U}_x$	Spatial domain $\mathcal{U}_x \subset \mathbb{R}^d$
$\partial\mathcal{U}$	Boundary of the domain $\mathcal{U}$
d	Spatial dimensionality
T	Time horizon
$\boldsymbol{x}$	Spatial coordinate vector $(x_1, \ldots, x_d)$
$F[\cdot]$	PDE differential operator, $F[u](\boldsymbol{x}, t; \boldsymbol{\lambda}) = 0$
$B[\cdot]$	Boundary condition operator, $B[u](\boldsymbol{x}, t; \boldsymbol{\lambda}_B) = 0$
$g(\cdot)$	Initial condition function, $u(\boldsymbol{x}, 0) = g(\boldsymbol{x}; \boldsymbol{\lambda}_0)$
$\boldsymbol{\lambda}$	Parameter vector of the PDE system
$\boldsymbol{\lambda}_B, \boldsymbol{\lambda}_0$	Boundary condition and initial condition parameters, respectively
Λ	Input parameter space, $\Lambda \subset \mathbb{R}^{n_p}$
n_p	Dimensionality of the input parameter space Λ
$S(\cdot)$	Numerical solver mapping configurations to solution trajectories
h	Discretisation configuration (Δ_x, Δ_t)
Δ_x, Δ_t	Spatial and temporal step sizes

N_x Number of spatial grid points per dimension

N_t Number of time steps in a trajectory

$\boldsymbol{X}^t$ Discrete solution state (spatial field) at time step t

$\boldsymbol{X}$ Solution trajectory $\{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^{N_t}\}$

$\boldsymbol{X}^{[t]}$ Training sample consisting of multiple trajectory steps associated with time step t

Deep Learning

$f_{\boldsymbol{\theta}}(\cdot)$, $f(\cdot; \boldsymbol{\theta})$ Neural network model parameterised by $\boldsymbol{\theta}$

$\boldsymbol{\theta}$ Model parameters vector, $\boldsymbol{\theta} \in \mathbb{R}^P$

P Model size (number of parameters)

N Dataset size (number of samples)

$\mathcal{D}$ Dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$

$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$ Training and test datasets

$p_\star(\cdot)$ True (population) data distribution

$p_{\text{data}}(\cdot)$ Empirical data distribution

$\hat{\boldsymbol{y}}$ Model prediction of $\boldsymbol{y}$ for input $\boldsymbol{x}$

$\ell(\cdot, \cdot)$ Per-sample loss function

$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ Empirical risk (average loss over dataset $\mathcal{D}$)

B Mini-batch size

Deep Surrogates

$f_{\boldsymbol{\theta}}^{(k)}$ k sequential (autoregressive) applications of the NN model

$r_{\boldsymbol{\theta}}(\boldsymbol{x})$ PDE residual evaluated at the NN output (PINNs)

$\mathcal{L}_r(\cdot)$, $\mathcal{L}_u(\cdot)$ Residual and data losses (PINNs)

$\mathcal{S}$ Set of collocation points (PINNs)

γ_s Linear difficulty coefficient for the s-th order spatial derivative (APEBench)

δ_{op} Nonlinear difficulty coefficient for operator op (APEBench)

Online Training

N_S Number of concurrent solver clients (Melissa)

N_B Buffer capacity (number of training samples)

$\mathcal{B}_{\text{buf}}$ Training buffer in online training (Melissa)

$\mathrm{S}, \mathrm{L}, \mathrm{C_j}$ Melissa Server, Launcher, and Client processes

$\mathrm{S_D}, \mathrm{S_T}, \mathrm{S_R}$ Melissa Server threads: data Receiver, Trainer, and Resampler

Active Learning

$\mathcal{S}_c$ Concentration set: points sampled in a loss-driven manner (Breed)

$\mathcal{S}_u$ Uniform set: points sampled uniformly for exploration (Breed)

$\varepsilon^{(i)}$ Concentrate-explore control value at resampling round i (Breed)

N_c, N_u Number of concentrated and uniform points ($N_c = \lfloor N \cdot \varepsilon^{(i)} \rfloor$, $N_u = N - N_c$)

$\varepsilon^0, \varepsilon^c, c$ Concentrate-explore schedule hyperparameters (Breed)

σ Neighbourhood width for Gaussian sampling (Breed); $\boldsymbol{\sigma}$ for dimension-specific widths

p Resampling period (number of training iterations between resamplings)

δl_{jti} Delta-loss statistic for sample $\boldsymbol{X}_j^{[t]}$ at training iteration i

δl_j Aggregated delta-loss statistic for simulation j (over time steps and iterations)

$R[N, N_t]$ Recording dictionary storing aggregated delta-loss values

$\mathcal{Q}$ Recency queue of size N_p storing unique simulation indices

N_p Sliding window size for the recency queue (online Breed)

M Number of new solver input parameters sampled per resampling event

β Width-reduction factor for adaptive OOB handling ($\beta \in (0, 1)$)

Part I

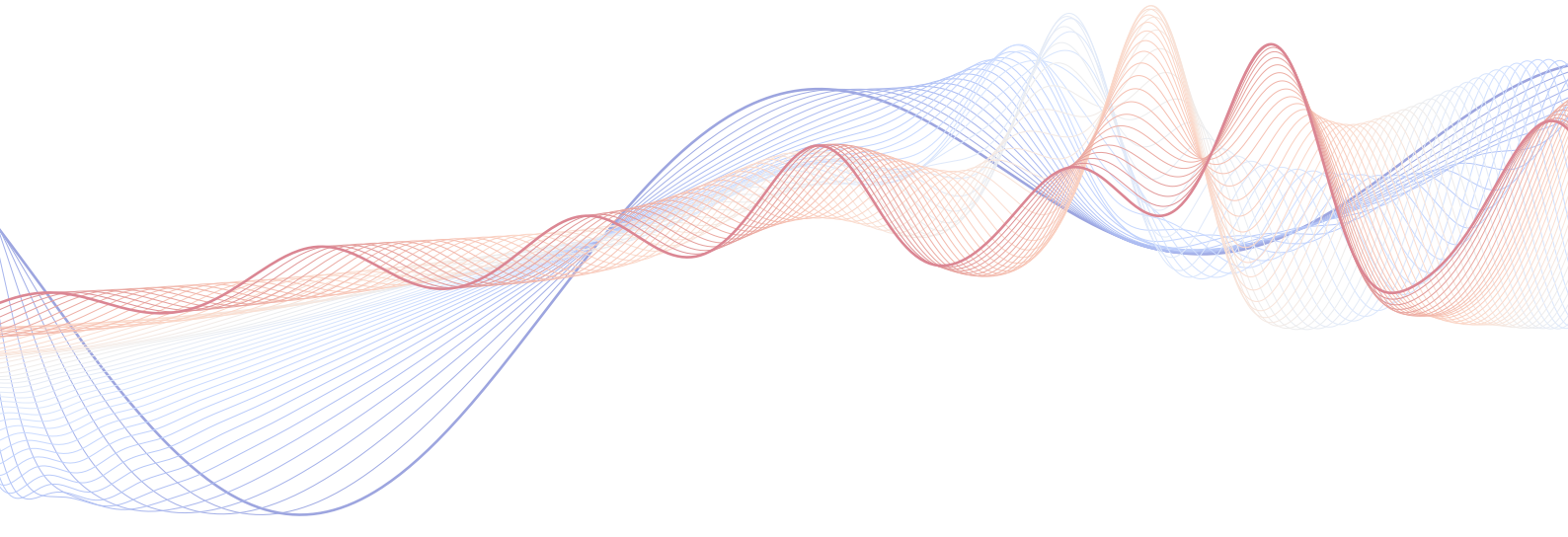
Background

Introduction to Part I

Part I serves as the foundational background for the thesis, introducing the terminology, notation, and related literature to provide the necessary context to position the thesis contributions presented in Part II. The part is organised into four chapters that progressively narrow from fundamental concepts to the specific efficiency questions that the thesis addresses.

- Chapter 1 covers prerequisites in deep learning and scientific computing, from neural networks basics to numerical simulations of Partial Differential Equations (PDEs).
- Chapter 2 lays out the principles of deep surrogate models for PDEs, discussing their motivation, state-of-the-art methods, limitations, and synthetic data specifics.
- Chapter 3 examines I/O efficiency in deep surrogate training, contrasting offline and online training paradigms, which transforms the process from storage-bound to compute-bound.
- Chapter 4 explores data-centric compute efficiency via active learning (AL), reviewing classical methods, prior work on AL for deep surrogates, and their limitations.

The part concludes by identifying a research gap that motivates the thesis contributions and methodological chapters that follow in Part II.



Chapter 1

Introduction to Deep Learning and Simulations

This chapter provides the reader with the necessary background on deep learning (DL) and scientific computing by introducing the key concepts, terminology, and notation used throughout the rest of the thesis. It is organised into two sections: Section 1.1 is dedicated to DL fundamentals, which include supervised learning objectives and neural networks training; Section 1.2 is dedicated to scientific computing fundamentals, which include partial differential equations, numerical methods for solving them, and simulation applications. The chapter concludes by discussing limitations of numerical simulations that motivate the use of DL-based surrogates, which are covered in the next chapter.

1.1 Deep Learning

In this section, we start with a brief introduction to machine learning fundamentals, specifically focusing on supervised learning from a probabilistic perspective. We then present deep learning (DL) models concepts, i.e., neural networks and their optimisation, and conclude with a discussion of DL successes from a computational perspective. We take [157] and [65] as the base references for this section.

Machine Learning (ML) is a subfield of artificial intelligence (AI) that develops mathematical algorithms and computer programmes able to automatically learn patterns and representations from data, and extrapolate that knowledge to new data. We start with a formal definition by Mitchell, 1997 [151]:

> *"Machine learning algorithms are defined as algorithms that given a task T, performance measure P, and experience E, can learn from E to improve their performance P on the task T for unseen experiences."*

Based on the type of experience, ML algorithms are categorised into three main approaches: supervised, unsupervised, and reinforcement learning.

In supervised learning, the experience E is a collection of input-output pairs, features and targets, which is called *dataset*. The goal of supervised learning is to learn a function that maps inputs to outputs based on the provided dataset, e.g., regression tasks with continuous targets and classification tasks with discrete targets. The targets in supervised

learning are often provided by human experts or derived directly within the data collection process.

In unsupervised learning, the experience E is a dataset of only input data points. The goal of unsupervised learning is to learn patterns or structures in the data, e.g., clustering tasks and anomaly detection, or to learn a representation of the data that captures its essential features, e.g., dimensionality reduction tasks. The targets in unsupervised learning are often derived from the data itself or from other sources.

In reinforcement learning, the experience E consists of interactions with an environment, where an agent takes actions and receives rewards. The goal of reinforcement learning is to learn a policy that maximises the cumulative reward over time.

In this thesis, we primarily focus on supervised learning methods, the core concepts of which are introduced below.

1.1.1 Supervised Learning Setting

Further terminology and notation are largely adopted from the ML textbook by Murphy, 2012 [157].

The goal of supervised learning is to learn a mapping f from inputs to outputs based on a given dataset by improving a performance measure P.

The dataset is a collection of N input-output pairs, $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathcal{X}$ are the inputs (features) and $\boldsymbol{y}_i \in \mathcal{Y}$ are the outputs (targets). Typically, the inputs are vectors, matrices, or tensors of real numbers, i.e., $\mathcal{X} = \mathbb{R}^{D_x}$, otherwise the inputs are preprocessed and transformed into such format. The outputs can be discrete or continuous, depending on the task. We consider only regression tasks in this thesis, where $\mathcal{Y} = \mathbb{R}^{D_y}$. This collection is drawn from an unknown true data distribution $p_\star(\boldsymbol{x}, \boldsymbol{y})$, also called *population distribution*. The dataset distribution is called *empirical distribution* $p_\mathcal{D}(\boldsymbol{x}, \boldsymbol{y})$.

The mapping $f : \mathcal{X} \to \mathcal{Y}$, called *model*, is typically parameterised as $f(\boldsymbol{x}; \boldsymbol{\theta})$ (or $f_\theta(\boldsymbol{x})$, or just $f_{\boldsymbol{\theta}}$) by a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^P$, where P is the model size. We often denote with $\hat{\boldsymbol{y}}$ the model's output, also called *prediction*, for the input $\boldsymbol{x}$, when it is not ambiguous to the parameters $\boldsymbol{\theta}$.

The performance measure, called *loss function*, indicates the discrepancy between the model's predictions $\hat{\boldsymbol{y}}$ and the target $\boldsymbol{y}$. The most common loss function for regression tasks is the squared error (SE):

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) := ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2, \tag{1.1}$$

where $\|\cdot\|_2$ is the L_2 norm and defined as:

$$||\boldsymbol{a}||_2 = \sqrt{\sum_{i=1}^{D} a_i^2}, \quad \boldsymbol{a} \in \mathbb{R}^D. \tag{1.2}$$

The average loss over the training dataset $\mathcal{D}$, called *empirical risk*, is defined as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})), \tag{1.3}$$

and for the squared error loss it is also called *mean squared error* (MSE).

The *model fitting* or *training* problem is to find the optimal parameters $\boldsymbol{\theta}^\star$ that minimise the empirical risk. Then the training objective is the following optimisation problem, called *empirical risk minimisation* (ERM):

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}). \tag{1.4}$$

The ultimate goal of supervised learning is to learn a model that generalises well to unseen data points, i.e., the model performs well on data points that are not in the training dataset. The generalisation performance is measured by the expected loss over the true data distribution, called *population risk*:

$$\mathcal{L}(\boldsymbol{\theta}; p_\star) = \mathbb{E}_{p_\star}\left[\ell(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))\right] \tag{1.5}$$

While we hope that the dataset is representative of the true data distribution, it is often not the case in practice due to the limited size of the dataset and potential biases in data collection. The bigger the gap between the empirical and true data distributions, the harder it is for the model to generalise to unseen data points. The difference between the empirical and population risks, $\mathcal{L}(\boldsymbol{\theta}; p_\star) - \mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$, is called *generalisation gap*. High generalisation gap indicates an *overfitting* of the model to the training dataset.

As we do not have access to the unknown $p_\star$, we can not calculate and minimise the population risk. Instead, we typically split the dataset into two disjoint subsets: the training set $\mathcal{D}_{train}$ and the test set $\mathcal{D}_{test}$. The training set is used to fit the model, while the test set is used to evaluate its performance on unseen data. We define the *test risk*, which we use to approximate the population risk, as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{test}) = \frac{1}{|\mathcal{D}_{test}|} \sum_{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}_{test}} \ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})). \tag{1.6}$$

High error on both training and test sets indicates an *underfitting* of the model, which means that the model was not trained long enough or itself is not complex enough to capture underlying data mapping. Model's complexity is also referred as *cognitive capacity* or ability and is related to the model size P and architecture.

Both underfitting and overfitting phenomena can be interpreted through the bias-variance dilemma [157, 109]. It states that the expected prediction error can be decomposed into three components: bias, variance, and noise (irreducible error imposed by the data); for MSE it is expressed as:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[(\boldsymbol{y} - f(\boldsymbol{x}; \theta))^2\right] &= \text{Bias}^2 + \text{Variance} + \text{Noise} = && (1.7)\\
&= \left(\mathbb{E}_{\mathcal{D}}\left[f(\boldsymbol{x}; \theta)\right] - \mathbb{E}_{p_\star}\left[\boldsymbol{y}|\boldsymbol{x}\right]\right)^2 + && (1.8)\\
&\quad + \mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}\left[f(\boldsymbol{x}; \theta)\right] - f(\boldsymbol{x}; \theta)\right)^2\right] + && (1.9)\\
&\quad + \mathbb{E}_{p_\star}\left[\left(\boldsymbol{y} - \mathbb{E}_{p_\star}\left[\boldsymbol{y}|\boldsymbol{x}\right]\right)^2\right], && (1.10)
\end{aligned}$$

Underfitting is characterised by high bias and low variance, while overfitting – by low bias and high variance, leading to a trade-off between these two as the error is minimised. Finding the optimal balance is the ultimate goal of supervised learning.

This decomposition parallels to a total uncertainty decomposition into aleatoric and epistemic uncertainties [68]. The noise term corresponds to aleatoric uncertainty – unavoidable, irreducible uncertainty due to randomness in a system, and epistemic uncertainty – reducible uncertainty induced by the lack of knowledge about the system – is

related to both bias and variance terms. The epistemic uncertainty stems from the model's limited capabilities to capture the underlying data mapping, which can be reduced by increasing the model size, and from the limited representativeness of the training dataset, which can be reduced by increasing the training dataset size or improving its quality.

In subsection 2.2 of Chapter 2 we further discuss supervised learning within the context of scientific ML.

1.1.2 Neural Networks Basics

Deep learning (DL) is a subfield of machine learning that focuses on using a special type of parameterised model, i.e., deep neural networks (NNs), to learn complex patterns and representations from data. We give a brief introduction to basic NNs and their training here, and in Section 2.3 of Chapter 2, we give an overview of advanced architectures that are used in scientific ML.

Neural network (NN) is a combination of linear parameterised functions, called *layers*, organised in acyclic graph, with non-linear functions, called *activations*, applied element-wise to the outputs of layers. For example, a NN with a simple linear graph of layers, is a multi-layer perceptron (MLP) [183]. An MLP of L layers and activation function φ for regression task is defined as:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = h_L \circ \varphi \circ h_{L-1} \circ \cdots \circ \varphi \circ h_1(\boldsymbol{x}), \tag{1.11}$$

where $\circ$ is the function composition operator and a linear function h_i is defined as:

$$h_i(\boldsymbol{a}) = \boldsymbol{W}_i\boldsymbol{a} + \boldsymbol{b}_i, \tag{1.12}$$

where $\boldsymbol{W}_i \in \mathbb{R}^{D_i \times D_{i-1}}$ is the weight matrix and $\boldsymbol{b}_i \in \mathbb{R}^{D_i}$ is the bias vector for layer i. The input layer size is $D_0 = D_x$ and the output layer size is $D_L = D_y$, while for the intermediate layers, called *hidden layers*, the sizes $D_1, \ldots, D_{L-1}$, also called *widths*, are chosen. The collection of all weight matrices and bias vectors is the model's set of parameters $\boldsymbol{\theta}$ and the model size is $P = \sum_{i=1}^{L}(D_i \cdot D_{i-1} + D_i)$. The design of the MLP's architecture is the choice of L, hidden layers widths, and the activation function φ.

The nonlinearity of the activation function is what allows NNs to learn complex patterns in data. The most common choice for the activation function is the rectified linear unit (ReLU) [6]:

$$\varphi(x) = \max(0, x). \tag{1.13}$$

The optimisation of the empirical risk is typically done using gradient-based methods, such as stochastic gradient descent (SGD) [26] and its variants, e.g., Adam [102].

The optimisation is typically performed in an epoch-based manner with mini-batches of data points [124]. At each *epoch*, the training dataset is randomly split into small batches $\mathcal{D}_{batch}$ of size B. Then, total of N/B iterations of a gradient descent step is performed over these batches. Calculating the gradients over a small batch of data points serves two purposes. First, it reduces the computational cost of an optimisation step, when compared to calculation on the whole dataset. Second, it introduces noise in the gradients as the estimation is based on smaller set, which helps to escape local minima and improves generalisation. Compared to using a single data point (as in pure SGD), mini-batches provide a better estimation of the gradients.

Model parameters update rule, i.e., an optimisation step iteration, using mini-batch SGD is defined as:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_{\text{old}}; \mathcal{D}_{\text{batch}}), \tag{1.14}$$

where $\eta > 0$ is the learning rate. The update rule for Adam optimiser is more complicated as it uses adaptive learning rates for each parameter and momentum terms to accelerate convergence. The gradients of the loss function with respect to the model parameters are computed using backpropagation [183], which is an efficient algorithm for computing gradients in NNs, especially with auto-gradient libraries, such as PyTorch [166] and JAX [60].

The training process is performed for a fixed number of epochs given training budget or until convergence criterion is satisfied. The training configuration consists of the choice of the optimisation algorithm, its parameters like the learning rate η, and batch size B, as well as other hyperparameters, such as weight decay, learning rate scheduler, or weights initialisation scheme.

1.1.3 Successes of Deep Learning from Compute Perspective

The DL field originates as early as almost a century ago but its notable successes emerged only recently. The first reference to NNs dates back to 1943 [146], while the modern version of NNs was introduced in the 1980s [183]. The fast-growing progress in computing hardware [201], in particular graphics processing units (GPUs), has a significant impact on the advancement of DL starting in the 2010s. Apart from efficient parallelisation and high throughput, which enables faster training of large models on large datasets, the cost-effectiveness of GPUs has made DL more accessible, leading to a surge in research and applications [2]. Exponential growth is observed in number of publications and models developed in AI research [143], advancing the state-of-the-art DL methods and their adoption in various domains.

Since 2015, the field has witnessed a distinct and rapid rise of large-scale models [196], fundamentally transforming modern AI. Its beginning is marked with release of AlphaGo [199] and includes large-language models like GPT-3 [30], image generation models like DALL-E [177], and transformer models [45]. These large-scale models were developed and trained mainly by large industry corporations and research labs, requiring significant computational resources [17]. Overall, the compute used to train the most notable AI models has grown by 4.1 times per year since 2010 (4-5 times based on [195]), alongside comparable growth in training dataset sizes [143], enabling these breakthroughs.

These advancements have been enabled by the co-evolution of the DL field and computing hardware technologies, where advances of one shape the other's. Specialised hardware, accelerators, and infrastructures have been developed to meet the computational demands of DL workloads: tensor processing units (TPUs), AI chips, cloud AI services, AI-specific supercomputers [198, 170]. These innovations have enabled further scaling of DL methods, shifting their application focus from consumer-oriented tasks towards complex scientific problems, like protein structure prediction [94].

At the same time, the compute growth trend has raised concerns in the context of economic and environmental costs, and some studies suggest that its continuation may face physical and practical limits [213]. For scientific ML, this means designing models that maintain high accuracy even with limited compute resources.

1.2 Differential Equations and Simulations

In this section, we provide an introduction to scientific computing, in particular, we focus on partial differential equations and autoregressive mesh-based numerical solving

methods. We take [50], [66], and [55] as the base references for this section.

Scientific computing, also called computational science or scientific modelling, is one of the ways to study the nature's system of interest. The system may be studied through (1) observations, (2) experiments, (3) analytical solutions, and (4) simulations. *Observations* capture the system's behaviour without interference from the outside, e.g., weather observations, astronomical observations, etc. Capturing can be done with various instruments, e.g., cameras, sensors, telescopes. While it is the most accurate way to study the system, it is often limited by the system's accessibility, incurred costs, instruments precision and capabilities. *Experiments* manipulate the system to study its behaviour under different conditions, e.g. chemical experiments, manufacturing tests, etc. Associated costs and dangers are often a concern. Simple examples: we do not observe such weather phenomena as hurricanes or tornadoes often, as they are rare and dangerous; experimenting with nuclear reactions is extremely expensive and dangerous. As the world is governed by physical laws, we can often describe (or try to describe) the system with mathematical equations. But for most of the real-world systems, finding the analytical solution to these equations is impossible, for example, due to the complexity of the equations or the domain. What is left is to *solve these equations numerically*, i.e., approximating continuous space with discrete points, and simulate the system on the computer.

In the last decades, the numerical solving methods have matured and are widely used in many fields of science and engineering. With the advancement of computing power, running simulations has become quicker and cheaper, thus, more affordable and accessible. Supercomputers, large clusters of computational resources joined into a single infrastructure, have been built mainly for scientific computing purpose to run simulations of complex systems. Some simulations can take days or weeks to run over thousands of compute modes, which is still much faster and cheaper than real-world experiments. Though the accuracy of the solution approximations is affected by truncation, rounding, precision, and other errors, numerical simulations are the main tool for science and engineering. It enabled such achievements as weather prediction, aircraft design, drug discovery, and many more. So what are these equations? And how are they solved?

Differential equations are the mathematical way to describe the physical model of the system. The system has certain values of interests, e.g., temperature, pressure, velocity, etc., that change in space and time, meaning they can be written as functions of space and time. The changes in these values and their relations can be described with equations of derivatives, i.e., differential equations. Specifically, these equations are governed by the physical laws, e.g., conservation of mass, momentum, energy, etc. We will not dive into the details of the system design and the derivation of these equations, but rather focus on the equations themselves and their numerical solving methods.

1.2.1 Example of Differential Equation and Numerical Solving Method

We give a simple illustrative example. Throwing a ball in the air, we want to know its position at any time t, function $x(t)$. The ball is affected by the gravity force g, which causes the acceleration a downwards. The acceleration is the second derivative of the position x with respect to time t,

$$a = \frac{d^2x}{dt^2} = -g, \tag{1.15}$$

where g is the gravity constant. This simple equation can be solved analytically. But how would we solve it numerically? We can discretise the time in small steps Δt. What is derivative? It is the rate of change, i.e., $\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}$. To take second derivative, we do it twice:

$$\frac{d^2x}{dt^2} = \lim_{\Delta t \to 0} \frac{\frac{dx}{dt}(t + \Delta t) - \frac{dx}{dt}(t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{\frac{x(t+2\Delta t) - x(t+\Delta t)}{\Delta t} - \frac{x(t+\Delta t) - x(t)}{\Delta t}}{\Delta t}. \tag{1.16}$$

This simplifies to

$$\frac{d^2x}{dt^2} = \lim_{\Delta t \to 0} \frac{x(t + 2\Delta t) - 2x(t + \Delta t) + x(t)}{\Delta t^2}. \tag{1.17}$$

Taking one step back in time, removing one Δt, we get

$$\frac{d^2x}{dt^2} = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - 2x(t) + x(t - \Delta t)}{\Delta t^2}. \tag{1.18}$$

Then the derivative can be approximated:

$$\frac{d^2x}{dt^2} \approx \frac{x(t + \Delta t) - 2x(t) + x(t - \Delta t)}{\Delta t^2} = -g, \tag{1.19}$$

where approximation becomes more accurate as Δt becomes smaller. Rearranging the equation, we get the update rule:

$$x_{next} = 2x_{current} - x_{previous} - g\Delta t^2. \tag{1.20}$$

Knowing the initial position $x(0)$ and velocity $v(0)$ (which gives $x(-\Delta t)$), we can iteratively compute the position of the ball at any time step t. The programme would look like this:

```
# Initial conditions
x0 = 10.0  # initial position [m]
v0 = 10.0  # initial velocity [m/s]

# Constants
g = 9.81   # gravity [m/s^2]
T = 1.0    # total time [s]

# Discretisation
dt = 0.01  # time step [s]

# Initialising variables
xs = []
x_previous = x0 - v0 * dt
x_current = x0

# Time-stepping loop
for t in range(0, int(T / dt)):
    x_next = 2 * x_current - x_previous - g * dt * dt
    xs.append(x_next)
    x_previous = x_current
    x_current = x_next
```

The solution accuracy will depend on the time step Δt, smaller it is more accurate the solution is, but also more time steps are needed to reach the same desired time t, thus, the longer it takes to compute.

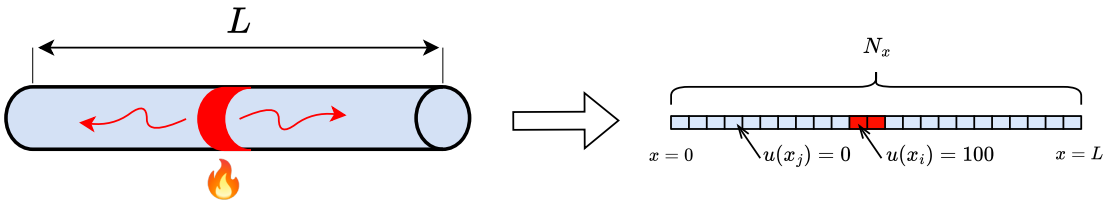


Figure 1.1: Illustrative example: Discretisation of the 1D spatial domain (a rod) into a mesh with grid points. The solution values are computed at the grid points.

However, for this very simple example, there is an analytical solution:

$$\begin{aligned} \frac{d^2x}{dt^2} &= a(t) = -g \\ \frac{dx}{dt} &= v(t) = \int (-g)dt = -gt + v_0 \\ x(t) &= \int (-gt + v_0)dt = -\frac{1}{2}gt^2 + v_0 t + x_0 \\ \boxed{x(t) &= -\frac{1}{2}gt^2 + v_0 t + x_0} \end{aligned} \tag{1.21}$$

It is easy to compute, and it does not require any iteration. But for complex systems integrals are intractable and analytical solutions are not available.

The described numerical method is called finite difference method, and the equation is an Ordinary Differential Equation (ODE), as it depends on a single variable, time t.

When the system is described by functions of multiple variables, Partial Differential Equations (PDE) are used. For example, the heat transfer equation depends on time and space, $u(x, t)$ is a temperature value of interest in space-time domain, and the equation is:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \tag{1.22}$$

where α is the thermal diffusivity constant. We use ∂ to denote partial derivatives, as the function depends on multiple variables, opposed to d for total derivatives. As in the previous example, the solution will depend on some initial state of the system, which now are defined over the spatial domain and time interval. For example, let the spatial domain be a metal rod of length L, i.e., $x \in [0, L]$, where the ends are insulated (meaning no heat goes through them), and there was a heating element at $100C^\circ$ in the centre of the rod, which was turned off at the start of the simulation. These conditions, boundary (BC) and initial (IC), can be written as functions. What we want is to find the temperature distribution across the rod during the next T seconds.

The numerical solving method is similar to the previous one, but now we need to discretise both time and space. The spatial domain is discretised into a mesh. Mesh is a collection of discrete cells, described by vertices, edges, and faces, which represent a larger continuous spatial domain, often of complex geometry. The simplest mesh is a regular rectangular grid, where vertices are spaced uniformly in each dimension, thus, eliminating the need to store the mesh structure explicitly. Further the solution values are computed at the vertices, i.e., grid points or interval end points. The complexity of computation depends on number of grid points and time steps, i.e., $O(N_x \times N_t)$, where

N_x is the number of spatial points and N_t is the number of time steps. See Figure 1.1 for the illustration.

Similarly, the derivatives are approximated using finite differences. The update rule for the heat equation would look like this:

$$u_i^{n+1} = u_i^n + \alpha \Delta t \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}, \tag{1.23}$$

where u_i^n is the value at spatial point i and time step n, Δx is the spatial step, and Δt is the time step. The programme of a simple explicit finite difference method for the heat equation would look like this:

```
# Initial conditions
T_c = 100.0  # initial temperature at the center [C]
T_rod = 0.0  # initial temperature of the rod [C]

# Constants
a = 0.01  # thermal diffusivity [m^2/s]
L = 10.0  # domain length [m]
T = 10.0  # total time [s]

# Discretisation
dx = 0.1
dt = 0.01
# Number of points
nx = int(L / dx) + 1
nt = int(T / dt)

# Initialising the field at t=0
u = [T_rod for _ in range(nx)]
u[int(nx / 2)] = T_c

# Time-stepping loop
for n in range(0, nt):
    u_next = u.copy()
    for i in range(1, nx - 1):
        u_next[i] = u[i] + a * dt / (dx * dx) \
                    * (u[i+1] - 2 * u[i] + u[i-1])
    u = u_next
```

For large grids, the computational and memory requirements are significant even for this simple example. The discretisation parameters have to be chosen fine enough to both capture the solution dynamics and satisfy the stability conditions. Given large spatial domain and long time interval, as it often is in real-world applications, the number of grid points and time steps increase largely, leading to high computational and memory costs, as well as long runtimes. To make it feasible, the computations can be parallelised over multiple processes by partitioning the grid, as the update of each grid point only depends on its neighbours at the previous time step, and by performing synchronisation on the partition boundaries after each time step. Thus, well-optimised distributed computing is essential to make running large-scale simulations feasible. As we already mentioned, high-performance computing (HPC) infrastructures (clusters, supercomputers) have been created mainly for this exact purpose.

There are other methods as well, e.g., implicit methods, which are more stable to discretisation parameters but require solving systems of equations (thus more computational

resources), spectral methods, which are more accurate for smooth solutions but require more complex implementation and sometimes special domain types.

The aforementioned methods use structured grids to represent the spatial domain and approximate the derivatives using the grid points. However, more complex domains require unstructured grids, including triangular and hexagonal, hybrid meshes, with splitting, or curvilinear, which can better represent complex geometries but require storing explicitly neighbourhood relationships and additional computational resources to build a mesh.

Instead of mesh-based methods, there are particles-based methods, which use a set of particles to represent the domain and approximate the derivatives. These methods are more flexible in handling complex geometries and moving boundaries, but they are often more computationally expensive. For further details we refer the reader to [55].

1.2.2 General PDE Notation and Terminology

Here we establish the general notation and terminology for PDEs used throughout the thesis, which is mainly adopted from [50] with some adaptations.

We have a function of interest $u(\boldsymbol{x}, t)$ defined on a spatial domain $\mathcal{U}_x \subset \mathbb{R}^d$, $\boldsymbol{x} \in \mathcal{U}_x$, and time interval $[0, T]$ for some time horizon $T > 0$, $t \in [0, T]$. We further denote spatio-temporal domain as $\mathcal{U}_x \times [0, T] =: \mathcal{U} \subset \mathbb{R}^{d+1}$. The spatial point is represented as $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$. The function u has continuous derivatives up to order k in $\mathcal{U}$, i.e., $u \in C^k(\mathcal{U})$.

A mapping from a set of functions to another set of functions is called an *operator*, for example, differentiation is an operator, which takes a function and returns its derivative which is also a function. More generally, a k-th order differentiation operator D^k is defined as:

$$D^k = \frac{\partial^{|k|}}{\partial x_1^{k_1} \partial x_2^{k_2} \ldots \partial x_d^{k_d}}, \tag{1.24}$$

where $k = (k_1, k_2, \ldots, k_d)$ is a multi-index with non-negative integers k_i and $|k| = k_1 + k_2 + \ldots + k_d$. Applying such operator to a function u (denoted by square brackets), we can denote all partial derivatives of order k as $D^k u$:

$$D^k u = \left\{ D^{k'}[u] \;\middle|\; \forall k' : |k'| = k \right\} \tag{1.25}$$

A PDE is an expression of these derivatives, the function u, its variables $\boldsymbol{x}$ and t, and parameters $\boldsymbol{\lambda} \in \mathbb{R}^n$, which includes PDE expression coefficients. The number n of these parameters depends on the problem being addressed. For example, in the heat transfer example, coefficient α is a parameter that affects the rate of heat diffusion but does not change the equation's form. If we want to study how different materials affect heat diffusion, we can treat α as a parameter and solve the PDE for different values of α. We introduce parameter vector $\boldsymbol{\lambda}$ to be able to represent family of PDEs with different parameter values in a compact form.

We then define a general form of parametric k-th order PDE like this:

$$F(D^k u, D^{k-1} u, \ldots, Du, u, \boldsymbol{x}, t; \boldsymbol{\lambda}) = 0, \tag{1.26}$$

for $(\boldsymbol{x}, t) \in \mathcal{U}$, where function F is given. This notation is explicit and formal. However, it is often simplified to a more compact form, which is common in DL literature:

$$F[u](\boldsymbol{x}, t; \boldsymbol{\lambda}) = 0, \tag{1.27}$$

The equation states that the solution function u must satisfy the relation defined by the *differential operator* F at every point in the domain $\mathcal{U}$.

The heat equation from before is an example of an initial-boundary value problem (IBVP). The problem is defined by the PDE and initial and boundary conditions (IC/BC), to which the solution is subject to. These conditions are what makes a solution unique.

The BC can be written as:

$$B[u](\boldsymbol{x}, t; \boldsymbol{\lambda}_B) = 0, \quad (\boldsymbol{x}, t) \in \partial\mathcal{U}, \tag{1.28}$$

where B is the boundary differential operator, $\boldsymbol{\lambda}_B \in \mathbb{R}^{n_B}$ are BC parameters, $\partial\mathcal{U}$ is the boundary of the domain $\mathcal{U}$. In the heat transfer example, the insulated rod ends BC can be written as $\frac{\partial u}{\partial x} = 0$ at $x = 0$ and $x = L$. The BC parameters $\boldsymbol{\lambda}_B$ can include the positions of the boundaries, e.g., 0 and L. This type of BC is called Neumann BC.

The BC can be of various types. Dirichlet BC specifies the functions values on the boundary, like $u(\boldsymbol{x}, t) = 0$. Neumann BC specifies the functions derivative values, like in the example. Robin BC combines both, like $\frac{\partial u}{\partial x} + u = 0$. Periodic BC specifies that the function values are the same on opposite sides of the boundary, like $u(0, t) = u(L, t)$ for a 1D domain of length L. Mixed BC uses different types of BC on different parts of the boundary.

The IC can be written as:

$$u(\boldsymbol{x}, 0) = g(\boldsymbol{x}; \boldsymbol{\lambda}_0), \quad \boldsymbol{x} \in \mathcal{U}_x, \tag{1.29}$$

where g is the initial condition function, and $\boldsymbol{\lambda}_0 \in \mathbb{R}^{n_0}$ are IC parameters. In the heat transfer example, the heating element temperature and position define the IC function g and its parameters.

Combining all together, the IBVP, defined by $(F, B, g; \boldsymbol{\lambda}, \boldsymbol{\lambda}_B, \boldsymbol{\lambda}_0)$ and domain $\mathcal{U}$ is written as:

$$\begin{cases} F[u](\boldsymbol{x}, t; \boldsymbol{\lambda}) = 0, & (\boldsymbol{x}, t) \in \mathcal{U}, \\ B[u](\boldsymbol{x}, t; \boldsymbol{\lambda}_B) = 0, & (\boldsymbol{x}, t) \in \partial\mathcal{U}, \\ u(\boldsymbol{x}, 0) = g(\boldsymbol{x}; \boldsymbol{\lambda}_0), & \boldsymbol{x} \in \mathcal{U}_x. \end{cases} \tag{1.30}$$

1.2.3 General Numerical Solving Method Notation and Terminology

Numerical solving methods approximate the solution of the PDE on a discrete set of points in the spatio-temporal domain.

While there are many numerical solving methods, in this thesis we work with finite difference methods with structured rectangular grids which solve the PDE sequentially in time, i.e., autoregressively. We give notations for this specific case.

Discretisation. Let's assume the spatial domain is a d-dimensional cube of length L in each dimension, i.e., $\mathcal{U}_x = [0, L]^d$. It is discretised into a uniform grid of N_x points in each dimension, thus, total N_x^d points. The spatial step is $\Delta_x = \frac{L}{N_x - 1}$. The time interval $[0, T]$ is discretised into N_t steps, where $\Delta_t = \frac{T}{N_t - 1}$. Then the spatio-temporal domain is discretised into a grid of $N_x^d \times N_t$ points. Let discretisation configuration be denoted[1] as $h = (\Delta_x, \Delta_t)$.

[1]The letter h comes from the *horizontal difference between points* that is used in the definition of differentiation, $x + h$, but here serves as a general notation for differences in each dimension.

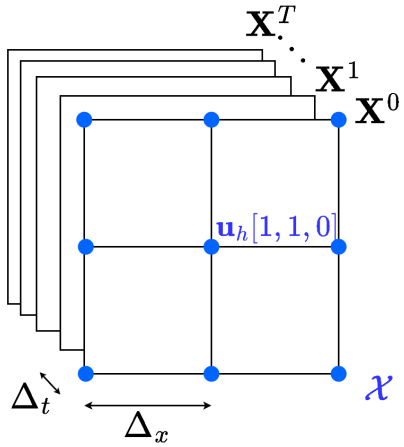


Figure 1.2: Example of discretisation grid for $N_x = 2$ grid points per dimensions and $d = 2$ spatial dimensions. Blue dots represent grid points where the solution values are computed, for example $u_h[1, 1, 0]$ is the value at spatial point $(1 \cdot \Delta_x, 1 \cdot \Delta_x)$ and time step 0.

The numerical solution of u is computed at these grid points, obtaining a tensor $\boldsymbol{u}_h \in \mathbb{R}^{N_x^d \times N_t}$, where $\boldsymbol{u}_h[i_1, \ldots, i_d, j]$ is the *approximated* value of u at spatial point $\boldsymbol{x} = (i_1\Delta_x, \ldots, i_d\Delta_x)$ and time step $j\Delta_t$.

Normally, we are not interested at specific spatial points, but rather the whole discretised spatial state at a given time step. We denote an ordered set of all spatial grid points as $\mathcal{X}$, and the discrete spatial state at time step j as $\boldsymbol{X}^j$. To ease the notation, we substitute index j with t, which before was used for continuous time variable. As we are now in the discrete setting, it does not create confusion, and makes it more readable. Finally, the solution grid at time step $t \in \{0 : N_t\}$ is:

$$\boldsymbol{X}^t \coloneqq \boldsymbol{u}_h[:, t] = \{\hat{u}(\boldsymbol{x}, t \cdot \Delta_t) \mid \forall \boldsymbol{x} \in \mathcal{X}\} \tag{1.31}$$

The solution *trajectory* is the sequence of all solution states at each time step, and we denote it as simply as:

$$\boldsymbol{X} = \left\{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^{N_t}\right\}, \tag{1.32}$$

and for a subtrajectory, sequence of consecutive steps from t_1 to t_2, we write $\boldsymbol{X}^{t_1:t_2}$.

Solver. The solver S of the PDE, independent of its method (explicit, implicit, spectral), is a programme that sequentially advances a discrete state as $S(\boldsymbol{X}^t) = \boldsymbol{X}^{t+1}$ starting from the initial state $\boldsymbol{X}^0$ defined by $g(\lambda)$.

Generally, the programme takes as input [2] the PDE configuration $(F, B, g; \boldsymbol{\lambda}, \boldsymbol{\lambda}_B, \boldsymbol{\lambda}_0)$, the domain configuration (L, T), and the discretisation configuration h. It then discretises the domain, initialises the state with IC, and iteratively applies the update rule defined by the numerical method to compute the next state from the current one, while applying BC at each step. During the programme run, it yields step-by-step the solution trajectory $\boldsymbol{X}$ defined by the input configurations. Formally, the solver is a mapping:

$$S : (F, B, g; \boldsymbol{\lambda}, \boldsymbol{\lambda}_B, \boldsymbol{\lambda}_0, L, T, h) \mapsto \boldsymbol{X}. \tag{1.33}$$

[2] It depends on the implementation of the solver which inputs are required. Sometimes some configurations are hardcoded, pre-computed and fixed, or omitted. However, any programme can be rewritten to take all configurations as input, so we use the general form.

A slightly more generic form that is independent of the mesh type would be:

$$S : (F, B, g; \boldsymbol{\lambda}, \boldsymbol{\lambda}_B, \boldsymbol{\lambda}_0, \mathcal{G}, \eta) \mapsto \boldsymbol{X}, \tag{1.34}$$

which allows for arbitrary geometry domain $\mathcal{G}$ and discretisation η and includes time-independent PDEs. Further in the thesis, we will often omit some configurations in the notation for brevity, assuming they are given or fixed within the context.

1.2.4 Scientific Computing Applications

Numerical simulations are widely used in many fields of science and engineering, such as fluid dynamics, weather prediction, climate modelling, structural analysis, and many more. Regardless of the application field, the simulations are purposed for various scenarios. Here we attempt to list these scenarios. The list is not exhaustive, but rather illustrative, as we aim to give the reader the idea of diverse usage of numerical simulations and imposed requirements.

Scenarios of Numerical Simulations Usage.

I. **What-If Analysis and Deterministic Prediction**

The task is to *predict system behaviour* across different scenarios and in long-term forecasts. The goal is to have a better understanding of the system, to explore possible outcomes across scenario space, identify risks, and validate hypotheses. Virtual exploration with simulations has lower costs and risks than testing real-world prototypes. The simulations must maintain reliable accuracy across the scenario space, and faster runtimes allow for more extensive exploration.

Example: Predicting climate change impacts on different aspects of human life, such as agriculture and health, depending on the mitigation scenario [160]. Simulations can take days to weeks, depending on the horizon and complexity.

II. **System Design Optimisation**

The task is to *find optimal configurations* of the system design before physical fabrication or experiments. Compared to the previous task, the system design is not fixed but its optimum is sought, which is typically done automatically. That means the number of simulation runs is not predefined but depends on the convergence of the optimisation algorithm.

Similarly, the simulations are required to be robust but also flexible to system components changes (e.g., geometry). Optimisation methods, in order to obtain gradient information, typically rely on either running large numbers of simulations or on adjoint solvers, which are not always available for complex systems or can take significant time to develop. Fast and differentiable simulation methods are of great interest to enable efficient optimisation.

Example: Aerodynamic shape optimisation via adjoint methods [88], which takes days to weeks and 100-1000 simulation evaluations typical, instead of months to years and multiple physical prototypes.

III. **Probabilistic Forecasting and Uncertainty Quantification (UQ)**

The task is to quantify uncertainty and provide probabilistic predictions by running large ensembles of simulations with perturbed inputs. The goal is to characterise

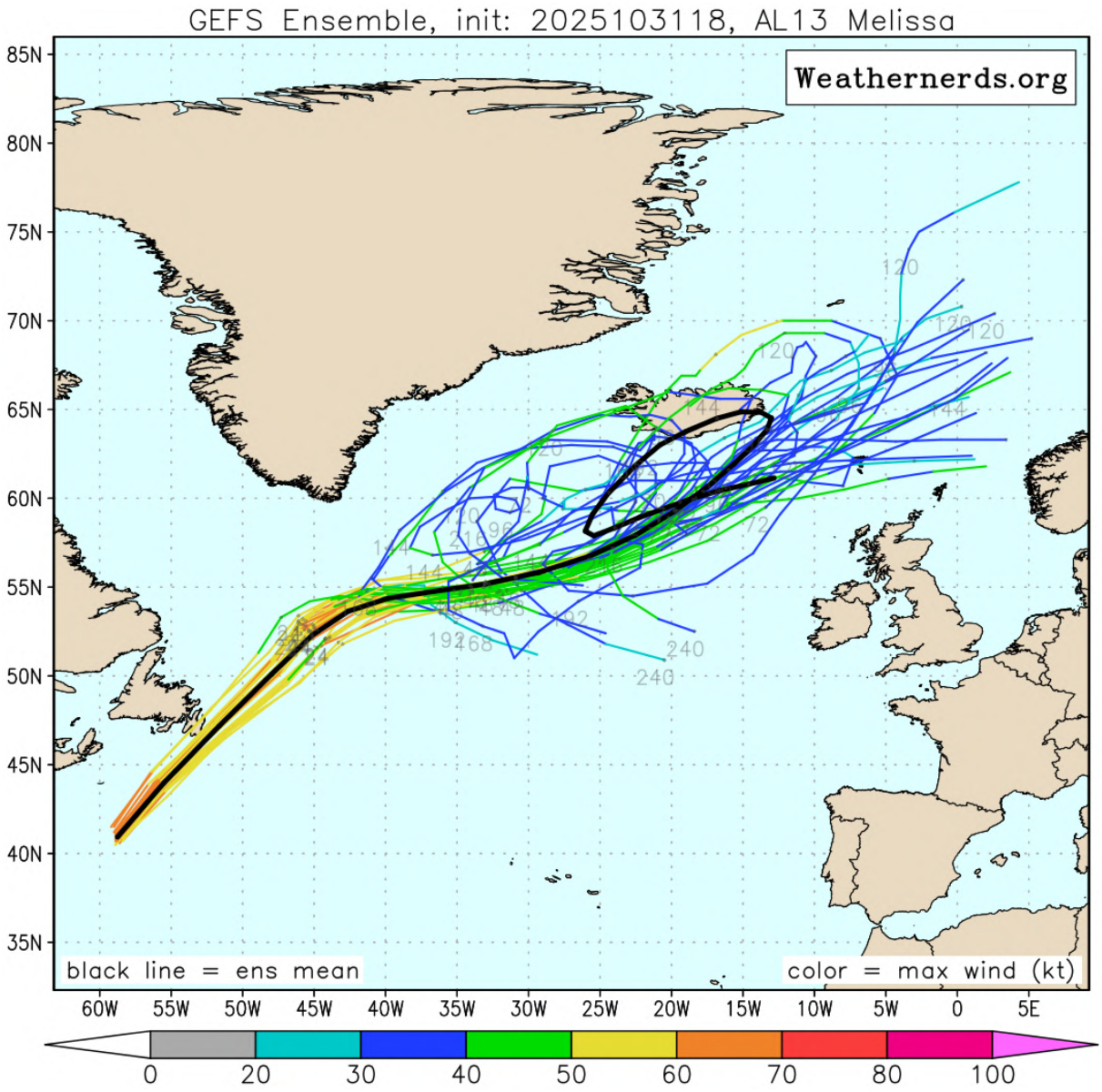


Figure 1.3: Example of ensemble forecasting via Global Ensemble Forecast System (GEFS). The image shows an ensemble of predictions of hurricane Melissa trajectory from the 31st of October, 2025. Each member represents a possible future scenario, capturing uncertainty in initial conditions. Thanks to https://weathernerds.org for the provided image.

how the uncertainty in inputs propagates through the simulation to produce uncertainty in outputs. See Figure 1.3 for an example of ensemble forecasting for a hurricane Melissa trajectory. Faster runtimes are crucial to enable large ensembles which yield more accurate statistics. Simulations of lower fidelity can be used to reduce the computational cost, but they must maintain sufficient accuracy to capture the relevant uncertainty characteristics. In some cases, the simulations must be completed within a strict time window to be actionable, which further emphasises the need for faster solvers. Stochastic PDEs and solvers are often employed to directly model uncertainty within a single run.

Example: Wind power forecasting [160], weather and environment forecasting. The number of simulations directly affects the reliability of predictions and estimates and ranges from hundreds to thousands of simulations.

IV. **Parameter Studies and Sensitivity Analysis (SA)**

The task is to *identify how input parameters influence outputs* systematically. While similar to UQ, the goal is to understand the relationship between inputs and outputs: determine how variations in each input parameter affect the outputs, rather than characterising overall uncertainty. The insights identify which parameters to refine or control to reduce risks or improve performance.

Example: Sensitivity analysis of aerodynamic coefficients in flight simulations [160]; power grid stability analysis. Depending on the method, from tens to thousands of simulations.

V. **Data Assimilation, Inference, and System Identification**

The system is observed (fully, partially, or sparsely) while the mathematical model is not fully known. The goal is to *infer/adjust system states or/and model from the observations*. The solvers themselves are not designed to solve inverse problems or to incorporate observational data, so auxiliary methods are used, which iteratively call the solver to minimise the mismatch between simulations and observations. Verification and validation (V&V) procedures ensure models reliability. These tasks often require running many simulations, sometimes, with varying parameters, initial and boundary conditions, and domain configurations.

Example: Ocean state estimation via ensemble Kalman filter [51], material parameter identification from experiments. Minutes to hours for real-time assimilation; days for complex parameter inference.

VI. **Digital Twins**

The simulations can be used to *monitor and control system behaviour* in real time. The goal is to make optimal and rapid decisions based on the current system state. Thus, the simulations are required to run at a very low latency.

Example: Model predictive control of chemical reactors [5], flight control systems, autonomous robotics. Milliseconds to seconds per control cycle.

Many of these applications require running a large number of simulations, sometimes, with varying parameters, initial and boundary conditions, and domain configurations.

While examples can be found in many application areas, the most prominent are associated with Computational Fluid Dynamics (CFD) [55] – the area of scientific computing that focuses on the problems involving fluid flows, such as air and water. As fluid flows

occur in large range of systems of different scales and complexities, CFD is widely used in numerous fields of science and engineering, such as aerospace, automotive, energy, environmental science, and many more.

For example, aerospace engineers optimise the design of an aircraft to improve its aerodynamic performance, preserve safety, reduce fuel consumption, and minimise environmental impact. This involves simulating the airflow around an aircraft with a high resolution, which is governed by the Navier-Stokes equations – a set of nonlinear PDEs on a complex geometry. Running a single simulation can take hours or even days on a supercomputer, depending on the method and fidelity. But to optimise the aircraft design, many simulations with different parameters and configurations are queried. These can take weeks or months of compute time, however, it is still significantly faster and cheaper than building and testing real-life prototypes. To shorten the design cycle and further reduce the costs and time-to-market, developing faster solvers is of great interest.

In case of such fields as drug-discovery, epidemiology, or climatology, faster simulations can directly translate into saving human lives. For example [1], the prediction time for the simulation of typhoon tornadoes, which hit Japan in 2024, was drastically reduced[3] from 11 hours to 80 minutes, while using only 5% of Fugaku supercomputer's resources. This made the predictions to be practicably applicable: a tornado occurrence can be predicted four hours in advance and help the government to take necessary actions to protect the population in time.

Numerical simulations play a crucial role in science and engineering but their high computational cost often limits their applicability and scalability. A prominent area of research which aims to mitigate this issue is *reduced order and surrogate modelling* – it develops techniques that approximate full mathematical models or simulations with ones of lower complexity and cost, while maintaining satisfactory accuracy for the task at hand. This field dates back to the 1970s [140] and is actively researched with many methods proposed [23].

1.3 Conclusion

In this chapter, we introduced the fundamental concepts of deep learning and scientific computing and highlighted the advantages and limitations of both fields. The advance of DL in scale and capabilities has paved the way for its applications in science, particularly in scientific computing. To address the limitations of numerical solvers, DL-based surrogate models have been proposed as a promising solution. The next chapter delves into the concept of deep surrogate models, discussing their motivations, state-of-the-art methods, and challenges.

[3] This example is, in fact, a spoiler into the next chapter, as the simulation time was reduced thanks to the use of DL.

Chapter 2

Deep Learning for Simulations

Numerical simulations are an essential tool in engineering and science for analysing, designing, and predicting complex systems, yet their applicability is often hindered by high computational costs, arising from both computationally intensive single forward runs and the need for auxiliary methods and large ensemble runs in scenarios like inverse problems, uncertainty quantification, and optimisation.

With the advances of deep learning (DL) and its success in domains of computer vision and natural language processing, the research community actively explores DL as an alternative approach to surrogate modelling in scientific computing applications. This research direction falls under the broad umbrella of AI for Science (AI4Science) and Scientific Machine Learning (SciML) communities, which has grown rapidly in recent years, as demonstrated by recurring Ai4Science workshop[1] at major AI conferences and a number of dedicated surveys on SciML perspectives [120, 234, 161].

Motivation for DL adoption in scientific computing is two-fold: *from the technical point of view*, DL leverages hardware accelerators, i.e., GPUs and TPUs, distributed computing, optimised hardware, software and techniques, e.g., automatic differentiation; *from the methodological point of view*, DL research community has developed numerous advanced algorithms that work efficiently with high-dimensional data, learn complex patterns from data, offer flexible designs, and generalise robustly across tasks. This established work can be reused and adapted for scientific computing problems, translating into (1) accelerating forward solver runs via DL-based surrogates, and (2) extending traditional solver workflows via DL-based surrogates that handle complex tasks beyond forward simulations.

In this chapter, we review the existing works on DL-based surrogate models for PDE simulations, discuss their advantages and limitations. It is organised as follows:

- Section 2.1 examines two perspectives on DL-based surrogates;
- Section 2.2 presents the methodological design space of DL-based surrogate models;
- Section 2.3 surveys the state-of-the-art DL-based surrogate models;
- Section 2.4 discusses training efficiency limitation arising from relying on synthetic data generation.

We conclude the chapter by summarising the key findings and identifying open challenges for surrogate modelling with DL.

[1] https://ai4sciencecommunity.github.io

2.1 Roles of Deep Learning in Surrogate Modelling

To address the computational cost limitations of numerical PDE solvers, DL-based surrogate models have been proposed as a promising alternative [231, 210].

A DL-based surrogate model for PDEs is a neural network (NN) designed to approximate the solution function of a given system, acting as a (possibly) computationally efficient alternative to traditional numerical solvers. In contrast to typical NNs in vision or language tasks that learn from real-world data, i.e., observational data, a surrogate can be trained using simulations generated by numerical solvers, i.e., (numerically[2]) synthetic data.

These models are often called simply *deep surrogates*, and *neural emulators* or *neural surrogates* also appear in the literature. In this thesis we discuss only DL-based surrogate models, so we often refer to them in the text as just *surrogates* for brevity.

We propose viewing deep surrogates from two perspectives based on their functional role in relation to traditional numerical solvers: (1) *solver surrogates*, which directly accelerate expensive forward solver runs by replacing them fully or partially; and (2) *extended surrogates*, whose capabilities enable addressing tasks like inverse problems or uncertainty quantification that are usually achieved by combining forward solvers with auxiliary methods. We elaborate on these in the following subsections.

2.1.1 Solver Surrogates: Faster Forward Inference

Once trained to a required accuracy, the surrogate predicts the solution orders of magnitude faster than the numerical solver, especially when leveraging modern hardware accelerators like GPUs or TPUs. In application scenarios with ensembles of simulations, where multiple forward simulations with varying parameters are required, surrogates can provide significant speed-ups while maintaining reliable accuracy.

The acceleration from surrogates can be achieved in various ways, depending on the application and design choices.

Often the speed is traded off with accuracy. In parameter space exploration and experimental design, lower accuracy can be tolerated, and the surrogate can be used to only provide guesses for promising values of interest, which then can be explored in more detail using the accurate but expensive numerical solver.

To accelerate the solver by replacing it completely, however, the accuracy of the surrogate must match the numerical solver, which may be achievable given sufficient training data and model capacity. In weather forecasting, where geometry, physics, and parameters are fixed, both observational and synthetic historical data are readily available to train high-quality surrogates. The synthetic data is produced by solvers that are calibrated from various observational measurements. In such scenarios, the numerical solver may be replaced completely by the surrogate, leading to real-time predictions [25]. The speed-up is attributed to both the ability of NNs to learn complex mappings accurately and the efficiency of NNs on modern hardware.

[2]The term "synthetic" may refer to data created without deterministic forward models, for example, in social sciences, synthetic data emulates empirical patterns from real datasets via generative and statistical models [92]. Here, we specifically refer to data generated by numerical solvers and omit the "numerical" qualifier for brevity as it is clear from the context.

Replacing the solver completely is not always necessary, and for complex systems requiring high accuracy and resolution, it may not be feasible. Instead, surrogates can be used to assist the numerical solver in various ways, leading to *hybrid approaches*, where the solver and surrogate are combined and decrease the overall runtime and errors [218, 232]. For example, in multiscale phenomena, where different scales of the system are modelled, solvers typically struggle to capture all the scales accurately and efficiently. Surrogates can be used to model the small-scale phenomena, while the solver handles the large-scale dynamics [39, 227]. Surrogates can be used to accelerate only certain components of the solver that are specifically expensive [49].

The speed-up can be achieved through NNs' flexible design that allows them to predict the solution with fewer steps compared to traditional numerical solvers. Instead of solving the equations in an autoregressive manner, i.e. predicting the next time step based on the previous one, step by step, the surrogate can be designed to predict the solution at a desired time step directly from the initial conditions. This is often referred to as *one-shot* or *direct* prediction. This approach does not require creating all the intermediate steps, which can lower the error accumulation and decrease the computational time. In such scenarios where the steady state solution is of interest, intermediate and transient states can be skipped, making direct surrogates particularly interesting.

Finally, surrogates can offer resolution independence, mesh-free representations that avoid explicit mesh discretisation, and easier handling of complex or varying geometries – issues where numerical solvers may often struggle, incurring higher computational costs and requiring tuning and configuration by domain experts [112, 233].

2.1.2 Extended Surrogates: Beyond Forward Inference

DL models can be used to address the limitations of numerical solvers in terms of flexibility and applicability to certain tasks. Their design can be adapted to the specific objectives, enabling capabilities that go beyond what numerical solvers can achieve.

Tasks that such surrogates can address beyond forward inference include:

- **Inverse problems**, i.e., discovering system parameters or physics laws from observations, where surrogates can be designed to be invertible [38, 42, 93].
- **Uncertainty quantification**, i.e., estimating the uncertainty of the solution approximation, where surrogates can be designed to provide probabilistic outputs or uncertainty estimates [107, 142, 131, 197].
- **Data assimilation**, i.e., incorporating observational data to update and correct model states during simulation, where surrogates benefit from being inherently data-driven and can effectively handle sparse, noisy, or limited observational data [176, 210, 141, 228].
- **Optimisation**, i.e., finding optimal designs or control strategies, where surrogates can be designed to be differentiable and integrated into gradient-based optimisation loops [181].
- **Generalisation** over varying parameters and physics, where surrogates can be designed to learn mappings that generalise across different scenarios and enable transfer learning techniques [247, 9].

Overall, these illustrate how deep surrogates evolve from computational accelerators that simply mimic numerical solvers to versatile frameworks addressing complex scientific tasks. By providing unified approaches to inverse problems, uncertainty quantification, data assimilation, and optimisation – tasks that would traditionally require multiple specialised methods – DL-based surrogates offer improved computational efficiency and flexibility. This versatility is likely to continue growing as DL techniques advance and integrate further with scientific computing.

While this thesis focuses on surrogates for forward inference acceleration, the methods and the challenges, in particular those related to training efficiency, are relevant to both perspectives. In the next section, we present the basic concepts of surrogate models, current state-of-the-art methods, and their limitations.

2.2 Deep Surrogates Methodology

A deep surrogate is an NN $f_{\boldsymbol{\theta}}$ that approximates the solution function of a given PDE system provided through a numerical solver S. As previously defined in Subsection 1.2.3, a solver S produces a solution trajectory $\boldsymbol{X}$ for a given configuration $(F, B, g; \boldsymbol{\lambda}, \boldsymbol{\lambda}_B, \boldsymbol{\lambda}_0, \mathcal{G}, \eta)$ – a set of governing equations, their parameters, a geometry domain, and a discretisation scheme. Within its design capabilities, the solver is able to obtain solutions for any valid configuration of the system, subject to variation in approximation accuracy and computational cost.

The surrogate can be designed to obtain solutions for a fixed configuration or to generalise over a range of configurations, depending on the application and the surrogate's role. For example, in earth system modelling, the equations and geometry are typically fixed, but the IC/BCs and parameters may vary. One might design a surrogate to generalise over different IC/BCs and parameters for the fixed equations and geometry, either through training on multiple simulations or by learning temporal dynamics from a single trajectory (e.g., weather models trained on ERA5 reanalysis [117]). When multi-resolution simulations are required, the surrogate can be designed to generalise over different spatial and temporal scales [71]. The NN's objective, architecture, and training data should be chosen accordingly to enable the desired generalisation capabilities, which we call *behaviour design*. Ashton et al., 2025 [13, Section 3] discusses the axes of generalisation for CFD modelling surrogates, highlighting the scale of conditional space for a surrogate to generalise over and the challenges associated with it.

In this thesis, we mainly consider surrogates that generalise over varying IC parameters because today this is the most common design in practice, and it is a fundamental step towards more complex generalisation capabilities. This behaviour design can be formulated as:

$$f_{\boldsymbol{\theta}}(x_i) \approx S(x_i), \quad \forall \boldsymbol{\lambda}_{0i} \in \Lambda_0, \tag{2.1}$$

where x_i denotes an arbitrary NN's input associated with the IC parameters $\boldsymbol{\lambda}_{0i}$, and $\Lambda_0 \subset \mathbb{R}^{n_0}$ is the range of IC parameter values to generalise over.

Hereafter, this design is considered by default, unless specified otherwise. For brevity, we drop the subscript 0 and denote IC parameters as $\boldsymbol{\lambda} \in \Lambda \subset \mathbb{R}^{n_p}$.

2.2.1 Methods Categorisation

Two independent axes distinguish surrogate methodologies: the optimisation approach (when discretisation occurs) and the physics-data duality (which information sources drive learning).

Optimisation approach. There are two main approaches for training optimisation: *discretise-then-optimise* and *optimise-then-discretise*.

To obtain approximate simulations with surrogates, we can first discretise the solution space, and then train a NN on these discrete solutions, which yields *discrete space models*. Discrete space models operate on discretised representations of the solution, learning mappings between grid-based states using standard neural network architectures.

Alternatively, we can first train a NN to learn the solution mapping in a continuous space, and then obtain discretised solutions, which yields *continuous space models*. Continuous space models learn solution mappings in continuous space, enabling solution approximation at any point in the domain without being tied to a specific discretisation.

Physics-Data duality. Surrogates can leverage two types of information during training: solution data from simulations or observations, and explicit physics knowledge such as governing equations or conservation laws. The relative availability of these resources – ranging from data-abundant/physics-poor to physics-rich/data-scarce to balanced hybrid settings – motivates three approaches.

Data-driven surrogates learn exclusively from solution samples, treating the surrogate problem as standard supervised learning. The training objective minimises prediction error on observed solution states without explicit knowledge of the underlying equations.

Physics-informed surrogates incorporate physical constraints explicitly into the learning process. Physics knowledge is embedded through additional loss terms (e.g., PDE residuals), architectural constraints (e.g., symmetry-preserving layers), or data augmentation strategies (e.g., conservation-law enforcement).

Hybrid surrogates combine both, enabling adaptation to varying data availability: data-rich scenarios rely on observations, while data-scarce scenarios exploit physics knowledge more heavily.

2.2.2 Handling Temporal Dimension

For time-dependent PDEs, there are two aspects of surrogate design related to the temporal dimension: how to predict the solution trajectory over time and how to train the surrogate with respect to time.

Prediction Strategies

A surrogate can be designed to predict solution trajectory states in *autoregressive* or *direct* manner.

Autoregressive surrogates predict the states step-by-step using the previous state as input, similar to traditional numerical solvers, like:

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}^t) \approx \boldsymbol{X}^{t+1}. \tag{2.2}$$

To obtain the solution trajectory over multiple time steps, typically, the surrogate is applied recursively, using previous predictions as inputs for the next ones, like:

$$\left\{\boldsymbol{X}^0, f_{\boldsymbol{\theta}}(\boldsymbol{X}^0), f_{\boldsymbol{\theta}}^{(2)}(\boldsymbol{X}^0), \ldots, f_{\theta}^{(N_t)}(\boldsymbol{X}^0)\right\}, \tag{2.3}$$

where $f_{\boldsymbol{\theta}}^{(k)}$ denotes k sequential applications of the NN model.

Direct surrogates predict the state of the system at a future time directly from an initial state, like:

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}^0, t') \approx \boldsymbol{X}^{t'}, \tag{2.4}$$

where t' is the time interval to predict ahead, which can be expressed as a discrete value (number of time steps) or a continuous value (time duration). This approach is typically used when the interest is in the steady state solution or specific future time steps.

Beyond these two main approaches, several alternative prediction schemes have been proposed. Instead of predicting the state itself, Lippe et al., 2023 [132] proposed to predict the difference between the states, like:

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}^t, \Delta t) \approx \boldsymbol{X}^{t+\Delta t} - \boldsymbol{X}^t. \tag{2.5}$$

This residual learning approach improves stability in autoregressive rollouts and enables iterative refinement of predictions, though at the cost of requiring multiple forward passes for refinement. As a generalisation of both autoregressive and direct approaches, Herde et al., 2024 [75] proposed an *all-to-all* approach, where the surrogate is trained to predict from any time step t to any other time step $t + t'$ within a given range, like:

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}^t, t') \approx \boldsymbol{X}^{t+t'}, \tag{2.6}$$

which enables flexible temporal predictions at arbitrary time intervals, at the cost of significantly increased training time as the model must learn mappings across all possible time step combinations. To reduce computational cost from the recursive application of the model, Diab and Al Kobaisi, 2025 [43] proposed an approach, where the surrogate predicts multiple future time steps in a single forward pass, but it comes with a trade-off in terms of accuracy.

Training Strategies

From the optimisation perspective, there are two main approaches to calculate the prediction loss and to perform backpropagation in respect to time steps.

The first is a *one-step* (or one-to-one) manner [217], when the NN's loss is backpropagated only through a single prediction, like:

$$\ell(f_{\boldsymbol{\theta}}(\boldsymbol{X}^t), S(\boldsymbol{X}^t)), \tag{2.7}$$

where $\boldsymbol{X}^t$ is arbitrary NN's input associated with a time step t, and $S(\boldsymbol{X}^t)$ is the corresponding solver reference.

The second is an *unrolled* (or multistep) manner [219, 237], when the NN's loss is backpropagated through multiple sequential predictions, like:

$$\sum_{k=1}^{K} \ell(f_{\boldsymbol{\theta}}^k(\boldsymbol{X}^t), S^k(\boldsymbol{X}^t)). \tag{2.8}$$

This approach enables the surrogate to learn long-term dynamics and to mitigate error accumulation over time, but it is computationally expensive due to the need to backpropagate through multiple time steps. Alternatively, Brandstetter et al., 2023 [29] proposed a "push-forward trick" approach as a compromise: the model autoregressively rolls out predictions for K time steps, but gradients are backpropagated only through the last prediction step K, reducing computational cost while still training on long-term dynamics. Unrolled loss is often used solely for validation, while one-step loss is used for training, as to evaluate the long-term performance of the surrogate without incurring the high computational cost of unrolled training.

Koehler et al., 2024 [108] introduces a novel taxonomy based on unrolling steps during training and the solver reference length, which unifies these schemes, like:

$$\sum_{k=0}^{K-B} \sum_{b=1}^{B} \ell(f_{\boldsymbol{\theta}}^{k+b}(\boldsymbol{X}^t), S^b(f_{\boldsymbol{\theta}}^{k}(\boldsymbol{X}^t))), \tag{2.9}$$

where K is the number of unrolling steps and B is the solver reference length. Then the one-step manner corresponds to $K = B = 1$ and unrolled manner to $K = B > 1$. Additionally, a case of *diverted chain* learning is distinguished, with any K and $B = 1$, where the model is unrolled for K steps, but the loss is always calculated with respect to a one-step solver reference. This configuration requires the solver to be available during training and differentiable, as the solver is called within the computational graph to generate one-step references from the surrogate's intermediate predictions, through which gradients must flow during backpropagation.

We will denote training samples consisting of multiple trajectory steps as $\boldsymbol{X}^{[t]}$, where the superscript $[t]$ indicates that the sample starts with a state from time step t. For example, in a two-step training setup:

$$\boldsymbol{X}^{[t]} = (\boldsymbol{X}^t, \boldsymbol{X}^{t+1}, \boldsymbol{X}^{t+2}). \tag{2.10}$$

2.3 State-of-the-Art Deep Surrogates

Early works on deep surrogates for numerical simulations adapted backbone architectures from broad DL. For example, Rasp et al., 2018 [179] used fully-connected NNs and de Bezenac et al., 2018 [41] employed CNNs for climate tasks, while [100, 214] utilised CNNs for fluid simulations in context of computer graphics. Standard architectures like UNet [248] and ResNet [74] remain widely used as baselines [108], as PDE solution fields on regular grids share structural similarities with images, enabling convolutional architectures from computer vision to achieve strong performance. These approaches are *data-driven*, learning directly from simulation data.

In contrast, methods specific to SciML have emerged that leverage the underlying physics and mathematical structure of PDEs more explicitly. These can be categorised into two paradigms: *physics-informed* surrogates that embed physical laws directly into the learning process, enabling training with limited or no solution data; and *operator learning* approaches that learn mappings between function spaces from simulation data, enabling discretization-invariant predictions and generalisation across parameter families. In this section, we provide an overview of the two foundational approaches for each paradigm: physics-informed neural networks (PINNs) and neural operators (NOs), which have inspired a wide range of subsequent innovations in surrogate modelling for PDEs.

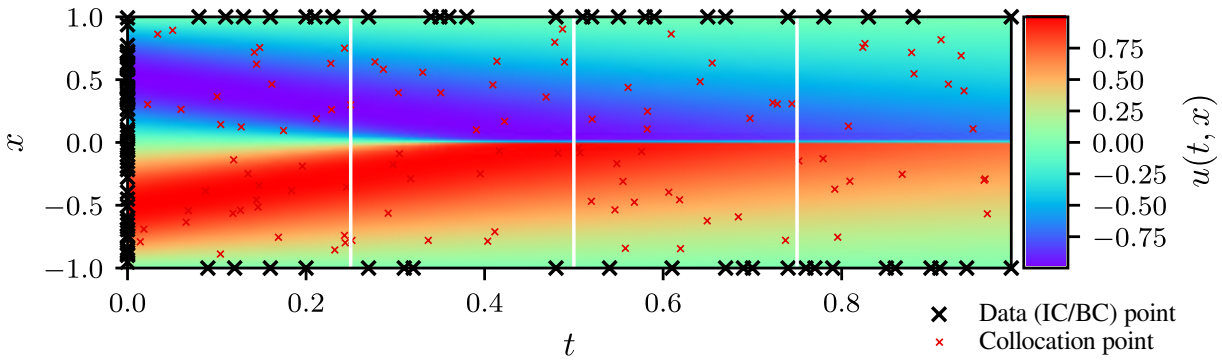


Figure 2.1: Spatio-temporal domain and solution for Burgers equation with initial and boundary data points (black crosses) and collocation points (red crosses). Adapted from Raissi et al., 2019 [176].

2.3.1 Physics-Informed NNs

PINNs were proposed by Raissi et al., 2019 [176] as an alternative to surrogates that learn from discrete solution state data. These aim to learn continuous point-wise solutions of a PDE directly by embedding the governing equations into the NN objective function.

Consider a general PDE problem as defined in Eq. (1.27). The unknown solution function $u(\boldsymbol{x}, t)$ is approximated by a NN $f_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ with parameters $\boldsymbol{\theta}$. The PDE residual is defined as:

$$r_{\boldsymbol{\theta}}(\boldsymbol{x}, t) \coloneqq F[f_{\boldsymbol{\theta}}; \boldsymbol{x}, t], \tag{2.11}$$

where partial derivatives for the NN in the right-hand side are calculated via automatic differentiation [21], allowing the residual to be evaluated at any point in the spatio-temporal domain.

The core optimisation objective minimises the *residual loss*, which enforces the NN to satisfy the governing equations at collocation points $\{(\boldsymbol{x}_f^i, t_f^i)\}_{i=1}^{N_f}$ sampled in the spatio-temporal domain:

$$\mathcal{L}_{\mathrm{r}}(\boldsymbol{\theta}) = \frac{1}{N_f} \sum_{i=1}^{N_f} \ell(r_{\boldsymbol{\theta}}(\boldsymbol{x}_f^i, t_f^i)), \tag{2.12}$$

where $\ell(\cdot)$ is a loss function.

To enforce known solution values, a *data loss* term is added, which can incorporate IC/BCs when specified as in Eq. (1.30) and additional measurements from observations or simulations when these are available, like:

$$\mathcal{L}_{\mathrm{u}}(\boldsymbol{\theta}) = \frac{1}{N_u} \sum_{i=1}^{N_u} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_u^i, t_u^i) - u^i), \tag{2.13}$$

where $\{(\boldsymbol{x}_u^i, t_u^i, u^i)\}_{i=1}^{N_u}$ are data points with known solution values. We provide an illustrative example adapted from [176] in Figure 2.1, which shows the spatio-temporal domain with collocation and IC/BC data points.

The overall loss function then becomes:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{r}}(\boldsymbol{\theta}) + \mathcal{L}_{\mathrm{u}}(\boldsymbol{\theta}). \tag{2.14}$$

This formulation makes PINNs a flexible framework: they can rely solely on governing equations (including IC/BCs), making them data-free surrogates, or can optionally incorporate available data, turning them into hybrid physics-data surrogates.

For a comprehensive review of PINNs, we refer the reader to Toscano et al., 2024 [216], which provides an overview of recent advances, including the latest algorithmic developments, applications, theoretical advances, computational frameworks and software. In particular, we highlight the chronological overview of key advancements presented in the paper's Appendix A.

Limitations. PINNs are challenging to train mainly due to the multi-objective nature of the optimisation problem, leading to failure modes and information bottlenecks [113, 10]. Extensive research has addressed these training difficulties [225, 226, 180], demonstrating that while PINNs are a powerful tool in SciML, they pose significant challenges in terms of training stability, convergence, and accuracy. Various optimisation methods have been proposed to address these issues, including natural gradient approaches such as ANaGRAM [188, 189] and energy natural gradients [156], conflict-free gradient methods [134], and meta-learned optimisation strategies [27]. A survey by Torres et al., 2025 [215] identifies transfer learning and meta-learning as promising directions to improve PINN training efficiency and generalisation capabilities.

Alternatively, active learning approaches can be used to guide the sampling of collocation points to improve training efficiency and convergence [235]. We detail related work in Section 4.4 and present our contribution for active learning of PINNs in Chapter 5.

2.3.2 Neural Operators

The Neural Operator (NO) learning framework was established by Kovachki et al., 2021 [112], with the earliest proposed operator-learning architecture being DeepONet by Lu et al., 2021 [136].

NOs are designed to learn solution operators that map between infinite-dimensional function spaces. This enables prediction at any point in the solution domain, making NOs discretisation-invariant. This allows a trained model to perform inference on different mesh resolutions without retraining, unlike grid-based CNNs where architecture dimensions are fixed to specific discretisations.

The key capability of NOs is that they take functions as inputs and outputs, allowing them to generalise over a family of PDEs and act as universal operator approximators. In practice, input and output functions are represented through discretised samples from high-fidelity simulations, making NOs data-driven surrogate models.

We omit mathematical details of NOs here for brevity and refer to Kovachki et al., 2021 [112] for comprehensive details; instead, we provide just the general idea of the architecture. The approach consists of three main components: (1) *Lifting*: input functions are mapped point-wise to a latent representation; (2) *Iterative kernel integration*: hidden representations are recurrently updated through multiple layers of non-local integral kernel operators; and (3) *Projection*: the final hidden representation is mapped point-wise to the output function.

The intuition behind this architecture is as follows: the lifting step expands the data into a richer space where non-locality – how points in one location are affected by distant ones – is easier to learn; the integral kernel operators then capture these long-range dependencies by computing weighted combinations of information from across the entire spatial domain; and projection maps back to the output function space.

NOs have become a foundation for extensive architectural innovations spanning representations and backbone designs. FNO [127] leverages Fourier space parametrisation of integral kernels for computational efficiency, while spectral methods [52] utilise frequency-domain representations of functions and operators, enabling accurate and efficient operator learning. Alternative operator formulations [190, 19] aim to reduce discretisation constraints through coordinate-based and representation-equivalent frameworks. Architectural variants include convolutional operators [178] for structured grids, U-shaped architectures [174] for multi-resolution processing, and state-space models [84, 83] for efficient long-range dependencies. Many recent architectures build upon these foundational extensions. We refer the reader to Azizzadenesheli et al., 2024 [14][3] for a comprehensive overview of the main advances in NO architectures and applications.

Unlike standard PINNs, which learn solutions for specific PDE configurations and require retraining for new parameters, NOs learn operators that generalise across instances of the same PDE with varying inputs or parameters. This makes PINNs suitable for sparse data and inverse problems, while NOs excel at many-query scenarios once trained on sufficient input variations. However, the main challenge for NOs and other data-driven surrogates training is the substantial cost of generating training data, as many high-fidelity simulations across the parameter space are required for high accuracy and generalisation.

2.3.3 Recent Advances

Surrogate modelling has advanced rapidly, with innovations addressing key challenges in generalisation, data efficiency, and computational scalability. Recent methods increasingly adopt state-of-the-art architectures from vision and language, while introducing approaches tailored to the unique challenges of learning PDE solution mappings. We highlight recent advances across three directions that shape the current research frontier: handling complex geometries, scaling across problems, and reducing data requirements.

Handling complex geometries. Early surrogates relied on regular grids, limiting applicability to the complex domains common in real-world applications, whereas graph-based architectures enable learning on unstructured meshes. GNS [185] introduces the encode–process–decode paradigm for particle-based graphs, GNO [126] employs graph-based integral kernel operators, and MP-PDE [29] utilises neural message passing to generalise across resolutions, geometries, and boundary conditions on regular and irregular grids. Building on this, other approaches propose to embed geometric properties more explicitly. GeoFNO [128] learns a deformation from irregular domains to a regular latent grid for Fourier-based processing, RIGNO [155] constructs multi-scale regional meshes to achieve approximate resolution invariance, and CALM-PDE [72] learns adaptive query points and applies continuous convolutions for flexible geometry handling. Further architectures combine graph and Fourier operators with explicit geometry encodings and transformers for large-scale 3D problems [129, 233].

AB-UPT [9] introduces branched transformer with anchor attention, enabling surrogate modelling of industry-scale problems; specifically, it achieves industry-standard predictive accuracy for automotive CFD simulations with **up to 150 million cells and**

[3]The ICML 2024 tutorial on NOs (video and slides): https://icml.cc/virtual/2024/tutorial/35235

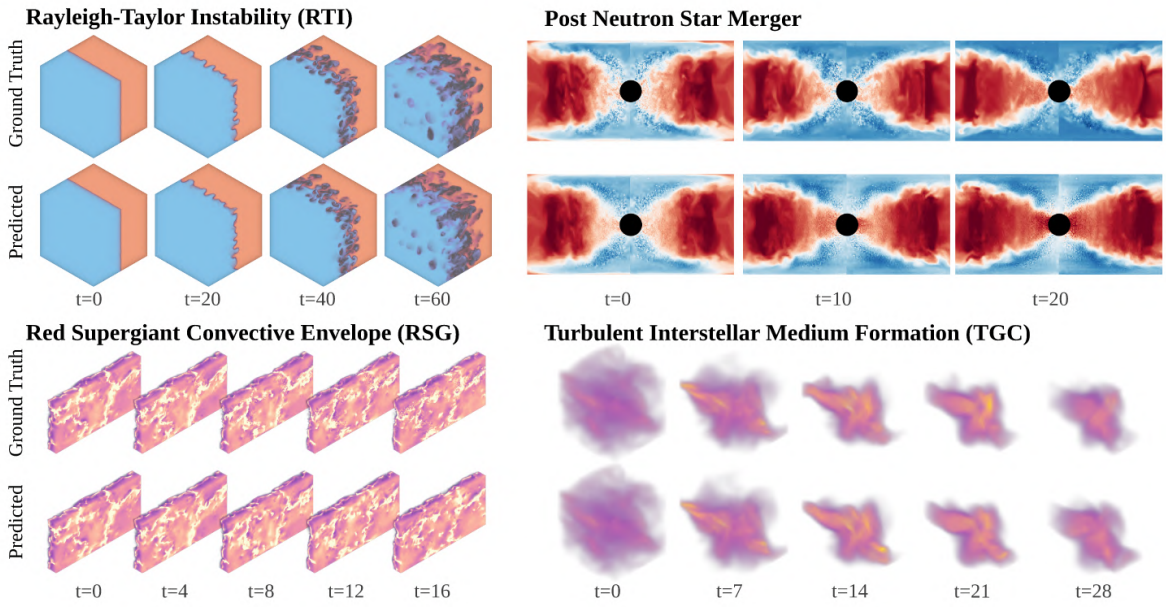


Figure 2.2: Examples of 3D research-frontier level simulations predicted by Walrus (bottom) compared to solver reference (top). From McCabe et al., 2025 [145].

inference time within seconds. Such applications, however, require training data spanning diverse geometries and parameters, whose configuration space grows rapidly with geometric complexity.

Scaling across problems. Foundation models aim to generalise across PDEs through pretraining on diverse simulation datasets and improved transformer-based architectures. Early work [144] demonstrates improved transfer accuracy on downstream tasks by learning shared representations from multiple physics domains, which capture broadly useful dynamics. Further architectural innovations enable efficient scaling: Poseidon [75] employs multiscale operators with time conditioning for flexible temporal predictions, AROMA [191] employs local neural fields for efficient spatial representation, PDETransformer [82] handles multiple physics via channel-wise self-attention, and P3D [81] uses hybrid CNN-Transformer design targeting 3D problems efficiently.

Walrus [145] achieves exceptional breadth through harmonic stabilization, compute-adaptive tokenization, and pretraining on millions of trajectories, enabling **generalisation across 19 continuum dynamics scenarios** spanning astrophysics, geoscience, rheology, plasma physics, acoustics, and classical fluids, in both 2D and 3D (Figure 2.2).

Reducing data requirements. Various methodological advances have been proposed to improve the data-efficiency and generalisation of data-driven surrogates. Hybrid methods leverage physics knowledge for improved generalisation with limited data: PIDeepONet [67] incorporates PDE residuals into DeepONet, while PINO [130] embeds physics constraints into Fourier operators, mitigating reliance on solution data for training. Context-learning approaches [103, 192, 37] enable adaptation to unseen dynamics without retraining, while evolution operators [154] handle partial trajectories. Physics-guided conditioning [205, 111] and function-attention mechanisms [175] further enhance generalisation across families of parameters and input conditions.

Beyond modifying the learning paradigm, multi-fidelity approaches leverage cheaper simulations. Tang et al., 2024 [208] achieve **81% reduction in data generation costs** with multi-fidelity FNO for geological carbon storage. Multi-fidelity DeepONet [138] reduces error by one order of magnitude when combining multiple fidelities versus high-fidelity data alone.

Community infrastructure. Frameworks and benchmarks enable standardised development and evaluation of surrogates: DeepXDE [137], PhysicsNeMo [4], and SciML [172, 173] facilitate implementation and training; benchmarks like PDEBench [206], APEBench [108], and Flowbench [207] provide diverse datasets, solver references, and evaluation protocols for rigorous comparison. The Well [162] exemplifies this community-building effort: **15TB across 16 datasets** (6.9GB to 5.1TB each) spanning diverse physics domains, showcasing large-scale resource sharing for training and evaluating foundation surrogates.

2.4 Synthetic Data Bottleneck

The advances presented demonstrate the field's impressive capabilities – fast inference speed, high predictive accuracy, and generalisation across varying configurations – but also reveal increasing data requirements.

This thesis addresses the *training efficiency challenge*, which arises from a fundamental paradigm shift in data availability: from pre-collected, static datasets to on-demand, simulation-generated data. Unlike observation-based tasks, synthetic data is *controllable yet computationally expensive* to produce, which means that standard practices treating the dataset as fixed and focusing solely on model improvements become insufficient. Inefficiencies in I/O, storage, and compute arise from two key aspects of this data paradigm: large data scale and complex input parameter spaces.

2.4.1 Large Data Scale

Scientific computing problems often involve high-dimensional spatio-temporal domains (2D/3D/4D), high-resolution grids (>1M points), multiple channels for different physical quantities, multiscale and multiphysics phenomena, complex geometries, and long trajectories. Compared to typical vision/language tasks, these characteristics lead to significantly larger data samples that are expensive to generate, store, and train with.

As an illustrative[4] example, consider a 3D flow simulation over a 256^3 grid with 5 physical quantities (velocity components, pressure, temperature) over 100 time steps. A single trajectory would require storing $256 \times 256 \times 256 \times 5 \times 100 \approx 8.4 \times 10^9$ floating-point numbers, which is approximately 32 GB of data in single precision. Storing a dataset of just 1000 such trajectories would require around 32 TB of storage. Generating a single trajectory using a high-fidelity solver can take hours or days on HPC clusters, leading to significant computational costs. In addition, writing high-fidelity simulation data to disk slows down the data generation process due to I/O bottlenecks. Training deep learning models on such large datasets also demands substantial computational resources and time, often requiring distributed training across multiple GPUs. Additionally, as the dataset does not fit into memory, it necessitates efficient data loading and I/O strategies to avoid bottlenecks during training.

[4]We note that in industry-standard applications, the simulations are often run at much larger scales.

The scale of the individual samples alone is challenging. Moreover, the number of samples needed to achieve good generalisation typically increases with the data complexity, which further exacerbates the storage, I/O, and compute challenges.

2.4.2 Complex Input Parameters Space

To achieve reliable generalisation, the training data must be representative of the underlying problem, i.e., it should represent the full range of dynamics and scenarios that the model is expected to predict. This requires creating large datasets that span a high-dimensional continuous input parameters space with complex input-output relations to the resulting simulations.

For example, the behaviour design in equation (2.1) requires generating data over a range of IC parameters Λ. We choose a prior distribution over the IC parameters $p(\boldsymbol{\lambda})$ that has support over Λ and create data accordingly. The resulting joint distribution is then:

$$p_{\text{data}}(\boldsymbol{X}, \boldsymbol{\lambda}) = p_S(\boldsymbol{X}|\boldsymbol{\lambda})\, p(\boldsymbol{\lambda}), \tag{2.15}$$

where $p_S(\boldsymbol{X}|\boldsymbol{\lambda})$ is the intractable likelihood of the data given the IC parameters, which is implicitly defined by the black-box solver S. We cannot access it explicitly, but we can sample from it by running the solver with given IC parameters $\boldsymbol{\lambda}_i \sim p(\boldsymbol{\lambda})$ to get the corresponding trajectory $\boldsymbol{X}_i \sim p_S(\boldsymbol{X}|\boldsymbol{\lambda})$.

Marginalising over the parameter distribution gives the distribution of training trajectories:

$$p_{\text{data}}(\boldsymbol{X}) = \int_{\Lambda} p_S(\boldsymbol{X}|\boldsymbol{\lambda})\, p(\boldsymbol{\lambda})\, d\boldsymbol{\lambda}. \tag{2.16}$$

In practice, we get samples from this distribution via the data generation process, for selected $\boldsymbol{\lambda} \sim p(\boldsymbol{\lambda})$.

The prior $p(\boldsymbol{\lambda})$ gives the ability to control data generation. However, choosing a suitable prior is challenging. Without any knowledge about the "right" prior, the choice has to be objective [90]. A common choice is a uniform distribution, which corresponds to creating data samples that cover the input parameters space Λ as evenly as possible. However, this choice does not guarantee optimality in terms of dataset diversity and representation of complex dynamics. In practice, uniform priors often oversample regions inducing trivial dynamics (e.g., near-equilibrium states) while undersampling regions with rich, informative behaviour.

For the rod heat transfer example (see 1.2.1), the IC parameters could define two temperatures: the initial temperature everywhere along the rod and the initial heat in the centre of the rod. A uniform prior over these two parameters may lead to many samples where these temperatures are very similar, leading to trajectories with minimal temperature changes over time. These samples have limited utility and waste both storage and computational resources.

In this example we can intuitively reason about which samples would have more value and design a better prior accordingly. In systems with more complex and nonlinear dynamics, however, it is less straightforward. As the utility of a sample cannot be assessed a priori, i.e., before creating training set and performing the training with it, the uniform prior remains the default choice in practice. With increasing dimensionality of the input parameters space, the challenge of representative sampling becomes more acute due to the curse of dimensionality.

2.4.3 Mitigating the Bottleneck

In practice, common solutions to the data bottleneck are straightforward: increasing resource budgets, reducing data scale through resolution downsampling, or training on smaller datasets. Other works focus on model improvements to achieve better data efficiency, without tackling data generation or I/O constraints directly. Yet these challenges are not inherent limitations of DL-based surrogate modelling – they are consequences of treating data as fixed, as in observation-based tasks that dominate broader DL. Approached from a data-centric perspective, they become opportunities.

Synthetic data is a unique aspect of surrogate modelling that distinguishes it from broader DL tasks, as **we can directly participate in the data generation process.** In principle, this ability enables unlimited on-demand data creation. In practice, it is constrained by storage, I/O, and compute costs. Ashton et al., 2025 [13] discusses the data generation bottleneck within the context of foundation surrogates for CFD, introducing scaling laws to quantify it.

Online training is a promising solution to alleviate the storage and I/O bottleneck by coupling data generation with the training process, effectively eliminating the need for storing and reading large datasets. It transforms the training of surrogates from a storage-bound to a compute-bound process, enabling the use of larger datasets.

Active learning mitigates the compute bound by selecting the high-utility samples for training, thus reducing the number of samples needed to achieve the desired generalisation.

Moreover, online training is a natural setting for active sampling within the continuous input parameter space, as data is generated on-the-fly. This allows the data generation process to adapt based on the current model state, enabling more efficient sampling of informative data points. This thesis contributes with a novel active learning strategy tailored for synthetic data generation in an online training setting.

2.5 Conclusion

In this chapter, we reviewed the state-of-the-art in DL-based surrogate modelling, covering both the methodological design space and key architectures such as physics-informed networks and neural operators.

These models have demonstrated strong performance across diverse PDE problems – they provide accurate solutions at a fraction of the time compared to traditional methods. However, despite these advances, their training efficiency remains significantly limited. Specifically, the large scale of synthetic data incurs high storage and compute costs, and the challenge of representative sampling from complex input parameters spaces hinders model generalisation, further exacerbating these costs. Addressing these limitations requires moving beyond model-centric improvements to a data-centric perspective that actively shapes the data generation process: rather than coarsening data or limiting dataset size to fit resource budgets, we can steer the data generation process on-the-fly to produce more informative samples that enhance model training efficiency.

The following chapters develop this data-centric perspective. Chapter 3 examines online training to alleviate storage constraints, Chapter 4 presents active learning strategies for mitigating compute costs via improving data efficiency, and Part II introduces our contributions that unify these approaches for efficient surrogate training.

Chapter 3

I/O Efficiency of Deep Surrogates Training

The scale of simulation data required for training surrogates and the curse of dimensionality in sampling the solver input parameters space pose significant training efficiency challenges when the offline training approach is used. As discussed in Section 2.4, this approach suffers from storage limitations and I/O bottlenecks, while online training offers a compelling alternative by coupling data generation with model training and eliminating the need for intermediate storage.

In this chapter, we illustrate the I/O inefficiency problem of offline surrogate training in Section 3.1, describe the online training approach in Section 3.2, present the Melissa framework [187] that provides implementation of this approach in Section 3.3, and conclude the chapter in Section 3.4.

3.1 Offline Training

The offline training approach workflow for data-driven surrogates is as follows:

1. Create a set of solver input parameters $\{\lambda_i\}_{i=0}^{N}$ sampled from the input parameter space Λ according to the behaviour design of the surrogate;
2. Create a dataset $\mathcal{D}$ of simulation trajectories $\{\boldsymbol{X}_i\}_{i=0}^{N}$ by running the solver S for each input parameter λ_i and saving the results to the disk;
3. Train the surrogate model $f_{\boldsymbol{\theta}}$ for K epochs using the dataset $\mathcal{D}$ read from the disk.

We illustrate offline training in Figure 3.1.

We attempt to examine the offline training from the resource budget perspective. Through illustrative formulations of storage and compute costs, we highlight associated bottlenecks and discuss the limitations.

3.1.1 Storage and Compute Budget Limits

To obtain a surrogate within a fixed HPC system and IO setup, we are given a storage budget D in bytes, used to store the training dataset, and a compute budget T in hours[1], divided between generating the dataset and training the surrogate model.

[1] We use walltime instead of compute hours, as the latter can be allocated differently within a fixed walltime limit.

The dataset[2] volume is determined by the number of trajectories N, the discretisation scheme $h = (\Delta x, \Delta t)$, which defines the trajectory sample size $|\boldsymbol{X}| = N_x \times N_t$. The storage budget thus satisfies:

$$\mathsf{D} \geq D(N, h) = N \cdot |\boldsymbol{X}|. \tag{3.1}$$

Both dataset creation T_{data} and training T_{train} time consist[3] of time T^{FLOP} spent on arithmetic operations (FLOP) and time T^{IO} spent on storage read/write (I/O) operations. In practice, I/O is mostly overlapped with compute, but a non-overlapped residual I/O cost typically remains and contributes to the total wall-clock time [122], which is what T^{IO} captures. The total compute budget thus satisfies:

$$\mathsf{T} \geq T_{\text{data}} + T_{\text{train}} = T^{\text{FLOP}} + T^{\text{IO}}. \tag{3.2}$$

3.1.2 Compute Budget: Dataset Creation

The computational cost of generating N trajectories depends on three factors: the simulation discretisation scheme $h' = (\Delta'_x, \Delta'_t)$, the numerical solver complexity to produce one trajectory, which defines the time spent T_{solver} depending on h', and chosen I/O writing strategy and memory limit, which determine the I/O write cost T_{write} depending on storing resolution h. The total compute used for dataset creation T_{data} in hours is modelled as:

$$T_{\text{data}}(N, h, h') = N \cdot \left(T_{\text{solver}}(h') + T_{\text{write}}(h)\right). \tag{3.3}$$

We distinguish between the discretisation resolutions h' used by the solver during dataset creation and the resolutions h used to store the dataset on disk. In the deep surrogate literature [158, 82, 75, 13], it is common to use coarser resolution for storing the data compared to the one used by the solver to create it, i.e., $\Delta'_x \leq \Delta_x$ and $\Delta'_t \leq \Delta_t$. Numerical methods often require fine resolutions to ensure stability and accuracy. Downsampling helps to reduce the sample size and, consequently, increase the number of samples in the dataset that can fit in the storage budget.

There are ways to optimise the I/O write cost using such techniques as batching multiple samples together before writing to disk or using efficient file formats, e.g., HDF5 [212]. This cost, nevertheless, is inevitable, and during dataset creation, compute resources are idle waiting for the I/O operations to complete.

3.1.3 Compute Budget: Surrogate Training

After the dataset is created and stored on disk, the surrogate model $f_{\boldsymbol{\theta}}$ is trained[4] for K epochs using this dataset. The computational cost of training similarly depends on the complexity of a forward-backward pass T_{NN} of the optimisation algorithm given the batch size B and the architecture $\mathcal{A}$, and chosen I/O reading strategy and memory limit, determining the I/O read cost T_{read} affected by dataset size N and discretisation h. The total compute budget used for training the surrogate in hours is expressed as:

$$T_{\text{train}}(K, N, h, P) = K \cdot \left(N/B \cdot T_{\text{NN}}(B, \mathcal{A}) + T_{\text{read}}(N, h)\right), \tag{3.4}$$

where the dependence on B and $\mathcal{A}$ is implied through N, h, and model size P.

[2]Without loss of generality, we assume a fixed domain and uniform discretisation of a single field, yielding equal-sized trajectory samples.

[3]For simplicity, we omit time spent on such processes as network and memory operations.

[4]For simplicity, we assume the NN architecture and training configuration are fixed and optimal.

3.1.4 Competing for Compute

The total compute cost for dataset creation and surrogate training is modelled as:

$$\begin{aligned} \mathsf{T} &\geq T_{\text{data}}(N, h, h') + T_{\text{train}}(K, N, h, P) = \\ &= T^{\text{FLOP}}(N, K, h, h', P) + T^{\text{IO}}(N, K, h) = \\ &= N \cdot T_{\text{solver}}(h') + K \cdot N/B \cdot T_{\text{NN}}(B, \mathcal{A}) + N \cdot T_{\text{write}}(h) + K \cdot T_{\text{read}}(N, h). \end{aligned} \tag{3.5}$$

While in traditional ML/DL applications the compute budget is mostly dedicated to training the model, in data-driven surrogates **both dataset creation and surrogate training compete for the same compute budget**. This results in complex trade-offs between the dataset size N, training time parameterised by epochs K, and model size P needed to achieve desirable surrogate performance under fixed resource budgets. When the task is to generalise across a high-dimensional, structurally complex parameter space Λ, the trade-off becomes even more constrained. The I/O overhead further exacerbates this issue and limits scalability.

Increasing N improves coverage of Λ and is essential for good generalisation, but it increases the cost of data generation and, in the offline setting, is directly capped by the storage cost limit $N \leq N^{\star}$. In principle, these costs can be amortised by reusing stored datasets across multiple experiments, and limited N can be compensated through epoch-based training. However, additional passes over a fixed dataset yield diminishing improvements and increase overfitting risk. At the same time, achieving low generalisation error for such tasks usually requires higher-capacity models (larger P) to capture the underlying mappings, and more training time (larger K) for optimisation to converge, together increasing the training cost.

The trade-off is further hindered by the I/O bottleneck, when part of the compute budget is spent on I/O operations instead of computations dedicated to dataset creation and surrogate training, effectively tightening the limits on feasible N and K. The problem is self-reinforcing: larger N and K incur larger I/O overhead, in turn constraining these further. While effective I/O strategies can be applied and I/O time can be maximally overlapped with computations, a residual overhead remains [152, 122]. At scale, it can dominate total training time, with reports of I/O accounting for up to 60-90% of per-epoch time in large DL workloads [171, 89].

In the context of large language models, the Chinchilla scaling law suggests that, under a fixed compute budget, "the model size and the number of training tokens should be scaled equally" [79] – simply increasing model size (P) or dataset size (N) alone is suboptimal. For surrogates, however, each additional training sample carries the full cost of running a numerical simulation, sharpening the competition between data-generation and training costs. In the context of CFD surrogates, Ashton et al., 2025 [13] derives scaling laws that explicitly account for the cost of data generation and distinguishes between regimes where either the data generation or the training cost is the dominant factor in the total compute. Ashton et al., 2025 [13, Table 4] reports estimates on the computational time and cost to develop foundation models for CFD applications, for example, 1.3×10^6 GPU hours at \$10.3M with 400TB storage at \$48k/year to create a dataset, and 4.963×10^4 GPU hours at \$397k to train a surrogate, illustrating the scale of resources required for such models. However, the analysis does not account for the I/O cost explicitly, even though I/O can become a significant contributor to end-to-end training time at these scales.

Scaling deep surrogates to industrially relevant and scientifically challenging applications remains challenging due to competing compute demands, the I/O bottleneck, and storage limits, motivating alternative training approaches that mitigate these constraints and improve scalability.

3.2 Online Training

The online training approach addresses the I/O inefficiencies of offline training by **interleaving data generation and model training**, streaming data samples directly from the numerical solver to the training process without intermediate storage. Recent work by Ashton et al., 2025 [13] highlights that the online approach is one potential solution to the storage and I/O costs in training data-driven CFD surrogates at scale. Essentially, the online paradigm converts the training task from storage-bound to compute-bound, which is beneficial in the context of modern HPC systems, where the gap between compute and storage-I/O performance is growing [153].

3.2.1 Defining Online

The terms *online* and *offline* are used with different meanings across disciplines and even within DL. We discuss these terms from four perspectives.

I. Connection. The terms online and offline indicate whether two entities are connected or disconnected, for example, a user being connected to the Web. The connection can be physical or logical, and need not be continuous, as the presence of a communication path is what matters. In our context, the connection is between the data generation and model training processes, which may operate in isolation (offline) or be directly coupled and exchanging data (online).

II. Decision-making. These terms indicate whether decisions are made before the main task (offline) or in parallel with it (online). Online decision-making allows using knowledge about the ongoing main task to make adaptive and active decisions. Here, the main task is surrogate training, and the decisions concern data generation, such as selection of solver input parameters, which can be informed by the training dynamics.

III. Data availability. The terms are well-established in computer science: an online algorithm processes its input piece-by-piece in the order it is received, without having the entire input available from the start. In contrast, an offline algorithm is given all data upfront and is required to output a solution for the problem at hand [96]. In online training, a surrogate receives data samples one-by-one as they are generated by the solver, instead of operating on a dataset precomputed upfront.

Within DL, the concept of continual (also called incremental or lifelong) learning is closely related to this perspective [224]. Training data arrive sequentially, often with shifts in the data distribution when a dynamical system is studied, and the model must adapt to new data without forgetting previously learned knowledge. This concept adds two important aspects that are missing in the aforementioned definition: the model *learns* from a (possibly non-stationary) data stream, not just processes it, and the model has no or only limited access to past samples. Different continual-learning scenarios are then distinguished based on these two aspects [224].

IV. Data placement. Similar terminology appears as *in-situ* (online) and *ex-situ* (offline), from the Latin “on-site” and “off-site”. Algorithms can be executed where the

data are produced (in-situ), instead of first moving data to a separate location (ex-situ). In our context, data are generated within the solver processes, while the training algorithm that consumes this data runs on separate resources.

In HPC and scientific computing, in-situ approaches are widely used for analysis and visualisation of large-scale simulations: algorithms are executed close to where data are created to avoid expensive I/O operations [20]. For data-driven surrogates, it is likely infeasible for training to reside entirely in the solver memory due to the complexity and distribution of both processes and often different hardware requirements. The data have to be moved at least between these processes' resources, but intermediate storage can be avoided by transferring it memory-to-memory between them, which can still be interpreted as an in-situ approach.

Definition. We attempt to define the online training in our context by combining these perspectives together, as each of them captures some important aspects of the approach. We illustrate online training in Figure 3.2.

The online training of a data-driven deep surrogate is a distributed program, where two connected processes run simultaneously and continuously (*I.*):

> **Producer**: a numerical solver *generation process that produces* training data for chosen under control (*II.*) input parameters and
>
> **Consumer**: a surrogate *training process that consumes* the produced data.

Created samples are streamed (*III.*) memory-to-memory (*IV.*) from Producer to Consumer to be used for training.

3.2.2 Comparison to Offline Training

As data creation and surrogate training are now interleaved, we perform as many $\widetilde{K}$ training iterations as it takes to generate $\widetilde{N}$ trajectories within the compute budget T, which is expressed as:

$$\widetilde{N} \cdot T_{\text{solver}}(h') \approx \widetilde{K} \cdot T_{\text{NN}}(B, \mathcal{A}) = \widetilde{T}^{\text{FLOP}}(\widetilde{K}, \widetilde{N}, h', P) \leq \mathsf{T}. \tag{3.6}$$

The relative speed of data generation and training determines both $\widetilde{K}$ and $\widetilde{N}$, but can be controlled through the parameters of processes (h', B, P), as well as the parallelisation strategy and the allocation of computational resources (running processes on larger/smaller number of nodes, using more/less parallelism within each process, etc.).

Compared to the offline approach with Equation (3.5), the online training approach eliminates the I/O cost T^{IO}, replaces the storage-bound data generation time T_{data} with the compute-bound T_{solver}, and allows for more training iterations $\widetilde{K}$ and more training samples $\widetilde{N}$ within the same compute budget T. The approaches can be compared illustratively via Figures 3.1 and 3.2, and we elaborate on the implications of these differences below.

Since training data are not stored on disk, the saved storage budget can be reinvested into compute budget ($\widetilde{N} \uparrow$, $\widetilde{K} \uparrow$). Moreover, data size $\widetilde{N}$ and downsampling from the fine resolution h' to a coarser h become design choices rather than constraints imposed by the storage limit D.

Without disk I/O and as the memory-to-memory transfer time is typically negligible relative to the residual I/O overhead, the compute budget is now fully dedicated to FLOP

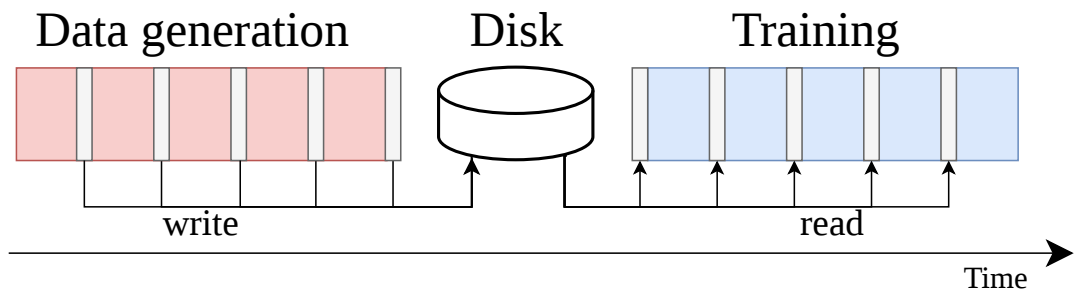


Figure 3.1: Offline training workflow: data are generated upfront and written to disk, then data are read from disk during training; residual I/O overhead stalls both generation and training compute.

execution ($\widetilde{N} \uparrow$, $\widetilde{K} \uparrow$). For simulations that produce large data within a short time, i.e., for I/O-bound solvers where I/O cost dominates the compute, this can result in significant improvements in computational efficiency.

Because new training samples are continuously generated and consumed on-the-fly, the concept of a fixed dataset no longer applies, and epoch-based training with repeated data samples is not required to compensate for the lack of data. The surrogate effectively trains for one epoch over a potentially infinite data stream, exposed to diverse samples with minimal repetition. This aligns with the scaling laws [79, 13], which suggest that performance gains should arise from model and data scaling rather than repeated exposure to the same information.

Finally, the online training approach opens opportunities for adaptive and active data generation strategies, where decisions about data generation are made on-the-fly using acquired knowledge about the training process. This can further improve surrogate performance and computational efficiency.

3.2.3 Challenges

While the online training approach offers significant advantages, it also introduces several challenges that need to be addressed for effective implementation.

Sequential nature of data generation and of simulation trajectories themselves introduce two types of biases in the training data stream that can affect the surrogate performance: intra-simulation (within a single simulation trajectory) and inter-simulation (between different simulation trajectories) biases. In addition, as the model has no or only limited access to past data samples, at any time only a subset of all generated data is available due to memory constraints. These biases can lead to catastrophic forgetting, where the model forgets previously learned information when new data is introduced [104]. Addressing these biases requires careful design of the training process to ensure that the model retains knowledge from earlier data while effectively learning from new samples.

Balancing the speeds of data production and consumption is crucial for efficient use of resources and effective learning. When consumption is faster than production, the model may exhaust available data: pausing training causes idle GPUs, while repeating data samples to keep training ongoing risks overfitting and exacerbated catastrophic forgetting. Conversely, when production outpaces consumption, training becomes a bottleneck, leading to idle solver resources or backlogs of unconsumed data that can cause memory issues.

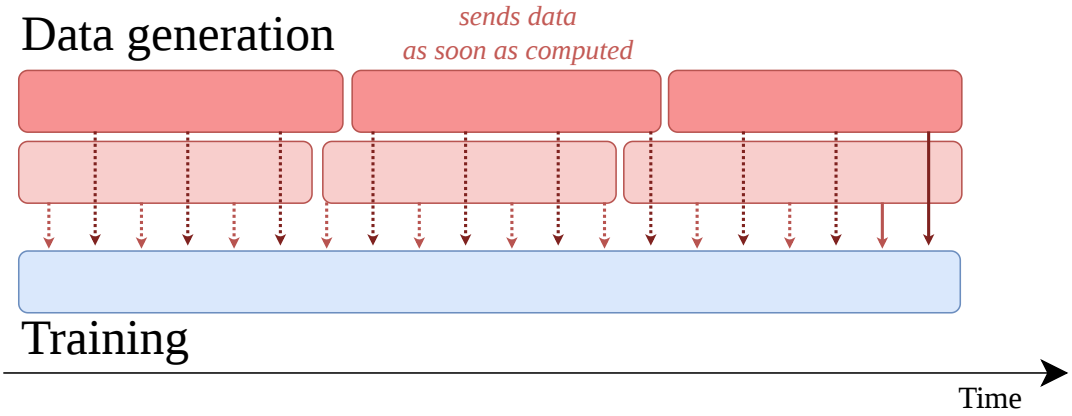


Figure 3.2: Online training workflow: training is overlapped with data generation, data from solvers are sent as soon as produced directly to the training process, avoiding storage and eliminating I/O overhead.

Implementing controlled data generation strategies that adapt to the model's learning progress adds further complexity to the workflow. It requires mechanisms for monitoring the training process and making real-time decisions about data generation parameters, which must be realised without degrading throughput or introducing additional biases in the training stream.

Orchestrating two large-scale distributed processes that run simultaneously and continuously also poses system-level challenges, including but not limited to efficient data transfer, resource allocation, and robust handling of faults and failures. These issues become more pronounced at scale, where communication overheads, stragglers, and heterogeneous hardware can significantly affect overall utilisation.

A remaining open question concerns the treatment of simulation data once it has been generated and consumed by the model. In the current framework, simulation data are regenerated for each training run and are not persistently stored, which removes storage and I/O bottlenecks but may be wasteful when simulations are strongly compute-bound. It is not yet clear whether this regenerating, storage-free regime is preferable to traditional offline workflows, or whether a hybrid strategy that selectively stores informative trajectories could better amortise generation costs while retaining the benefits of online training. Characterising this trade-off between storage, data generation cost, and surrogate performance is, therefore, a key open challenge for deploying online training in practice.

3.2.4 Related Work

Concurrent DL model training with data generation is commonly found in deep reinforcement learning (DRL) [204], where actors (data generation) explore an environment following a policy while a learner (DL model) continuously updates based on the environment simulation rewards, creating the scenario of constant back-and-forth interactions between the two processes. This paradigm shares structural similarities with online training of surrogates: both parallelise data generation and training, and DRL has driven the development of orchestration tools for large-scale online learning, such as Rllib [238], as well as mechanisms like replay buffers for data mixing and mitigating the off-policy effect

[245, 186].

However, key differences limit direct comparisons. First, DRL simulations are typically sequential or multithreaded, running on single nodes, whereas scientific surrogate training requires integration with massively parallel PDE solvers using distributed memory to achieve HPC-relevant scales. Second, DRL neural architectures required for that scale of simulations are typically smaller (shallow MLPs or compact CNNs) that the high-capacity architectures employed for deep surrogates. Third, DRL learns a policy for *interacting with an environment*, while surrogate modelling *learns the environment itself.*

Other related works include DeepDriveMD [28], which couples molecular dynamics ensembles with adaptive DL models for steering simulation campaigns, Colmena [230], which orchestrates large ensembles of simulations and ML models for decision-making in materials and chemistry workflows, and Proxima [243], a control-theoretic framework for integrating ML controllers with simulation codes. These frameworks share the idea of tightly coupling simulations with ML components and supporting adaptive, in-situ decisions at scale, but they target steering and control rather than continuous online surrogate training.

To our knowledge, the only existing framework that addresses online training of surrogates at HPC-scale is the Melissa framework [187], which we discuss in the next section.

3.3 Melissa Framework

The Melissa framework [187] is an open-source software for online data-processing from ensembles of large-scale data-intensive applications on HPC systems. It was initially developed for sensitivity analysis [211], further advanced for data assimilation [59], and recently expanded to deep surrogate training [148]. These applications involve running many instances of numerical simulations, which produce large amounts of data that need to be processed efficiently. The Melissa framework addresses the challenges of I/O bottlenecks and storage limitations by enabling online data processing, i.e., it eliminates intermediate file storage and processes data on-the-fly. Within the online paradigm, each application presents specific challenges and requirements, which are addressed by different components and strategies developed in Melissa.

Melissa is an implementation of the online training of data-driven deep surrogates. The objective of the framework is to maximise the throughput of simulation data production for distributed deep surrogate training at large scale (thousands of CPUs and multiple GPUs). To mitigate biases in the data produced sequentially and improve surrogates training convergence, the data is received into an intermediate memory buffer that supports various data management strategies.

We continue with a detailed presentation of the main components of Melissa, including data management strategies, and discuss empirical results that demonstrate the effectiveness of the framework. We follow the thesis of Lucas Meyer [147], which is dedicated to the development of Melissa, and refer the reader to it for more details.

3.3.1 Melissa Components

The framework is a client/server architecture where clients and server can have different levels of parallelism. Figure 3.3 illustrates the architecture of the Melissa framework. The **Clients** are solver codes that produce simulation data concurrently (production), and the **Server** performs (possibly on multiple GPUs) surrogate training with data received from

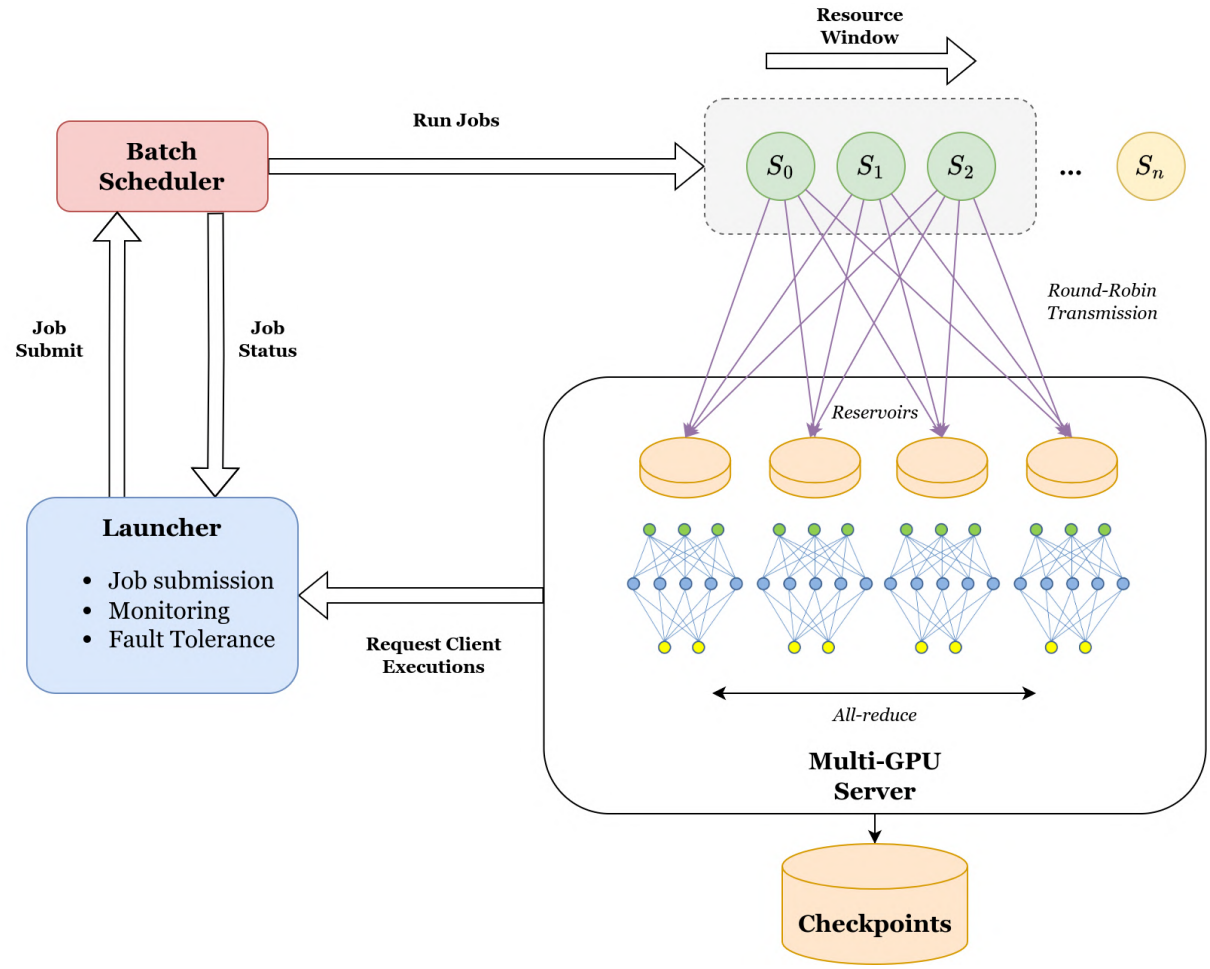


Figure 3.3: The Melissa framework architecture for online training of data-driven surrogates. Thanks to Abhishek Purandare for the original figure.

the Clients (consumption). Received data are stored in-memory in an intermediate **Buffer** to lessen synchronisation required between Clients and the Server and to improve data mixing. The workflow is orchestrated and monitored by a **Launcher** process. Settings of experiment studies are defined in a user-customised **configuration file**.

The data exchanges between the Clients and the Server are direct memory-to-memory communications, thus, **eliminating the I/O bottleneck and need for intermediate storage**. The framework supports **fault tolerance** protocol: connections between the Clients and the Server are dynamic, thus, a Client's halt, by a user, a scheduler, or due to a failure, does not lead to a halt of the Server. Moreover, in case of a Client or the Server failure, they restart automatically. The application is **elastic**, as the number of Clients running concurrently can be adapted continuously based on the availability of the resources.

Launcher. The Launcher is a Python programme that orchestrates and monitors the workflow. It interacts with the HPC system's batch scheduler, such as OAR [34] or SLURM [241], based on the directives provided by the user in the configuration file of the study. First, it starts the Server job on the allocated resources. Then, it receives from the Server a queue of requests of Client instances to run of size N — total number of simulations to be produced during the study. The Launcher builds the associated jobs in

```
#!/bin/bash
python3 solver.py --solver-parameters 1.0 4.2 59
```

client.7.sh

```
from mpi4py import MPI
from melissa_api import (
    melissa_init, melissa_finalize, melissa_send
)
import solver_generator # arbitrary solver generator that yields samples

solver_parameters = ... # parsed from command line arguments

comm = MPI.COMM_WORLD

field_previous_position = "X_t-1"
field_position = "X_t"
data_size = 100 # size of one sample

melissa_init(field_previous_position, data_size, comm)
melissa_init(field_position, data_size, comm)

for x_prev, x_cur in solver_generator(solver_parameters):
    melissa_send(field_previous_position, x_prev)
    melissa_send(field_position, x_cur)

melissa_finalize()
```

solver.py

Figure 3.4: Client code example with Melissa API for $i = 7$ and $\lambda_7 = (1.0, 4.2, 59)$.

batches of size N_S and submits them to the batch scheduler. Further, it is responsible for killing and restarting faulty Client jobs that were indicated to be unresponsive by the Server. At any moment, the Launcher is informed which of the Clients are currently running, which of them has finished, and which are waiting to be submitted.

Clients. Each Client S_j runs a simulation code $S(\cdot, \lambda_j)$ with a given set $\boldsymbol{\lambda}_j$ of variable inputs. The code can be written in C, C++, Fortran, or Python languages. It is usually parallelised with the Message Passing Interface (MPI) or similar standard to run over several nodes and cores. There are always at most N_S Clients running concurrently independently of each other.

Melissa requires adding in the solver code three API calls: communication initialisation, a simulation step send, and communication finalisation. The first connects the Client to the Server for a data transfer. The second is the data transfer. The third signals that no more data will be sent before disconnecting. See a Client code example in Figure 3.4.

The Client S_j sends every completed training sample $\boldsymbol{X}_j^{[t]}$ as soon as computed (non-blocking), where $[t]$ associates the sample with the timestep (see Eq. (2.10)) and j – with input parameter $\boldsymbol{\lambda}_j$ used in the CLient's script.

Server. The Server is mainly responsible for the surrogate training. It is an MPI code that supports distributed data-parallel training: the Server can run several Python pro-

cesses (MPI ranks $R_0, \ldots, R_{N_g}$), running an identical copy of the model, i.e., weights, architecture, and optimiser. After each batch, weight updates computed from each local batch are added, so that all model copies receive the same update.

Each Server process runs two threads. The first is the data management thread – the **Receiver** role (only on R_0). It manages connections to Clients, data reception, i.e., deletion/insertion from/into the Buffer, and communication with the Launcher. At the start of the study, the Receiver creates N Client script files `client.j.sh` with $j = 1, \ldots, N$, which include the solver input parameters $\boldsymbol{\lambda}_j$ and the path to the solver's executable. The set of parameters $\{\boldsymbol{\lambda}_j\}_{j=1}^{N}$ is created by the Server by uniformly sampling from the user-defined ranges for each parameter $\Lambda = \bigcup_{d=1}^{n_p} \Lambda_d$ in the configuration file, where n_p is a total number of parameters. Alternatively, to provide more uniform coverage of the parameter space for finite number of samples, sampling can be done quasi-uniformly, i.e., using such sequences as Latin Hypercube, Halton, Hammersley. The Receiver then sends a request to the Launcher to execute these scripts as Client jobs on the HPC system. Clients send data to the Receiver, which manages their reception into the Buffer according to the user-defined Buffer management strategy.

The second is the surrogate training thread – the **Trainer** role (on all ranks). It builds training batches from the Buffer and performs training optimisation step, i.e., batch loading to GPU, forward and backward passes (supports Torch and Tensorflow). Then all training ranks communicate through an all-reduce operation to aggregate the gradients for the NN update, and, finally, update their local copy.

A user customises the Server code by inheriting a base Server class in Python, where all main Server functions are already defined, and defines several required methods: data transformation to a training sample pair input-output; the NN model's architecture and optimiser definition; the logic for a single training step; and model's checkpoint and loading.

Configuration file. The configuration file consists of sub-configurations for each component. The two most important are the study options and DL configuration. The former sets the total number of simulations N, the number n_p and ranges of input parameters Λ. The latter sets the Buffer size N_B and the batch size B. The Launcher's configuration consists of batch scheduler directives to run the Server and a Client job. Finally, the Client's configuration consists of a path to the solver executable, its default arguments, and a list of commands to be executed on the job node prior to the solver execution (e.g., loading modules or setting flags). There is a set of minimum required configuration items to be included, however, the Server class has access to the whole dictionary, therefore, it can include more items, to customise the experiment study. The users have full freedom over this file, as they have over the Server class.

Transport library. The data is transferred from the Clients to the Server's training Buffer relying on ZeroMQ library [76] (MQ stands for Message Queue). It manages asynchronous transfers relying on internal buffers and threads on the Client and Server sides, to store data transfer messages when necessary to favour overlap of communication and computations. When the buffers are full, the data production is suspended until space becomes available.

3.3.2 Training Buffer

The training Buffer $\mathcal{B}_{\text{buf}}$ has a fixed capacity N_B and sits between the Receiver and the Trainer. When training is distributed on several GPUs (i.e., several Servers), each has its own Buffer, and data from Clients is distributed evenly across Servers, for example, in a round-robin fashion.

The purpose of the Buffer is to mitigate two inevitable biases specific to online training. The first is the *inter-simulation* bias – training batches are dominated by data from the N_S concurrently running simulations, reducing the diversity of dynamics represented. The second is the *intra-simulation* bias – training batches contain correlated trajectory states due to sequential production of timesteps by the solver and their ordered arrival to the Buffer.

Biased data affects the quality of training by introducing distributional shifts. The inter-simulation bias introduces a shift across the parameter space, while the intra-simulation bias introduces a shift over time. Both can lead to catastrophic forgetting – a phenomenon where the NN's performance degrades on previously seen data no longer represented in the current training batches [104].

The Buffer acts as a mixing pool, accumulating data from Clients over time, which decorrelates and diversifies the training batches sampled by the Trainer. The Buffer manages data through three operations: *insertion* of newly received data, *eviction* of existing items to make room as the capacity is fixed, and *selection* of samples to form training batches. The policies governing these three operations, together with the buffer size N_B, determine how effectively the biases are mitigated.

Data management strategies. The first and most simple strategy is First-In First-Out (FIFO). The Buffer can be implemented as a simple queue: to insert a new data item, the oldest item is evicted. While it provides some slack to keep the consumption on when the production is slower, it does not address the data biases.

The second strategy is First-In Random-Out (FIRO). When a new data item is ready to be received, if there is no room left, a Buffer item at a random position is evicted. To mitigate over-representation of the data received in the beginning of the study, there is a threshold, called *watermark*, that defines the minimum number of items in the Buffer before starting the training process. The first training iterations are critical for convergence of the NN, thus it is important to eliminate the bias caused by this initial filling period when the diversity is low.

Meyer et al., 2023 [148] introduced a new strategy, called **Reservoir Buffer**, to mitigate biases and enable data to be seen more than once. There is a distinction between the data that was seen by a NN and new unseen data. When an item is received, it is marked as "unseen"; after it is selected for a training batch, it is marked as "seen" and moved to a list of "seen" items.

This seen/unseen distinction defines the eviction and selection policies. When the Buffer is full and a new item is to be received, a random index is selected among the "seen" items and that item is evicted, freeing space for the new "unseen" item. If there is no "seen" item in the Buffer, the reception (and thus the production) is blocked. When building a training batch, the items are selected uniformly at random among both "seen" and "unseen" items. This allows sample repetition, avoiding GPU starvation while incorporating newly available data. The distribution of "seen" and "unseen" items in a batch is proportional to the population distribution in the Buffer. It is affected by the ratio between production and consumption speeds and the Buffer capacity N_B. Once production

has ended, the remaining data is consumed until the Buffer is empty, at which point the study terminates.

3.3.3 Empirical Results

The experiments in Meyer et al., 2023 [148] demonstrated that the Reservoir Buffer strategy enables a stable high throughput of data compared to FIFO and FIRO, and significantly mitigates the biases, which leads to better generalisation accuracy of the surrogate and faster convergence. When compared to offline training, the online framework enables to improve the surrogate accuracy due to larger training dataset size.

Meyer et al., 2023 [148] compares online training with offline training of a direct surrogate of a 2D heat transfer solver. The surrogate is an MLP of 514M parameters, which takes 5 IC/BC parameters and time step index as input and predicts the temperature field of 1000×1,000 grid points at that time step. The solver is written in Fortran and parallelised with MPI, and each simulation consists of 100 time steps.

Offline training used 250 simulations (450GB raw dataset, 95.5GB compressed) repeated over 100 epochs, with data written to disk and loaded via `mmap` across 4 GPUs with 8 parallel workers each, achieving 38 samples/sec and requiring 24 hours to train after 60k training batches. With the same batch budget and number of GPUs, online training streams data from 20,000 simulations (8TB total) generated on 5,120 cores directly to the training Server with 4 GPUs, processing 476 samples/sec and completing in under 2 hours. The offline approach exhibits clear overfitting, with validation loss converging above the decreasing training loss, while online training with a Reservoir Buffer improves validation loss by 47%, demonstrating superior generalisation. We provide Meyer et al., 2023 [148, Figure 6] that illustrates these results in Figure 3.5.

Although online training incurs 29% higher compute cost (€63.8 vs €49.1) due to larger amount of data generated, it avoids the prohibitive storage cost of €480 that would be required to persist the full 8TB dataset for an equivalent offline training. A practical production workflow would combine pre-training on a static reduced dataset for hyperparameter tuning with selective online retraining at scale using complementary data, balancing storage footprint against the compute cost of regenerating simulations.

In another publication, Meyer et al., 2023 [149] compares offline and online training with Unet, FNO, and Message Passing Neural PDE Solver surrogate models for mixed advection-diffusion, Lorenz's system, and Navier-Stokes PDEs. The results further demonstrate improved generalisation accuracy and processing of larger datasets for online training compared to offline training across a variety of surrogate architectures and applications.

3.3.4 Discussion

The Melissa framework provides the implementation of online training with many subtleties covered, supporting the deployment at scale on supercomputers. Such challenges as data bias mitigation, high-throughput data transfer, and distributed training at scale are addressed successfully, allowing scaling up the surrogate application. However, some open questions remain.

First, the Buffer is currently implemented to store training samples $\boldsymbol{X}_j^{[t]}$ ready for selection into batches, for example, pairs of consecutive time steps $\{\boldsymbol{X}_j^t, \boldsymbol{X}_j^{t+1}\}$. However, this means the Buffer contains duplicate states across training samples, for example, the

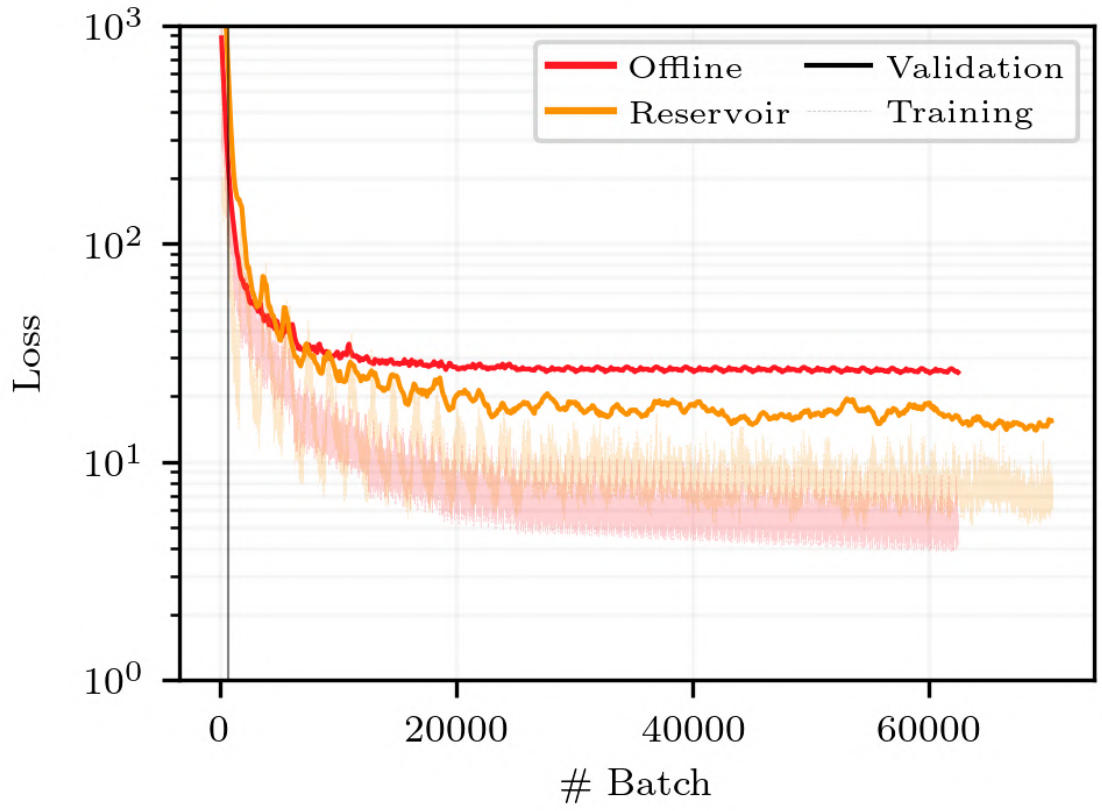


Figure 3.5: Comparison between offline and online training at equivalent numbers of batches from Meyer et al., 2023 [148, Figure 6]. Offline trains for several epochs on the data generated by 250 simulations. Online training is performed on 20,000 simulations managed by the framework.

state $\boldsymbol{X}_j^{t+1}$ appears twice in $\{\boldsymbol{X}_j^t, \boldsymbol{X}_j^{t+1}\}$ and $\{\boldsymbol{X}_j^{t+1}, \boldsymbol{X}_j^{t+2}\}$. It would be more memory-efficient to store individual states $\boldsymbol{X}^t$ and form training samples on-the-fly, in accordance with a training scheme (see subsection 2.2.2). For example, composing subsequences of n consecutive timesteps from available sequences in the Buffer for multistep training, or choosing pairs of timesteps with a prescribed interval for all-to-all training [75] Moreover, this would enable more flexible training schemes not yet explored, to our knowledge, in the context of deep surrogate training, such as non-regular multistep sequences (e.g., using $[t, t+1, t+3]$ when $t+2$ is unavailable) or gap-aware subsequence splitting (e.g., forming $[t, t+1]$ and $[t+3, t+4, t+5]$ from $[t, t+1, t+3, t+4, t+5]$ when $t+2$ is unavailable).

Second, the imbalance between data production and consumption speeds is currently managed heuristically, i.e., adjusting manually the number of concurrent Clients N_S, parallelisation size of the solver, Buffer size, batch size, number of training GPUs. If the production-consumption speed ratio is too imbalanced, one side may frequently stall, leaving Client or Server resources idle and reducing overall training throughput. These parameters could be adapted on-the-fly based on the observed speeds in order to maximise the resource utilisation, for example, using control theory approaches [69] or dynamic resource management [86]. Moreover, Melissa was not benchmarked on a challenging real-world applications yet, where the production-consumption imbalance is expected to be more pronounced due to the complexity of the solver and surrogate model. Overall, studying impact of these parameters on the surrogate training performance and resource

utilisation is an important work that remains to be done.

Third, the framework is intended to be used in a scenario where the surrogate model architecture and hyperparameters are found optimal before the online training. Otherwise, the training process is required to be restarted, losing all the data generated previously, and thus using computational resource to generate data again. An interesting direction for future work is to enable hyperparameter tuning [221] and architecture search [61] within the online training framework, which would further improve the usability of the framework for deep surrogate training at scale.

Last, we believe that more systematic comparison between offline and online training is still required to fully understand the advantages and disadvantages of both approaches across a variety of applications, surrogate architectures, and HPC systems. For example, profiling the I/O overhead incurred in offline training, even with state-of-the-art storage and I/O strategies, could show more clearly the benefits of online training, as the impact of I/O on DL training performance remains broadly understudied [152, 122]. Further, training large transformer-based surrogates for complex large-scale simulations could demonstrate the scalability of online training further, similar to Ashton et al., 2025 [13] study in the offline context.

Finally, Melissa does not support active steering of the data generation process, i.e., making active choices about which training data to produce by adapting the input parameters of the simulations based on the training progress. This would require a feedback loop between the Server and the Launcher to enable dynamic modification of the Client scripts during the experiment run. Beyond steering data generation, the Buffer management strategies could also be made active: making informed choices about which data to evict and which samples to prioritise in training batches. Potentially, eviction of less informative samples and prioritisation of more informative ones could improve the training efficiency, as well as balancing the distribution of samples in the Buffer based on how many times they were seen by the NN. Such strategies would require approaches that do not incur additional bias in the data distribution stemming from prioritisation, while maintaining high data throughput and resource utilisation.

The active steering of data generation is the research direction that we explore in this thesis. To pursue this goal, we choose Melissa, as it addresses the core challenges of online training at scale, extending and advancing it to support active strategies.

3.4 Conclusion

In this chapter, we highlighted the I/O inefficiency problem of traditional fixed-dataset (offline) training of data-driven surrogates, which impedes scaling to complex real-world applications due to limitations on storage and overhead of I/O operations. We discussed online training as an alternative approach that mitigates this problem by interleaving data generation and model training, converting the training task from being I/O and storage bound to being compute bound. This shift enables larger amounts of diverse training data and more training iterations within the same computational budget. To further advance scalability of surrogates, approaches that improve the computational efficiency of the model training and data generation have to be explored.

We overviewed the Melissa framework that implements online training of surrogates and presented results of Meyer et al., 2023 [148] demonstrating advantages of this approach over the offline training in terms of surrogate performance and scalability. While the framework addresses many challenges of online training, several research directions emerge

for further improvements.

Specifically, the Melissa framework has to be advanced to enable active steering of data generation process based on the information acquired about the surrogate performance and the solver behaviour during training. Currently, the data generation process is passive, as solver input parameters are simply sampled from a fixed uniform distribution. Incorporating a mechanism that selects next training data to be generated in an informed manner would make the training task truly online from the second perspective established in subsection 3.2.1.

In the next chapter, we explore active learning (AL) methods as a data-centric approach to improving computational efficiency of surrogate training by improving data efficiency. AL methods that work with continuous data parametrisation spaces, i.e., query-synthesis AL methods, fit naturally into the online training of data-driven surrogates, as they can be used to actively steer the data generation process by selecting the most informative solver input parameters to generate next training samples from.

Chapter 4

Compute Efficiency with Active Learning (AL)

Research on the computational efficiency of training NNs is a wide, highly active area of research that cuts across ML, systems, and hardware communities and is critical for both environmental and economic reasons. In the last decade, DL has made huge progress across many application domains, such as protein folding, robotics, text/image generation. During the last few years, capabilities and accuracy of generative models for images and texts have grown so rapidly that humans now struggle to distinguish AI-generated content from real ones, raising widespread concerns about manipulation and loss of trust in online media [44, 200]. The key factor behind these advances is scaling up DL models to billions or trillions of parameters and training them on massive amounts of data; but so have scaled the computational costs and carbon footprint of training these models [18]. Yet DL still falls short of performance requirements in many real-world and scientific applications, making further naive scaling unsustainable, both economically and environmentally. Improving the computational efficiency of DL training thus became a necessity and a major research direction.

The research direction organises solutions around three main axes: model-, resource-, and data-level optimisations [121, 18, 246]. Model-level optimisations focus on, for example, designing more efficient NN architectures and training/optimisation algorithms that converge faster or require fewer computations during training or inference, e.g., low rank approximation, weight quantisation, windowed attention mechanisms. Resource-level optimisations target arithmetic optimisation (e.g., mixed precision) and implementation/system optimisation (e.g., leveraging sparsity, neuromorphic hardware). Data-level optimisations aim to improve the quality/size of training and inference data (e.g., selective data collection, pruning, prompt engineering) and its maintenance (e.g., monitoring, storage). While significant progress has been made in all three areas, historically the AI community relied mainly on the model-centric perspective (model-/resource-level), with the data-centric approaches gaining attention only recently [246].

In the context of DL surrogates for scientific applications, the paradigm shift towards data-centric approaches is less evident. Most existing work focuses on model-centric solutions, such as physics-informed architectures [176], neural operators [112], efficient transformers [75, 82], leaving data-centric strategies underexplored or out of the spotlight. Ashton et al., 2025 [13] for the first time explicitly highlighted the challenges of surrogate training and scaling associated with data generation process.

However, the simulation data used for scientific DL surrogates brings unique oppor-

tunities and challenges. Unlike in observational-based DL, like vision and language tasks, the training data is synthetic, i.e., it is generated on-demand with solvers, which enables full control over its quantity, diversity, fidelity, and resolution. At the same time, such data is typically high-dimensional, multi-modal, and governed by complex physical laws. These characteristics both enable and necessitate data-centric strategies to effectively curate and utilise simulation data for training DL surrogates.

We argue that computational efficiency of training DL surrogates *can be significantly improved* through data-centric approaches that increase *data efficiency*. Specifically, by curating the training data generation process, these reduce the number of expensive simulations required to achieve a desired surrogate performance. The online training setting, discussed in the previous chapter, naturally provides this curation by running the data generation process alongside the surrogate training. We propose to leverage *query-synthesis active learning* [193] – methods that aim to synthesise the most informative training data samples based on the current state of the model – to guide the simulation data generation in an adaptive, online manner.

In this chapter, we present the concept of active learning in Section 4.1, discuss closely related fields in Section 4.2, review classical methods in Section 4.3 and recent advances for PDE surrogates in Sections 4.4 and 4.5, discussing their limitations and highlighting existing gaps that motivate this thesis contributions.

4.1 Improving Data Efficiency with AL

Data efficiency (also termed sample efficiency in the literature) measures how effectively each data sample contributes to a model performance relative to the costs of acquisition, labelling, processing, and storage, essentially, how to "make the most out of the data". Comparable model performance can often be achieved using either a large, low-quality dataset or a smaller, high-quality one. For example, a BERT large language model trained on a carefully curated subset of its original dataset achieves near full-data performance using only about 15% of the original training data [53]. This finding demonstrates that improving data quality can significantly reduce associated costs while maintaining accuracy, highlighting the importance of data efficiency. Moreover, when datasets are carefully collected and labelled initially, the saved resources can be reinvested into increasing dataset size and diversity, further enhancing model generalisation.

Active Learning (AL) methods focus on improving data efficiency by selectively curating the data labelling process. Labelling a data sample, or more generally, obtaining the corresponding output for an input, is done by an *oracle*, typically a human expert, which incurs significant costs. Instead of labelling every available data sample, AL methods aim to select those expected to maximise data efficiency and, consequently, model performance.

There are three main categories of AL methods, distinguished by the way unlabelled data arrives for labelling [193]:

1. **Pool-based**: A large, static pool of unlabelled data samples is available all at once. The goal is to select the N most informative samples from this pool for labelling.

2. **Stream-based**: A continuous stream of unlabelled data samples becomes available sequentially, without control or advance knowledge of future samples. The goal is to decide on-the-fly for each sample whether to label or permanently discard it.

3. **Query-synthesis (or query-based)**: An input space from which the data samples can be generated at will is available, without necessarily having knowledge of the input-output mapping or distributions. The goal is to query N points in the input space to generate the expectedly most informative samples.

The pool-based setting is the most widely studied in classical ML applications, primarily as large unlabelled datasets, such as images or texts collected from the Web or sets of data from various sensors, are simply very common [114, 193]. The stream-based setting is less common but poses unique challenges, such as data distribution shifts over time and the critical need for real-time decision-making and fast algorithms, as seen in applications like fraud detection, spam filtering, and manufacture monitoring [32]. This setting is also referred to as online active learning in some literature. The query-synthesis category is the least explored due to its unusual setting where data can be generated. However, with the rise of generative AI models, it is gaining attention, as these enable generating synthetic images and texts for training [222].

4.2 Related Concepts

Several fields share the AL's goal of improving learning efficiency from a data-centric perspective or face similar challenges.

Experimental design, also known as design of experiments (DOE) [56], aims to design experiments whose outcomes provide the most information about a system under study. Bayesian optimal experimental design (BOED) [36] models this task in a probabilistic framework, where experiments are selected to maximise the expected information gained about the system parameters. The task can be further extended to design sequences of experiments and to utilise information gathered from past experiments to construct *adaptive* experimental designs [184]. Recent works [57, 58, 87] use DL methods to amortise costs and enable real-time applications. [230, 28] integrate DOE with HPC workflows for large-scale scientific applications.

This field closely relates to AL, especially the query-synthesis setting, as both involve strategically selecting points in an input space to maximise information gain about an unknown system. In the AL context, experiment outcomes correspond to labelled (generated) data samples, the system under study is the input-output mapping being learned, and information about it is implicitly captured through the predictive model's performance. By selecting data samples (experiments) that are expected to yield the most informative outcomes, AL methods aim to efficiently learn the underlying mapping with minimal labelled data. However, DOE provides a more formal framework, where principled information-theoretic objectives are maximised, while AL methods often employ heuristic criteria. Both fields benefit from each other's advances and are recognised as related, as evidenced by the shared community workshop series "Modern Experimental Design and Active Learning in the Real World" [3].

Simulation-Based Inference (SBI) methods perform Bayesian inference of a system's model parameters when the likelihood probability distribution is intractable but simulation data are available [38]. Modern SBI approaches use DL to perform amortised inference, where the likelihood, posterior, or their ratio are approximated with NNs [164, 165, 64]. Sample efficiency is a crucial shortcoming of SBI and [38] recognises AL "as a key idea to improve the sample efficiency of various inference methods". In [139]

AL guides the acquisition of simulation parameters to generate training data for a neural density estimator.

Reinforcement learning (RL) similarly employs data-centric approaches for improving sample efficiency and model performance. *Experience Replay* technique [54] improves stability and efficiency of training Deep RL agents by selectively reusing past experiences based on, for example, priority sampling [186]. It parallels AL's data curation goal but through *reusing* existing samples rather than *generating* new ones. The *exploration-exploitation trade-off* [204] is fundamental in RL, where an agent must balance between discovering actions with potentially higher rewards – exploration – and leveraging known high-reward actions – exploitation – risking convergence to suboptimal policies. This trade-off parallels query-synthesis AL's balance between sampling within unknown regions of the input space and within known informative regions, risking redundancy.

Data *pruning*, *distillation*, and *coreset selection* aim to reduce training set size to decrease computational costs while preserving model performance [246], similarly to the example of BERT model mentioned earlier. *Curriculum learning* [22] curates the order in which training samples are presented to the model to improve learning speed and robustness, usually starting with easier samples and gradually increasing difficulty. Data *augmentation* techniques [246] perform various transformations on samples to increase dataset diversity and improve generalisation.

Semi-supervised learning utilises both unlabelled and labelled samples during training to improve model performance. AL can be viewed as a form of semi-supervised learning with the common goal of learning more from the unlabelled data, in this case by labelling it selectively. *Continual (lifelong) learning* [224] aims to enable models to learn from a continuous stream of data while retaining previous knowledge, and *federated learning* [239] focuses on training models across distributed devices with data being decentralised. Both fields share similarities with stream-based AL, as they cope with distribution shifts and real-time unlabelled data. *Knowledge distillation* methods [133] intersect with query-synthesis AL, as they involve a teacher model acting as an oracle to label data for training a student model. *Transfer learning* [224] focuses on leveraging knowledge from pre-trained models to improve learning efficiency on new tasks, and can be considered as a method to improve data efficiency, as it reduces the amount of new labelled data required for training.

Lastly, to avoid confusion, we clarify several terms. *Active* vs *passive* differentiates whether there is a strategy to make a decision or if a decision is made randomly. *Sampling* vs *selection* distinguishes between creating new samples or choosing from existing ones. *Adaptive* vs *non-adaptive* refers to whether a strategy changes based on previous experiences or remains fixed.

Adaptive or active sampling, in some literature, refers to techniques that strategically select training samples to form mini-batches for stochastic gradient descent (SGD) optimisation to improve convergence [8, 135, 97, 98, 150]. While they share AL's goal of improving learning efficiency through data curation, they differ in that AL focuses on selecting samples for labelling from unlabelled data, whereas active sampling for SGD selects from already labelled data to form training batches.

4.3 Classical AL Methods

An AL method consists of two base components: an acquisition function, which scores the informativeness of individual samples, and an acquisition strategy, which queries samples based on these scores. However, the distinction between scoring and querying is not standardised, and many prior works implicitly combine both into a single method.

In this section, we provide an overview of key AL methods categories, which is based mainly on the two surveys by Settles, 2009 [193] and Li et al., 2024 [123]. The former is a fundamental work in AL, while the latter is a recent review of the state-of-the-art AL methods for Deep Learning (DAL).

Uncertainty-based: *Select samples where predictions are most uncertain.*

Uncertainty may be measured in multiple ways; for example, for classification this includes distance to a decision boundary or class prediction probability. These scores are directly provided by the model at inference time but are not always reliable [193], motivating function variants to mitigate this issue, e.g., margin and entropy sampling. For regression tasks, these can be obtained from models with predictive distributions (e.g., Gaussian Processes) or via additional classifiers on residuals.

Ensemble techniques provide uncertainty estimates by measuring disagreement among multiple models, like Query-by-Committee [194]. Disagreement metrics include output variance, vote entropy, or Kullback-Leibler divergence, requiring training multiple instances of the model (with different initialisation or architecture). Monte Carlo dropout approximates model uncertainty by performing multiple stochastic forward passes with dropout enabled at inference time, producing an empirical output variance across passes. Recent DAL methods leverage training dynamics [115, 223, 85] or loss approximations [242] to estimate uncertainty.

Influence-based: *Select samples that most change the model or its performance.*

These approaches aim to estimate the expected changes in the model or its performance in a decision-theoretic manner. Initially these methods are proposed for classification tasks, where the expectation is taken over the possible class labels for each sample in the pool of "future samples".

One class measures the *change in the model's parameters*; for example, it can be measured through the norm of the NN's gradient update when the sample is added to the training set. Another class measures the *reduction in the expected future error* of a model, which aims to reduce future possible incorrect predictions. It can be viewed in information-theoretic Bayesian frameworks as maximising the mutual information between the selected sample and the pool of unlabelled samples, for example, [123] distinguishes this as a separate category called Bayesian approaches. A related variant measures the *reduction of the error's variance*, where the loss expectation is minimised indirectly by reducing the variance, using the bias-variance decomposition of the error. This method can be applied in query-synthesis setting, as the variance can be approximated over the input space. This approach is closely related to experimental design methods.

Representative-based: *Select samples that increase diversity or coverage.*

This complements informativeness-based methods by ensuring selected samples are not redundant. These methods are also called diversity or density-based. Diversity can be measured in various ways, e.g., distance in the input space, using clustering methods, density approximation, similarity measures. Often various lower dimensional representations of the data are used to compute diversity [105, 12, 203]. Empirical studies show that combining diversity with informativeness (uncertainty or influence) improves performance in batch DAL settings [12].

Limitations. Many classical AL methods were designed for pool-based setting and for relatively low-dimensional problems. They face practical limitations when scaled to large datasets or high-dimensional spaces [123]. Uncertainty-based methods rely on either model's ability to provide reliable uncertainty estimates, training several models, or performing several forward passes, all increasing the costs, and might not be feasible in large-scale cases. Influence-based methods are computationally expensive for large data and model spaces, as they, for example, require estimating the expected changes over the pool of unlabelled samples. In query-synthesis setting, this requires approximating the expectation over the input space, which is not trivial in high dimensions. Diversity-based methods similarly suffer from the curse of dimensionality, however, working in a lower-dimensional feature space can mitigate this issue. Recent DAL methods address several limitations through batch acquisition and learned embeddings [123].

These limitations are particularly pronounced in the context of training data-driven PDE surrogates, where data generation is expensive, input and output spaces are high-dimensional spatio-temporal domains, and physics imposes additional constraints. AL research has become particularly active in this area and especially in the closely related field of PINNs, where expensive data generation is replaced by strategic collocation point sampling to enforce physics residuals. In the following sections, we overview state-of-the-art AL strategies in the context of both PINNs and data-driven surrogates.

4.4 AL for PINN Collocation Points Selection

Although not traditionally framed as AL, collocation point selection in PINNs shares the same core principle: adaptively identifying the most informative points to minimise training costs. As discussed in Section 2.3.1, the main computational bottleneck in PINNs is computation of residuals at collocation points, i.e., calculating partial derivatives of governing equations at each selected spatio-temporal domain point. While the cost is relatively lower compared to generating simulation data with solvers for selected parameters (for data-driven surrogates), the choice of collocation points can significantly affect convergence, training stability, and solution propagation in PINNs [180]. We review prominent methods, primarily residual-based sampling strategies that use residual loss as a way to quantify uncertainty.

The first adaptive sampling method for PINNs is Residual Adaptive Refinement (RAR) proposed by Lu et al., 2021 [137], motivated by inefficiency of uniform collocation point sampling for certain PDEs that exhibit steep gradients. It is conceptually similar to FEM refinement methods. RAR iteratively adds points in regions with high residuals by selecting points with the largest residuals from a pool of randomly sampled locations. Gao and Wang, 2021 [62] samples points proportionally to residuals using Metropolis-Hastings

MCMC algorithm, and outperforms RAR. Nabian et al., 2021 [159] employs importance sampling with piecewise constant residual approximations. Wu et al., 2022 [235] proposes Residual Adaptive Distribution (RAD) method that samples collocation points from a distribution proportional to residuals and provides a comprehensive benchmark of non-adaptive and adaptive strategies over a large set of experiments. It shows that residual-adaptive methods (RAD, RAR-D, RAR) substantially outperform non-adaptive methods (uniform, Latin Hypercube Sampling, Halton) in terms of accuracy and convergence speed (2-10× reduced relative error), with RAD being the best overall.

Subsequent works build upon these residual-based methods by improving methods to approximate residual distributions for sampling. Yang et al., 2022 [240] uses triangular mesh-based estimation, Tang et al., 2022 [209] employs generative models, and Peng et al., 2022 [167] utilises node generation techniques. More advanced methods include failure probability posterior models Gao et al., 2023 [63], Gaussian mixture models Jiao et al., 2023 [91], and Neural Tangent Kernel (NTK) based training dynamics Lau et al., 2024 [119], which jointly optimise collocation and experimental points. The latter is computationally expensive due to NTK calculations, although the experiments show faster convergence yields superior quality at comparable overall training times. Daw et al., 2023 [40] proposes a retain-release-resample (R3) algorithm, which does not require a pool of candidate points. Instead, at each iteration, it retains points with residuals higher than average, releases other points, and substitutes these by sampling new random points. Experiments show that R3 significantly outperforms RAD, while being computationally efficient.

Uncertainty-based methods offer principled alternatives to sampling from residual distribution approximation. These include work of Zanisi et al., 2023 [244] that applies uncertainty-based AL with MC dropout for tokamak turbulence modelling and Aikawa et al., 2024 [7] that uses variational dropout to assess solution uncertainty.

Instead of selecting collocation points based on residuals, Anagnostopoulos et al., Anagnostopoulos et al., 2023, 2024 [10, 11] propose residual-based attention (RBA) reweighting of existing collocation points to improve gradient homogeneity, addressing the information bottleneck in PINNs from an optimisation perspective.

4.5 AL for Surrogates Data Generation

The literature that focuses on AL for data-driven PDE surrogates remains quite limited. Here, we review several representative works highlighting the current state and key challenges.

One of the most relevant recent works is by Musekamp et al., 2025 [158], which introduces a benchmark AL4PDE for evaluating AL methods applied to deep surrogates and provides an excellent overview of related work. It includes multiple parametric PDEs, state-of-the-art surrogate models (UNet, SineNet, FNO), and common pool-based batch AL algorithms, i.e., uncertainty- and feature-based, including SBAL [106] and LCMD [80]. The paper provides extensive evaluation of these methods and shows that AL can improve data efficiency up to 4×. Key findings include:

- AL significantly reduces worst-case errors, improving reliability of the model predictions;

- actively selected training data is beneficial for models not used during selection, enhancing reusability of the datasets;
- pool-based AL methods require sufficient pool sizes for optimal performance;
- uncertainty-based methods with just two small ensemble members outperform random sampling;
- model training instability significantly impacts AL methods performance.

The benchmark considers only offline settings that enable pool-based AL methods directly, as these require 1) retraining the surrogate after each AL selection round and 2) iteratively selecting points from a pre-sampled pool of input parameters. The former requirement limits applicability to large-scale surrogates like foundation models due to retraining costs, while the latter raises concerns about pool size affecting both selection quality and computational costs. In online setting, high compute costs for AL selection are infeasible. The time it takes to select new points is shown to be comparable to the training time, which is viable only in offline setting.

Another limitation is the absence of a time-equivalent random baseline: training a single random dataset matching AL's final size, but for the total duration of the AL cycle, i.e., including selection and multiple retrainings, rather than using incremental retraining as in AL. This approach more accurately reflects the true computational advantage of AL in terms of data efficiency and model training time.

Bajracharya et al., 2024 [15] investigates benefits of applying AL methods for data-driven surrogate training, i.e., examines the efficacy of traditional pool-based AL methods for surrogate modelling. It considers two NN models, ConvNet and UNet, in the context of two sources heat diffusion problem. Three AL methods are considered: uncertainty-based using entropy measure obtained with dropout, uncertainty-based using loss estimation obtained with temporal output discrepancy measure proposed by Huang et al., 2021 [85], and diversity-based using a coreset constructed based on input parameters diversity. The exponential weighted MAE loss function is used to adjust for the specific case of two sources heat diffusion. The experiments show that actively selecting simulation data, especially when estimated surrogate loss is used, consistently improves surrogate performance compared to selecting randomly. It is especially pronounced in the regions of high error, similar to the finding of Musekamp et al., 2025 [158]. However, it is noted that the entropy-based uncertainty estimation method imposes substantial computational demands. The study further emphasises that the surrogate model must be chosen carefully to ensure stable training and to fully benefit from AL methods. Finally, it outlines the need for establishing appropriate HPC infrastructure to enable efficient on-the-fly data selection during surrogate training.

Pestourie et al., 2020 [168] outlines that the training cost of surrogates is hindered by the expense of data generation, especially for high-dimensional input spaces. To address this, they apply AL, and successfully reduce the number of required simulations for training by an order of magnitude compared to uniform sampling. The surrogate model targets optical-surface components in photonic-device metasurface design, which is based on Maxwell equation solvers. The approach is uncertainty-based and employs an ensemble of NNs to estimate uncertainty following Lakshminarayanan et al., 2017 [116]. The method is applied in an iterative, pool-based manner. The results show that certain regions of the input parameter space are significantly easier for the NN to learn than others, highlighting the importance of AL. The authors use a five-member ensemble of

small NNs that produce both predictions and corresponding error estimates for a given set of parameters. Although the authors describe the algorithm as online, it is in practice an iterative pool-based method. The NNs, however, are not retrained from scratch like in Musekamp et al., 2025 [158] but are trained incrementally as new data points are added. The follow-up work of Pestourie et al., 2023 [169] applies AL for training physics-enhanced surrogates and achieves data-efficiency gains too.

Vardhan et al., 2022 [220] highlights the challenge of acquiring sufficient training data and questions expensive Bayesian ensemble-based AL methods. The authors propose a simple and scalable AL approach that works in a student-teacher manner, using an approximation of „likelihood to fail" as the acquisition metric, and explore several acquisition strategies. The approach is applied to training surrogates for three engineering design problems: pressure vessel design, propeller design, and underwater vehicle hull design. The approach still relies on a pool-based data selection scheme. The results indicate that the proposed method maintains accuracy comparable to baseline random data selection while reducing the required number of samples by 40%. However, the presented plots appear inconclusive, and the study does not include statistical analyses to verify the significance of the results.

Guo et al., 2024 [70] proposes to apply AL in a reduced low-dimensional feature space to address challenges of adaptive learning with high-dimensional input/output data and the exploration-exploitation problem. The authors combine distance-based and uncertainty-based methods and use error estimation as the acquisition metric. Experiments are conducted for training a surrogate that models a manufacturing process simulations of electron beam melting, where generating a single data point can take up to 30 minutes. The approach is pool-based and iterative, as data generation is computationally expensive and not trajectory-driven. The results demonstrate meaningful improvements in surrogate accuracy through adaptive sampling, as validated across several benchmark problems.

Li et al., 2024 [125] identifies collecting training data for FNOs as a costly bottleneck. The authors propose a multi-resolution AL method that dynamically selects input functions and resolutions to reduce costs. They modify the FNO model to integrate multi-resolution examples and to perform posterior inference. The method employs a pool of candidates with a utility-cost ratio acquisition function, using an influence-based strategy based on mutual information function and expected prediction change. Experiments include Darcy flow, Burgers, diffusion, and Navier-Stokes equations. The approach reduces data generation costs and improves surrogate's quality, though it requires retraining the surrogate model.

Subedi and Tewari, 2025 [202] studies active data collection strategies for operator learning with bounded linear operators. The authors show that active strategies achieve significantly faster theoretical convergence rates compared to passive collection, which is limited to linear decay. The theoretical bounds are empirically validated for the heat and Poisson equations. The authors emphasise that query-synthesis AL setting is particularly suitable for surrogate training.

Wu et al., 2023 [236] employs an interactive neural process to train a surrogate model that emulates stochastic simulators. This approach provides uncertainty quantification to drive AL method. The framework iteratively queries the solver using Bayesian AL, enabling continuous improvement of the surrogate with minimal data. This approach demonstrates substantial acceleration of complex stochastic simulations, such as large-scale epidemic models.

Wang et al., 2025 [229] introduces a query-synthesis AL method that optimises selected data points in a continuous space to maximise a policy function that quantifies informativeness, for example, expected information gain, by using gradient-based update procedure with explicit control of batch diversity. We believe this method can work well in an online setting, if the policy function and its optimisation are computationally efficient and scalable. The method is evaluated within an offline pool-based setting using AL4PDE benchmark [158] by complementing other AL methods (SBAL, LCMD), thus, its performance in online setting remains to be seen.

Kim et al., 2025 [101] proposes to select data points within a temporal axis in addition to the input parameter space. This method is designed to choose which time steps to simulate with the numerical solver or to approximate with the surrogate model based on an acquisition function, for example, approximate variance reduction. Similarly to the previous work, this method is evaluated in an offline pool-based setting by being combined with SBAL, leaving its applicability in online setting yet to be explored. We believe that this method fits well within online setting, as the solver and surrogate are interleaved, but the orchestration of such interplay needs to be carefully designed to ensure no idle time for either component when synchronised.

Overall, existing work on AL for data-driven PDE surrogates demonstrates that actively selecting data substantially improves data efficiency and reduces worst-case errors across a range of benchmarks. Yet most of the methods rely on pool-based selection with retraining or ensembles, which is computationally demanding and can be difficult to scale to large surrogates. Taken together, these confirm AL's promise for PDE surrogate data efficiency while highlighting key challenges: scaling to online settings, reducing selection overhead, and exploiting query-synthesis for simulators. This motivates methods effective under realistic budgets that integrate with large-scale architectures.

4.6 Conclusion

In this chapter, we reviewed AL as a data-centric approach to improve computational efficiency of DL surrogates training. Prior works in this direction are mostly focused on pool-based AL methods, which are not directly applicable in the online training setting. On the contrary, AL for PINN collocation points is closer to our setting, as it belongs to query-synthesis AL. However, since it does not involve data generation, the methods developed in this context may overlook the challenges associated with it.

In the next part, we present our contributions to the development of AL methods for online data-driven surrogate training, addressing its challenges and demonstrating the efficiency of the proposed methods through extensive experiments. We first develop a query-synthesis AL method within the context of PINNs, and then advance it to work within online data-driven settings.

Part II

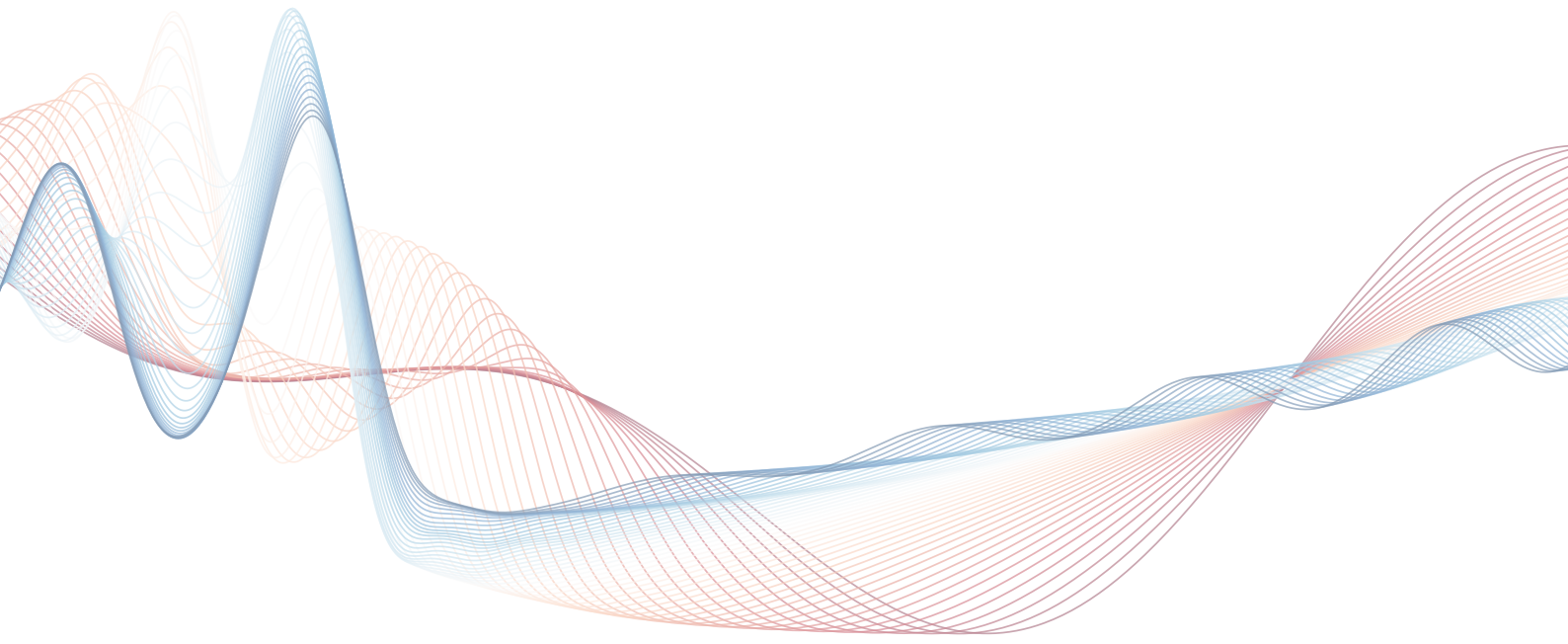
Contributions

Introduction to Part II

Building on the background and motivation developed in Part I, Part II presents the contributions of this thesis in the domain of active learning (AL) for online PDE surrogate training. The part is organised into three chapters that progress along three axes: from offline to online training, from explicit-distribution to generative AL methods, and from low-dimensional simple problems to large-scale nonlinear PDEs.

- Chapter 5 introduces Breed, a loss-driven AL method for adaptive sampling of training points, and demonstrates its effectiveness on two offline benchmarks: a data-driven toy-simulator surrogate and a PINN.
- Chapter 6 extends Breed to the online training setting, introduces an active sampling mechanism within the Melissa framework, and evaluates the resulting methods through a sequence of experiments of increasing complexity on several PDEs.
- Chapter 7 presents OGAS, a generative AL method that efficiently learns a high-dimensional evolving sampling distribution, and demonstrates its superior performance compared to Breed and adapted pool-based AL baselines in large-scale experiments across multiple PDEs and surrogate architectures.

The part is followed by an overall thesis conclusion that summarises these contributions, discusses their implications and limitations, and outlines future research directions.



Chapter 5

Adaptive Sampling of Collocation Points for PINNs

This chapter is based on the following publication:

[46] Sofya Dymchenko and Bruno Raffin. Loss-driven sampling within hard-to-learn areas for simulation-based neural network training. In *MLPS 2023 - Machine Learning and the Physical Sciences Workshop at NeurIPS 2023 - 37th Conference on Neural Information Processing Systems*, pages 1–5, New Orleans, United States, December 2023. URL https://hal.science/hal-04305233.

As described in Section 2.3.1, physics-informed learning framework was introduced by Raissi et al., 2019 [176] in order to constrain deep surrogates to respect physical laws described by PDEs. These laws are incorporated as regularising components into the NN loss function, which is evaluated to comply with the governing equations at *collocation points* within the spatial-temporal PDE domain.

While PINNs have shown promising results in various applications [216], training them efficiently and accurately remains a significant challenge [180]. In particular for high-dimensional and highly-nonlinear problems, a careful selection of collocation points is necessary to capture steep gradients and sharp transitions in the solution, while keeping the computational cost manageable. Strategies that adaptively select collocation points based on the model's performance have been proposed to address this challenge, as reviewed in Section 4.4.

In this chapter, we present our contribution to adaptive sampling strategies for PINNs [46], where we introduce a novel method called Breed. It adaptively samples collocation points based on the residual loss of the model by building a proposal distribution that approximates the loss distribution over the input domain. The method addresses exploration-exploitation trade-off by balancing the sampling between high-loss regions and uniform distribution in a scheduled manner. The experiments demonstrate Breed to converge faster and achieve lower errors compared to both uniform and R3 [40] sampling methods when evaluated on two benchmarks: a data-driven toy-example of imbalanced pits gradient descent (IPGD) and a PINN solving the Allen Cahn equation in 1D.

The chapter is organised as follows: Section 5.1 outlines the problem setup and presents related work methods, Section 5.2 describes and discusses the proposed method, Section 5.3 details the experimental benchmarks and discusses the experimental results, and finally, Section 5.4 concludes the chapter and outlines future directions.

5.1 Background

We first outline the problem setup for PINNs and then present related work methods for adaptive sampling of collocation points in PINNs.

5.1.1 Problem Setup

For PINNs, we consider the problem of training a NN $f_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ to learn the solution $u(\boldsymbol{x}, t)$ of a given PDE by (1) approximating residuals of the governing equation at collocation points in the domain and minimising the residual loss $\mathcal{L}_r$ and, optionally, (2) fitting IC/BC and data points, by minimising the data loss $\mathcal{L}_u$ (Eq. (2.14)).

Specifically, the residual loss is defined as:

$$\mathcal{L}_r(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{S}\sim p(\boldsymbol{x})}\left[r_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{i=1}^{N} \ell(r_{\boldsymbol{\theta}}(\boldsymbol{x}^i)), \tag{5.1}$$

where $\mathcal{S} = \{\boldsymbol{x}_j\}_{j=0}^{N}$ is a set of collocation points $\boldsymbol{x} \coloneqq (\boldsymbol{x}, t) \in \mathcal{U} \subset \mathbb{R}^d$ sampled from distribution $p(\boldsymbol{x})$, and $r_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the residual of the governing equation evaluated at the NN output as in Eq. (2.11).

The baseline approach is to sample collocation points uniformly across the domain, i.e., $p(\boldsymbol{x}) \sim \text{Unif}(\mathcal{U})$, and keep the training set static during training. Alternatively, quasi-uniform sequences, like Latin Hypercube Sampling (LHS) or Hammersley, can be used to fill the domain more evenly. Otherwise, to avoid overfitting to a static set of points, collocation points can be resampled uniformly periodically during training, i.e., every p training iterations.

Contrastingly, the training set can be updated in adaptive manner, i.e., based on the model's performance. From the active learning (AL) perspective, the goal is to optimise the training set $\mathcal{S}$ by selecting the most informative points in the domain that improve the model's performance. While AL methods aim to reduce labelling costs, adaptive sampling in PINNs aims to reduce computational costs by using fewer collocation points and fewer training iterations to achieve the same generalisation error. Both non-adaptive and adaptive strategies were extensively studied in Wu et al., 2022 [235].

Adaptive selection of IC/BC and data points is also possible but we focus on collocation points only, as they are the main computational bottleneck in training PINNs, following the related work methods (at the time this contribution was done).

5.1.2 Related Work

In Section 4.4, we gave an overview of adaptive sampling strategies for selecting collocation points in PINNs. Here, we focus on two related works that were the state-of-the-art at the time this contribution was done: the group of *Residual-based Adaptive* (RA) methods [137, 159, 235], and the *Retain-Resample-Release* (R3) method [40]. These methods rely on per-sample residual losses to manage the training collocation set, but differ in their strategies for selecting and updating points.

The group of residual-adaptive methods perform selection of collocation points from a large finite pool $\mathcal{P}$ of candidate points sampled uniformly from the domain. The training set is updated periodically, i.e., after every p training iterations, by evaluating the residual loss at all points in the pool using the current state of the NN. The methods differ in how they update the training set $\mathcal{S}^{(i)}$ at each resampling round i: RA-Refinement (RAR)

[137] adds m points with the highest residual loss to the existing training set, where $m \ll |\mathcal{S}^{(i)}| \lll |\mathcal{P}|$; RAR with distribution (RAR-D) [159] adds m points sampled proportionally to their residual loss values; RA-Distribution (RAD) [235] replaces the entire training set by sampling $|\mathcal{S}^{(i)}|$ points from the pool proportionally to their scaled residual loss values. RAR and RAR-D incrementally grow the training set and possibly include repetitive points, as the same points can be selected multiple times across iterations, while RAD maintains a constant training set size by replacing all points at each iteration.

These methods have two main limitations stemming from a pool-based approach. First, the cost of evaluating the residual loss over the entire pool during each resampling round grows with size of the pool and the model complexity. Second, to adequately represent the domain, the pool must be sufficiently large and well-distributed, which can be challenging in high-dimensional spaces. This can lead to missing critical regions of the domain that require more collocation points for accurate learning. The RAR and RAR-D methods also suffer from potential overfitting due to repeatedly adding the same high-loss points to the training set without removing old points, while RAD may cause catastrophic forgetting by replacing the entire training set.

To mitigate catastrophic forgetting and pool-based related limitations, Daw et al., 2023 [40] proposed the R3 method, which maintains a constant-size training set by *retaining* high-loss points from the current training set, *releasing* not-retained low-loss points, and *resampling* new points uniformly to fill the remaining slots. The method updates the training set at every iteration, i.e., $p = 1$, as there is no need to evaluate the residual loss over a large pool, which allows for more frequent adjustments based on the model's performance. At every training iteration i, the method uses computed per-sample residual losses $r_j = \ell(r_{\boldsymbol{\theta}}(\boldsymbol{x}_j))$ for every collocation point in the current set $\mathcal{S}^{(i)}$, and calculates the threshold loss value τ as the average of these losses. The next training set is formed as follows:

$$\begin{aligned} \mathcal{S}^{(i+1)} &= \mathcal{S}^{\text{ret}} \cup \mathcal{S}^{\text{new}}, \\ \text{where} \quad \mathcal{S}^{\text{ret}} &= \{\boldsymbol{x}_j \in \mathcal{S}^{(i)} | r_j \geq \tau\}, \\ \text{and} \quad \mathcal{S}^{\text{new}} &= \{\boldsymbol{x}_k \sim \text{Unif}(\mathcal{U}) | k = 1, \dots, N - |\mathcal{S}^{\text{ret}}|\}. \end{aligned} \tag{5.2}$$

The conducted experiments [40] have shown that R3 outperforms every aforementioned residual-adaptive method and baselines in terms of convergence speed and final accuracy on several benchmarks.

5.2 Methodology of Breed

In Dymchenko and Raffin, 2023 [46], we proposed a novel query-synthesis AL method called Breed (**B**alance **R**atio and **E**nhanc**E** **D**ensity), which can be applied to adaptively sample collocation points for PINNs and for general NNs with parametrised input space as well. Using computed per-sample losses, the method samples new points in Gaussian neighbourhoods of existing points proportionally to their loss values to enhance the sampling density in high-loss regions. The method balances the sampling between high-loss regions and uniform distribution in a scheduled manner to address exploration-exploitation trade-off.

Algorithm 1 Offline Breed AL method: Training points sampling algorithm.

Require: PDE domain $\mathcal{U}$; NN model $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, per-sample loss function $\ell_{\boldsymbol{\theta}}(\boldsymbol{x})$, maximum number of training iterations M; size of the training points set N, resampling period p, neighbourhood width σ, maximum number of OOB attempts $\text{oob}_{\max}$, concentrate-explore control value schedule $(\varepsilon^{(0)}, \varepsilon^{(c)}, c)$.

$\mathcal{S} \leftarrow \{N$ points sampled from $\text{Unif}(\mathcal{U})\}$
$i \leftarrow 0$
while $f_{\boldsymbol{\theta}}$ is not converged and $i \times p < M$ **do**
  Train $f_{\boldsymbol{\theta}}$ using $\mathcal{S}$ for p iterations
  $r_j \leftarrow \ell_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$ for each $\boldsymbol{x}_j \in \mathcal{S}$
  Obtain probability vector $(l_1, \ldots, l_N)$ ▷ *Eq. 5.3*
  $N_c \leftarrow \lfloor N \cdot \varepsilon^{(i)} \rfloor$ with updated $\varepsilon^{(i)}$ ▷ *Eq. 5.7*
  Select N_c parent points from $\mathcal{S}$ with replacements with $(l_1, \ldots, l_N)$ and
    obtain children points count m_j for each $\boldsymbol{x}_j \in \mathcal{S}$
  $\mathcal{S}_c \leftarrow \emptyset$
  for $j = 1, \ldots, N$ **do**
    for $k = 1, \ldots, m_j$ **do**
      $\boldsymbol{x}_j^{(k)} \sim \mathcal{N}(\boldsymbol{x}_j, \sigma^2 I)$
      Counter of attempts oob $\leftarrow 0$
      while $\boldsymbol{x}_j^{(k)} \notin \mathcal{U}$ **do** ▷ *Resample until in bounds*
        oob $\leftarrow$ oob $+ 1$
        if oob $> \text{oob}_{\max}$ **then** ▷ *If too many failed attempts, sample uniformly*
          $\boldsymbol{x}_j^{(k)} \sim \text{Unif}(\mathcal{U})$
        $\boldsymbol{x}_j^{(k)} \sim \mathcal{N}(\boldsymbol{x}_j, \sigma^2 I)$
      $S_c \leftarrow S_c \cup \{\boldsymbol{x}_j^{(k)}\}$
  $N_u \leftarrow N - N_c$
  $\mathcal{S}_u \leftarrow \{N_u$ points sampled from $\text{Unif}(\mathcal{U})\}$
  $\mathcal{S} \leftarrow \mathcal{S}_c \cup \mathcal{S}_u$
  $i \leftarrow i + 1$
return $f_{\boldsymbol{\theta}}$

5.2.1 Sampling Algorithm

We describe the method for the PINNs training setup. In Algorithm 1, we outline the complete training procedure with Breed sampling algorithm for general NNs, which can be directly applied to PINNs by replacing the loss function $\ell_{\boldsymbol{\theta}}(\boldsymbol{x})$ with the residual loss $\ell(r_{\boldsymbol{\theta}}(\boldsymbol{x}))$ evaluated at collocation points. In Figure 5.1, we illustrate the sampling procedure.

Similar to the related work methods, every p training iterations, the current set $\mathcal{S}$ of N training points $\boldsymbol{x}$ is updated at the resampling round i using per-sample residual loss r_j for every point in the current training set using the current NN parameters $\boldsymbol{\theta}_{p \times i}$.

A probability vector $(l_1, \ldots, l_N)$ is constructed based on these losses:

$$l_k := \frac{r_k}{\sum_{j=1}^{N} r_j}, \tag{5.3}$$

and used to choose a random combination *with replacements* of N_c points from the current set $\mathcal{S}$, that will serve as *parents* for sampling new points – *children* – in their neighbourhoods. The number of times each point $\boldsymbol{x}_j$ is selected is denoted as m_j, which determines

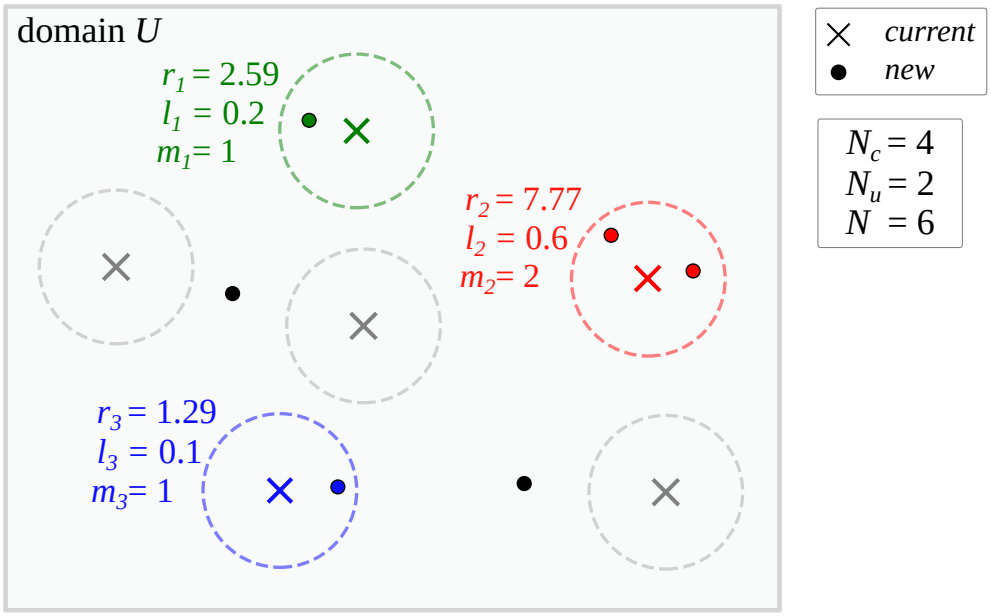


Figure 5.1: Breed sampling algorithm illustration. A training set size is $N = 6$, a concentrate-explore control value is $\varepsilon^{(i)} = 0.66$, which gives $N_c = 4$ and $N_u = 2$, as indicated. Symbols "×" indicate points of the current training set $\mathcal{S}$: in grey colour are points with zero children count, in colour are points with non-zero children count and text near them indicating their loss r_k, probability l_k, and children count m_k values. Symbols "∘" indicate points of the new training set: in black colour are points of the uniform set $\mathcal{S}_u$, in colour are points of the concentration set $\mathcal{S}_c$ sampled around parents of the respective colour.

how many children will be sampled around this parent point. As the selection is done with replacements, it is possible that some points are not selected at all ($m_j = 0$), while others can be selected multiple times ($m_j > 1$). The total number of selected parents is $N_c = \sum_{j=1}^{N} m_j$.

A children point $\boldsymbol{x}^{\text{child}_j}$ is sampled from a Gaussian neighbourhood around its respective parent $\boldsymbol{x}_j$:

$$\boldsymbol{x}_j^{\text{child}} \sim \mathcal{N}(\boldsymbol{x}_j, \sigma^2 \boldsymbol{I}), \tag{5.4}$$

where σ is a scalar hyperparameter defining the neighbourhood width. If a sampled child point falls outside the input domain $\mathcal{U}$, it is resampled until it lies within the domain or the maximum number of resampling attempts $\text{oob}_{\max}$ is reached, after which a uniformly random point from the valid input space is generated.

A *concentration* set $\mathcal{S}_c$, a set of points sampled in a loss-driven manner, contains all these children points, like:

$$\mathcal{S}_c = \bigcup_{k=1}^{N} \left\{ \boldsymbol{x}_j^{\text{child}_k} \middle| k = 1, \ldots, m_j \right\} \tag{5.5}$$

A *uniform* set $\mathcal{S}_u$, a set that ensures exploration of the input space and retains previously learned knowledge, is created by sampling $N_u = N - N_c$ points uniformly from the input domain $\mathcal{U}$, like:

$$\mathcal{S}_u = \left\{ \boldsymbol{x}^{\text{uniform}_k} \sim \text{Unif}(\mathcal{U}) \middle| k = 1, \ldots, N_u \right\}. \tag{5.6}$$

Finally, the training set is updated as $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_u$ and used for the next p training iterations.

The size N_c is defined through a *concentrate-explore control* value $\varepsilon^{(i)} \in [0, 1]$, i.e. $N_c = \lfloor N \cdot \varepsilon^{(i)} \rfloor$, which balances the number of concentrated points and uniformly sampled points. The value $\varepsilon^{(i)}$ can be scheduled over resampling rounds i with various strategies. We propose a linear-constant schedule that is expressed as:

$$\varepsilon^{(i)} = \varepsilon^{(0)} + \min\left(i/c, 1\right) \cdot (\varepsilon^{(c)} - \varepsilon^{(0)}), \tag{5.7}$$

where triplet $(\varepsilon^{(0)}, \varepsilon^{(c)}, c)$ are hyperparameters of the schedule, i.e., $\varepsilon^{(0)}$ is the starting value which linearly grows to the ending value $\varepsilon^{(c)}$ over c resampling rounds, and then remains constant.

5.2.2 Discussion

Concentration-exploration balance. A controlled balance between concentrating and exploring training points is a known approach in reinforcement learning, where it addresses the exploration-exploitation dilemma and is called an *epsilon-greedy policy* [204]. The approach also relates to stability-plasticity dilemma and catastrophic forgetting phenomenon [99]. All of these concepts regard the model's ability to learn new knowledge while retaining previously learned knowledge, which is essential in non-stationary environments with distribution shifts in data. Compared to the R3 method, Breed provides a more explicit control over the ratio of concentrated and uniform points.

We explored different non-adaptive control schedules, i.e., linear, exponential, polynomial, step function, and found that the linear-constant schedule works best in our experiments. We hypothesise that it is due to a warm-up phase, where more random points are present, followed by a gradually increased exposure of the model to harder points. This technique is similar to the curriculum learning concept [22], where the model is exposed to easier examples first, and then progressively harder ones. However, tuning the schedule hyperparameters $(\varepsilon^{(0)}, \varepsilon^{(c)}, c)$ is tedious and problem-dependent. An adaptive control approach that adjusts the mixing ratio based on the model's performance could be more effective.

Computational complexity. The computational complexity of the Breed method is moderate and comparable to related work methods.

First, re-calculating per-sample losses incurs an overhead for large models and training sets, especially if several passes are required, i.e., when the number of points exceeds the maximum possible batch size. Compared to the related work methods, the proposed method does not mitigate this overhead, however, the overhead is smaller than for RA-based methods that require evaluating losses over a large pool of candidate points.

Second, weighted selection of N_c points with replacements has complexity $O(N_c)$, which is comparable to or lower than: sorting in RAR $O(|\mathcal{P}| \log |\mathcal{P}|)$, weighted selection in RAR-D $O(m)$ and in RAD $O(N)$, and ranked selection in R3 $O(N)$. Vectorised implementations and parallelisation can further reduce the overhead.

Third, sampling from a multivariate Gaussian distribution has complexity $O(d^3)$, which is decreased to $O(d)$ for diagonal covariance matrices. Compared to related work methods, this is an additional overhead, which **can create a bottleneck for very high-dimensional input spaces**. In practice, for PINNs, the input dimensionality is typically low, making this overhead negligible.

Fourth, complexity of concentrate set creation is $O(N_c)$ but can be parallelised across parents, as sampling around each parent is independent of others. Similarly, complexity of uniform set creation is $O(N_u)$, which can be parallelised.

Total complexity of a resampling round of the method (not vectorised and without parallelisation) is:

$$O\left(p(N, d, |\boldsymbol{\theta}|) + N + N_c \cdot d\right), \tag{5.8}$$

where $p(N, d, |\boldsymbol{\theta}|)$ is the complexity of computing per-sample losses for N points of d dimensions for model size $|\boldsymbol{\theta}|$, which for the PINNs residual loss will additionally depend on the PDE complexity, i.e., derivative orders.

For other methods, the total complexity of a resampling round is:

- RAR: $O\left(p(|\mathcal{P}|, d, |\boldsymbol{\theta}|) + |\mathcal{P}| \log |\mathcal{P}| + m\right)$,
- RAR-D: $O\left(p(|\mathcal{P}|, d, |\boldsymbol{\theta}|) + m\right)$,
- RAD: $O\left(p(|\mathcal{P}|, d, |\boldsymbol{\theta}|) + N\right)$,
- R3: $O\left(p(N, d, |\boldsymbol{\theta}|) + N\right)$.
- Uniform resampling: $O\left(N\right)$.

Overall, the method incurs a small computational overhead compared to R3 method, while being more efficient than RA-based methods that require evaluating losses over a large pool of candidate points $\mathcal{P}$. For all sampling methods, $p(N, d, |\boldsymbol{\theta}|)$ dominates the total complexity, being the main bottleneck that has to be addressed.

Non-adaptive neighbourhoods. Another limitation of the Breed method is the use of diagonal covariance matrix for the Gaussian neighbourhoods. While this choice simplifies the sampling procedure and reduces computational overhead, it may not capture the true geometry of high-loss regions in the input space. Anisotropic covariance matrices, i.e., with non-zero off-diagonal elements, that adapt to the local structure of the loss landscape could potentially improve the sampling efficiency and model performance. However, estimating such covariance matrices with low computational cost is a challenging task.

Additionally, the width hyperparameter σ is fixed for all parents, all resampling rounds, and is used to create neighbourhoods of the same width in all directions. Similarly, while being a straightforward choice, it may not be optimal for all regions of the input space. Moreover, for points near the boundaries of the domain, a large σ can lead to excessive resampling outside the domain, resulting in increased resampling attempts and computational overhead. An adaptive approach that adjusts σ based on the local loss landscape, past loss history, joint loss distribution over the points, or the distance to the domain boundaries could enhance the sampling efficiency and model performance.

Out-of-boundary resampling. While rejection sampling with a maximum number of attempts and fallback to uniform sampling is a simple and effective method for handling out-of-boundary (OOB) samples, it has its limitations. This strategy may repeatedly hit the oob_{max} limit, which adds sampling overhead. This effect worsens in higher-dimensional parameter spaces, where the acceptance rate of rejection sampling tends to decrease [182]. Drawing uniformly from the entire input space even when only one dimension violated its bounds is too coarse, as it discards the locality information provided by the parent sample, thus potentially hindering the ability to focus on informative regions of the parameter space.

Relation to importance sampling. The proposed method echoes an adaptive multiple importance sampling (AMIS) approach, where samples are drawn from a mixture of proposal distributions that are adapted based on the target distribution [31], and specifically the Population Monte-Carlo (PMC) algorithm [35]. In Breed, the target loss distribution is approximated over the input domain, and the proposal distributions are Gaussian neighbourhoods centred at previously sampled training points. However, the main conceptual difference with AMIS is that the target distribution is non-stationary and defined implicitly through the model's performance.

Elvira and Chouzenoux, 2022 [48] proposes an optimised PMC algorithm that updates not only the locations but also the covariances of the proposal distributions, exploiting geometric information of the target distribution. This is similar to the aforementioned discussion of the non-adaptive neighbourhoods. Pal et al., 2023 [163] employs a normalizing flow generative model to construct more expressive proposal distributions in AMIS framework, a particularly interesting direction for improving the sampling efficiency of the Breed method, especially in high-dimensional input spaces.

Additionally, the sampling algorithm resembles evolutionary and genetic algorithms [157], where a population of candidate solutions is evolved over iterations through selection, mutation, and crossover operations, and cross-entropy methods [157], where a distribution over the solution space is iteratively updated based on the performance of sampled solutions – some ideas from these fields could be potentially adapted to further improve the sampling efficiency and model performance of the proposed method.

5.3 Experiments

To evaluate the performance of the proposed Breed method, we conduct experiments on two distinct benchmarks against the R3 [40] and a uniform resampling methods. We choose to compare to R3 as it was shown to outperform other residual-adaptive methods in Daw et al., 2023 [40], and uniform resampling was shown to be a strong baseline in Wu et al., 2022 [235]. Each experiment is run with 5 different random seeds to account for variability due to random initialisation and sampling. We assess the results quantitatively by measuring validation loss over training iterations and qualitatively by visualising the loss distribution over the input domain. The Python implementation of Breed, experimental setup, and validation datasets can be found at `https://gitlab.inria.fr/breed/breed`.

5.3.1 Benchmarks

Here we present the two benchmarks: (1) a data-driven NN surrogate of a toy-simulator of gradient decent on a surface, which we designed specifically to test adaptive sampling methods, and (2) a PINN learning the solution of Allen Cahn equation in 1D, following the setup from Daw et al., 2023 [40].

Imbalanced Pits Gradient Descent (IPGD) Toy-Simulator

Let's first introduce the toy-simulator benchmark. Consider a surface in 3D space expressed as a sum of M Gaussian pits, defined as:

$$z = f(x, y) = \sum_{i=1}^{M} w_i \cdot \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{2\omega_i^2}\right), \tag{5.9}$$

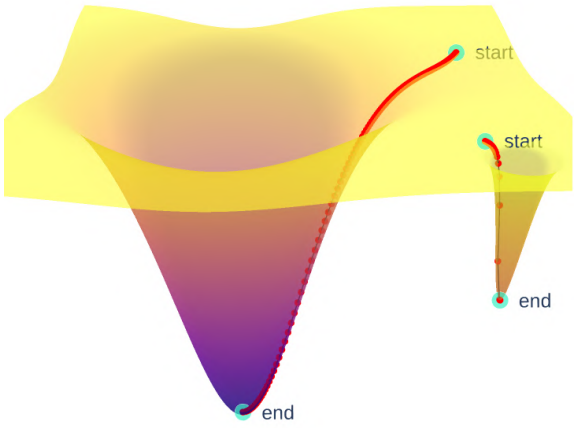


Figure 5.2: Illustration of the IPGD surface and two toy-simulator trajectories starting from different points.

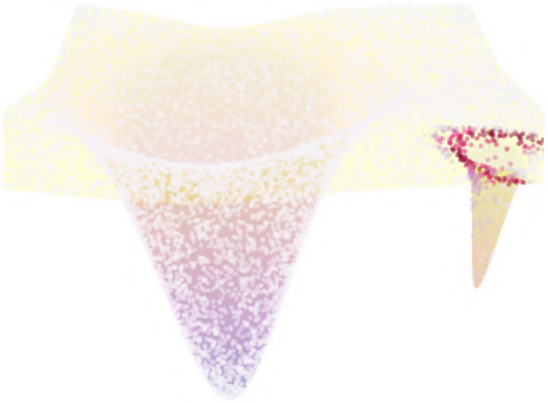


Figure 5.3: Illustration of the sensitive region on the IPGD surface (red-coloured points), where the gradient is near-zero and the output is highly sensitive to small changes in the input.

where centres (x_i, y_i), weights w_i, and widths ω_i are configurable parameters. Specifically, we configure an *imbalanced pits* (IP) surface with $M = 4$ and the following parameters:

- input domain $\mathcal{U} = [-2, 2]^2$,
- centres $C_1 = (-0.5, -0.5)$, $C_2 = (1.5, 1.5)$, $C_3 = (1.5, -1.5)$, $C_4 = (-1.5, 1.5)$,
- weights $\boldsymbol{w} = (0.895, 0.005, 0.05, 0.05)$, and widths $\boldsymbol{\omega} = (0.4, 0.008, 0.5, 0.5)$.

It results in two highly imbalanced pits with centres C_1, C_2 and two additional wide and shallow pits with centres C_3, C_4. The former two pits create a surface with a sensitive region in their intersection, where the gradient is near-zero. The latter two pits are added to prevent flat regions outside the first two pits.

The toy-simulator performs *gradient descent* (GD) on this surface. The input of the simulator is a 2D point (x_s, y_s), which is projected onto the surface as $(x_s, y_s, f(x_s, y_s))$. From this starting point, the simulator runs GD until convergence in a local minima point (x_m, y_m, z_m). The output of the simulator are the coordinates (x_m, y_m). The goal of the surrogate NN is to learn this mapping from input points (x_s, y_s) to the corresponding minima points (x_m, y_m). See an illustration in Figure 5.2.

Given the IPGD surface configuration, there are two possible minimum points, one in each of the two imbalanced pits. The challenging aspect of this benchmark is the *high sensitivity of the output to small changes in the input* within the near-zero gradient intersection of the pits. A small perturbation in the input can lead to a switch in the converged minimum from one pit to another, resulting in a significant change in the output. This behaviour creates a complex decision boundary in the input space that the NN surrogate must learn to accurately predict the correct minimum for any given input.

Uniform sampling of training points results in the sensitive region being represented worse than other regions of the input space due to its relatively smaller area. As illustrated in Figure 5.3, the sensitive region is small and concentrated, while the majority of the points lie in regions with smooth gradients. As a result, number of training points required to sufficiently cover the sensitive region increases.

The goal of adaptive sampling methods is to find a sampling distribution that increases data efficiency by focusing on informative regions of the input space. In IPGD

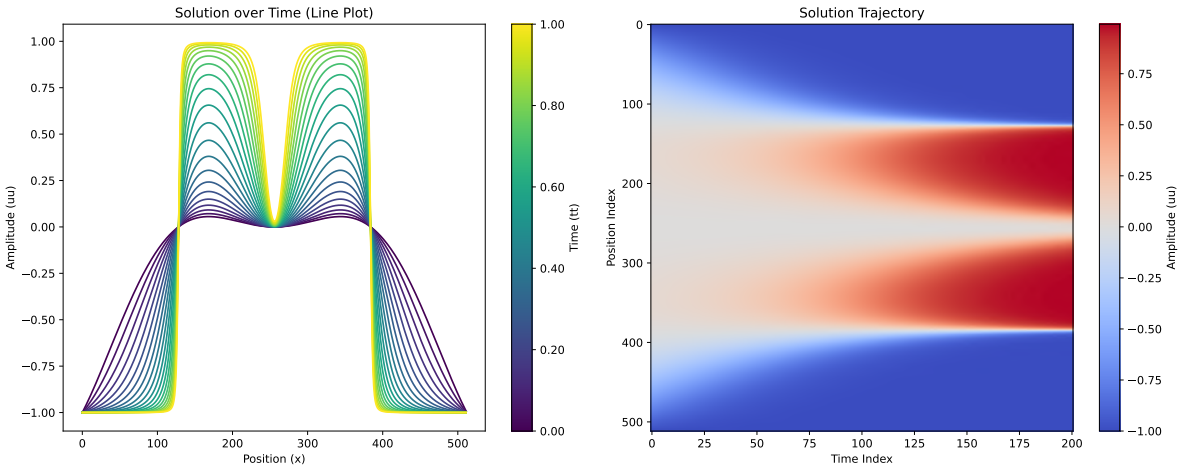


Figure 5.4: Example of the 1D Allen Cahn equation solution. On the left, the solutions at different time steps are shown as line plots, with x-axis representing the spatial dimension, y-axis representing the solution value, and colour of the line indicating associated time step. On the right, the same trajectory is shown as a heatmap, where the y-axis represents the spatial dimension, the x-axis represents time, and the colour indicates the solution value.

case, one can visually identify such regions, or use gradient magnitude of the surface as a proxy for informativeness. However, for not-toy problems with black-box solvers and complex governing equations, defining optimal distribution a priori is not as straightforward. Therefore, to guide the sampling process, adaptive methods must rely on the information that can be gained during training.

In our experiments, to evaluate a surrogate performance within the sensitive region, we create an "area-of-interest" hard validation set, which consists of 10k points sampled uniformly within the region where the surface's gradient is less than 4×10^{-3}. The regular validation set of 10k points is sampled uniformly across the whole input domain.

Training and Breed hyperparameters. For the surrogate model, we use a shallow fully-connected NN architecture of 4 hidden layers ([64, 32, 16, 8]) and ReLU activations. It is trained for 100 epochs using a training set of size $N = 1000$, which is resampled every $p = 1$ epoch. We use Huber loss, which is a combination of L1 and L2 losses, where small errors (below a threshold $\delta = 1.0$) are penalised quadratically, and large errors (above the threshold) are penalised linearly. We choose Huber loss over MSE loss as it is more robust to outliers, which are present in the sensitive region due to abrupt changes in the output.
The model is trained using Adam optimiser with a learning rate of 10^{-2}, weight decay of $5 \cdot 10^{-5}$, and batch size 50. The learning rate uses a reduce-on-plateau scheduler, where it is reduced by a factor of 0.1 when the regular validation set loss does not significantly improve for 5 consecutive epochs. We tuned these hyperparameters to achieve the best performance when trained with uniform baseline.
For Breed, we use schedule parameters ($\varepsilon^{(0)} = 0.15, \varepsilon^{(c)} = 0.7, c = 25$), width $\sigma = 0.005$, i.e., neighbourhoods widths are about $0.005 \times 6 = 0.03$ in diameter, which for the domain length of 4 ($\mathcal{U} = [-2, 2]^2$) results in approximately 0.75% in each dimension.

Allen Cahn Equation PINN

The second benchmark is based on [40], where a PINN model is trained to solve Allen Cahn (AC) equation in 1D. It is a nonlinear reaction-diffusion equation that models interface motion in time, e.g., phase separation in alloys. The solution exhibit steep gradients and sharp transitions, making the choice of collocation points critical for accurate approximation. See the illustration of the solution in Figure 5.4.

We follow the same experimental setup as in Daw et al., 2023 [40]. The AC PDE is defined on the spatial domain $\mathcal{U}_x = [-1, 1]$ on time interval $[0, 1]$ as:

$$\frac{\partial u}{\partial t} - 0.0001\frac{\partial^2 u}{\partial x^2} + 5u^3 - 5u = 0, \tag{5.10}$$

with initial condition (IC) and periodic boundary conditions (BC):

$$u(x, 0) = x^2 \cos(\pi x), \quad u(t, -1) = u(t, 1), \quad \frac{\partial u}{\partial t}\Big|_{x=-1} = \frac{\partial u}{\partial t}\Big|_{x=1}. \tag{5.11}$$

The PINN model is a fully-connected NN architecture of 4 hidden layers ([128, 128, 128, 128]), with hyperbolic tangent activations, and periodic encoding, which encodes input (t, x) into a vector:

$$\left[t, 1, \cos\left(\frac{2\pi}{L}x\right), \sin\left(\frac{2\pi}{L}x\right), \ldots, \cos\left(\frac{2\pi M}{L}x\right), \sin\left(\frac{2\pi M}{L}x\right)\right], \tag{5.12}$$

where $M = 10$ and $L = 2$, as in Daw et al., 2023 [40], thus the input dimension is 22. The model is trained for 50,000 epochs using a collocation training set of size $N = 1000$, which is resampled every $p = 1$ epoch, to minimise residual MSE loss. IC/BC losses are also included in the total loss computation using fixed training sets obtained from the ground-truth solution. The ground-truth solution is discretised on 128 spatial grid points over 101 timesteps (including initial condition), which is used for validation and IC/BC training sets, following Daw et al., 2023 [40]. The total loss is a weighted sum of residual loss over collocation points and IC/BC data losses with weights 1 and 100, respectively. The model is trained using Adam optimiser with a learning rate 10^{-3} and scheduled to decay by a factor of 0.9 every 5000 iterations.

For Breed, we use the schedule parameters ($\varepsilon^{(0)} = 0.15, \varepsilon^{(c)} = 0.7, c = 2000$), width $\sigma = 0.001$, i.e., neighbourhoods widths are about $0.001 \times 6 = 0.006$ in diameter, which for the spatial domain size 2 ($\mathcal{U}_x = [-1, 1]$) results in approximately 0.3% of the spatial domain.

5.3.2 Results

In both benchmarks, Breed shows faster convergence, lower validation errors, and more reliable training compared to the R3 and uniform resampling methods.

For the IPGD benchmark, we present the average validation loss over training iterations in Figure 5.5 for regular (5.5a) and hard (5.5b) validation sets. In both cases, Breed achieves about 3× lower error than uniform resampling and one magnitude lower error than R3, outperforming them consistently throughout training. Moreover, we can see how big is the difference in scale of errors between regular and hard validation sets, which highlights the challenge of learning within the sensitive region. Specifically, on the regular validation set, uniform resampling reaches average error around 8.5×10^{-3},

while on the hard validation set, the error is around 1.9×10^{-1}. For Breed, the respective errors are around 3×10^{-3} and 0.7×10^{-1}. This highlights that non-uniform adaptive sampling methods are necessary to improve performance in such regions, and Breed is able to achieve this effectively while reducing overall error.

To visually analyse and assess the performance within the sensitive region, we illustrate predictions and loss distributions over the input domain after training with Breed in Figure 5.7 and uniform resampling in Figure 5.8. We use both scattered point-plots and histograms to present these distributions. For Breed, we observe a more clear separation between the two minima in predictions, indicating that the model has learned the decision boundary more accurately. The loss value is distributed more uniformly across the input space, with fewer high-loss points, especially within the sensitive region. The histograms further confirm this, showing that Breed has a tighter right-tail loss distribution with fewer outliers.

For Allen Cahn benchmark, we present the relative L^2 validation error over training iterations in Figure 5.6. We observe higher variability across random seeds for this benchmark compared to IPGD, which indicates that the training is less stable and more sensitive to initialisation and sampling. Motivated by this observation, we present both the average error over random seeds (5.6a) and individual runs (5.6b) to provide a comprehensive view of the results.

Breed shows notably faster and more reliable convergence compared to both R3 and uniform resampling. Uniform resampling manages to converge in 1/5 runs at approximately 30k iterations, while R3 converges in 3/5 runs at approximately 13k, 20k, and 25k iterations, respectively. In contrast, Breed converges in 5/5 runs at approximately 7k iterations. Interestingly, the moment of reaching the highest $\varepsilon^{(c)}$ ratio (at 2k iterations) coincides with the start of the decrease in validation error for all runs, with a significant drop later (at 5k iterations), indicating that focusing more on high-loss regions at this stage of training is beneficial for convergence.

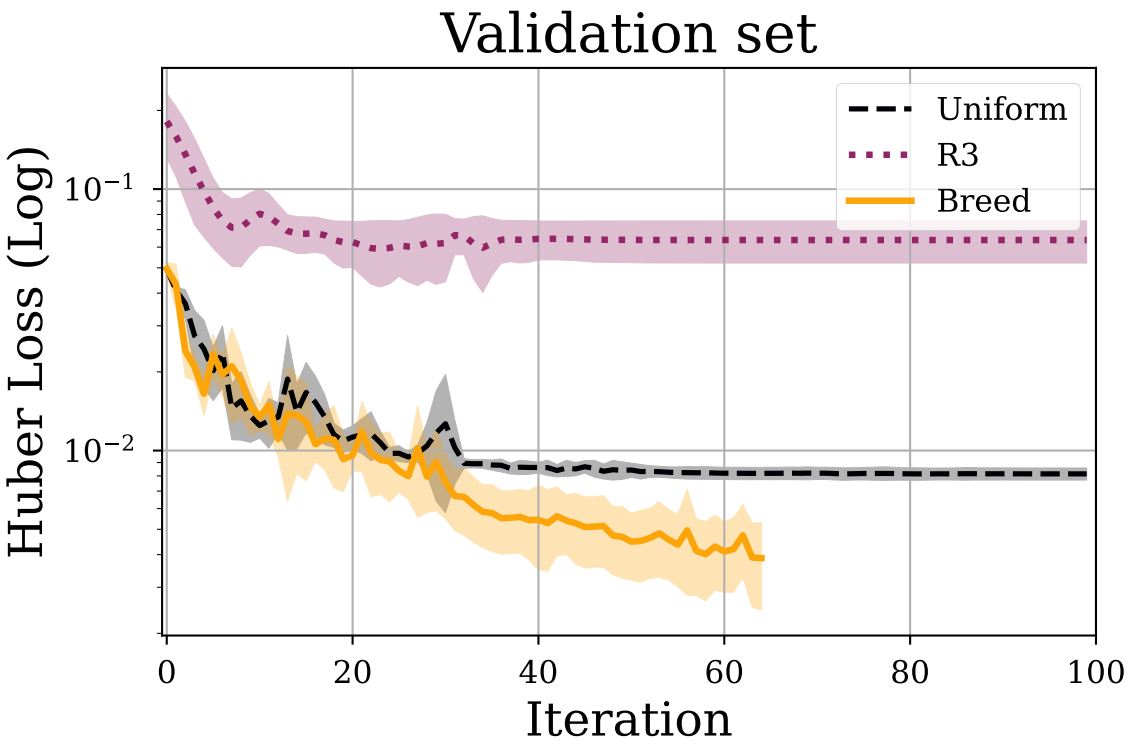
5.3.3 Discussion

The experimental results demonstrate effectiveness of the proposed method, however, the experimental scope is limited to two low-dimensional benchmarks and two baseline methods. Performing more extensive experiments for higher-dimensional nonlinear and chaotic problems would strengthen the evidence of the method's effectiveness. An ablation study of the method's components, such as the sampling schedule and neighbourhood width, is necessary to provide insights into their contributions to the overall performance.

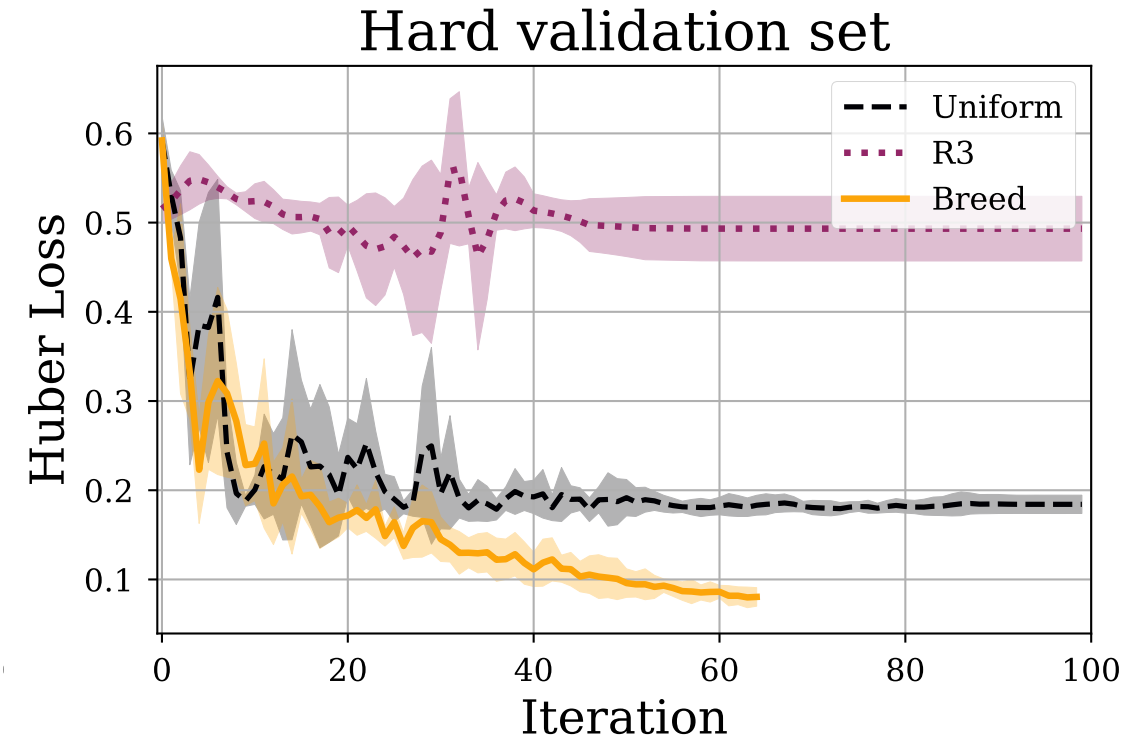
Considering experiments with PINNs, prior related work methods have performed sampling of collocation points only, without adapting IC/BC and data points, and the proposed method follows this approach. However, we noticed in our experiments that optimising the residual loss alone can create imbalance among loss components in the PINN's multi-objective optimisation. This imbalance hinders the model to converge to a correct solution, as it fails to propagate IC/BC and data information. Therefore, extending the method to adaptively sample these points jointly with collocation points could improve convergence and stability of PINNs, as it was proposed by Lau et al., 2024 [119].

5.4 Conclusion

In this chapter, we presented the Breed method for adaptive sampling of training points for data-efficient training of NNs. By sampling in Gaussian neighbourhoods of current training points proportionally to their loss values, Breed enhances the density of training points in high-loss regions. Using concentrate-explore control value, the method balances the sampling between high-loss regions and uniform distribution in a scheduled manner to address exploration-exploitation trade-off, with a linear-constant schedule proposed to gradually increase the exposure of the model to harder points. The method was evaluated on two benchmarks: a data-driven NN surrogate of a toy-simulator of gradient descent on an imbalanced surface, that is specifically designed to test adaptive sampling methods, and a PINN learning the solution of Allen Cahn equation in 1D. The results demonstrated that Breed outperforms both R3 and uniform resampling methods in terms of convergence speed, final accuracy, and training reliability, showing visible improvements in capturing complex regions of the input space. In the next chapter, we build upon the presented method and extend it for data-driven surrogate training within an online setting.

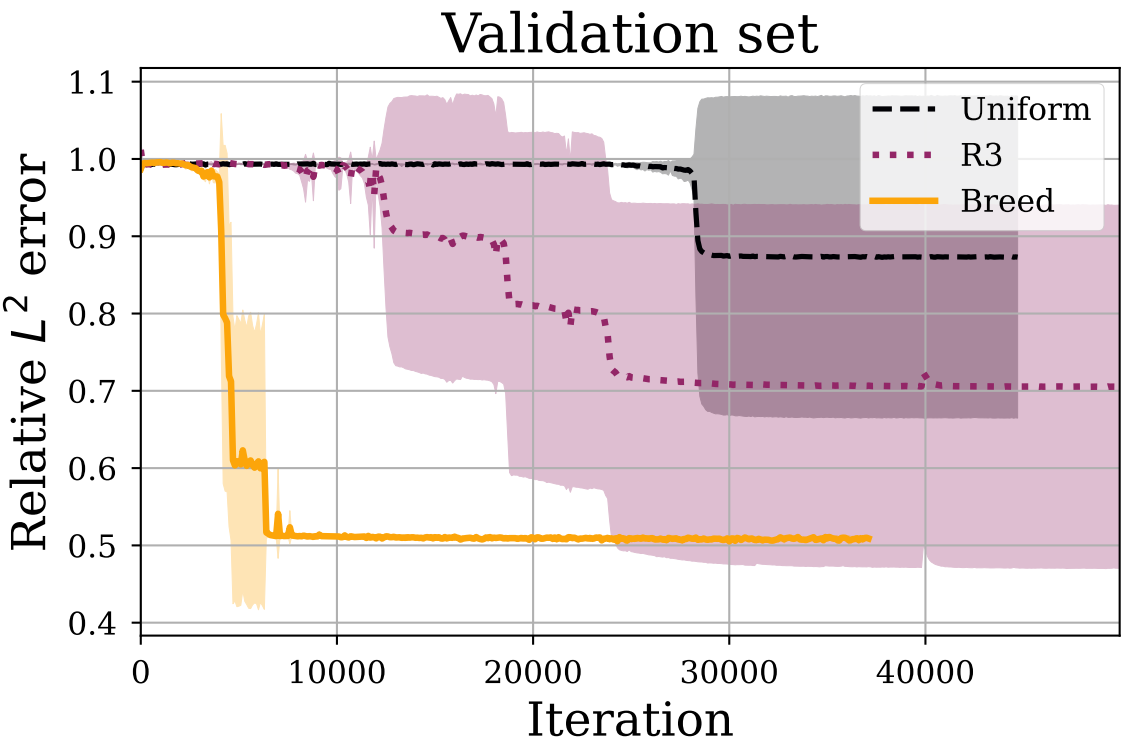


(a) Regular validation set. The y-axis is in log-scale.

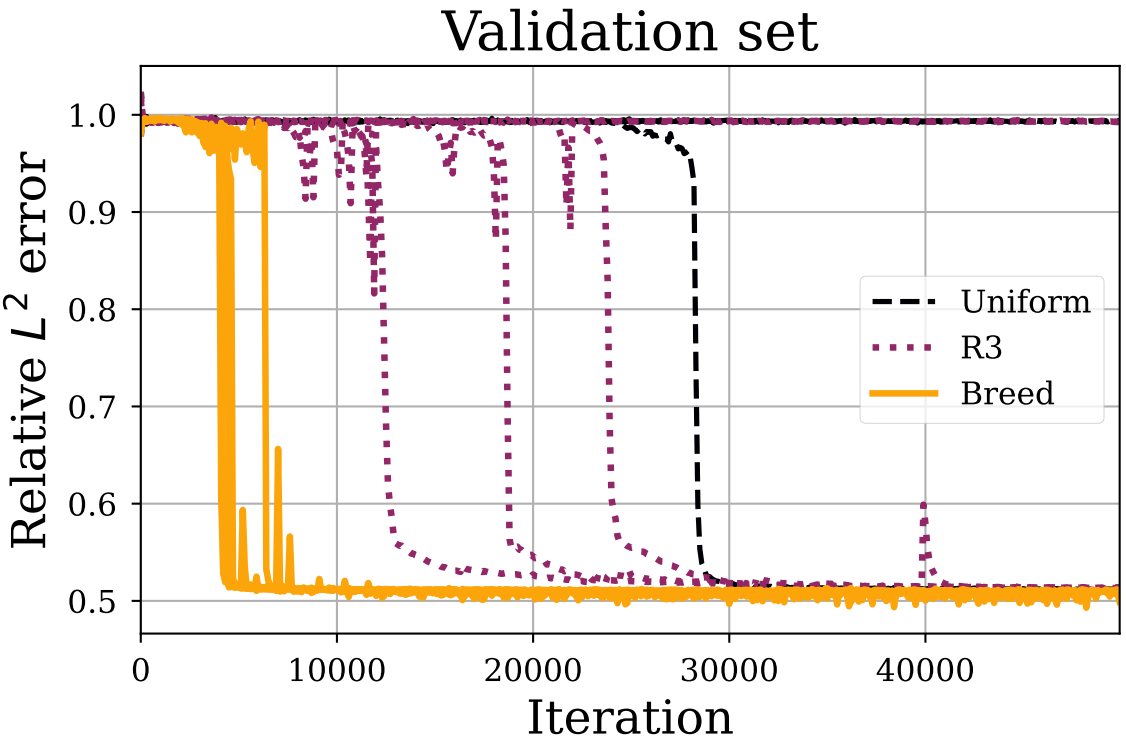


(b) Hard validation set of points within the area-of-interest where the surface gradient is less than 4×10^{-3}.

Figure 5.5: IPGD benchmark: Validation loss over training iterations for Breed, R3 and uniform sampling. The plots present distribution over 5 random seeds as mean (line) and standard deviation (shaded area). Evaluation is done on (a) regular and (b) hard validation sets.



(a) Distribution over 5 random seeds.



(b) Individual runs for 5 random seeds.

Figure 5.6: Allen Cahn benchmark: Validation error over training iterations for Breed, R3 and uniform sampling. The plots present (a) distribution over 5 random seeds as mean (line) and standard deviation (shaded area), and (b) individual runs for each random seed.

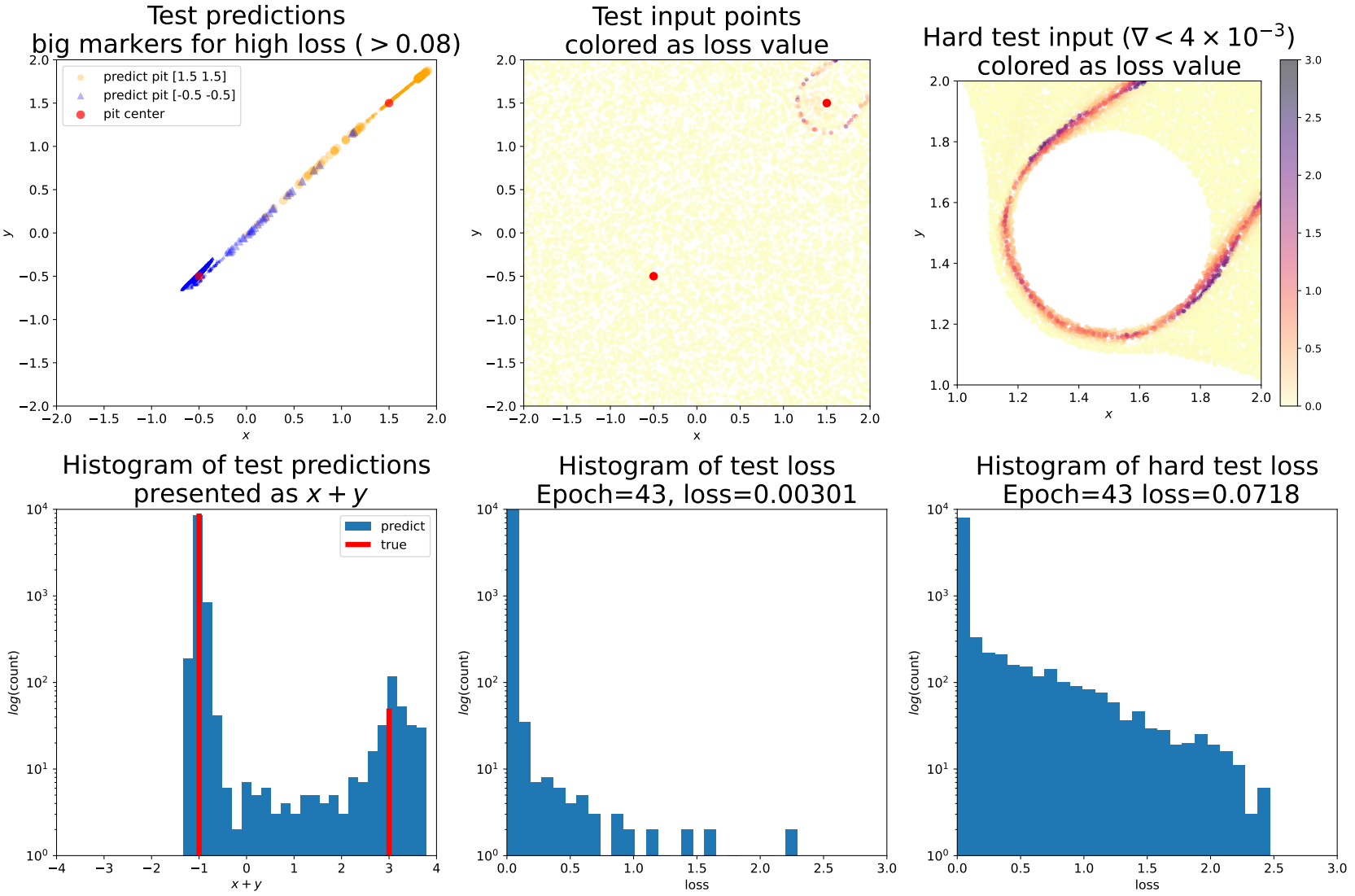


Figure 5.7: IPGD benchmark. Loss distribution over the input domain after 43 training iterations using **Breed method** (left to right, top to bottom): a scatter-plot of the model's predictions over the domain (blue colour are points that correspond to the minimum of C_1, orange – for C_2, red points indicate both C_1, C_2); two scatter-plots of loss distribution over the input points for regular and hard validation set (colour indicates loss value – darker is higher); a histogram of test predictions with x-axis corresponding to $x_m + y_m$ (red lines indicate ground truth values); two histograms of test losses for regular and hard validation sets.

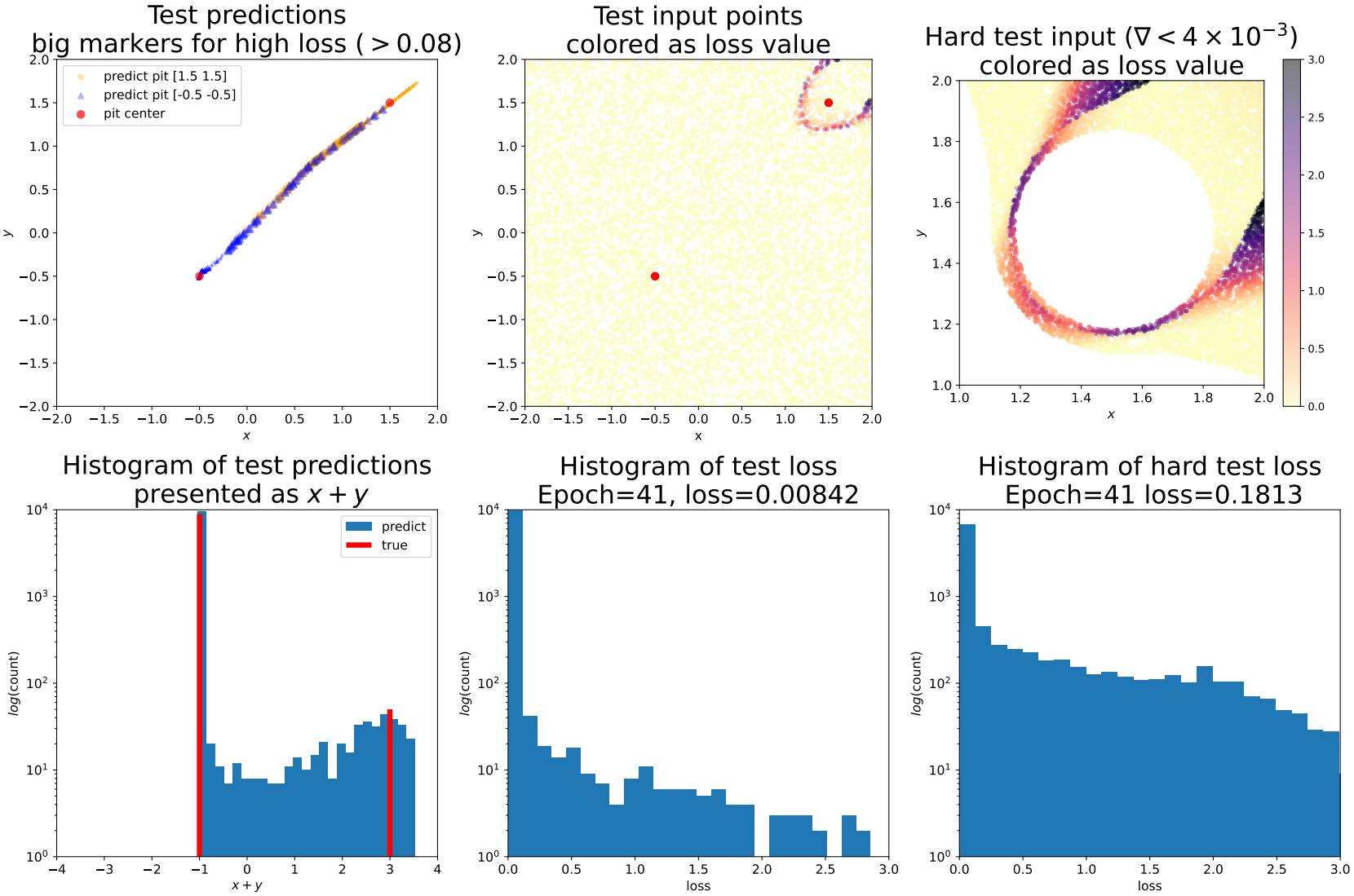


Figure 5.8: IPGD benchmark. Loss distribution over the input domain after 41 training iterations using **uniform resampling method** (left to right, top to bottom): a scatter-plot of the model's predictions over the domain (blue colour are points that correspond to the minimum of C_1, orange – for C_2, red points indicate both C_1, C_2); two scatter-plots of loss distribution over the input points for regular and hard validation set (colour indicates loss value – darker is higher); a histogram of test predictions with x-axis corresponding to $x_m + y_m$ (red lines indicate ground truth values); two histograms of test losses for regular and hard validation sets.

Chapter 6

Online Active Learning for Data-Driven Surrogates

The previous chapter established Breed as an effective AL method for offline surrogate training. Translating this approach to the online setting, however, is not straightforward: the continuous production-consumption loop that defines online training imposes constraints that are fundamentally different from those of epoch-based workflows. An AL method that is efficient and principled in the offline setting may be disruptive, too slow, or simply inapplicable when data generation and model training proceed concurrently and resources are shared.

This chapter develops the online variant of the Breed method and its integration into the Melissa framework, progressing from design principles to a proof-of-concept validation and then to large-scale experiments on challenging PDEs. Section 6.1 motivates and formalises design considerations for AL methods for online surrogate training, proposing the *pillars of online AL* and requirements for an *active sampling mechanism*. Section 6.2 describes the resulting method: the adaptation of Breed to the online setting and the active sampling mechanism implemented within Melissa. Section 6.3 validates the approach on 2D heat transfer, a tractable problem that permits systematic study of the method's behaviour and hyperparameters. Section 6.4 scales the experiments to nonlinear, chaotic PDEs and autoregressive surrogates, introduces an improved implementation of the active sampling mechanism addressing limitations identified in the proof-of-concept stage, and provides a broader empirical analysis of the method's effectiveness and limitations.

6.1 Online Training with AL Design

The online approach to training data-driven surrogates brings several challenges that have to be addressed when designing an AL method and its implementation. The main goal of online training is to maintain an optimised production-consumption process with minimal idle resources. Therefore, the AL method must be designed to operate efficiently within these constraints and not disrupt the workflow, as well as to work with the specificities of online training.

In this section, first, we attempt to formulate the requirements for an AL method in the context of online data-driven surrogate training, which we refer to as the *pillars of online AL method*. These pillars serve as guidelines for developing AL methods that are compatible with online training and can effectively enhance the training process without introducing significant overhead or delays. Then, we discuss requirements to a mechanism

that provides the necessary infrastructure for the AL method to operate within the online training framework, which we call the *active sampling mechanism*.

6.1.1 Pillars of Online AL Method

I. *The AL method must not require pauses in the surrogate training process to compute acquisition metrics.*

Traditional AL methods often require additional passes through the NN to compute acquisition metrics over a candidate pool or a training dataset. While this is feasible in offline training, within online training these additional passes would stop consumption of generated data, leading to idle solver resources. One possible solution is to maintain a separate copy of the NN for the AL purposes, however, this doubles the GPU resources required.

II. *The AL method must work with limited availability of generated data.*

Within online training, only a subset of generated data is available in the buffer for training, and therefore, for the AL method. However, solver input parameters are stored for all generated simulations, and thus, the AL method can use them to select new samples for generation.

III. *The AL method must be light-weight in compute and memory requirements, i.e., inferior to surrogate training and data generation.*

Similarly to the previous two pillars, this constraint stems from the requirement of maintaining uninterrupted production-consumption process under limited resources. The AL method should not compete with the surrogate training or data generation for resources, as it would lead to suboptimal utilisation of resources and increased idle times. Therefore, the AL method should be designed to be light-weight in terms of both compute and memory requirements. Again, AL method can be executed on separate resources to alleviate this constraint, however, this increases required resources and can introduce communication overheads.

Moreover, the AL method has to be *reactive*, i.e., sampling of input parameters for next simulations to produce has to be fast enough as to still be relevant for the NN state when these newly generated samples arrive for training. The time it takes for new samples to be generated depends on the solver complexity, and it is increased by the complexity of the AL method to sample new input parameters.

IV. *The AL method should work with solver input parameters and simulation trajectories, where NN training samples are substeps of the trajectories.*

The typical AL setup is based on selecting points in the input space of the NN model and obtaining corresponding output labels, usually low-dimensional, from an oracle. In our context, the input space corresponds to the solver input parameters that, typically, parametrise the initial trajectory state and its further progression, which is obtained by running the solver. The AL method has to select input parameters[1] to query the solver to generate a full trajectory. Next, the NN model is trained on subsets of the generated

[1]It is also possible to generate an initial trajectory state directly, but generated states have to be consistent and physically valid for the solver to evolve the trajectories.

trajectories, e.g., pairs of consecutive time-steps. Thus, an acquisition metric has to be defined with respect to the solver input parameters based on the NN's performance on the corresponding trajectory states, possibly at different NN training iterations due to the online training setup.

V. *The AL method should be agnostic to the NN model and training objective, and should not require its modification.*

This constraint is rather desirable than mandatory and stems from the intention of having a general-purpose AL method that can be used for any data-driven surrogate training within online training. Therefore, the AL method should not require specific model architectures or training objectives, as it would limit its applicability. For example, such AL methods that require uncertainty estimates of the NN predictions as acquisition metrics would not be applicable to models that do not provide such estimates.

6.1.2 Requirements to Active Sampling Mechanism

An online training framework has to provide a mechanism that enables an AL method to interact with both the surrogate training and the data generation processes. This mechanism, which we refer to as the *active sampling mechanism*, has the following requirements

- Minimal to none delays induced to the production-consumption process to ensure no idle resources, i.e.:
  - Pausing or blocking of the training process when communicating training information to the AL method;
  - Pausing of running simulations or blocking of submission of new simulations when communicating new input parameters sampled by the AL method.
- Reactive communication between the AL method, surrogate training, and data generation processes, i.e.:
  - New training information has to be communicated to the AL method as soon as available, to ensure that the AL method works with up-to-date information.
  - New input parameters sampled by the AL method have to be communicated to the earliest possible data generation processes, to minimise the delay between the AL sampling and the arrival of newly generated samples for training.

These pillars and requirements serve as a tentative foundation for designing AL methods and implementations compatible with online training of data-driven surrogates. We propose them as initial guidelines for future development, regardless of the specific online training framework or AL method being used.

Next, we present our AL method and its implementation for online training: specifically, we develop the active sampling mechanism within the Melissa framework and adapt the Breed AL method to this setting.

6.2 Melissa Active Sampling with Breed

This section is partially based on the following publication:

[47] Sofya Dymchenko, Abhishek Purandare, and Bruno Raffin. MelissaDL x Breed: Towards Data-Efficient On-line Supervised Training of Multi-parametric Surrogates with Active Learning, October 2024. URL http://arxiv.org/abs/2410.05860.

We presented Breed [47] in Section 5.2.1 as an AL method for offline training of data-driven surrogates and Melissa [187] in Section 3.3 as a framework for online training of data-driven surrogates. This section first outlines the required modifications and our proposed adaptations to Breed, then describes the active sampling mechanism within Melissa, and concludes with the full details of the online Breed method.

6.2.1 Necessary Modifications to Breed

The Breed method for offline training relies on loss values computed on the training dataset with the latest NN state to calculate acquisition scores and build a sampling distribution accordingly. To conform to the pillars of online AL design presented in the previous section, several modifications are required to adapt the method to the online training context, which are summarised as follows:

- It is no longer possible to compute loss values for all training samples with the latest NN state, as it would require pausing the training process (pillar I) and working with the full dataset (pillar II). We propose to use loss values computed during the previous training iterations.
- The method already works with the solver input parameters but requires modification to aggregate loss values over trajectory time steps (pillar IV). We propose to average the loss values over time steps for each trajectory.
- The complexity bottleneck of the original method is the computation of loss values for all training samples with the latest NN state. As we propose to use previously computed loss values, this bottleneck is alleviated. Other steps of the algorithm have linear complexity with respect to (1) the number of parents and children samples and to (2) the dimensionality of the solver input parameter space. The former is controllable via hyperparameters and can affect sampling quality, while the latter is fixed and can be high for complex solvers and surrogate behaviour designs. However, linear complexity is acceptable within the online training context (pillar III).
- The total training data size is continuously growing and epochs are no longer defined, which requires modifications to the resampling strategy, i.e., when to resample, how many samples to consider for building the sampling distribution, and how many new samples to generate at each resampling (pillar II and III). We propose to resample periodically every p training iterations, considering the most recent N_p samples for building the sampling distribution, and generating as many new samples as needed for the next p training iterations.

- The original OOB handling strategy may repeatedly fail in higher-dimensional spaces, since the acceptance rate of rejection sampling tends to decrease with dimensionality, leading to significant delays (pillar III). Moreover, falling back to uniform sampling discards the locality information provided by the parent sample, potentially hindering the AL method's ability to focus on informative regions of the parameter space. We propose adaptive width reduction and soft-boundary repair to improve acceptance rates and preserve locality.

6.2.2 Melissa with Active Sampling Mechanism

Here, we describe the first implementation of the active sampling mechanism within Melissa to enable AL methods for online training. We provide a schematic overview of the mechanism in Figure 6.1, full algorithm details in Algorithm 2, and describe the main components next.

At the start of the framework execution, the Server (S) initialises input parameters $\{\boldsymbol{\lambda}_j\}_{j=1}^{N}$ for N simulations to be executed and creates the corresponding simulation scripts $\{S_j\}_{j=1}^{N}$ (Alg. 2, lines 1-2), which are sent to the Launcher process (L). The Launcher launches these scripts N_S at a time to the batch scheduler (lines 3-4), which executes them as N_S Clients ($\mathrm{C_j}$) running concurrently (line 5, lines 7-10).

Within the Server, there are two threads: the data Receiver ($\mathrm{S_D}$), which receives generated data into the Buffer $\mathcal{B}_{\text{buf}}$, and the Trainer ($\mathrm{S_T}$), which consumes it for training the surrogate $f_{\boldsymbol{\theta}}$. We implement the following modifications to the Server to enable the active sampling mechanism.

To the Trainer, we add a Resampler functionality. At every training iteration i, the *Trainer-Resampler* calculates required training information array $\boldsymbol{a}_i$, defined according to the AL method, for each training sample $\boldsymbol{X}_j^{[t]}$ in the batch $\mathcal{B}_i$ (line 20). It then updates the resampling internal state, as defined by the AL method (line 21), and triggers resampling based on the AL method condition (line 22).

When resampling is triggered (lines 23-29), the Receiver is blocked to prevent submission of new simulation jobs by the Launcher. The Trainer-Resampler queries the Launcher to get a consistent view of the job queue and identifies the maximum index k of currently running clients given this view. Then, the Trainer-Resampler determines the number of parameters required to be resampled, which is equal to the average number M of simulations generated between two consecutive resampling events, and samples M new input parameters for simulations $\{S_{k+N_S}, \ldots, S_{k+N_S+M}\}$. When new samples calculated according to the AL method, the Trainer-Resampler overwrites the corresponding simulation scripts and unblocks the Receiver.

Subsequent Clients ($\mathrm{C_j}$) use the updated simulation scripts when they start their corresponding job and generate new trajectories with the newly sampled input parameters. The generated data will be sent to the Server for training the surrogate, thus closing the active sampling loop.

The offset $k + N_S$ helps to avoid inconsistencies with simulations that may have been already submitted but not running yet: from the moment of determining the maximum index k of currently running simulations and to the update of the simulation scripts, there is a possibility that some simulations (with index lower than k) will finish and new jobs (with indices higher than k) will be submitted to the batch scheduler. By applying new parameters starting from index $k + N_S$, we ensure that the new parameters are used

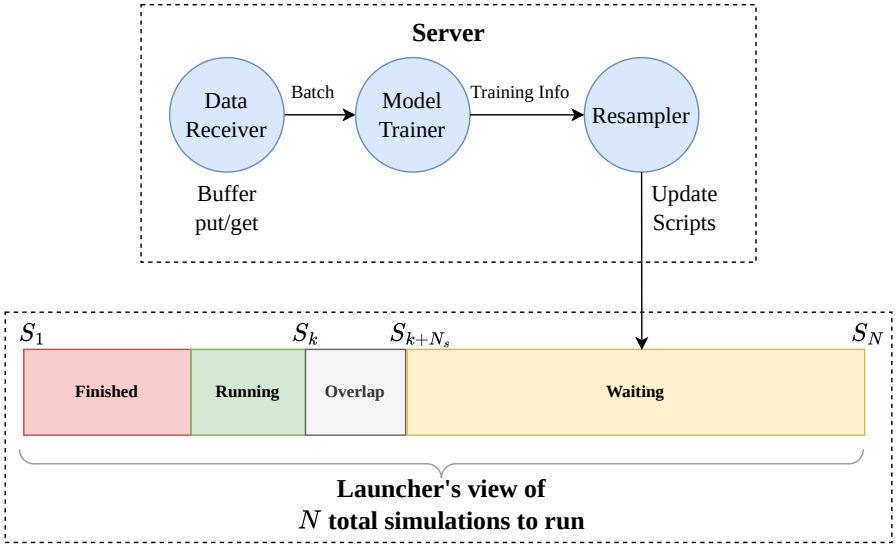


Figure 6.1: Illustration of the active sampling mechanism within Melissa: the Trainer-Resampler process calculates training information from batches, updates internal states, and triggers resampling; when trigger condition is met, it blocks the Receiver, obtains the maximum index of currently running simulations from the Launcher, samples new input parameters for the next simulations to be submitted, and updates simulation scripts. Thanks to Abhishek Purandare for the original figure.

to update the scripts of simulations that are not running yet, avoiding inconsistencies between the parameters used for data generation and those expected by the AL method.

The creation of M new parameters instead of all yet-to-be-submitted simulations, i.e., $N - k - N_S$, allows to reduce the resampling time, maintaining the reactivity of the mechanism.

Limitations. The presented implementation of the active sampling mechanism within Melissa, while simple and functional, does not entirely conform to the requirements outlined earlier and has two main limitations.

First, it potentially introduces delays when an AL method is queried to provide new solver input parameters: the Receiver is blocked until new parameters are received, which stops the Launcher from submitting new Client jobs during this waiting period. This approach ensures that new parameters are used as soon as possible without discontinuing currently running simulations. If the AL resampling algorithm is relatively fast compared to simulation time, the delay is negligible since new parameters are ready when the next batch of simulations would be submitted. However, for higher-complexity AL methods with longer resampling times, the delay can become significant, leading to underutilisation of computational resources and increased overall training time.

Second, it potentially hinders the reactivity of the AL method: the starting index from which solver parameters are updated is calculated with an offset based on the number of concurrent solvers N_S, ensuring consistency of the parameters used for data generation with those expected by the AL method. However, this means that newly sampled parameters – based on the most recent surrogate state – are not used for the immediately following jobs, but rather for the N_S next ones. It creates a temporal mismatch between when parameters are sampled and when they are actually used, potentially reducing the

Algorithm 2 The Melissa framework with active sampling mechanism.

Processes: Launcher (L); Server threads: data Receiver (S_D), Trainer–Resampler (S_T); N_S Clients (C_j).

Require: Total number of simulations to create N, number of clients N_S, buffer $\mathcal{B}_{\text{buf}}$ with strategy BM; solver $S(\cdot, \boldsymbol{\lambda})$, total number of time steps N_t, solver input parameter space Λ;

1: (S_D) Initialise solver input parameters $\{\boldsymbol{\lambda}_j \sim \text{Unif}(\Lambda)\}_{j=1}^{N}$
2: (S_D) Create N client scripts $\{S_j\}_{j=1}^{N}$ with these parameters
3: (L) `wait` until scripts are created by S_D
4: (L) Create a job queue $\{C_j\}_{j=1}^{N}$ with these scripts distributed over N_S clients
5: (C_j) `run_solver`(S_j)
6: (S_D, S_T) `train_model`()

`run_solver`(S_j)

7: Initialise $\boldsymbol{X}_j^0$ with IC parameters from $\boldsymbol{\lambda}_j$
8: **for** $t = 0, \ldots, N_t - 1$ **do**
9: $\quad \boldsymbol{X}_j^{t+1} \leftarrow S(\boldsymbol{X}_j^t, \boldsymbol{\lambda}_j)$
10: $\quad$ `send` training data $\boldsymbol{X}_j^{[t]} \rightarrow S_D$

`train_model`()

11: Initialise model parameters $\boldsymbol{\theta}_0$, training iteration counter $i = 0$
12: **while** $\mathcal{B}_{\text{buf}} \neq \emptyset$ **do**
13: $\quad$ (S_D) `receive` $\boldsymbol{X}_j^{[t]} \leftarrow C_j$
14: $\quad$ (S_D) $\mathcal{B}_{\text{buf}} \overset{\text{BM}}{\leftarrow} \boldsymbol{X}_j^{[t]}$
15: $\quad$ (S_D) Create training batch $\mathcal{B}_i \overset{\text{BM}}{\sim} \mathcal{B}_{\text{buf}}$
16: $\quad$ (S_D) `send` $\mathcal{B}_i \rightarrow S_T$
17: $\quad$ (S_T) `receive` $\mathcal{B}_i \leftarrow S_D$
18: $\quad$ (S_T) $\boldsymbol{\theta}_i \leftarrow \text{update}(\boldsymbol{\theta}_{i-1})$ with $\mathcal{B}_i$
19: $\quad$ (S_T) $i \leftarrow i + 1$
20: $\quad$ (S_T) $\boldsymbol{a}_i \leftarrow \left\{ a_{jti} = a(\boldsymbol{X}_j^{[t]}, \boldsymbol{\theta}_i) \,\middle|\, \forall \boldsymbol{X}_j^{[t]} \in \mathcal{B}_i \right\}$ ▷ *Eq.* (6.3)
21: $\quad$ (S_T) `update_acquisition_scores`$(\boldsymbol{a}_i)$ ▷ *Alg. 3*
22: $\quad$ **if** resampling is triggered by S_T **then**
23: $\quad\quad$ (S_D) `wait` for unblock signal $\leftarrow S_T$
24: $\quad\quad$ (S_T) `receive` job queue view $\leftarrow$ L
25: $\quad\quad$ (S_T) $k \leftarrow$ maximum index of currently running jobs in job queue view
26: $\quad\quad$ (S_T) Starting update index $j' \leftarrow k + N_S$
27: $\quad\quad$ (S_T) $M \leftarrow$ number of simulations to resample
28: $\quad\quad$ (S_T) $\{S_{j'}, \ldots, S_{j'+M}\} \leftarrow$ `produce_new_parameters`(M) ▷ *Alg. 3*
29: $\quad\quad$ (S_T) `send` unblock signal $\rightarrow S_D$

effectiveness of the AL method.

Within this section, we accept these limitations as trade-offs for implementation simplicity, and we address them in Section 6.4.1 with a more advanced algorithm.

6.2.3 Online Breed Method within Melissa

We implement the online version of Breed with three key features: *(1)* it computes acquisition scores based on aggregated batch-wise normalised loss values obtained during previous training iterations, *(2)* it triggers resampling periodically and samples new solver input parameters based on a subset of the most recently processed simulations, *(3)* it optimises OOB samples handling to reduce delays and improve local sampling quality. The method is implemented within the active sampling mechanism of Melissa, as described in the previous subsection. We provide the full algorithm details in Algorithm 3 and describe these components next.

6.2.3.1 Delta-loss statistic for acquisition scores

Batch-wise normalised loss values. To compare per-sample loss values computed at different training iterations, we define *a delta-loss statistic* that normalises the loss values within each training batch.

We train the NN surrogate $f_{\boldsymbol{\theta}}$ to minimise a loss function $\ell(\boldsymbol{X}_j^{[t]}; \boldsymbol{\theta})$ between the NN prediction and the target output, according to the given training scheme and behaviour design for the training sample $\boldsymbol{X}_j^{[t]}$ associated with the trajectory state at time step t and solver input parameters $\boldsymbol{\lambda}_j$ of the j-th simulation.

At each training iteration i, a batch $\mathcal{B}_i$ of training samples of size B is created from the Buffer and used to update the NN parameters $\boldsymbol{\theta}_i$. For each sample $\boldsymbol{X}_j^{[t]}$ in the batch $\mathcal{B}_i$, we compute its loss value with the current NN parameters $\boldsymbol{\theta}_i$:

$$l_{jti} = \ell(\boldsymbol{X}_j^{[t]}; \boldsymbol{\theta}_i), \quad \boldsymbol{l}_i = \{l_{jti} \,|\, \forall \boldsymbol{X}_j^{[t]} \in \mathcal{B}_i\}. \tag{6.1}$$

We compute mean m_i and standard deviation s_i of the batch loss values $\boldsymbol{l}_i$:

$$m_i = \text{mean}(\boldsymbol{l}_i), \quad s_i = \text{std}(\boldsymbol{l}_i). \tag{6.2}$$

Then the delta-loss δl_{jti} for the sample $\boldsymbol{X}_j^{[t]}$ at iteration i is defined as:

$$\delta l_{jti} = \max\left\{\frac{l_{jti} - m_i}{s_i}, 0\right\}. \tag{6.3}$$

This statistic represents how much the loss value of the sample deviates from the mean loss value of the batch, normalised by the standard deviation of the batch loss values. The use of max with zero ensures that only samples with above-average loss values contribute to the delta-loss, which aligns with the goal of focusing on samples where the NN performs poorly.

Aggregation over NN iterations. Training samples can be seen multiple times during training. For a given sample $\boldsymbol{X}_j^{[t]}$, we can have multiple delta-loss values δl_{jti} computed at different training iterations i when the sample was included in the batch $\mathcal{B}_i$. Therefore, we

Algorithm 3 Online Breed AL method: Solver input parameters sampling.

Require: Resampling period p, schedule $(\varepsilon^{(0)}, \varepsilon^{(c)}, c)$, sliding window size N_p, initial $\boldsymbol{\sigma}$ and minimum $\boldsymbol{\sigma}_{\min}$ widths, width-reduction factor β, OOB attempts limit $\text{oob}_{\max}$, solver input parameter space Λ. Optional strict condition $T_{\min}$.

Set queue $\mathcal{Q}$ of size N_p is initially empty
Iterative recording dictionary $R[N, T]$ is initially empty
Widths array $\{\boldsymbol{\sigma}_j\}_{j=1}^{N}$ is initialised with $\boldsymbol{\sigma}$
Resampling iteration counter $r \leftarrow 0$

`update_acquisition_scores`$(\boldsymbol{\delta l}_i)$

for each $\delta l_{jti} \in \boldsymbol{\delta l}_i$ **do**
    $R[j, t] \leftarrow \delta l_{jti}$ ▷ *Update as in Eq.* (6.8)
    `set_queue_insert`$(j, T_{\min})$
if $i \mod p = 0$ **then**
    Trigger resampling

`produce_new_parameters`(M)

Compute δl_j from $R[j, :]$ for all j in $\mathcal{Q}$ ▷ *Eq.* (6.7)
Obtain probability vector $(l_1, \ldots, l_{N_p})$ ▷ *Eq.* (6.9)
$M_c \leftarrow \lfloor M \cdot \varepsilon^{(r)} \rfloor$ with updated $\varepsilon^{(r)}$ ▷ *Eq.* (5.7)
Select M_c parent points from $\mathcal{Q}$ with replacements with $(l_1, \ldots, l_{N_p})$ and
    obtain children points count m_j for each j in $\mathcal{Q}$
$\mathcal{S}_c \leftarrow \emptyset$
for each $j \in \mathcal{Q}$ **do**
    for $k = 1, \ldots, m_j$ **do**
        $\boldsymbol{\lambda}' \sim \mathcal{N}(\boldsymbol{\lambda}_j, \boldsymbol{\sigma}_j^2 I)$
        Counter of attempts oob $\leftarrow 0$
        while $\boldsymbol{\lambda}' \notin \Lambda$ and oob $\leq \text{oob}_{\max}$ **do** ▷ *OOB procedure*
            oob $\leftarrow$ oob $+ 1$
            $\boldsymbol{\sigma}_j \leftarrow$ reduce with β for OOB dimensions ▷ *Eq.* (6.11)
            Redraw $\boldsymbol{\lambda}' \sim \mathcal{N}(\boldsymbol{\lambda}_j, \boldsymbol{\sigma}_j^2 I)$
        Inherit reduced width $\boldsymbol{\sigma}' \leftarrow \boldsymbol{\sigma}_j$
        if $\boldsymbol{\lambda}' \notin \Lambda$ **then** ▷ *Fallback to soft-boundary*
            for each OOB dimension d **do**
                $\boldsymbol{\lambda}'[d] \leftarrow$ soft-boundary adjustment ▷ *Eq.* (6.13)
                Reset width $\boldsymbol{\sigma}'[d] \leftarrow \boldsymbol{\sigma}[d]/3$
        $\mathcal{S}_c \leftarrow \mathcal{S}_c \cup \{\boldsymbol{\lambda}'\}$
        Update associated $\boldsymbol{\sigma}'$ width in widths array
$M_u \leftarrow M - M_c$
$\mathcal{S}_u \leftarrow \{M_u$ points sampled from Unif$(\Lambda)\}$
$r \leftarrow r + 1$
return Shuffled $\mathcal{S}_c \cup \mathcal{S}_u$

`set_queue.insert`$(j, T_{\min} = 1)$

if number of elements $R[j, :] < T_{\min}$ **then** do not insert j into $\mathcal{Q}$, **return**
if j is in $\mathcal{Q}$ **then**
    Remove j from $\mathcal{Q}$ ▷ *Its position will be updated*
else if $\mathcal{Q}$ is full **then**
    Pop the oldest element from $\mathcal{Q}$
Insert j into $\mathcal{Q}$

need to aggregate these delta-loss values over the iterations to obtain a single delta-loss value per simulation j per time step t:

$$\delta l_{jt} = \mathrm{Agg_i}\left(\{\delta l_{jti} \,|\, \forall i \in \mathcal{I}_{jt}\}\right), \tag{6.4}$$

where $\mathcal{I}_{jt}$ is the set of training iterations i when the sample $\boldsymbol{X}_j^{[t]}$ was included in the batch $\mathcal{B}_i$, and $\mathrm{Agg_i}$ is an aggregation function over the iterations, e.g., mean, max, weighted exponential moving average, etc. In our implementation, we use simple averaging over the iterations:

$$\delta l_{jt} = \frac{1}{|\mathcal{I}_{jt}|} \sum_{i \in \mathcal{I}_{jt}} \delta l_{jti}. \tag{6.5}$$

Due to the online training setup, some samples may not have been seen yet by the NN, and thus, $\mathcal{I}_{jt}$ is empty. In this case, the delta-loss statistic is undefined for these samples.

Aggregation over trajectory timesteps. Similarly, to compute a single delta-loss value per simulation parameter $\boldsymbol{\lambda}_j$, we need to aggregate these delta-loss values over the trajectory timesteps t:

$$\delta l_j = \mathrm{Agg_t}\left(\{\delta l_{jt} \,|\, \forall t \in \mathcal{T}_j\}\right), \tag{6.6}$$

where $\mathcal{T}_j$ is the set of timesteps t which were processed by the NN so far, i.e., $\mathcal{I}_{jt} \neq \emptyset$, and $\mathrm{Agg_t}$ is an aggregation function over the timesteps. We aggregate using average over the timesteps:

$$\delta l_j = \frac{1}{|\mathcal{T}_j|} \sum_{t \in \mathcal{T}_j} \delta l_{jt}. \tag{6.7}$$

Similarly, it is possible that for some simulations, the NN has not seen any of the trajectory steps yet, and thus, $\mathcal{T}_j$ is empty, making the delta-loss statistic undefined for these simulations. The statistic is defined as soon as $|\mathcal{T}_j| \geq 1$, i.e., at least one trajectory step has been seen by the NN.

Optionally, we can apply stricter conditions on the statistic definition, requiring $|\mathcal{T}_j| \geq T_{\min} \in (1, N_t]$, where N_t is the total number of timesteps in the trajectory. This ensures that the statistic is computed only for simulations which trajectory states have been sufficiently seen by the NN.

Delta-loss value update. To store delta-loss values δl_{jti} and use them during resampling iterations, we maintain an iterative recording dictionary $R[N, N_t]$. It stores pairs (j, t) as keys and corresponding delta-loss values δl_{jt} and set size $|\mathcal{I}_{jt}|$ as values. When a delta-loss value δl_{jti} is received, the corresponding entry in the dictionary is updated as follows:

$$R[j,t] \leftarrow \begin{cases} \left(\frac{\delta l_{jt} \cdot |\mathcal{I}_{jt}| + \delta l_{jti}}{|\mathcal{I}_{jt}|+1},\ |\mathcal{I}_{jt}| + 1\right), & \text{if } (j,t) \in R[N, N_t] \\ (\delta l_{jti}, 1), & \text{otherwise} \end{cases} \tag{6.8}$$

which corresponds to iterative average computation of equation (6.5). The memory complexity of the recording dictionary is $O(N \times N_t)$, as we store only iterations-aggregated values instead of all received delta-loss values.

6.2.3.2 Sampling new solver input parameters

Unique queue with recency priority. To consider the most recent training samples for building the sampling distribution and to limit the computational complexity of the resampling step, we utilise a queue $\mathcal{Q}$ of *window size* N_p, which stores unique simulation indices j in an order of recency. When a delta-loss value δl_{jti} is received, the corresponding simulation index j is inserted into the queue $\mathcal{Q}$. If j is already in the queue, it is removed and re-inserted to update its position. If the queue is full, the oldest element is removed to make space for the new element. This way, the queue always contains the indices of the most recent N_p training simulations.

When a strict condition on the delta-loss statistic definition is applied, we only insert j into the queue when the condition is satisfied, which is checked using the recording dictionary $R[N, N_t]$.

Resampling algorithm. When the Resampler process receives the array of acquisition scores (losses) for the current batch $\mathcal{B}_i$, the AL method can decide whether to trigger resampling or not. We propose the simplest triggering criteria, i.e., resampling periodically every p training iterations, i.e., when $i \mod p = 0$.

When resampling is triggered, we obtain delta-loss values $\{\delta l_j\}_{j\in\mathcal{Q}}$ for the simulations in the recency queue $\mathcal{Q}$ by aggregating values δl_{jt} from the recording dictionary $R[N, N_t]$ using equation (6.7).

To build the sampling distribution and sample new input parameters, we follow the same steps as in the offline Breed method. First, we normalise the delta-loss values as:

$$\left(\frac{\delta l_j}{\sum_{j\in\mathcal{Q}} \delta l_j}\right)_{j\in\mathcal{Q}}. \tag{6.9}$$

Then, we get the number M of simulations to update and perform the resampling procedure as described in subsection 5.2.1, using these normalised delta-loss values as weights $(l_1, \ldots, l_{N_p})$, corresponding solver input parameters $\{\boldsymbol{\lambda}_j\}_{j\in\mathcal{Q}}$ as parents, and sampling M new points. The created set of M points is then used to update the corresponding simulation scripts $\{S_{k+N_s}, \ldots, S_{k+N_S+M}\}$, as described in subsection 6.2.2.

Neighbourhood width. The neighbourhood width $\boldsymbol{\sigma}$ is a vector of size n_p, where n_p is the dimensionality of the solver input parameter space Λ. Unlike the original scalar formulation, this allows dimension-specific widths when the parameter ranges vary across dimensions. Moreover, we can set the widths as a fraction of each dimension's domain length, i.e., for $d = 1, \ldots, n_p$:

$$\boldsymbol{\sigma}[d] = \sigma_{\text{frac}} \cdot |\Lambda_d|, \quad |\Lambda_d| = \Lambda_d^{\max} - \Lambda_d^{\min} \tag{6.10}$$

where $\cdot[d]$ denotes the d-th vector coordinate, $\sigma_{\text{frac}} \in (0, 1)$ is a hyperparameter and $\Lambda_d = [\Lambda_d^{\min}, \Lambda_d^{\max}]$ denotes the valid range of the d-th input parameter.

6.2.3.3 Out-of-bounds handling

We propose two modifications to the OOB handling strategy: dimension-wise adaptive width reduction with inheritance and soft-boundary sampling. To support these, we maintain and adaptively update a neighbourhood width vector $\boldsymbol{\sigma}_j$ for all parameters $\{\boldsymbol{\lambda}_j\}_{j=1}^N$, requiring $\mathcal{O}(N \times n_p)$ memory. The widths array $\{\boldsymbol{\sigma}_j\}_{j=1}^N$ is initialised to the user-prescribed global width: $\boldsymbol{\sigma}_j = \boldsymbol{\sigma}$ for all $j \in \{1, \ldots, N\}$.

Adaptive width reduction with inheritance. When a child sample $\boldsymbol{\lambda}'$ drawn from the parent $\boldsymbol{\lambda}_j$ is OOB in at least one dimension, we reject it and attempt to redraw a new child from the same parent, up to $\text{oob}_{\max}$ attempts, similarly to the original method.

To increase the acceptance rate in subsequent redraw attempts for the parent, we reduce the corresponding width $\boldsymbol{\sigma}_j$ in-place in dimensions where a rejected child was OOB, i.e., for each $d \in \{1, \ldots, n_p\}$ such that $\boldsymbol{\lambda}'[d] \notin \Lambda_d$:

$$\boldsymbol{\sigma}_j[d] \leftarrow \max(\boldsymbol{\sigma}_j[d] \cdot \beta, \boldsymbol{\sigma}_{\min}[d]), \tag{6.11}$$

where $\beta \in (0, 1)$ is a width-reduction hyperparameter, and $\boldsymbol{\sigma}_{\min}$ is a minimum-width hyperparameter vector that prevents near-zero values. In practice, we set $\boldsymbol{\sigma}_{\min}[d] = 5 \cdot 10^{-7} \cdot |\Lambda_d|$. As the reduction is performed in-place, subsequent redraw attempts and child draws from the same parent use the updated width $\boldsymbol{\sigma}_j$.

When a valid child sample $\boldsymbol{\lambda}'$ is finally obtained from the parent $\boldsymbol{\lambda}_j$ before reaching the $\text{oob}_{\max}$ limit, it inherits the parent's adapted width:

$$\boldsymbol{\sigma}' \leftarrow \boldsymbol{\sigma}_j. \tag{6.12}$$

When the framework updates the simulation parameters with newly generated children, the width array is updated accordingly, maintaining the correspondence $\boldsymbol{\lambda}_{j'} \leftrightarrow \boldsymbol{\sigma}_{j'}$. This inheritance ensures that if $\boldsymbol{\lambda}_{j'}$ is later selected as a parent, it uses the adapted width, further improving acceptance rates and preserving locality in subsequent iterations.

Soft-boundary sampling. If the $\text{oob}_{\max}$ limit is reached and the child $\boldsymbol{\lambda}'$ still violates bounds in some dimensions, we repair the violated coordinates by adjusting them to lie within Λ near the violated bound. For each dimension d where $\boldsymbol{\lambda}'[d] \notin \Lambda_d$, we set

$$\boldsymbol{\lambda}'[d] \leftarrow \begin{cases} \Lambda_d^{\min} + \epsilon, & \text{if } \boldsymbol{\lambda}'[d] < \Lambda_d^{\min}, \\ \Lambda_d^{\max} - \epsilon, & \text{if } \boldsymbol{\lambda}'[d] > \Lambda_d^{\max}, \end{cases} \tag{6.13}$$

where $\epsilon \sim \text{Unif}([0, \boldsymbol{\sigma}[d]/3])$ is sampled independently for each repaired coordinate to place it near the bound. For dimensions that were repaired, the child's width is reset to $\boldsymbol{\sigma}[d]/3$; for dimensions that remained within bounds, the child inherits the parent's adapted width $\boldsymbol{\sigma}_j[d]$. This reset provides a balance between exploration and boundary avoidance, as $\boldsymbol{\sigma}[d]/3$ is smaller than the initial width but less conservative than the possibly very small adapted $\boldsymbol{\sigma}_j[d]$ in repeatedly violated dimensions.

6.2.4 Conclusion

We presented the online version of the Breed AL method implemented within the active sampling mechanism of the Melissa framework.

To adapt Breed to the online training context, we introduced the delta-loss statistic for acquisition scores, recency-based sampling with periodic resampling, and adaptive out-of-bounds handling. These modifications enable the method to conform to the online AL design pillars outlined earlier. However, Breed retains the original method's limitations regarding sampling complexity in high-dimensional parameter spaces and hyperparameter sensitivity.

The active sampling mechanism, while functional, has limitations regarding potential delays and reduced reactivity that do not fully meet the requirements outlined in Section 6.1.2.

In the next section, we evaluate this implementation through experimental studies.

6.3 Experiments for Direct Linear PDE Surrogate

This section is partially based on the Dymchenko et al., 2024 [47].

In this section, we present experimental results of using the online Breed method within Melissa for training a direct surrogate for 2D heat transfer simulations with varied IC and BC. To assess the performance of the method, we conduct systematic studies across different surrogate model sizes and Breed hyperparameters, comparing to a baseline of uniform parameter sampling. We analyse the results and provide insights on the method behaviour and tuning.

6.3.1 Experimental Setup

We follow the setup of a 2D heat transfer PDE solver and surrogate model as in [149]. The heat transfer PDE has relatively low computational demands and provides easy interpretability of results, making it suitable for proof-of-concept experiments.

Heat transfer PDE solver. The solver is an in-house C++/Fortran programme that implements a finite difference method with an implicit Euler scheme. It takes as input 5 temperature values defining the IC and BC:

$$\boldsymbol{\lambda} = [T_0, T_l, T_r, T_b, T_t] \in \Lambda = [100, 500]^5, \tag{6.14}$$

where T_0 is the initial temperature inside the 2D domain, and T_l, T_r, T_b, T_t are the temperatures on the left, right, bottom, and top borders of the domain, respectively.

Other input parameters are fixed within the code, including the spatio-temporal domain $\mathcal{U} = [0, X]^2 \cup [0, T]$, with $X = 1\text{m}$ and $T = 1\text{s}$, the thermal diffusivity coefficient $\alpha = 1\,\text{m}^2\text{s}^{-1}$, and discretisation configuration (N_x, N_t), with $N_x = 100$ and $N_t = 100$.

Inside the programme, the initial state $\boldsymbol{X}^0$ is created as follows:

$$\boldsymbol{X}^0[i,j] = \begin{cases} T_0, & \text{if } 1 < i, j < N_x \\ T_l, & \text{if } i = 1 \\ T_r, & \text{if } i = N_x \\ T_b, & \text{if } j = 1 \\ T_t, & \text{if } j = N_x \end{cases} \tag{6.15}$$

The solver autoregressively produces the trajectory steps $\{\boldsymbol{X}^0, \boldsymbol{X}^1, \ldots, \boldsymbol{X}^{N_t}\}$ of the temperature field evolution over time.

Direct surrogate model. The surrogate $f_{\boldsymbol{\theta}}$ is trained to directly predict the discretised temperature field at time t for the given IC/BC parameters $\boldsymbol{\lambda}$:

$$\begin{gathered} f_{\boldsymbol{\theta}} : \mathbb{R}^5 \times \mathbb{R} \to \mathbb{R}^{N_x} \times \mathbb{R}^{N_x}, \\ f_{\boldsymbol{\theta}}(\boldsymbol{\lambda}, t) \approx \boldsymbol{X}^t_{\lambda} = S^{(t)}(\boldsymbol{X}^0; \boldsymbol{\lambda}), \quad \forall \boldsymbol{\lambda} \in \Lambda \quad \forall t \in \{1, \ldots, N_t\}. \end{gathered} \tag{6.16}$$

The training sample consists of $(\boldsymbol{\lambda}, t, \boldsymbol{X}^t_{\lambda})$, where $\boldsymbol{X}^t_{\lambda}$ is the true temperature field at time t for parameters $\boldsymbol{\lambda}$ produced by the solver.

Table 6.1: Fixed hyperparameters for each experiment. Asterisks (*) indicate varied parameters: (**Exp1**) model size, (**Exp2**) sampling parameters, (**Exp3**) schedule parameters.

	P	p	σ	$\varepsilon^{(0)}$	$\varepsilon^{(c)}$	c	H	L
Exp1	200	300	10.0	0.5	0.7	3	*	*
Exp2	*/200	*/200	*/5.0	0.5	0.9	3	16	1
Exp3	200	200	5.0	*/0.1	*/1.0	*/5	16	1

We choose an MLP with L hidden layers of H neurons with ReLU activations, which is trained to minimise the MSE loss between predicted and true temperature fields using Adam optimiser with a learning rate of $1e^{-3}$ and batch size $B = 128$.

For validation, we create a set of 200 simulation trajectories with parameters sampled quasi-uniformly using Halton sequences, which ensures low-discrepancy coverage of Λ. The validation set totals 20,000 samples.

Melissa configuration. The total number of simulations is $N = 800$, number of concurrent Clients is $N_S = 20$, and we use the Reservoir Buffer of size $N_B = 3000$ training samples and watermark of 300.

Choice of experiments. We conduct three systematic studies: **Exp1**, across different model configurations, comparing to uniform resampling baseline (Random), and ablations **Exp2, Exp3** across Breed's sampling and schedule hyperparameters. We vary one set of parameters at a time, while keeping the others fixed, see Table 6.1.

Exp1. The model configuration affects its capability to capture more complex data and directly connects to overfitting and underfitting phenomena. We vary hidden size $H \in \{16, 32, 64\}$ and number of layers $L \in \{1, 2, 3\}$ of the fully connected NN.

Exp2. Sampling hyperparameters affects the proposal distribution quality and reactivity. We vary windows size $N_p \in \{50, 600, 1000\}$, period $p \in \{10, 50, 100, 300, 500\}$, and width $\sigma \in \{1.0, 5.0, 10.0, 25.0\}$, which defines the width vector as $\boldsymbol{\sigma} = \mathbf{1}\sigma \in \mathbb{R}^5$.

Exp3. The $\varepsilon^{(r)}$ schedule directly affects exploration-exploitation trade-off. We study values $\varepsilon^{(0)} \in \{0.1, 0.5, 0.8, 1.0\}$, $\varepsilon^{(c)} \in \{0.7, 0.9\}$, and $c \in \{2, 4\}$.

6.3.2 Experimental Results

Exp1: varying model size. We present the results of model size variation in Figure 6.2. Each subplot corresponds to a specific model configuration and consists of line plots of training and validation loss curves for both Breed and Random methods.

As the heat transfer problem is relatively simple, the benefits of active learning are not very pronounced in terms of final average validation loss, where both methods achieve similar results. However, looking at the training and validation curves, we see a specific pattern: Random consistently results in overfitting, in contrast to Breed, where training and validation curves most of the time stay close. This effect is more pronounced for bigger models ($H = 64$), where for Breed the curves almost overlap. We attribute this to Breed steering sampling towards harder-to-learn trajectories, which for bigger models *creates more diverse training data, preventing overfitting.*

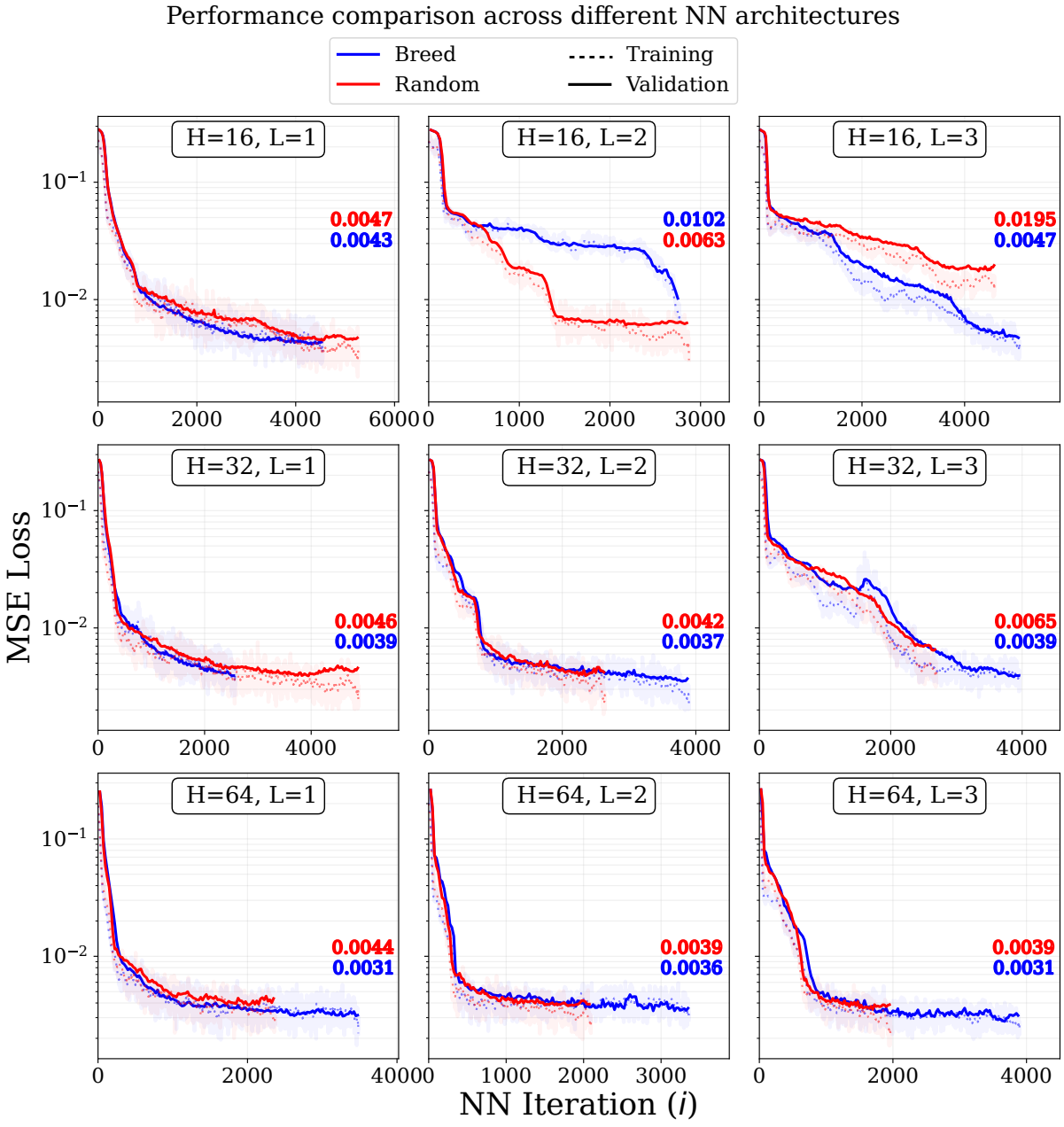


Figure 6.2: **Exp1.** Each subplot represents a run with varying MLP configurations, i.e., hidden layer size H (different rows) and number of layers L (different columns). In each plot, training (dotted line) and validation (solid line) loss curves are shown for Breed (blue colour) and Random (red colour) methods. The text annotations indicate the final validation loss values.

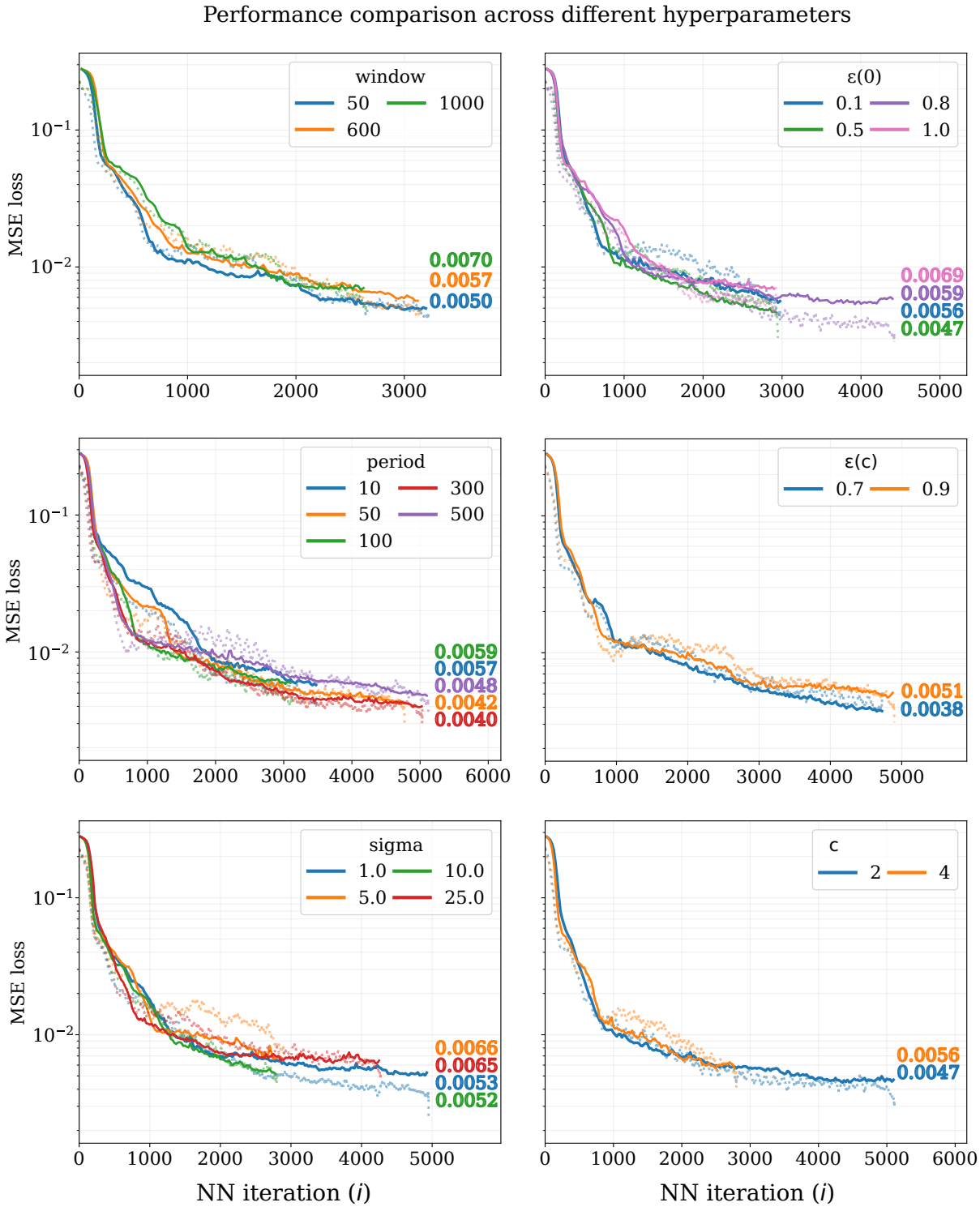


Figure 6.3: **Exp2 & Exp3.** Each subplot represents a run with varying hyperparameters, indicated in the corresponding legend title. In each plot, training (dotted line) and validation (solid line) loss curves are shown for different hyperparameter values, colours indicated in the legend. The text annotations indicate the final validation loss values.

Exp2 & Exp3: varying Breed hyperparameters. We present the results of Breed hyperparameter variation in Figure 6.3. Each subplot corresponds to a specific hyperparameter and consists of line plots of training and validation loss curves for different values of that hyperparameter.

Smaller window size N_p benefits the training process: *with smaller window size where the proposal is more focused, the training converges faster and to a better final validation loss.* When the window size N_p is large, the training diverges from optimal at early iterations, which affects further convergence. For window size 1000 (green line), validation loss is consistently the worst for the first 1000 training iterations, and resulting validation loss is worse compared to window size 50. Our hypothesis is that big window size makes the proposal distribution outdated, as it includes samples collected long ago, which do not reflect the current NN state. This stresses the importance of AL reactivity.

The resampling period p impacts the training dynamics: *frequent resampling destabilises training at early iterations, while infrequent resampling prevents adaptation to the evolving network.* When the resampling period is too small ($p = 10$, blue line), the validation loss is consistently the highest during the first 2000 iterations compared to other values, resulting in suboptimal final validation loss. On the other hand, higher resampling period ($p = 500$, violet line) leads to an early plateau around iteration 1000 and suboptimal final loss too. Values in between show elements of both behaviours, with the best final validation loss for $p = 300$ (red line), where stagnation happens later at around iteration 2000, and the convergence is fast at the beginning. This suggests that resampling should be triggered in a more adaptive manner, depending on the training dynamics, as a fixed period might not be optimal throughout the training or finding the right fixed value can be challenging.

The width σ affects the generalisation ability of the surrogate: *smaller values lead to a significant overfitting gap, while bigger values lead to stagnation of training at early iterations.* Within the σ study, we see that for the smallest value (blue line), the training loss is about 1.5 times lower than the validation loss at the end of training. The biggest value (red line) shows faster convergence than other values for the first 1000 training iterations but then stagnates, although there is no overfitting. Similarly to the previous hyperparameter, this suggests that an adaptive width schedule could be beneficial, as a fixed value might not be optimal throughout the training. Possibly, starting with a bigger value and gradually decreasing it could balance exploration and exploitation better.

The exploration-exploitation trade-off influences both convergence and generalisation: *high exploitation at earlier iterations leads to divergence and overfitting, while gradual increase from low to optimal exploitation leads to stable convergence and good generalisation.* In the $\varepsilon^{(0)}$ study, we see that high value of 0.8 (purple line) shows divergence at earlier iterations, and resulting relative to validation generalisation gap is almost 30%, while low value of 0.1 (blue line) shows fast convergence at the start which further slows down, with resulting training loss higher than validation loss. In the $\varepsilon^{(c)}$ study, lower value (blue line) shows slightly better convergence compared to higher one (orange line). As for the c study, the difference is inconclusive due to small difference of chosen varied parameters, but we include it for completeness.

6.3.3 Sampled Parameters Distribution Analysis

Beyond quantitative performance metrics, we conducted exploratory analysis of the distribution of sampled solver input parameters $[T_0, T_l, T_r, T_b, T_t]$. We observe that *Breed steers*

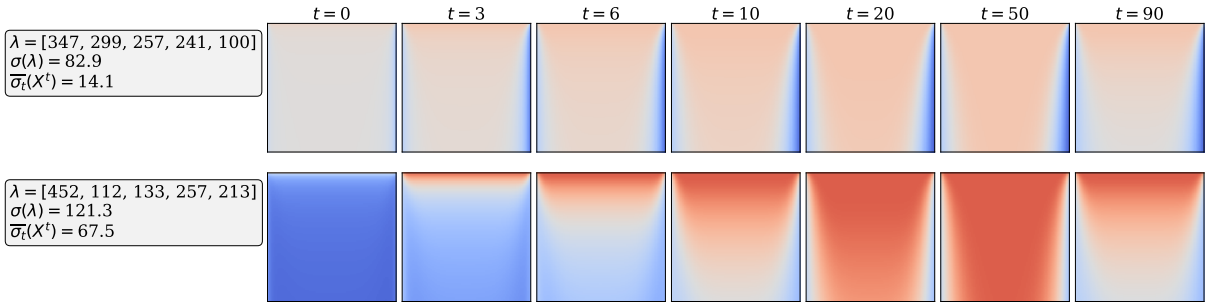


Figure 6.4: Example of trajectories with low (top) or high (bottom) difference in input temperature parameters.

sampling toward regions of the parameter space that yield harder-to-learn trajectories.

We use the standard deviation of parameter vector values to characterise the difficulty of learning a trajectory. The intuition is that trajectories with larger differences in temperatures at the initial step are more dynamic and exhibit larger differences between consecutive time steps, thus making them harder for the surrogate to learn. We illustrate this with two example trajectories in Figure 6.4.

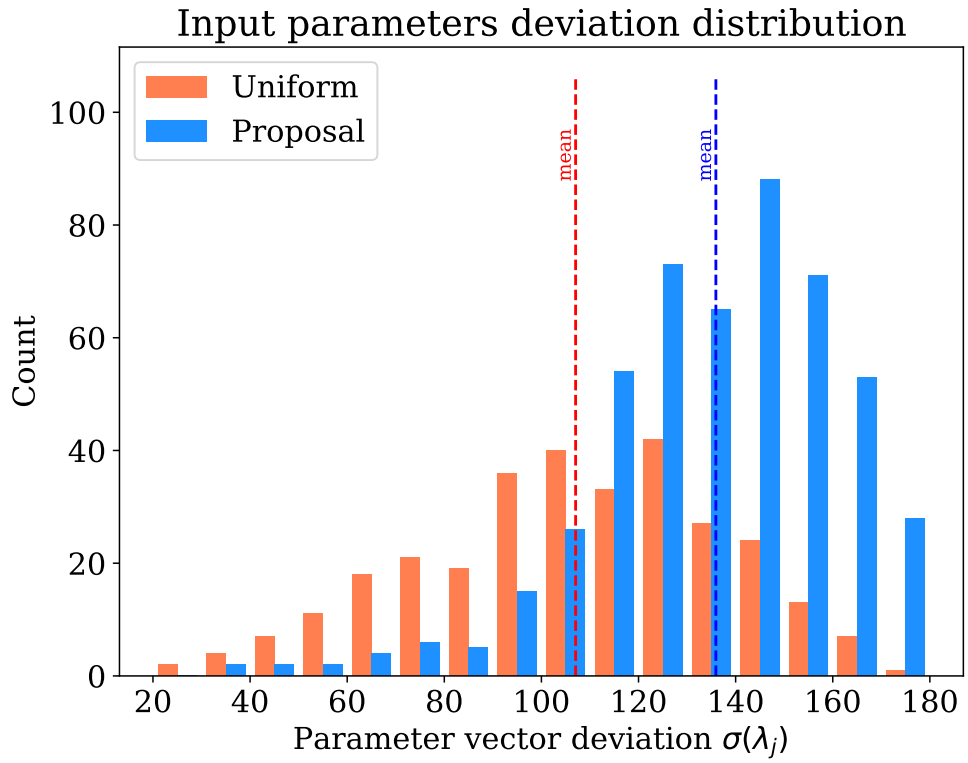
For one experiment run, we calculate the standard deviation of each input parameter vector $\boldsymbol{\lambda}_j$ and analyse the distributions across different sample categories. For Breed, sampled input parameters come from two sources: the uniform set $\mathcal{S}_u$ and the proposal concentration set $\mathcal{S}_c$. In Figure 6.5a, we compare the standard deviation distributions of these two categories collected from one Breed run. We see a clear distributional shift from uniform to proposal samples, indicated by a higher mean and longer right tail in the proposal distribution.

We compare standard deviation distributions between one Random run and one Breed run. As shown in Figure 6.5b, the Breed distribution is shifted to the right compared to Random, though the difference in means is less pronounced since Breed includes both uniform and proposal samples. However, examining the right tails shows that Breed produces about 2-3× more samples with deviation values > 150 compared to Random.

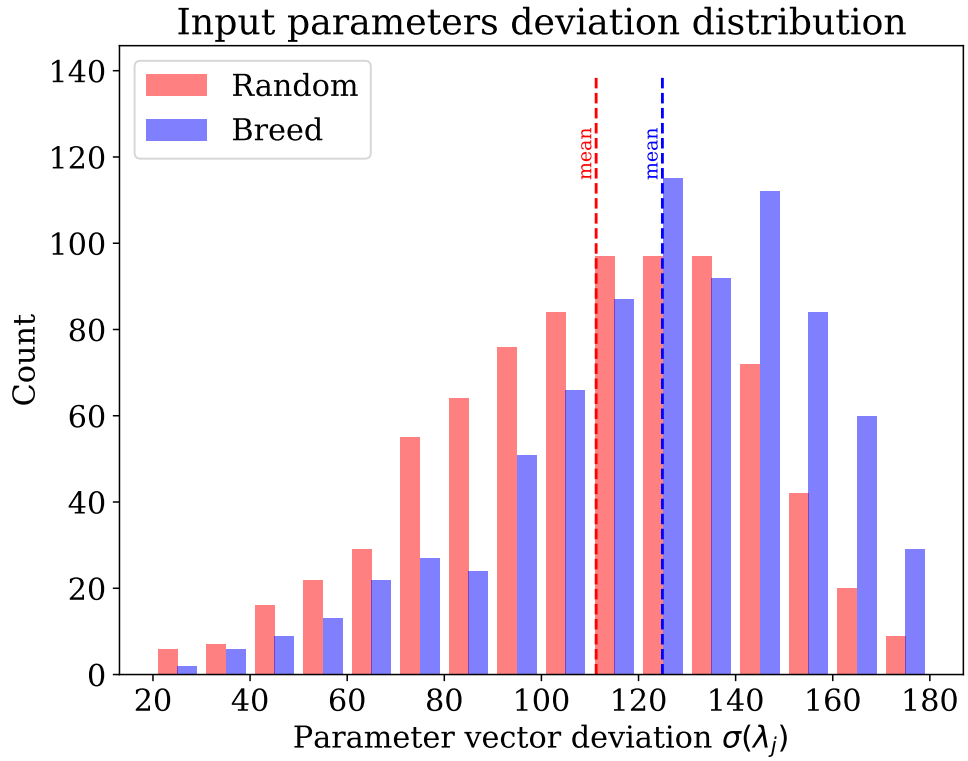
6.3.4 Conclusion and Limitations

Our experiments show that Breed steers sampling toward solver input parameters that produce harder-to-learn trajectories, specifically more dynamic trajectories with larger differences in input temperature parameters. Including such training samples improves surrogate generalisation, reflected in a reduced gap between training and validation losses. The method achieves this with minimal computational overhead, as it is designed to run online during training without interrupting the training process. We studied the method's hyperparameters through ablation experiments, providing insights for their tuning, and identified further improvement, such as adaptive resampling and width schedules.

While our experiments used a simple setup (2D heat PDE with a small direct surrogate model), they provide proof-of-concept for the method's online integration and benefits. Beyond this simplicity, the study has several limitations: inconclusive performance gain of Breed over Random sampling; limited exploration of Breed hyperparameters; absence of significance testing since only one run per configuration was conducted; lack of computational cost statistics to quantify overhead. These limitations motivate further investigation in more complex scenarios, which we discuss next.



(a) Uniform set samples (orange) vs. Concentrated set samples (light blue) within one Breed run.



(b) Breed run samples (blue) vs. Random run samples (red).

Figure 6.5: **Parameters distribution analysis.** Histograms of solver input parameters standard deviation $\sigma(\boldsymbol{\lambda}_j)$ distribution for samples from different categories. The mean is plotted as a vertical line of corresponding colour to emphasise the distributional shift.

6.4 Advanced Experiments for Autoregressive Nonlinear PDE Surrogates

Building on the proof-of-concept results from the previous section, we extend the experiments to more challenging settings with complex PDEs and higher-dimensional problems. We also develop a second implementation of the active sampling mechanism within Melissa that addresses the limitations of the first version. We first describe the new implementation, introduce Breed hyperparameter regimes we designed to reduce the hyperparameter tuning burden, then describe the experimental setup, and conclude by discussing the results and limitations.

6.4.1 Improved Active Sampling Mechanism

The first implementation of the active sampling mechanism within Melissa has certain limitations that we discussed in subsection 6.2.2, i.e., delays caused by the block of the production-consumption loop, and hindered AL reactivity due to the batched job submission by the Launcher and uncertainty in the job start times. We address these limitations with two key modifications: separating the Resampler from the Trainer using an additional MPI rank with synchronisation at a resampling trigger, and redesigning the job submission logic.

Separate active sampling rank. The Server's workflow is centred around the divergence of MPI ranks into two distinct roles: training ranks $\{R_0, \ldots, R_{N_g}\}$ (the Trainer) and a dedicated active sampling rank R_b (the Resampler) that handles resampling and distribution updates. We illustrate the new workflow in Figure 6.6 and in Algorithm 4, where this separation is indicated with blue (the Trainer) and red (the Resampler) elements. This divergence enables overlapping active sampling and training, while preserving samples throughput.

As before, the data reception and communication with the Launcher are handled on R_0 as the Receiver role (Alg. 4, lines 1, 8, 11). Following a data-parallel training scheme, training ranks perform forward and backward passes on their assigned batches and compute per-sample acquisition scores a_{jti} to store locally (lines 14-15).

At the end of each training iteration, if no resampling is currently ongoing (line 17), the Trainer globally checks if the resampling trigger conditions are met (line 19). If so, training ranks send their local a_{jti} values to the Resampler (line 20), which aggregates these and computes per-trajectory scores a_j (line 29). In addition, the information about the most recently processed training samples is sent to the Resampler to update the recency queue $\mathcal{Q}$ (described in subsection 6.2.3.2).

Training is continued while the Resampler performs the sampling procedure, and the Trainer asynchronously probes for sampling completion at the end of each training iteration (line 18). Once finished (line 21, line 33), submission and training are temporarily blocked (line 10, lines 22-23), and the update of parameters for next simulations is performed (line 34).

This design allows the surrogate training to proceed with minimal interruptions from the active sampling mechanism and minimises the delay between when the new parameters are sampled and when the corresponding simulation data is used for training, consequently improving AL reactivity.

Algorithm 4 The Melissa server with improved active sampling mechanism.

Require: Total simulations N, concurrent clients N_S, buffer $\mathcal{B}_{\text{buf}}$; parameter space Λ

Launcher (L): Continuously waits for requests from S_D and submits to cluster

Receiver ($\mathbf{S_D}$ on R_0) + Trainer ($\mathbf{S_T}$ on $R_0, \ldots, R_{N_g}$)

(S_D) Create N_S client scripts $\{S_j\}_{j=1}^{N_S}$ using parameter matrix from S_R
(S_D) `send` submission request to L for $\{S_j\}_{j=1}^{N_S}$
(S_D) Last submitted simulation index $k \leftarrow N_S$
(S_T) Initialise model parameters $\boldsymbol{\theta}_0$, iteration counter $i = 0$
while $\mathcal{B}_{\text{buf}} \neq \emptyset$ **do**
    // S_D: Lazy one-by-one submission
    Upon receiving training sample from a client C_j: Update $\mathcal{B}_{\text{buf}}$
    if some C_j finished **then**
        if pause signal received from S_T **then** `wait` for resume signal from S_R
        $k \leftarrow k + 1$; Create script S_k using $\boldsymbol{\lambda}_k$; `send` request to L
    // S_T: Training and active sampling coordination
    Every training iteration i:
        Update $\boldsymbol{\theta}_i$ using batch $\mathcal{B}_i$ from $\mathcal{B}_{\text{buf}}$
        Locally store acquisition scores $\boldsymbol{a}_i$ and recency info for $\mathcal{B}_i$
        $i \leftarrow i + 1$
    if sampling in progress **then**
        Ask S_R if completed (non-blocking probe)
    else if resampling trigger conditions met **then**
        `send` sampling request, all $\boldsymbol{a}_i$, and recency info $\rightarrow \text{S}_\text{R}$
    if sampling complete **then**
        `send` pause signal $\rightarrow \text{S}_\text{D}$; `send` $k \rightarrow \text{S}_\text{R}$
        `wait` for resume signal from S_R
Signal end of study

Resampler ($\mathbf{S_R}$ on R_b)

Create parameter matrix $\{\boldsymbol{\lambda}_j \sim \text{Unif}(\Lambda)\}_{j=1}^{N}$ (memory-mapped)
while not end of study **do**
    `wait` for sampling request from S_T
    if sampling request received **then**
        Aggregate $\boldsymbol{a}_i$ from all training ranks S_T
        Update recency queue $\mathcal{Q}$ and scoring matrix $R[j, t]$
        // M is the average number of submissions between triggers
        $\mathcal{S} \leftarrow$ `produce_new_parameters`(M)
        `send` completion signal $\rightarrow \text{S}_\text{T}$ and receive last submitted index k
        Update parameter matrix: $\{\boldsymbol{\lambda}_j\}_{j=k}^{k+M} \leftarrow \mathcal{S}$
        `send` resume signal $\rightarrow \text{S}_\text{T}$, S_D

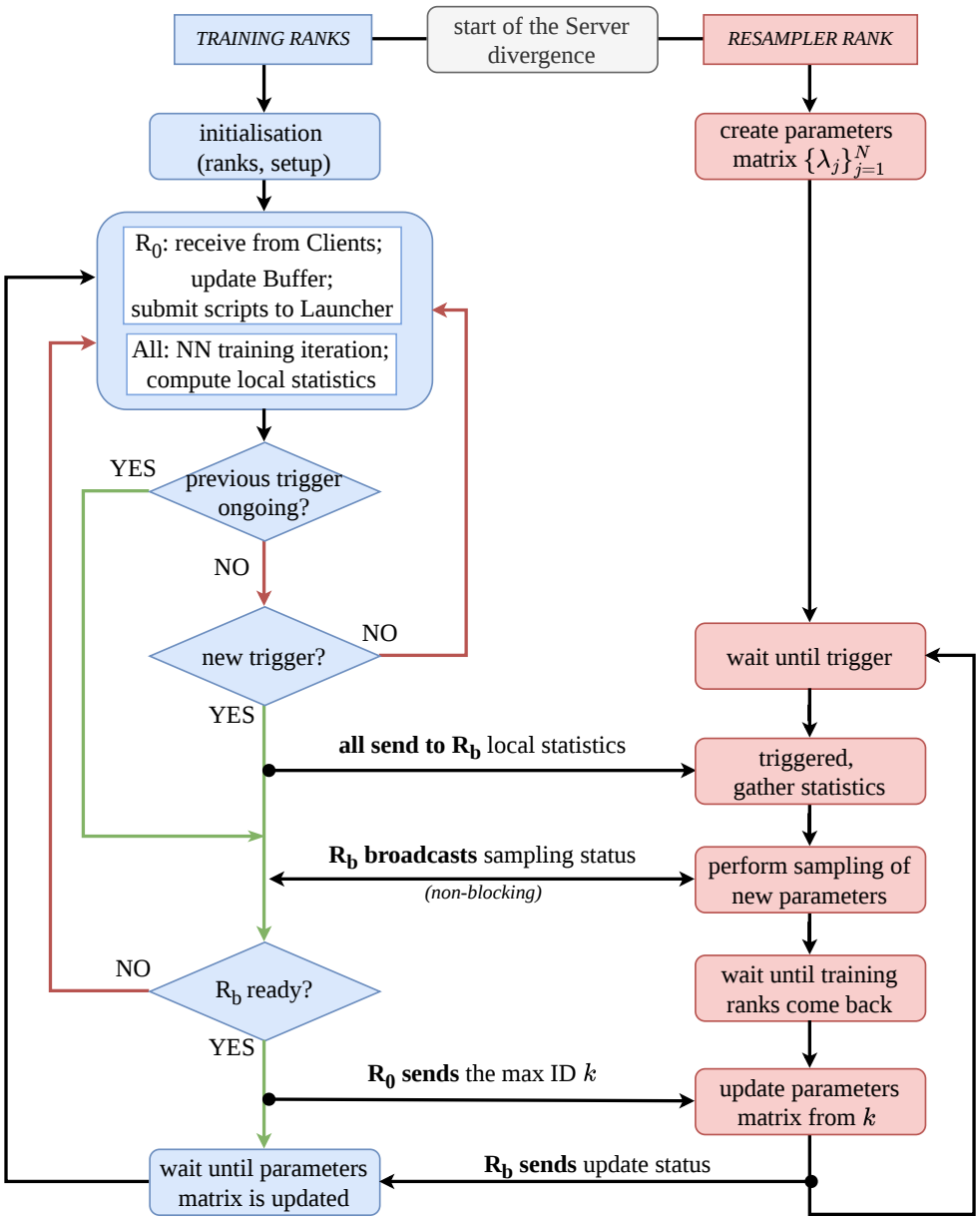


Figure 6.6: The Melissa Server with a separate active sampling rank.

New job submission logic. The job submission strategy has been redesigned to be more reactive to resampling. Initially, the Receiver would create N simulations scripts and request the Launcher to submit all of them at the start of the study, and the Launcher buffered these requests in batches of N_S concurrent solvers for submission to the cluster. Update of the parameters was performed by changing the scripts that certainly had not yet started.

Now, the requests are handled in a lazy, one-by-one approach. At the start of the study, the Receiver creates N_S simulations scripts and requests the Launcher to submit them (lines 2-3). Then, as soon as some simulation finishes (line 9), a new script is created and its simulation is requested (line 11). The Resampler manages a memory-mapped parameters matrix that stores the solver inputs of all N simulations (line 25). The Receiver gets the parameters for the new script from this matrix (line 11).

When a resampling is triggered and finished, i.e., the new set of parameters is prepared by the Resampler, the Trainer temporarily blocks the one-by-one submission and training across ranks (line 10, lines 22-23), communicates the last submitted simulation index to the Resampler (lines 22, 33), and waits (line 23) for the Resampler to complete the update of parameters in the memory-mapped matrix starting from that index (line 34). Once the update is done (line 35), the Receiver resumes submission with the new parameters and training resumes across ranks.

This strategy minimises the delay between when parameters are sampled and when they are used for new simulations, while avoiding race conditions from modifying parameters of already-submitted jobs.

6.4.2 Breed Hyperparameters Regimes

The Breed method described in Section 6.2.3 remains the same, but we attempt to combine the hyperparameters into *regimes* to lessen the burden of hyperparameter tuning. Instead of analysing the effect of each of the hyperparameters separately using grid of values, we design four regimes that follow different strategies based on our understanding of the method behaviour:

- *The precise regime* is designed to resample less often, to focus on the most informative samples, to use the most recent and full information.
- *The broad regime* is designed to resample more often, to cover a wider range of the parameter space, to use big interval of but less complete historical data.
- *The mixed regime* is a combination of the previous two.
- *The soft regime* has higher ratio of uniform mixing, resamples more often than other regimes, uses the biggest interval of historical data.

Detailed per-regime hyperparameters values are provided in Table 6.2. Additionally, we attempt to connect the hyperparameter values with the Melissa hyperparameters, such as memory Buffer size N_B and number of concurrent solvers N_S. The goal is to improve interpretability of the hyperparameters and to ensure that the method scales well with different training setups. The AL method performance depends on the Melissa parameters as they affect the speed ratio between the surrogate training and data generation and the distribution of the data in the buffer.

Table 6.2: Hyperparameters for different method's regimes.

Hyperparameter	**Regime**			
	precise	broad	mixed	soft
Width σ_{frac}	0.05	0.1	0.1	0.1
Schedule start $\varepsilon^{(0)}$	0.5	0.75	0.5	0.75
Schedule end $\varepsilon^{(c)}$	0.75	0.9	0.9	0.5
Schedule clip c	5	3	3	5
Sliding window size N_p	N_B	$2 \cdot N_B$	N_B	$3 \cdot N_B$
Seen steps condition T_{min}	$0.9 \cdot N_t$	$0.25 \cdot N_t$	$0.33 \cdot N_t$	$0.5 \cdot N_t$
First resampling	N_S	$2 \cdot N_S$	$2 \cdot N_S$	$0.9 \cdot N_S$
Resample period p	$5 \cdot N_B$	$2 \cdot N_B$	$3 \cdot N_B$	N_B

6.4.3 Experimental Setup

We aim to evaluate Breed when training autoregressive surrogates for a range of complex nonlinear PDEs with diverse dynamics and sensitivity to initial conditions (ICs). To focus on that aspect, we keep the surrogate architecture and training setup fixed and work with a relatively small discretisation, which allows us to run extensive systematic experiments across different Breed regimes and PDEs. As a baseline, we compare against the default uniform sampling method, as there are no other AL methods designed for this setting to compare against. Compared to the intermediate experiments in the previous section, we introduce a more comprehensive set of metrics to evaluate the surrogate performance and analyse the results in more depth.

As a base for our experimental setup, we employ APEBench by Koehler et al., 2024 [108] — a comprehensive benchmark suite for evaluating autoregressive deep surrogates, which *(1)* provides modular components that can be easily integrated into our online training framework, *(2)* includes solvers for a wide range of challenging PDEs, and *(3)* introduces a "unique dynamics identifier" interface that allows a systematic control over the complexity of the discrete dynamics, thus, the difficulty of the surrogate learning problem.

6.4.3.1 Suite of PDE cases

We study three PDE dynamics, each with two difficulty levels, and four IC functions of the same form but different parameters. This yields 24 possible combinations, but we only use 10 PDE cases. We exclude 14 configurations because some IC parameters are either incompatible with the PDE due to solver instabilities, or they result in dynamics that the surrogate fails to learn adequately with our fixed training setup, which limits the effectiveness of AL methods. Next we detail each of the chosen PDEs, the difficulty parameters, and the IC functions.

PDEs. The PDEs are defined on a periodic 1D spatial domain $\mathcal{U}_x = [0, 1]$ and the solutions discretised into $N_x = 800$ points are provided by Fourier pseudo-spectral solvers from APEBench.

The Kuramoto-Sivashinsky equation is a fourth-order nonlinear PDE that is known for its chaotic behaviour, i.e., it is sensitive to IC changes. It is widely used to model

instabilities in laminar flame fronts, plasma and chemical reaction-diffusion dynamics, and fluid film flows [95].

It has two forms, conservative, which used in fluid film flows, and combustion, typical in flame propagation applications. They are given by:

$$\text{KS-c}(a_2, a_4, b): \quad \frac{\partial u}{\partial t} = -a_2 \frac{\partial^2 u}{\partial x^2} - a_4 \frac{\partial^4 u}{\partial x^4} - b \frac{1}{2} \frac{\partial u^2}{\partial x}, \tag{6.17}$$

$$\text{KS}(a_2, a_4, b): \quad \frac{\partial u}{\partial t} = -a_2 \frac{\partial^2 u}{\partial x^2} - a_4 \frac{\partial^4 u}{\partial x^4} - b \frac{1}{2} \left(\frac{\partial u}{\partial x} \right)^2. \tag{6.18}$$

The conservative form (KS-c) solution conserves the spatial integral of u over time, while the combustion (KS) form does not. KS exhibits more complex dynamics because it is not constrained by the conservation law, leading to richer patterns, while KS-c is more stable. See Figure 6.12 and Figure 6.11 for examples of trajectories. We see that the patterns of KS-c are more centred near zero.

Depending on the IC, the change from a decaying to a chaotic state happens at a different time horizon. The PDE coefficients impact the range of chaotic patterns. Originally in APEBench, the solver performs warmup steps to ensure that the dynamics are in the chaotic regime for most of the trajectory, but we choose to not perform these warmup steps to have more diverse trajectories and to better show the effect of IC parameters on the dynamics.

The Korteweg-de Vries (KdV) equation with dissipation is a fourth-order nonlinear PDE exhibiting decaying dynamics. The standard KdV equation models waves on shallow water surfaces, plasma physics, and lattice dynamics [110].

It is given by:

$$\text{KdV}(a_3, a_4, b): \quad \frac{\partial u}{\partial t} = a_3 \frac{\partial^3 u}{\partial x^3} - a_4 \frac{\partial^4 u}{\partial x^4} - b \frac{1}{2} \frac{\partial u^2}{\partial x} \tag{6.19}$$

While the standard KdV equation exhibits a soliton behaviour (waves that travel without changing shape), the dissipative version we use produces decaying wave structures rather than shape-preserving solitons, resulting in smoother yet nonlinear dynamics. The complexity is lower compared to KS equations, but non-linearities create challenges for surrogates. See Figure 6.7 for an example of trajectory.

Depending on the IC, the wave patterns vary in complexity and decay at different rates. Depending on the PDE coefficients, the dissipation strength and wave propagation characteristics differ.

Difficulty levels. The APEBench provides an interface to systematically control the complexity of the discrete dynamics through a set of coefficients that capture the difficulty of the surrogate learning problem. These *difficulty coefficients* are based on the Courant-Friedrichs-Lewy (CFL) stability condition numbers. Depending on the discretisation and PDE coefficients, the information propagation speed through the mesh and across time steps changes and can be characterised by these numbers. By generalising this concept, APEBench enables systematic benchmarking across a wide spectrum of PDE dynamics complexity.

For linear operators, for the s-th order spatial derivative term with physical coefficient

a_s, the corresponding *linear difficulty coefficient* γ_s is defined as:

$$\gamma_s = \frac{a_s \Delta t}{L^s} \cdot N_x^s \cdot 2^{s-1} \cdot N_d, \tag{6.20}$$

where Δt is the time step size, L is the domain length, N_x is the number of spatial discretisation points, and N_d is the number of spatial dimensions. The *nonlinear difficulty coefficients* δ_{op} are defined similarly depending on the operator, using Δt, L, N_x, N_d, and additionally expected maximum absolute value of the solution over the domain. For further details, we refer to [108].

Together, these coefficients characterise the dynamical behaviour of a chosen PDE system. According to APEBench, the KS equations are represented with linear difficulty coefficients $\{0, 0, \gamma_2, 0, \gamma_4\}$ and nonlinear coefficients $\{\delta_{\text{conv}}\}$ (convection, conservative) or $\{\delta_{\text{gn}}\}$ (gradient norm, combustion), while KdV is represented by $\{0, 0, 0, \gamma_3, \gamma_4\}$ and $\{\delta_{\text{conv_sc}}\}$ (single channel convection).

In addition to the *default set* of difficulty coefficients provided by APEBench, we select a *harder set* of coefficients for each PDE to increase the complexity of the dynamics and to better demonstrate the effect of AL methods.

As the spatial discretisation is increased from $N_x = 160$ in APEBench to $N_x = 800$ in our setup, the difficulty of learning the dynamics increases too, and the coefficients are scaled accordingly. Specifically, a coefficient γ_s in APEBench, in our setup is scaled by a factor of $(800/160)^s$, according to the Eq. (6.20). For example, for KdV equation, the γ_3 coefficient in APEBench is -14, and in our setup it is scaled to $-14 \cdot (800/160)^3 = -1750$. To avoid confusion and to keep the connection with the APEBench setup, in Table 6.3 we report the difficulty coefficients before scaling, and in the experimental code we use the scaled values.

Additionally, to quantify the difficulty of *each trajectory* in terms of its dynamics across time, we define a metric called diffstd. It measures the temporal variability of spatial variation in the trajectory by computing the standard deviation of spatial standard deviations of differences between consecutive timesteps. Higher values indicate more variability in the temporal changes, which is associated with more complex and chaotic dynamics.

The metric is defined as follows:

$$\begin{aligned}
\sigma_{jt} &= \sqrt{\frac{1}{N_x} \sum_{i=0}^{N_x-1} \left((\boldsymbol{X}_j^{t+1} - \boldsymbol{X}_j^t)_i - \mu_{jt} \right)^2}, \qquad \text{where } \mu_{jt} = \frac{1}{N_x} \sum_{i=0}^{N_x-1} (\boldsymbol{X}_j^{t+1} - \boldsymbol{X}_j^t)_i \\
\text{diffstd}_j &= \sqrt{\frac{1}{N_t} \sum_{t=0}^{N_t-1} (\sigma_{jt} - \mu_j)^2}, \qquad \text{where } \mu_j = \frac{1}{N_t} \sum_{t=0}^{N_t-1} \sigma_{jt}.
\end{aligned} \tag{6.21}$$

To show the visual difference between the two difficulty coefficient sets, we provide some trajectory examples for each set for each PDE. We provide the diffstd_j values to facilitate the comparison. In Figure 6.9 for KS, the patterns for the harder case are more complex and chaotic compared to the default case. In Figure 6.12 for KS-c, the harder case example trajectory shows more nonlinearities compared to the default case. In Figure 6.7 for KdV case, the difference is less pronounced, but still visible – the default case example trajectory decaying dynamics converge earlier and show less variability near the boundary.

Table 6.3: Sets of difficulty coefficients γ and δ for each PDE (before scaling). Each PDE is tested with a default and harder difficulty coefficients set.

PDE	Tag	Linear Coefficients	Nonlinear Coefficients
KS-c	default	$\gamma = \{0, 0, -2, 0, -18\}$	$\delta_{\text{conv}} = -1$
	harder	$\gamma = \{0, 0, -3, 0, -50\}$	$\delta_{\text{conv}} = -1$
KS	default	$\gamma = \{0, 0, -1.2, 0, -15\}$	$\delta_{\text{gn}} = -6$
	harder	$\gamma = \{0, 0, -3.3, 0, -30\}$	$\delta_{\text{gn}} = -6$
KdV	default	$\gamma = \{0, 0, 0, -14, -9\}$	$\delta_{\text{conv_sc}} = -2$
	harder	$\gamma = \{0, 0, 0, -20, -5\}$	$\delta_{\text{conv_sc}} = -2$

IC function form and parameters. For the IC function, we employ truncated Fourier series[16], which in 1D corresponds to a superposition of sine waves given by:

$$g_0(x; A_1, \ldots, A_K, \varphi_1, \ldots, \varphi_K) = \sum_{k=1}^{K} A_k \sin(kx + \varphi_k), \tag{6.22}$$

where K is the number of waves (the cutoff), A_k and φ_k are the amplitude and the phase shift of the k-th wave. Changing these parameters allows creation of a wide range of IC functions with different levels of detail, which affects the complexity of the dynamics. In APEBench, the amplitudes are sampled uniformly from $[-1, 1]$ and the phase shifts from $[0, 2\pi]$, which we adopt in our setup.

We make the following two modifications to the IC function generation.

In APEBench experiments, the cutoff value is set to $K = 5$. However, high cutoff results in IC functions with significant high-frequency content, which in the case of KS equations leads to chaotic behaviour appearing early for most of the ICs. Instead, we consider two cases, namely $K = 2$ and $K = 3$, which lead to more diverse trajectories. This effect can be seen by comparing Figures 6.8 and 6.10: with a cutoff of 3, the KS solution shows more complex patterns compared to a cutoff of 2.

Additionally, in APEBench, the generated initial states are normalised to have a maximum absolute value of 1. However, this breaks the dependency on the IC parameters, as different combinations of amplitudes can lead to the same initial state after normalisation. We instead normalise the initial state by simply dividing values by K, which ensures that the maximum value is at most 1, while preserving the dependency on the IC parameters. To study the effect of normalisation, for the KS equation we consider two cases, namely without normalisation and with our normalisation approach. The effect of normalisation can be seen by comparing Figures 6.8 and 6.9: without normalisation, the KS solution transitions into chaos earlier compared to with normalisation.

For the AL method, the IC parameters vector to sample from is defined as $\boldsymbol{\lambda} = (A_1, \ldots, A_K, \varphi_1, \ldots, \varphi_K)$, where K is either 2 or 3 depending on the experiment.

Case combinations. In total, there are 10 cases as we combine different PDEs (KS, KS-c, KdV), difficulty levels (default, harder), cutoff values (2, 3), and normalisation options (with, without):

- (2 cases) KS, default difficulty, $K = 2$: with or without normalisation;
- (4 cases) KS, harder difficulty: $K = 2$ or $K = 3$, with or without normalisation;

- (2 cases) KS-c, $K = 3$, with normalisation: default or harder difficulty;
- (2 cases) KdV, $K = 2$, with normalisation: default or harder difficulty;

Examples of dynamics We provide examples of trajectories for different PDE cases to illustrate the effect of chosen parameters on the dynamics visually and statistically. In Figures 6.7-6.12, we show pairs of trajectories for the same IC parameters (except KdV, see the caption note) for the default and harder difficulty levels (upper and lower subfigures, respectively).

Each subfigure contains the following plots (top to bottom, left to right), where we use the notation (x-axis, y-axis) or (x-axis, y-axis, colour) to indicate what each axis represents:

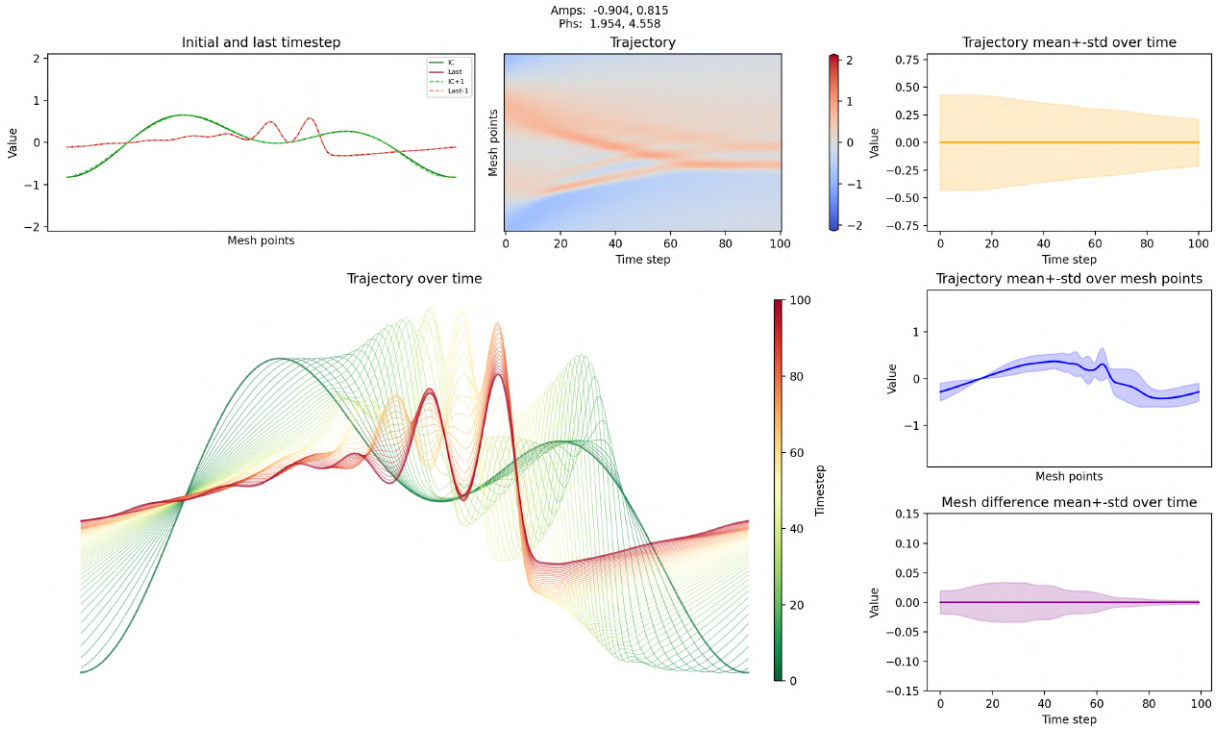
1. Line plot (space, solution) of the solution at timesteps $t = 0$ (green), 1 (dotted green), $N_t - 1$ (red dotted), and N_t (red). We can analyse the difference between the initial and final states, and assess the overall change in the solution over time.
2. Image plot (space, time, solution) of the full trajectory. The colour map is centred at zero (white), with negative values in blue and positive values in red. This shows how the solution evolves across space and time.
3. Line plot (time, mean solution) with shaded area representing one standard deviation. We can see how the average behaviour of the solution changes over time, and statistically assess the variability across time.
4. Set of line plots (space, solution, time) showing the solution at every 5th timestep with colour ranging from green (initial state, $t = 0$) to red (final state, $t = N_t$). This visualisation shows the evolution of the solution shape, the rate of divergence from the initial state, and temporal variability — particularly useful for chaotic dynamics.
5. Line plot (space, mean solution) with shaded area representing one standard deviation. We can see how the average behaviour of the solution changes over space, and statistically assess the variability across space.
6. Line plot (time, std of differences) of the standard deviation of the difference between consecutive timesteps over time. We can see the variability in temporal changes between timesteps.

Each vertical axis range and colour bar scale is fixed across all plots of the same type for better comparison between different cases and trajectories, except for the line plot (3) where the y-axis limits are set per trajectory as minimum and maximum of the solution values across space and time.

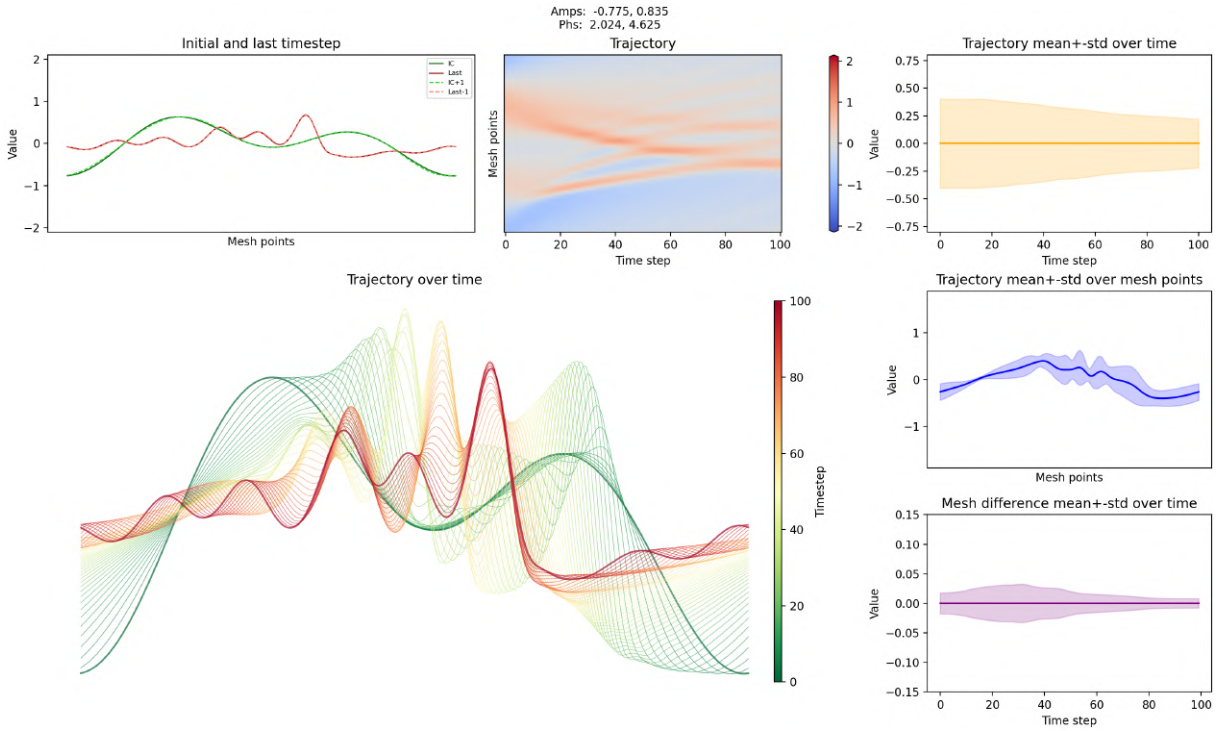
To avoid cluttering the manuscript, we provide only the examples needed to illustrate the effect of the IC parameters and difficulty sets on the dynamics.

6.4.3.2 Surrogate training details

As the AL method is designed to be model-agnostic, we choose one surrogate architecture for all experiments to focus on studying the method's performance across PDE cases. According to the APEBench results, the UNet architecture performs consistently well across

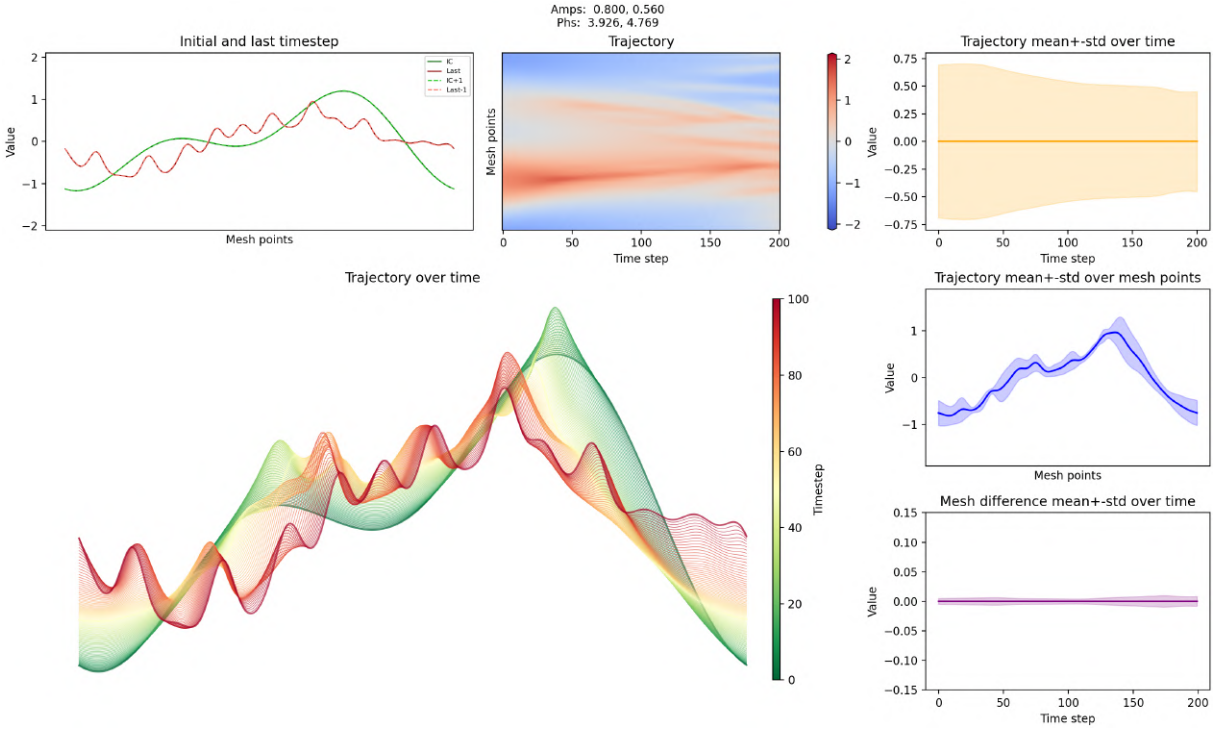


(a) Default, $\text{diffstd}_{1348} = 1.1e^{-2}$

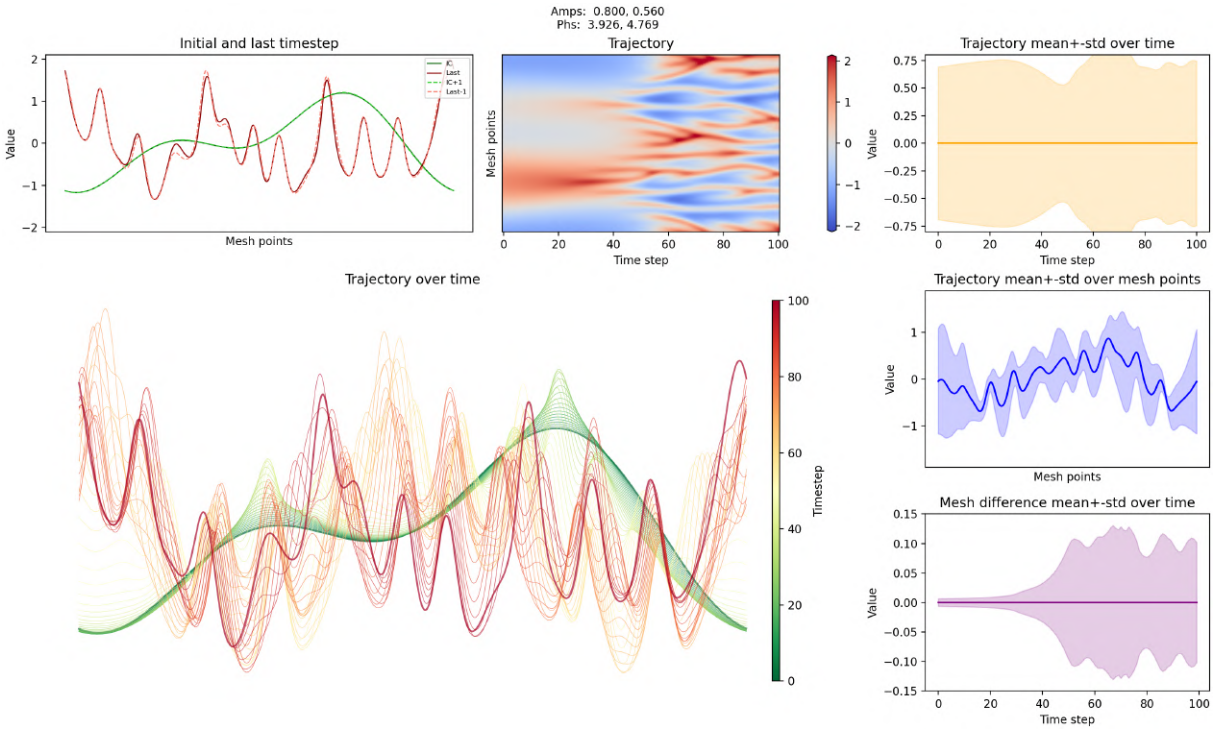


(b) Harder, $\text{diffstd}_{241} = 8.2e^{-3}$

Figure 6.7: KdV, $K = 2$, for $\boldsymbol{\lambda}_{1348}$ and $\boldsymbol{\lambda}_{241}$. These two IDs are almost the same IC parameters, but different dynamics due to different difficulty parameters.

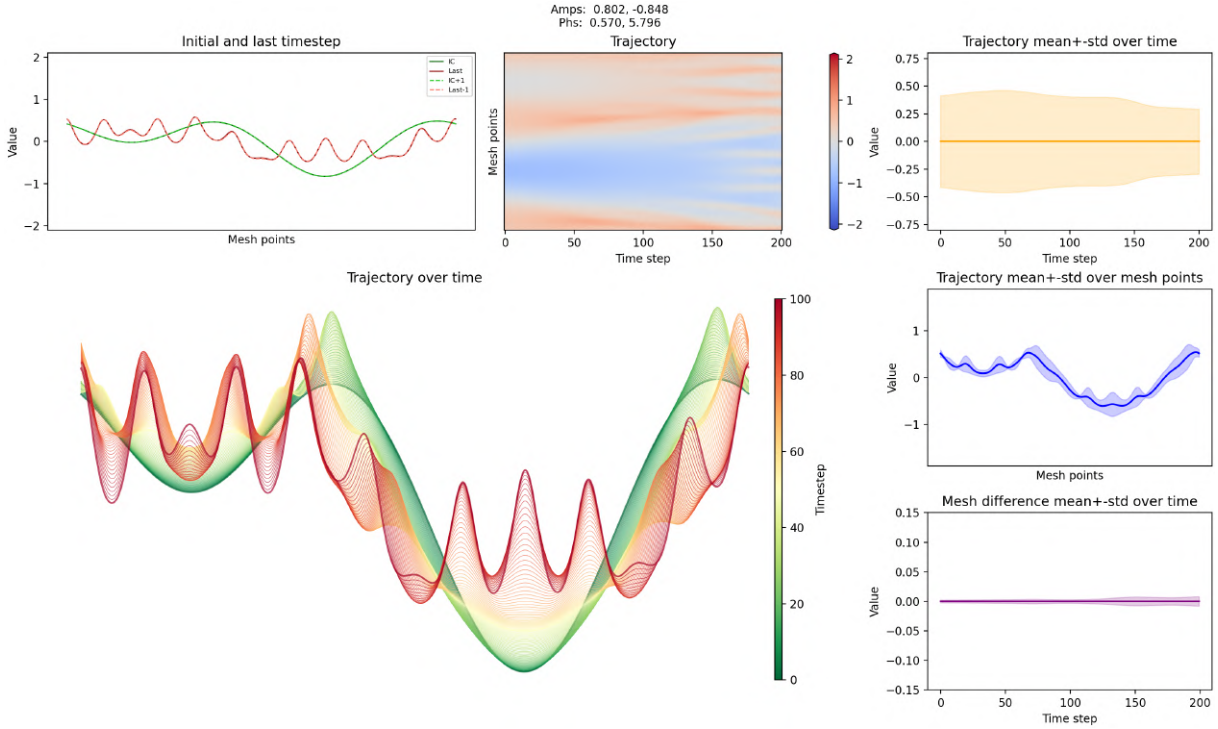


(a) Default, $\text{diffstd}_{70} = 1.6e^{-3}$

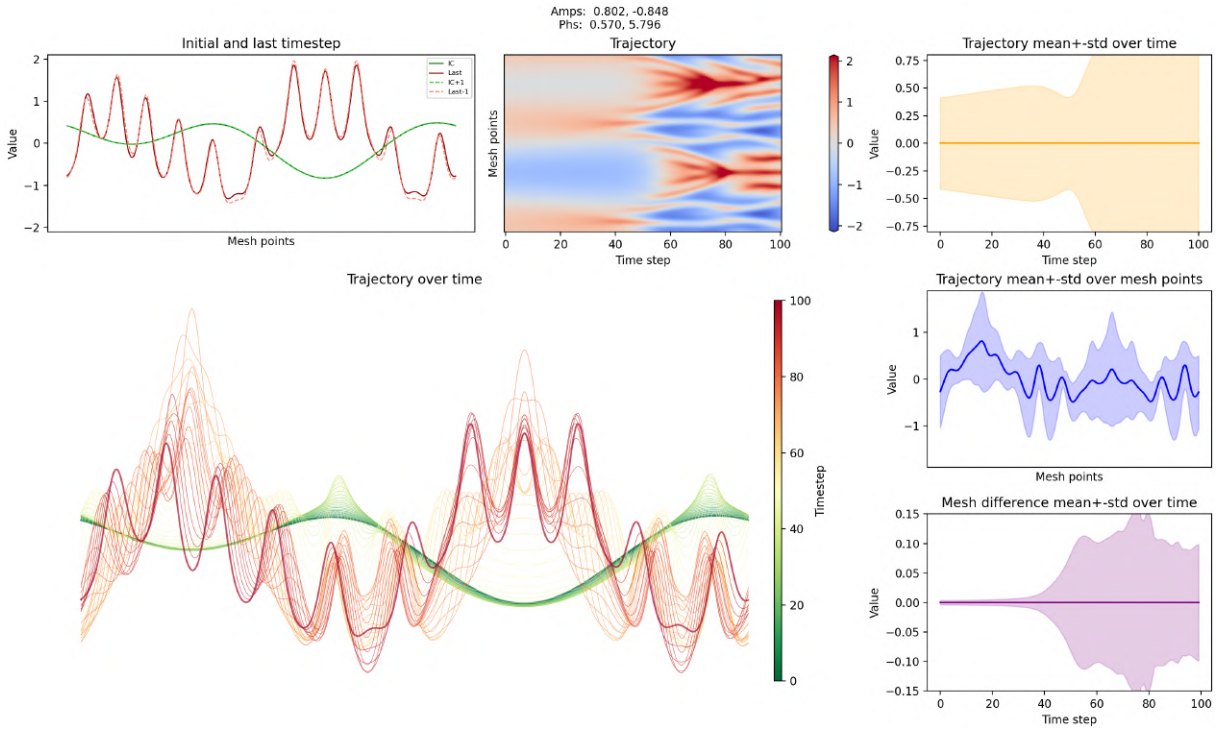


(b) Harder, $\text{diffstd}_{70} = 4.6e^{-2}$

Figure 6.8: KS, $K = 2$, for $\boldsymbol{\lambda}_{70}$

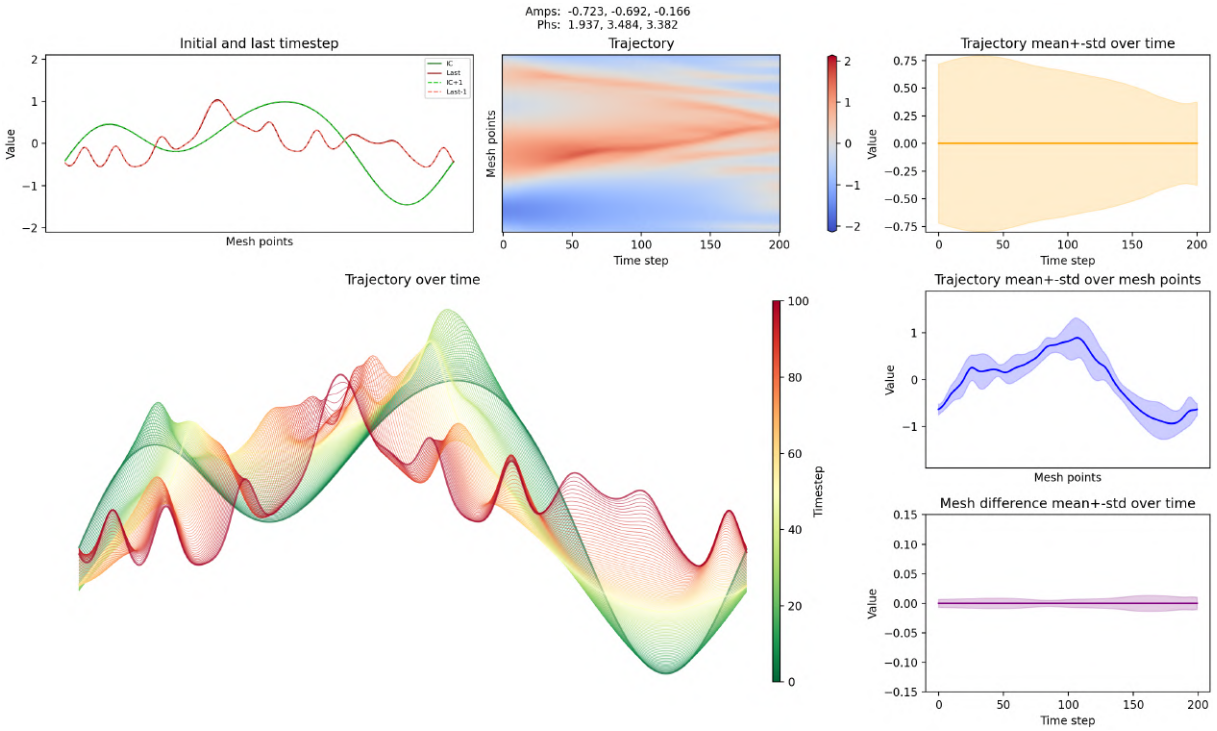


(a) Default, $\text{diffstd}_{320} = 1.8e^{-3}$

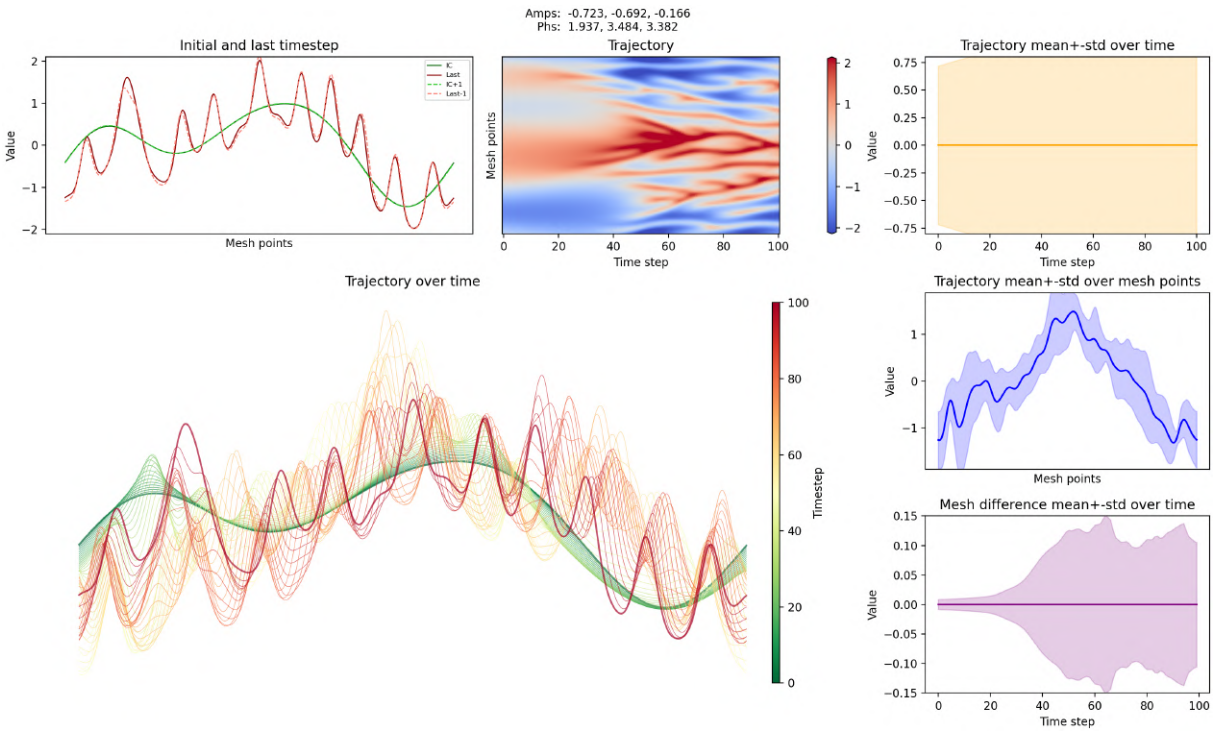


(b) Harder, $\text{diffstd}_{320} = 5.3e^{-2}$

Figure 6.9: KS, $K = 2$, with normalisation, for $\boldsymbol{\lambda}_{320}$

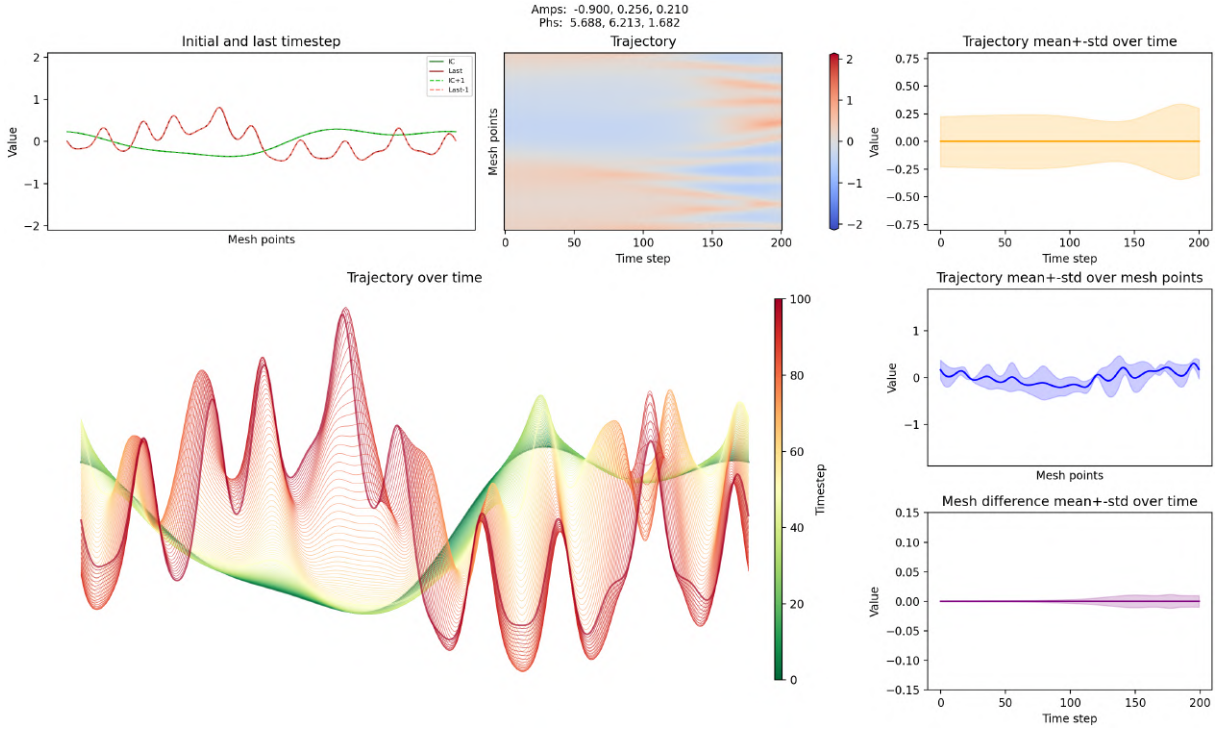


(a) Default, $\text{diffstd}_6 = 2.4e^{-3}$ (not used for the experiments)

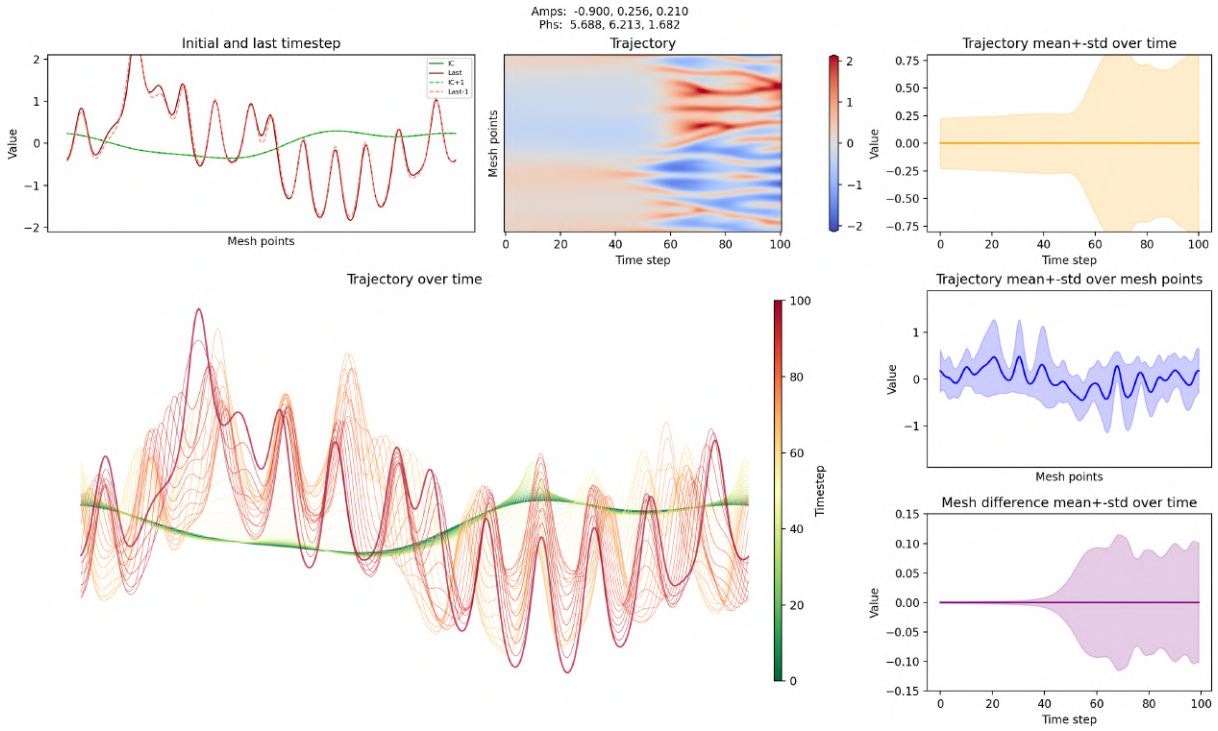


(b) Harder, $\text{diffstd}_6 = 4.9e^{-2}$

Figure 6.10: KS, $K = 3$, for $\boldsymbol{\lambda}_6$

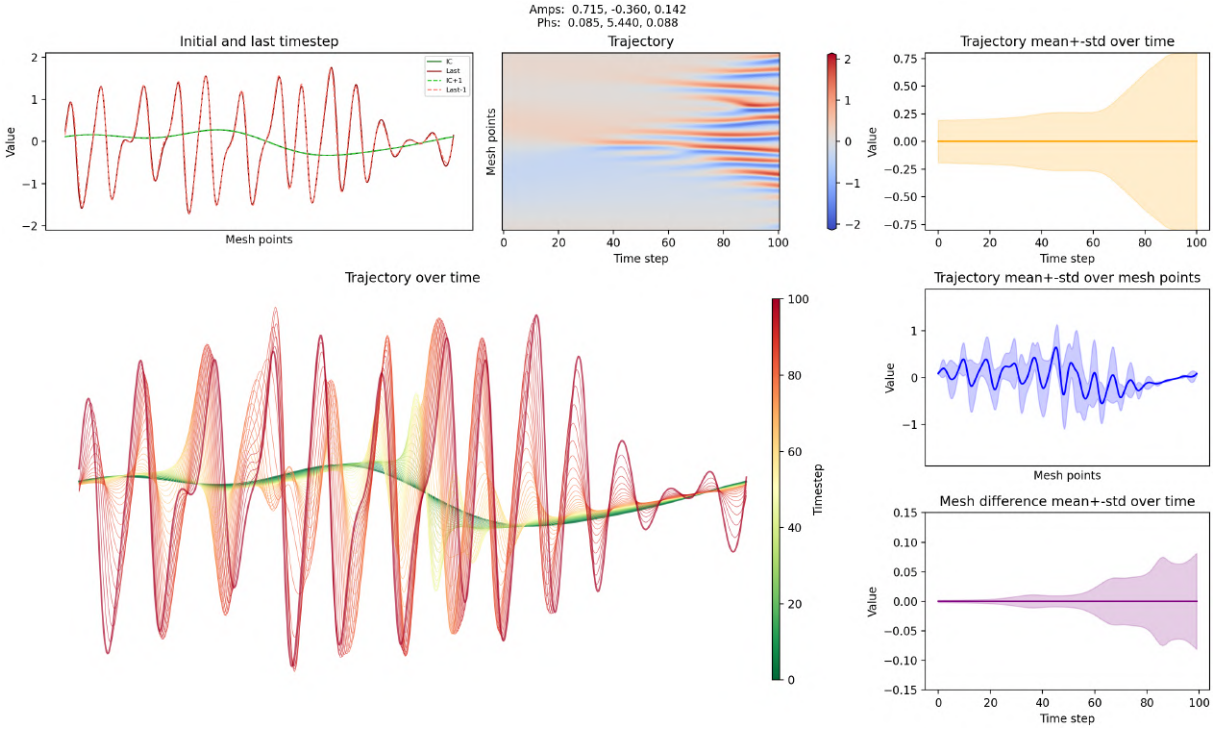


(a) Default, $\text{diffstd}_{973} = 4.0e^{-3}$ (not used for the experiments)

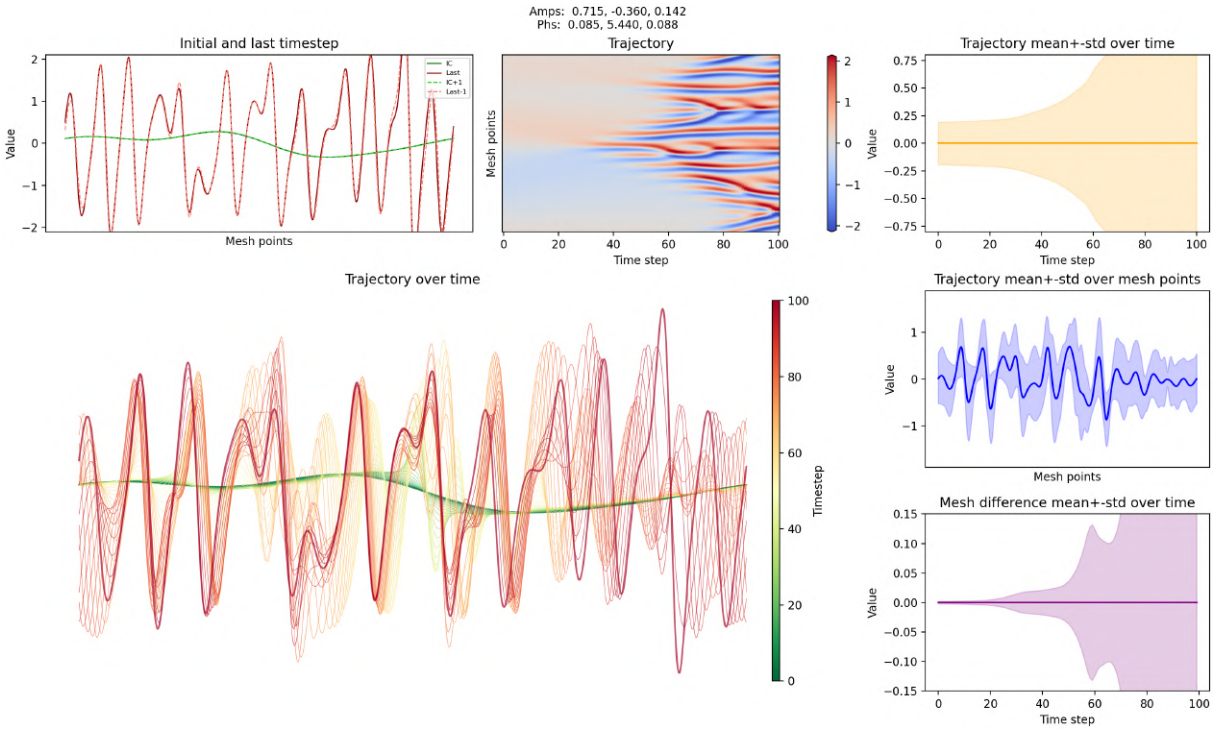


(b) Harder, $\text{diffstd}_{973} = 4.3e^{-2}$

Figure 6.11: KS, $K = 3$, with normalisation, for $\boldsymbol{\lambda}_{973}$



(a) Default, $\text{diffstd}_{979} = 2.4e^{-2}$



(b) Harder, $\text{diffstd}_{979} = 8.6e^{-2}$

Figure 6.12: KS conservative, $K = 3$, with normalisation, for $\boldsymbol{\lambda}_{979}$

different PDEs, often outperforming other architectures. Since the spatial discretisation in our experiments is higher compared to APEBench, we increase the depth of the network from 2 to 4, but decrease the width from 12 to 6 to keep the number of parameters manageable and the model size close to the one used in APEBench.

We use the default optimiser and loss, i.e., Adam and MSE loss, and we tune the training hyperparameters such as learning rate and batch size to ensure stable training and good convergence when uniform sampling is used. Specifically, the learning rate is set to 10^{-3} with decay to 10^{-4} over 5000 iterations, and the batch size is 256.

Following APEBench, we evaluate the surrogate's ability to predict beyond the training temporal horizon. We provide only the first 75% of a trajectory for training, while validation is performed on full trajectories (the Δt is kept the same). We set $N_t = 100$ for all PDE cases except default KS, $K = 2$ (with and without normalisation), where we set $N_t = 200$ to ensure that the dynamics transition into chaos within the training horizon for most of the ICs, which allows us to better demonstrate the effect of AL methods. In comparison, in APEBench training trajectories consist of 50 timesteps, and testing of 200 timesteps.

The total number of simulations generated is set to $N = 2000$ ($N = 4000$ for $N_t = 200$), which is considerably higher ($\times 40$) compared to APEBench experiments, where in an offline setting the dataset consists of 50 trajectories. However, for the online setting, even $N = 2000$ is relatively small-scale experiment, as the total training data produced takes only approximately 690 Mb (for $N_t = 100$) and we can fit all the data in the memory. We choose this number to be able to run a wide range of experiments. To mimic large-scale training setups, we intentionally limit the Buffer size to contain only 6.5% of the total data created, i.e., $N_B = 100$. This ensures that training is done in an online fashion and the surrogate sees only a fraction of the data at each training iteration, which is typical for large-scale training setups. The number of concurrent solvers is set to $N_S = 8$ to ensure diversity of the data in the Buffer and balance between the data generation and training speeds.

To compare the Breed regimes and the baseline fairly, we fix the model's initialisation and the initial set of IC parameters across all the compared methods and PDE cases, and we run each experiment once. For a subset of experiments, we perform multiple runs with different seeds generating the initial set of IC parameters. We include 3 PDE cases, namely KS with $K = 2$ and normalisation, KS with $K = 3$ without normalisation, and KS-c with $K = 3$ with normalisation, and we compare the broad Breed regime with uniform sampling. For each case and method, we perform 5 runs with different seeds. We choose to perform seeded runs only on a subset to reduce the computational burden, while still being able to provide statistically significant results. However, inherent randomness in the online training process—caused by asynchronous data generation, Buffer management, and GPU non-determinism—introduces inevitable variability in the results.

Duration of training. Duration of one experiment is set to 1 hour (2 hours for $N_t = 200$, 30 minutes for seeded runs), which results in approximately 3-5k training iterations using a single V100 NVIDIA GPU. The variation in the number of training iterations is caused by the asynchronous nature of the online training process. For the same amount of generated data, a surrogate may perform different number of gradient updates due to the variability in communication time between Clients and Server within the cluster and inherent GPU non-determinism. Thus, we report summary results at a given training iteration, which is available for all experiments, to ensure fair comparison. We choose to

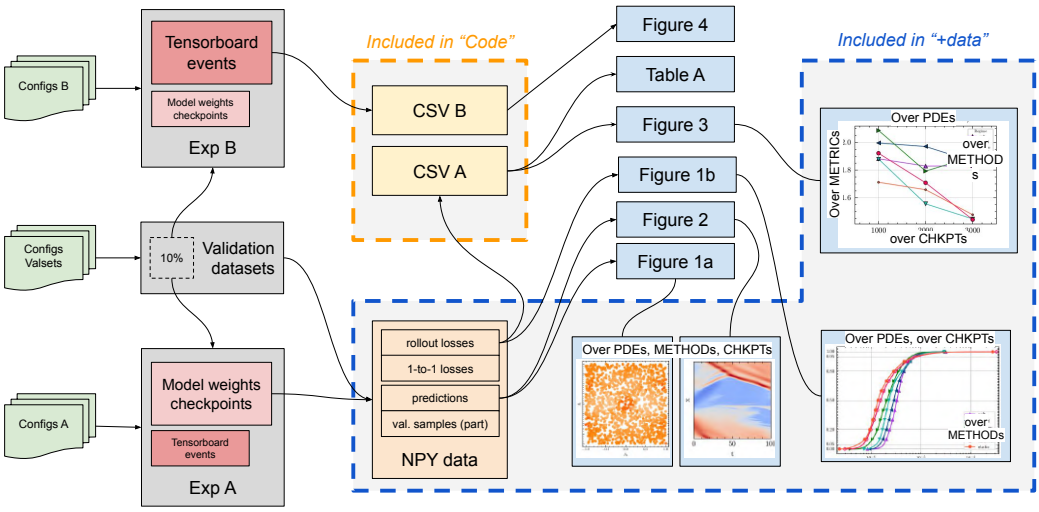


Figure 6.13: Assets organisation diagram.

report results at iteration $i = 3000$ as it is the last checkpoint available for testing for all experiments, while some experiments have checkpoints at the $i = 4000$ iteration.

Assets and reproducibility. The assets of the experiments are organised as shown in Figure 6.13. The code, configurations, and intermediate results data are publicly available at https://github.com/svdcvt/breed_neurips_experiments_assets. We provide detailed instructions to reproduce the results. Full assets are under restricted access, available with a token-link[2]. It provides all the test datasets, models' checkpoints, additional data arrays, figures, and environment container (in total around 28Gb).

6.4.3.3 Evaluation details

We use a test set $\mathcal{D}_{\text{test}}$ of 1500 trajectories. The IC parameters are sampled from a quasi-uniform distribution using Hammersley sequence to ensure low-discrepancy coverage of the space. Testing is performed separately after the training is finished, as to not interfere with the online training process. The model's weights checkpoint is saved every 1000 training iterations for testing.

Metrics. As mentioned before, we use a MSE loss for training. For evaluation, we report per-sample MSE over the test set through distribution statistics. Namely, mean (MSE), standard deviation (MSE_{sd}), and maximum (MSE_{max}). The mean of MSE captures the average error, while the standard deviation and maximum capture the variability and worst-case error, respectively, which are important for assessing the surrogate's reliability and generalisation.

These are defined as follows:

$$\text{MSE}_{\text{sample}}(\boldsymbol{\theta}; \boldsymbol{X}^t) = \frac{1}{N_x} \|\boldsymbol{X}^{t+1} - f(\boldsymbol{X}^t; \boldsymbol{\theta})\|_2^2 \tag{6.23}$$

[2] https://zenodo.org/records/15490672?token=... (hyperlink contains the full token)

$$\mathrm{MSE}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\boldsymbol{X}^t \in \mathcal{D}_{\text{test}}} \mathrm{MSE}_{\text{sample}}(\boldsymbol{\theta}; \boldsymbol{X}^t) \tag{6.24}$$

$$\mathrm{MSE}_{\text{sd}}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) = \sqrt{\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\boldsymbol{X}^t \in \mathcal{D}_{\text{test}}} \left(\mathrm{MSE}_{\text{sample}}(\boldsymbol{\theta}; \boldsymbol{X}^t) - \mathrm{MSE}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}})\right)^2} \tag{6.25}$$

$$\mathrm{MSE}_{\text{max}}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) = \max_{\boldsymbol{X}^t \in \mathcal{D}_{\text{test}}} \mathrm{MSE}_{\text{sample}}(\boldsymbol{\theta}; \boldsymbol{X}^t) \tag{6.26}$$

Visualisations. We use several visualisations to present the experimental results from different perspectives: final validation error distribution, dynamics of metrics across training iterations, sampling behaviour, and dependency of the surrogate's performance on the IC parameters. We describe each visualisation in detail.

Comparing test loss distributions. A line plot of the empirical cumulative distribution function (eCDF) of the per-sample MSE losses over the test set at a given training iteration. See Figure 6.15, where we provide this visualisation for each PDE case (indicated in the subtitle), using the checkpoint $i = 3000$ for testing, as it is available for every experiment. Horizontal axis corresponds to the loss value (log scale) and vertical to the fraction of samples with loss less than or equal to the corresponding loss value. Every line corresponds to a distribution of test losses of surrogate trained using a different method (colour and marker provided in the legend). Each point-marker of the line corresponds to every 10-th percentile of the distribution, including minimum, maximum, and median values (horizontal grid lines serve as guide for percentiles).

This visualisation allows clear comparison of different distributions and their properties, such as variance and maximum value.

Metrics progression during training. A point plot of a metric (y-axis) over the test set across training iterations (x-axis). See Figures 6.17, 6.18, where we provide this visualisation for each metric, PDE case, and method, using all the checkpoints available for testing. Each point corresponds to the metric value computed on the test set for a given model's checkpoint and a given method (different colours and markers provided in the legend). Points are connected with lines for guidance and highlighting the trends.

This visualisation allows assessing the convergence of the surrogate and the data efficiency of the method, as well as comparing the performance of different methods across training iterations.

Metrics progression with statistical significance. A line plot with error bars of the validation metric (y-axis, log scale) value over training iterations (x-axis) averaged across multiple seeded runs. See Figure 6.16, where we provide this visualisation for the subset of experiments with seeded runs, with each row corresponding to a different PDE case and each column to a different metric. Each solid line corresponds to the mean value of the metric across seeded runs for the compared method (different colours provided in the legend), with error bars representing 95% confidence interval around the mean. For better visualisation, we omit the first 199 training iterations, because at the beginning of training the metric values are very high and the differences between methods are minor,

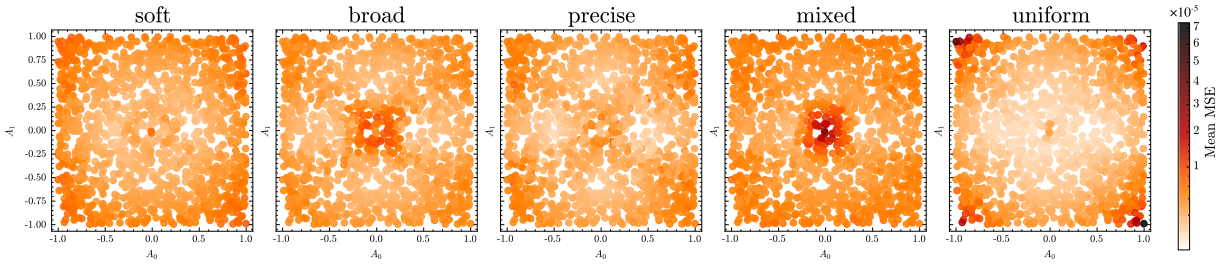


Figure 6.14: Comparison of the mean-over-trajectory test loss distribution over IC parameters between Breed (left) and uniform sampling (right) for default KdV at training iteration $i = 4000$.

which makes it less clear to see the trends and differences between methods in the later stages of training.

This visualisation allows assessing the statistical significance of the results and the variability of the method's performance.

Sampling distribution progression. Pairwise scatter plots of solver input parameters sampled across resampling iterations. We provide this visualisation only for one PDE case (KS with $K = 2$) for two Breed regimes (broad and precise) to avoid cluttering the manuscript. See Figures 6.19 and 6.20. Each subplot corresponds to a different pair of IC parameters (amplitude and phase shift parameters), indicated in the horizontal and vertical axes labels. Each scatter point corresponds to a sampled parameter vector from the proposal distribution at a given resampling iteration, with colour indicating the iteration index (colour bar is provided) and from random distribution (grey colour).

This visualisation allows assessing how the sampling distribution evolves across resampling iterations and its convergence towards certain regions of the parameter space.

Loss distribution over IC parameters. A scatter plot of per-sample loss values over the test set for different methods. We provide this visualisation only for one PDE case (default KdV), comparing Breed and uniform sampling, to avoid cluttering the manuscript. See Figure 6.14. Each point corresponds to a (A_1, A_2) pair of IC parameters for a sample $\boldsymbol{\lambda}_j$ from the test set. The colour of the point corresponds to an average per-sample MSE loss value across time steps of the trajectory $\boldsymbol{X}_j$ predicted by the surrogate with a given checkpoint. As the distribution of loss values across the test set is highly skewed towards zero, resembling log-normal distribution, we use logarithmic colour scale to have better visualisation of the loss distribution. The colour bar is provided for reference, with the same scale across all compared methods.

This visualisation allows assessing how the surrogate's performance varies across different regions of the IC parameter space, and whether certain regions are more challenging for the surrogate to predict.

6.4.4 Experimental Results and Analysis

6.4.4.1 Overall performance from distributional perspective

We summarise the results of all experiments in Table 6.4, where for each PDE case and method we report the mean, standard deviation, and maximum of the distribution of test losses at iteration $i = 3000$. Full distributions of the test losses are visualised in

Figure 6.15, allowing to assess form of the distribution (centre, modes, tails) and compare median and minimum values.

Lower worst-case errors and dispersion. Compared to the uniform baseline, the *Breed regimes consistently show lower standard deviation and maximum values* of the test loss distribution across all PDE cases. In some cases, the difference reaches one order of magnitude, for example, the worst-case error in default KS and KdV cases, and the deviation in harder KS with $K = 3$ and normalisation, default KS-c, and default KdV cases.

For tasks where worst-case error is important, such as safety-critical applications, this improvement is significant as it indicates that the surrogate is more reliable and less likely to produce large errors on outliers. The reduction in dispersion also suggests that the surrogate's performance is more consistent across different samples, which can be crucial for tasks where uniformity of performance is desired.

Resulting shift in median and minimum errors. We found that the improvement in the worst-case error *often results in increase of minimum and median values.* In 3 cases, namely harder KdV and default KS with $K = 2$ with and without normalisation, the shift of the distribution towards centre is specifically notable for Breed.

This apparent degradation of average performance can be beneficial: without AL, uniform sampling encourages overfitting to simpler dynamics, with minimum errors 2-3 orders of magnitude lower than the maximum. *Breed narrows the loss range, suggesting slower but potentially better convergence towards a global optimum.*

In two cases, however, namely default KdV and harder KS with $K = 3$ and normalisation, both minimum and median errors are lowered by Breed. Interestingly, the improvement of the median error in the former case is more notable than in the latter (visible in Figure 6.15), but the improvement of the average error is more notable in the latter case (visible in Table 6.4).

We acknowledge this inconsistency across different PDE cases. We hypothesise that Breed can exhibit different behaviour across different PDE cases mainly due to the direct way to model the proposal sampling distribution with an adaptive mixture of Gaussians. This approach, while simple, has low adaptability and expressivity, leading to mode collapse and difficulty in capturing complex distributions.

Multimodal loss distribution. We notice that for harder KS and both KS-c cases the loss distribution has *two modes and heavy right tail, spanning several orders of magnitude.* We tend to characterise these PDE dynamics as more complex, and, likely, more sensitive to the IC parameters. For some of these cases, in Figure 6.15, we can see Breed to slightly improve the central part of the distribution (harder KS-c, harder KS with $K = 2$ and no normalisation) or one of the modes (default KS-c, harder KS with $K = 2$ with normalisation). Additionally, for both harder KS with $K = 2$, the improvement of the worst-case error is the smallest among all PDE cases.

This observation supports the aforementioned limitations. Additionally, we argue that it is important to consider test error metrics beyond the average, which is the most common practice in the literature. For dynamics with high sensitivity to the parameters and intricate parameters–solution dependencies, proper assessment of the surrogate's performance is essential for development of reliable surrogates.

Table 6.4: Test metrics results for each PDE case and compared methods computed using surrogate's checkpoint at the 3000-th iteration. Lower is better, and we highlight the best with gray background.

			Method	broad	mixed	precise	soft	uniform
PDE	**Difficulty**	K	**Norm**			**Metric**		
						MSE		
KS	default	2	False	4.9e-5	5.1e-5	8.7e-5	3.4e-5	2.8e-5
			True	1.4e-5	2.2e-5	1.0e-5	9.2e-6	1.1e-5
	harder	2	False	1.8e-4	3.3e-4	2.3e-4	3.9e-4	4.1e-4
			True	2.1e-4	2.3e-4	3.2e-4	2.1e-4	3.0e-4
		3	False	9.7e-5	1.1e-4	1.4e-4	1.0e-4	1.5e-4
			True	8.9e-5	8.7e-5	7.2e-5	8.4e-5	1.5e-4
KS-c	default	3	True	5.8e-5	6.9e-5	6.9e-5	7.7e-5	7.1e-5
	harder	3	True	1.8e-4	1.6e-4	1.7e-4	1.3e-4	1.9e-4
KdV	default	2	True	5.9e-6	8.0e-6	4.0e-6	3.4e-6	8.1e-6
	harder	2	True	5.1e-6	2.6e-5	4.9e-6	4.3e-6	3.8e-6
						$\mathbf{MSE}_{\max}$		
KS	default	2	False	2.4e-4	4.5e-4	6.6e-4	3.6e-4	3.6e-3
			True	3.8e-4	6.1e-4	3.3e-4	3.8e-4	1.1e-3
	harder	2	False	2.2e-1	2.8e-1	2.5e-1	3.1e-1	3.3e-1
			True	1.4e-1	1.7e-1	1.8e-1	1.4e-1	1.8e-1
		3	False	8.7e-3	3.0e-2	2.8e-2	3.0e-2	4.2e-2
			True	7.0e-3	2.6e-2	2.5e-2	2.1e-2	6.2e-2
KS-c	default	3	True	2.4e-3	5.3e-3	4.1e-3	5.2e-3	7.9e-3
	harder	3	True	5.4e-3	6.3e-3	1.0e-2	1.1e-2	1.4e-2
KdV	default	2	True	5.5e-5	1.3e-4	3.3e-5	5.5e-5	5.7e-4
	harder	2	True	2.9e-5	1.2e-4	4.0e-5	3.2e-5	4.7e-4
						$\mathbf{MSE}_{\mathbf{sd}}$		
KS	default	2	False	3.2e-5	3.0e-5	6.0e-5	2.1e-5	5.7e-5
			True	1.4e-5	1.6e-5	1.4e-5	1.2e-5	2.2e-5
	harder	2	False	2.2e-3	3.7e-3	3.0e-3	4.3e-3	4.6e-3
			True	2.2e-3	2.9e-3	3.3e-3	2.4e-3	3.1e-3
		3	False	9.6e-5	3.9e-4	3.7e-4	4.0e-4	5.7e-4
			True	1.0e-4	3.6e-4	3.3e-4	2.6e-4	1.0e-3
KS-c	default	3	True	5.6e-5	7.6e-5	7.1e-5	8.2e-5	1.2e-4
	harder	3	True	1.6e-4	1.4e-4	1.8e-4	1.7e-4	2.3e-4
KdV	default	2	True	3.4e-6	6.3e-6	2.7e-6	2.8e-6	1.2e-5
	harder	2	True	2.3e-6	1.1e-5	3.2e-6	2.8e-6	9.4e-6

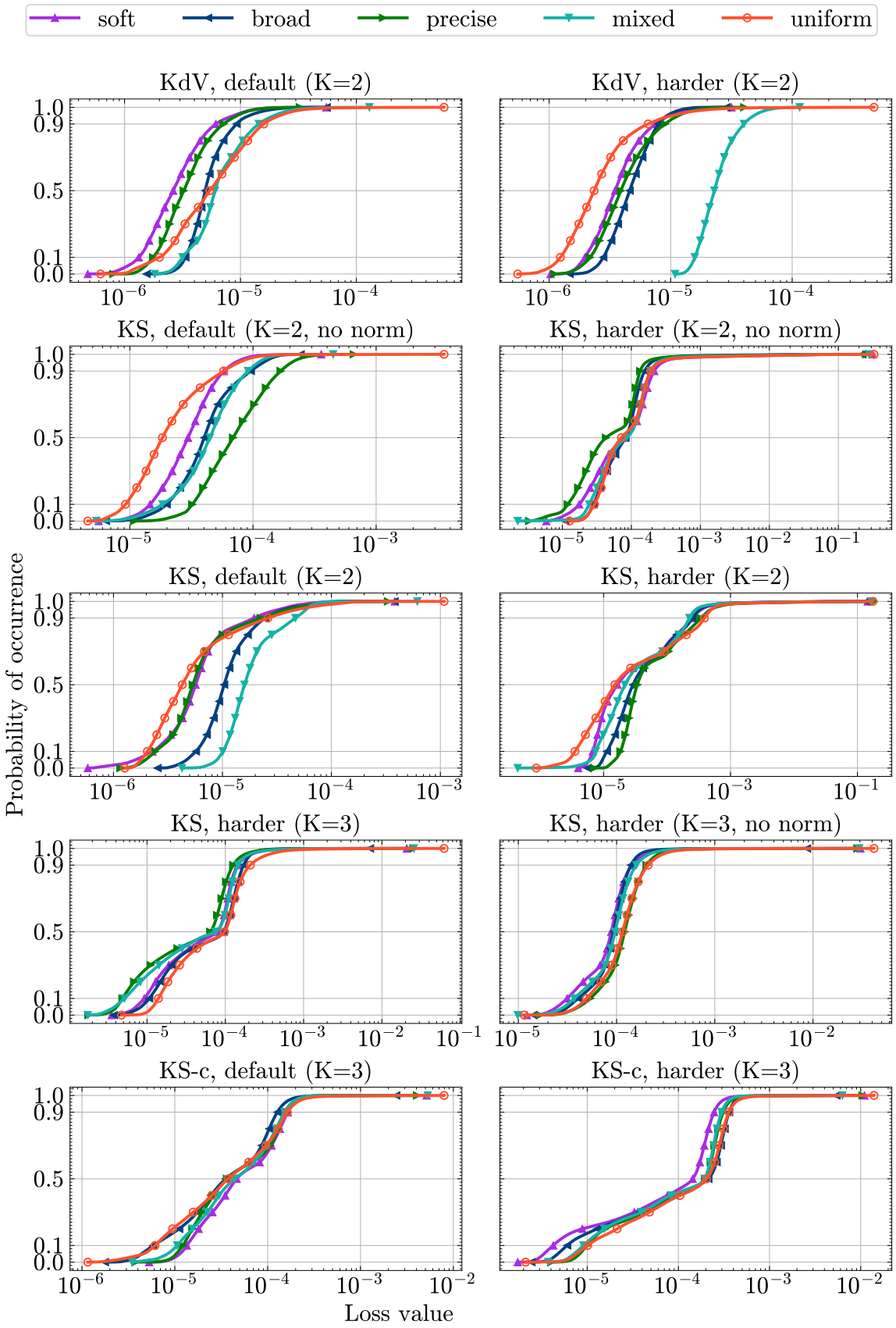


Figure 6.15: Empirical cumulative distribution functions of test per-sample MSE loss values (log scale) at iteration 3000 across 10 PDE cases (each subfigure) and 5 sampling methods (colours are provided in the legend). Leftward shifts indicate better performance (more frequent low loss values), narrower horizontal spread indicates lower variance, and maximum loss values correspond to the rightmost points where curves reach probability 1.0. Markers indicate every 10-th percentile for guidance.

Comparison of Breed regimes. Among the Breed regimes, in terms of worst-case error and dispersion, *the broad regime often shows the best performance*, while the mixed being the worst. For mean error, the soft regime performs on-par with the broad regime, while the precise and mixed regimes often show worse performance.

We attribute these differences to the exploration-exploitation trade-off and the responsiveness of the regimes. The soft regime explores the space the most, with a larger fraction of uniformly sampled points, more Gaussian components, and more frequent resampling. In contrast, the precise regime explores the least, focusing on narrow neighbourhoods with fewer components and rare resampling, which likely explains its poorer performance in both worst-case and mean error. The broad regime strikes a balance between these extremes but uses a stronger concentration on high-loss areas, which we argue is the main reason for its strong worst-case performance while maintaining competitive mean error.

This finding highlights the importance of optimal configuration of the method, which can be challenging due to the large number of hyperparameters and their complex interactions.

6.4.4.2 Dynamics across training iterations

Thus far we have focused on the results at a given training iteration, however, it is also important to assess the surrogate's convergence behaviour.

Test metrics dynamics. One of the AL goals is to improve the data efficiency of training, i.e., achieve desired performance with fewer data samples and training iterations. While we have already discussed the significant reduction of the worst-case error and dispersion by Breed, here we investigate how early Breed achieves these improvements. From Figures 6.17 and 6.18, we see that in most cases, Breed, particularly the broad and soft regimes, reaches the same level of worst-case error and dispersion as the uniform baseline by around the 2000-th iteration, i.e., using roughly half as many training iterations and data samples, and in some cases (KdV and default KS with $K = 2$ without normalisation) even by the 1000-th iteration.

As of the mean error, there are 4 cases where Breed reaches the same or lower mean error than the uniform baseline at the last checkpoint, namely both harder KS with $K = 2$, harder KS with $K = 3$ and normalisation, and default KS-c. Specifically for both harder KS with $K = 2$, the same level of mean error as the uniform baseline is achieved at around the 2000-th iteration for the broad regime.

These results show that *Breed can outperform the baseline with significantly fewer training data and iterations*, which is a crucial advantage for training surrogates for complex PDEs, where data generation can be computationally expensive.

Statistical significance. For the seeded runs subset, these findings are more substantiated, since the test metrics are calculated more frequently across training iterations, enabling better assessment of the trends, and the confidence intervals provide insight into the reliability of the results. From Figure 6.16, we see that Breed consistently shows lower worst-case error and dispersion than the uniform baseline across training iterations, with broad regime showing better performance than the mixed regime. For mean error, the results are more mixed, with some cases showing improvement by Breed (harder KS with $K = 3$ and normalisation) and others showing similar (default KS-c) or worse (default KdV) performance compared to the uniform baseline.

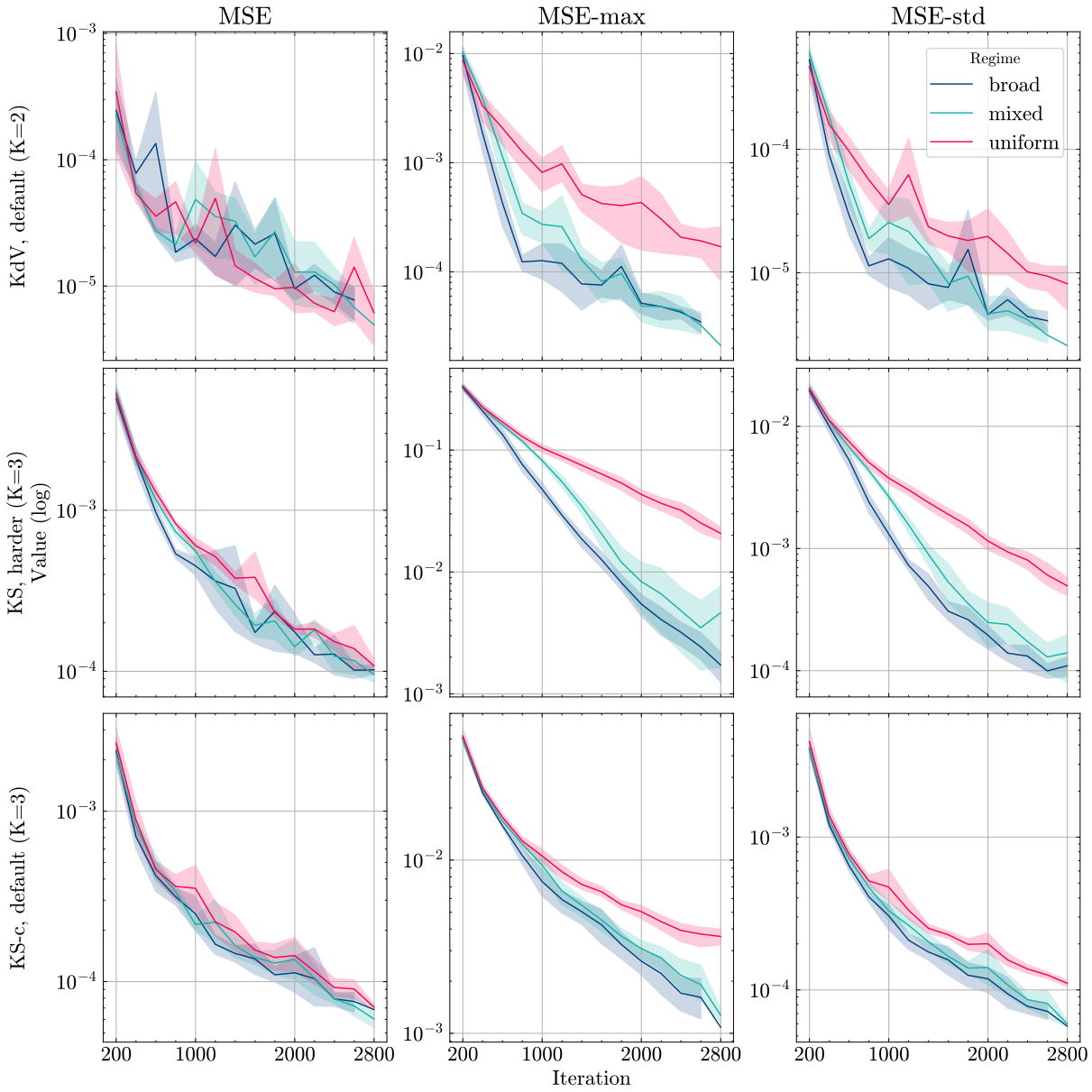


Figure 6.16: Evolution of test metrics (columns, y-axis, log scale) with statistical significance assessment for a subset of PDE cases (rows) and methods (colours are provided in the legend) across training iterations. Solid lines show mean values, shaded regions indicate 95% confidence intervals over 5 seeded runs. We show the range without first 200 training iterations to focus on the convergence phase.

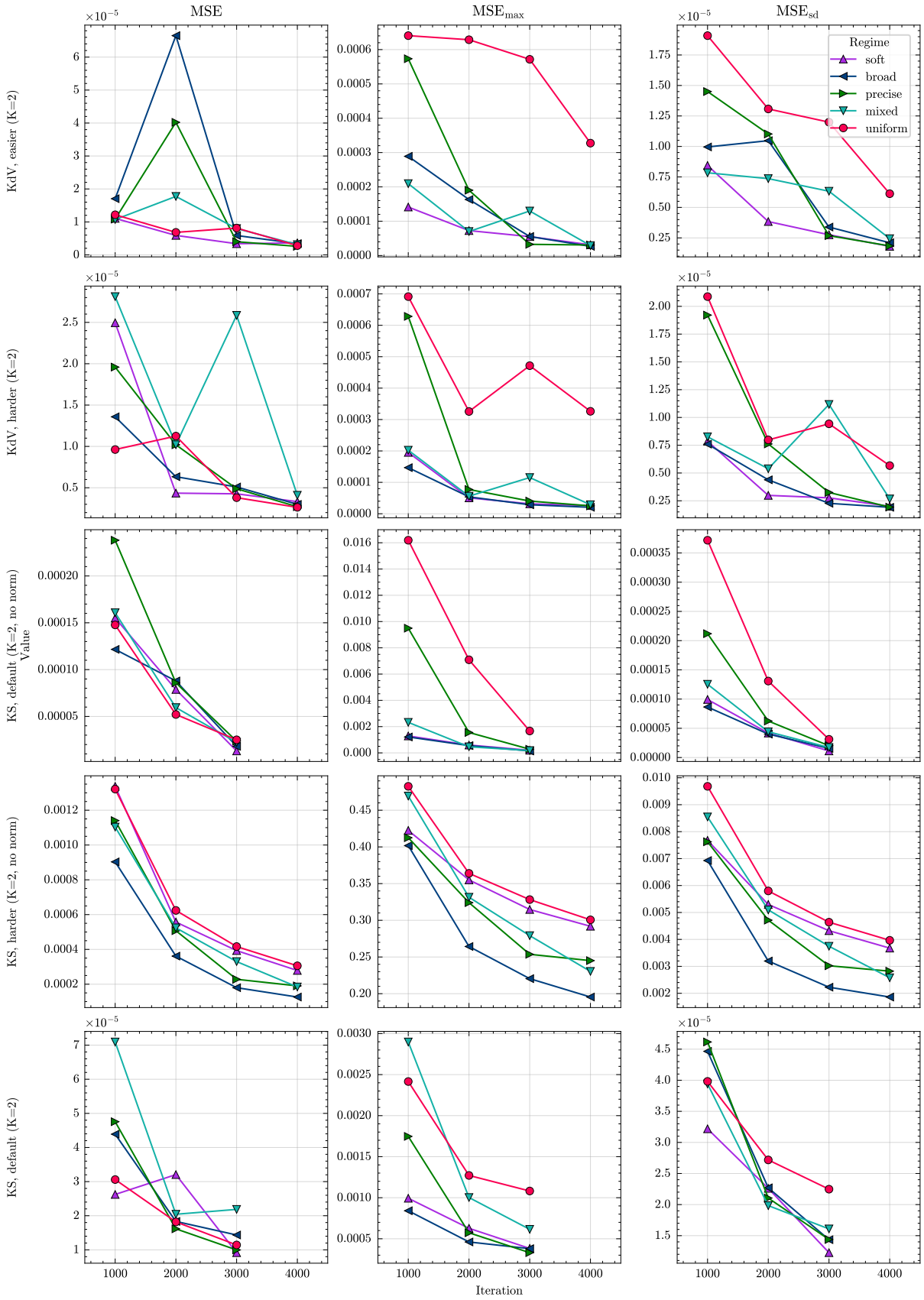


Figure 6.17: Evolution of the test metrics across (columns, y-axis) training iterations for different methods (coloured markers are provided in the legend). Every row corresponds to a different PDE case indicated on the left. Calculated using the checkpoints available for testing, which are every 1000 iterations (x-axis).

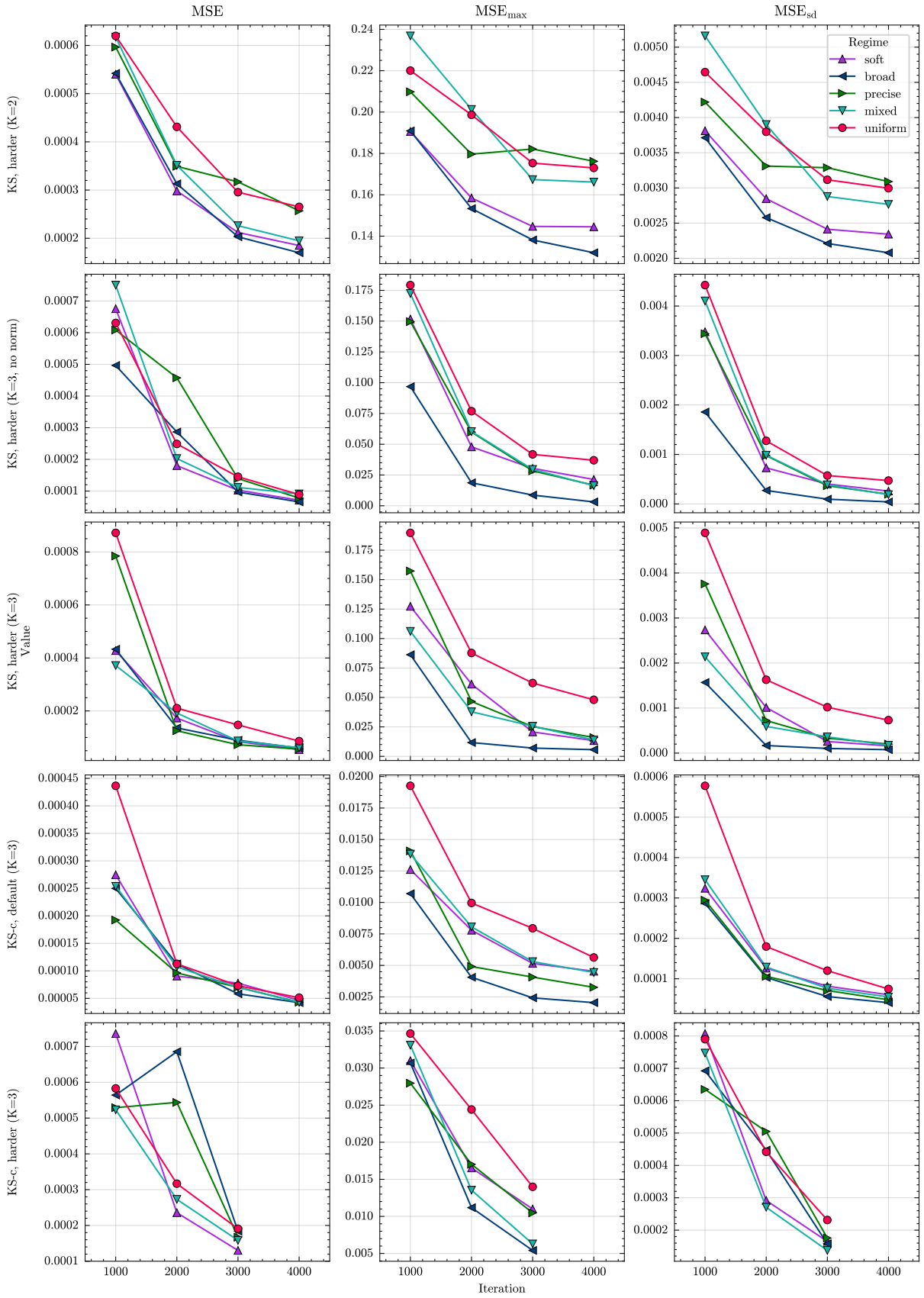


Figure 6.18: Continued: evolution of the test metrics (columns, y-axis) across training iterations for different methods (coloured markers are provided in the legend). Every row corresponds to a different PDE case indicated on the left. Calculated using the checkpoints available for testing, which are every 1000 iterations (x-axis).

6.4.4.3 Regions of interest in the parameter space

Apart from the quantitative metrics, we investigate what regions of the parameter space the method focuses on and how it correlates with the surrogate's performance across the parameter space.

Sampling behaviour. We provide the visualisation of the sampling distribution across resampling iterations for default KS with $K = 2$ and normalisation and two Breed regimes, namely broad and presise, in Figures 6.19 and 6.20, respectively. For both regimes, most samples concentrate in regions with higher amplitude values, especially the first component, which correspond to more complex dynamics and higher losses, while phase parameters exhibit no strong sampling pattern.

The difference in sampling behaviour between the broad and precise regimes is very noticeable. The precise regime tends to have clusters of samples in certain regions of the parameter space, especially at the later resampling iterations. It also exhibits "holes" in the sampling distribution, i.e., regions of the parameter space that are not sampled at all by the Gaussian components.

Error distribution over IC parameters. We visualise the test error distribution over the amplitude parameters for default KdV, comparing Breed regimes to the uniform baseline in Figure 6.14.

Breed regimes show more uniform distributions of losses, achieving reduced loss in the regions of higher amplitude values, compared to the uniform baseline. However, the values near zero amplitude values, which correspond to simple dynamics, have higher losses compared to the uniform baseline, especially broad and mixed regimes. This observation is consistent with the previous findings about the shift of the loss distribution towards the centre and increase of the minimum error by Breed.

6.4.5 Extended Experiments

The primary limitation of the experiments is insufficient training convergence observed in many cases, which we attribute to the limited number of iterations performed. This stems from hardware constraints of the GRICAD HPC cluster used for the experiments: low CPU-GPU bandwidth (PCIe 3.0, ∼16 GB/s), limited GPU-GPU bandwidth (PCIe, ∼32 GB/s), slow inter-node network (Omni-Path, 100 Gb/s), and modest compute capability (NVIDIA V100, 32 GB). Together, these constraints prevented reaching adequate convergence within the available computational budget of 1 hour per run. Additionally, surrogate test evaluation was performed post-hoc on a sparse set of checkpoints, since evaluating on the large test dataset requires pausing the online training process for several minutes. This limits how frequently the surrogate can be assessed during training, reducing the resolution at which test loss trends can be observed.

To address these limitations, we gained access to the Leonardo supercomputer at CINECA (ranked 10th in the TOP500), providing significantly higher bandwidth (NVLink 600 GB/s GPU-GPU, PCIe 4.0 ∼32 GB/s CPU-GPU, InfiniBand HDR 400 Gb/s inter-node) and more powerful compute (NVIDIA A100, 64 GB), allowing faster and more stable training. Additionally, we developed an external validation pipeline that performs evaluation in parallel with training, enabling more frequent surrogate assessment without interfering with the training process. We conducted additional experiments to validate

the main results and investigate the method's performance in greater depth with these improvements.

Setup. We selected a representative subset of experiments for reproduction under the improved setup: KdV harder, KS default with $K = 3$ and without normalisation, and KS-c default with $K = 3$. Training was extended to 40,000–60,000 iterations with 9 independent runs per configuration (3 different random seeds, each repeated 3 times to account for online training stochasticity). We used the same model architecture with adjusted learning rate and batch size. The comparison focused on uniform sampling versus Breed with parameters $\varepsilon^0 = 1.0$, $\varepsilon^c = 0.5$, $c = 5$, $N_p = 1000$, $p = 500$, and other settings matching the broad regime configuration.

The unusually high value of $\varepsilon^{(0)} = 1.0$ is motivated by an observed discrepancy between the expected and actual ε-schedule during early training. Before the first resampling iteration, clients use parameters sampled from the uniform distribution, and the Buffer is populated with training samples generated from these uniformly sampled parameters. After the first several resampling iterations, the Buffer still contains predominantly samples generated from the uniform distribution, as eviction is performed randomly without differentiating between samples from uniform or concentrated distributions. This results in a lower ratio of concentrated samples in the Buffer than intended by the ε-schedule. Setting $\varepsilon^{(0)}$ to a higher value compensates for this effect by increasing the proportion of concentrated samples during early training, which improves initial surrogate convergence and overall method performance.

Results. The results confirm our previous findings: as shown in Figure 6.21, Breed achieves lower variance and maximum loss values compared to uniform sampling across all tested cases. Furthermore, these extended experiments reveal that Breed also achieves lower mean loss values due to improved surrogate convergence enabled by the longer and more stable training process. The 10–90th percentile intervals demonstrate that Breed maintains more consistent performance throughout training.

The seeded-repetition experimental design (3 seeds, each repeated 3 times) proved particularly valuable for assessing method robustness under online training conditions. Figure 6.22 shows results for each individual run, revealing that even runs with identical seeds can exhibit variability due to training stochasticity, with occasional outliers. Notably, Breed demonstrates not only greater consistency across repetitions but also superior performance even in its worst-case runs. In contrast, uniform sampling demonstrates higher variability and worse outlier behaviour, highlighting Breed's robustness to the inherent randomness of online training.

6.4.6 Discussion and Limitations

Our results show that Breed can substantially reduce worst-case error and dispersion across PDE cases, often without degrading mean error and sometimes even improving it. At the same time, several limitations of the current formulation and evaluation setup constrain its applicability and point to directions for future work.

A first limitation concerns scalability with respect to the parameter space. Breed models the proposal distribution with Gaussian mixtures, whose complexity and expressivity become problematic as dimensionality grows. In this work we focused on one-dimensional PDEs at moderate resolution, with a relatively simple surrogate and a low-dimensional

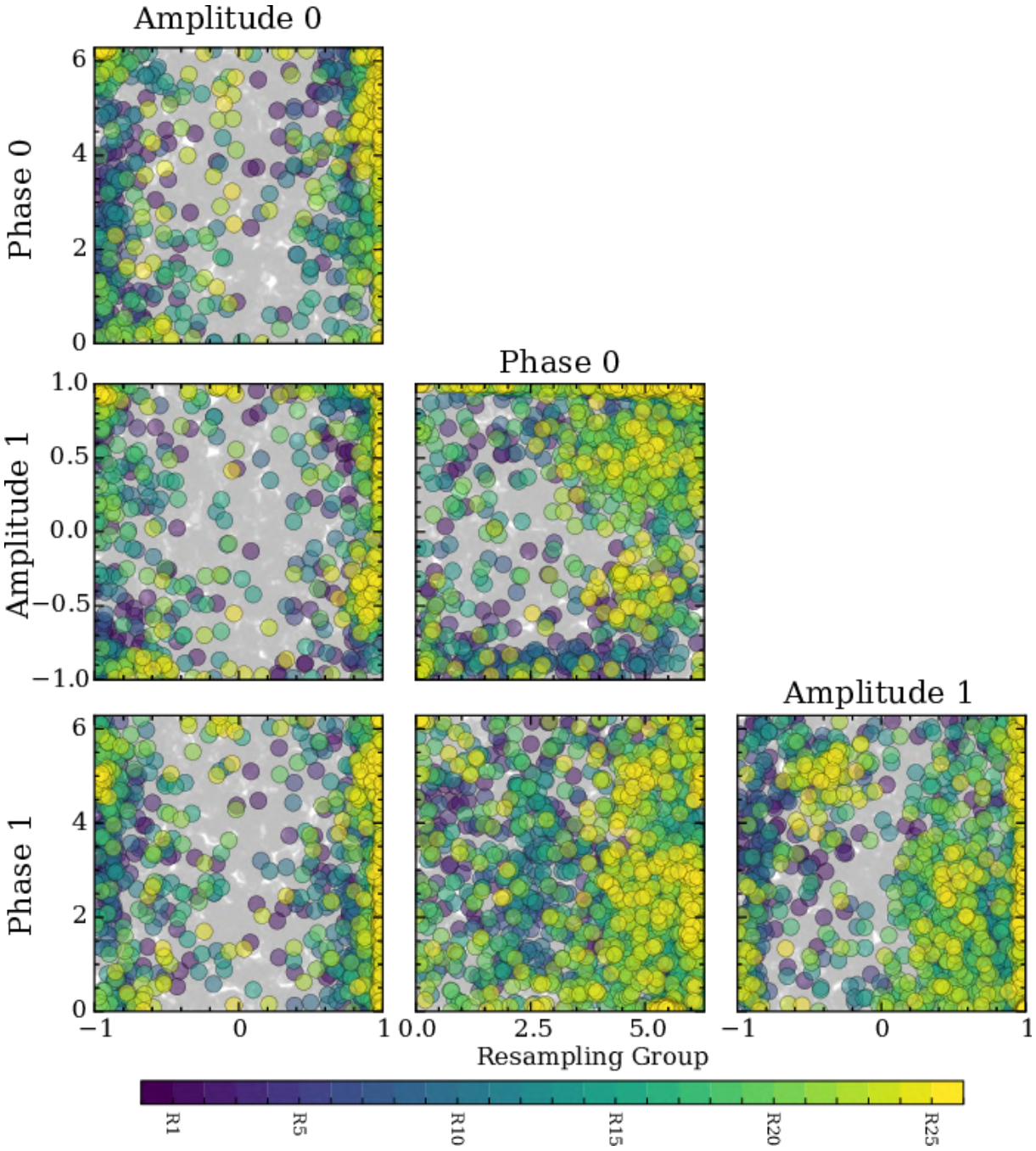


Figure 6.19: Pair plot of sampled IC parameters, i.e., amplitudes and phase shifts, across resampling iterations (colour) for KS default with $K = 2$ using broad regime.

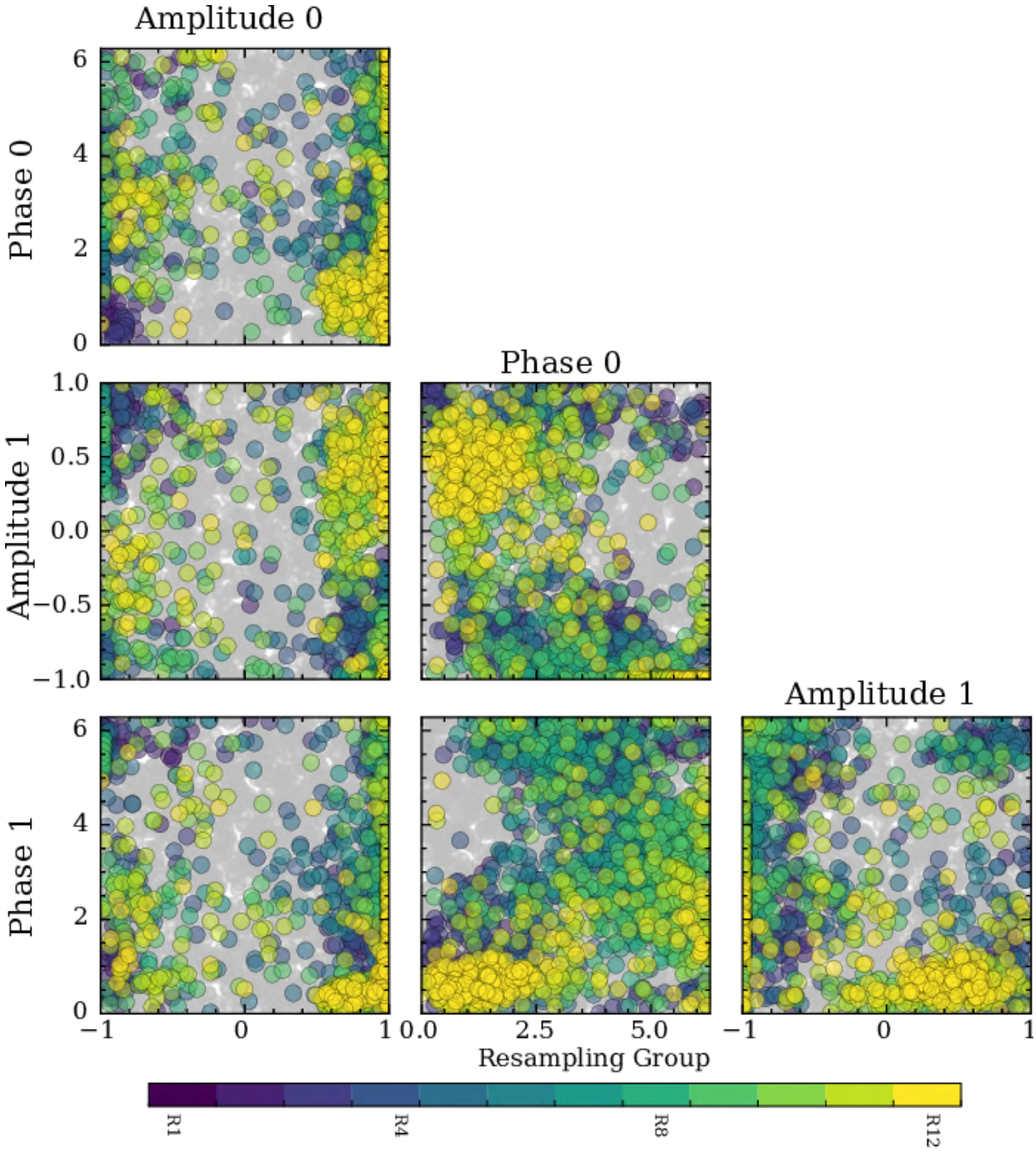


Figure 6.20: Pair plot of sampled IC parameters, i.e., amplitudes and phase shifts, across resampling iterations (colour) for KS default with $K = 2$ using precise regime.

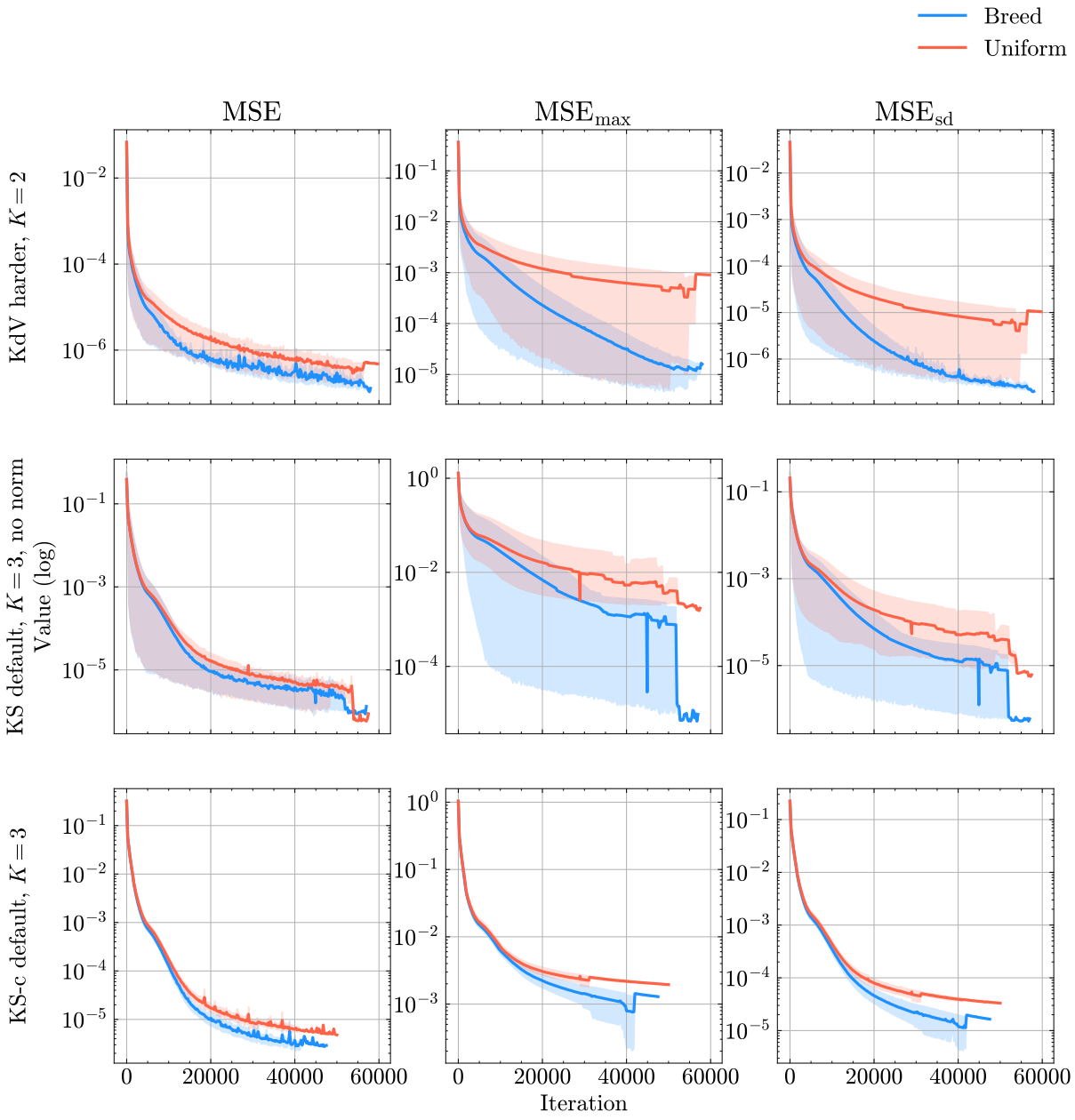


Figure 6.21: **Additional experiments:** evolution of test metrics (columns, y-axis, log scale) for Breed (blue) and uniform (orange) sampling across three PDE cases (rows). Solid lines show mean values, shaded regions indicate 10–90th percentile intervals over 9 repeated runs.

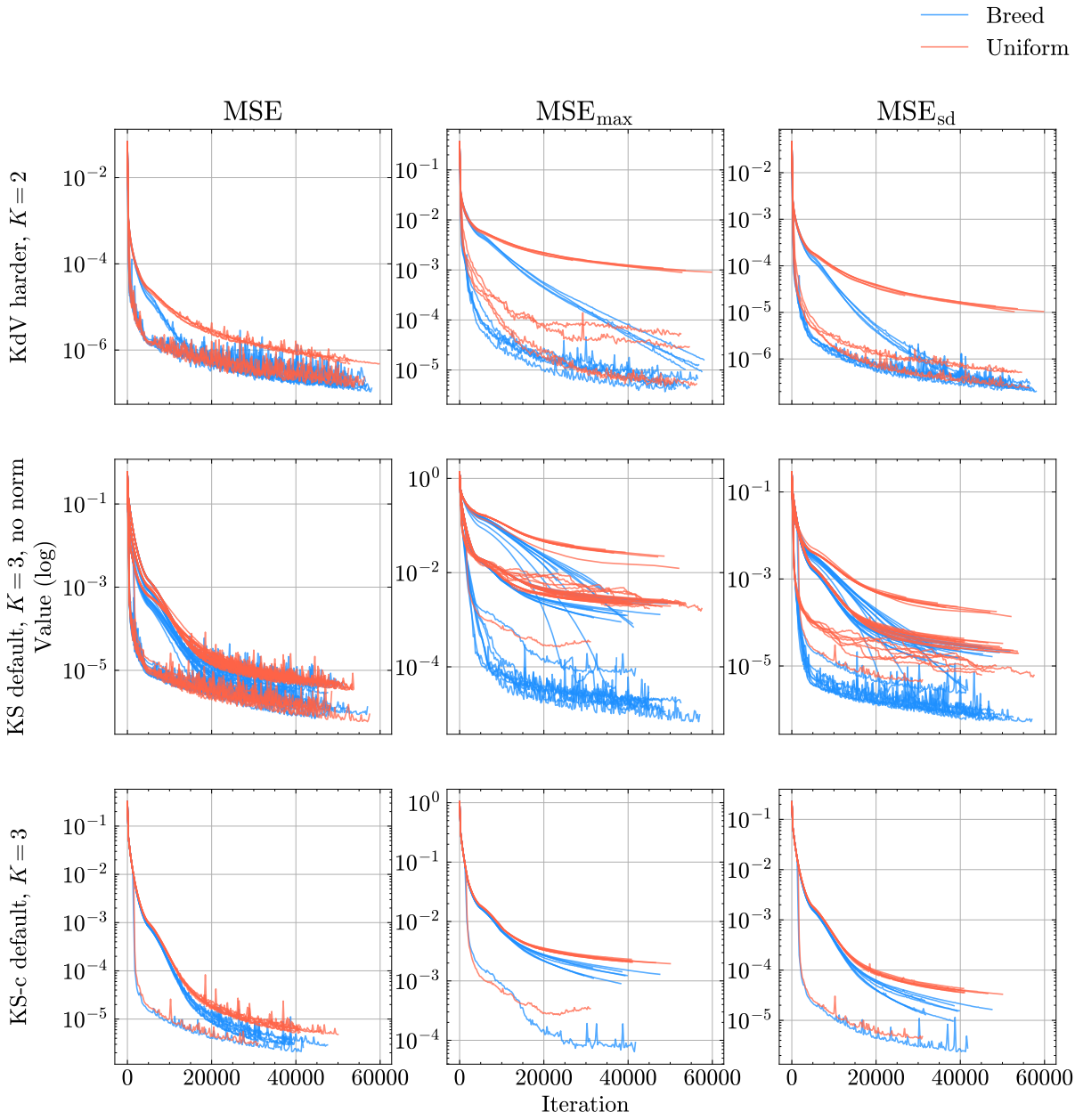


Figure 6.22: **Additional experiments:** evolution of test metrics (columns, y-axis, log scale) for Breed (blue) and uniform (orange) sampling across three PDE cases (rows). Solid lines show values for each of the 9 repeated runs.

parametrisation of initial conditions. Moving to two-dimensional PDEs, conditional surrogates with varying physical coefficients, or mixed continuous-discrete parameter spaces would dramatically increase the number and heterogeneity of parameters, and it is unclear whether the current mixture-based proposal can capture the resulting complex, multimodal structures in a robust way.

Second, our analysis highlights the need for more careful evaluation metrics and validation sets. Because the loss distributions are highly skewed, average errors are not sufficient to assess surrogate quality. Moreover, the unnormalised loss biases both evaluation and the Breed acquisition towards samples with large state magnitudes: high-amplitude configurations naturally induce larger absolute errors, which Breed may prioritise even when the underlying dynamics are not intrinsically harder, helping to explain the increased loss near zero amplitudes. Using normalised error metrics for both training and evaluation would mitigate this effect. In addition, our uniformly sampled validation sets inherit the same skew towards easier configurations. It would be informative to complement them with "hard" validation subsets that focus on challenging regions of the parameter space, in the spirit of our first contribution with IPGD benchmark (see Section 5.3.1), to better evaluate reliability under difficult conditions.

Third, the behaviour of Breed in terms of average error is not uniformly positive, even when variance and worst-case error clearly improve. Our extended experiments suggest that this is often due to insufficient convergence within the available training budget, and, for the hardest cases, to limited surrogate capacity to represent the dynamics at all. In such regimes, AL can reduce variability and extreme errors, and from the bias-variance decomposition (see Eq. (1.7)) it is reducing the variance of the errors, but the bias remains high, leading to worse average error. Similarly, from the perspective of uncertainty decomposition, AL reduces part of epistemic uncertainty related to lack of knowledge about the data, which can be reduced by sampling more and better data, but residual epistemic uncertainty due to model architecture or training procedure remains and must be addressed by improving the surrogate itself.

Finally, several methodology choices were kept intentionally simple and could be investigated in future work. We used a fixed parametric representation of initial conditions, but alternative representations, such as learned latent spaces, may better capture structure in complex dynamics. Similarly, we considered a single acquisition function and a particular temporal aggregation of trajectory losses. Exploring alternative criteria, for example variance-based scores or different aggregation schemes, could yield more informative sampling. Extending these ideas to surrogates trained over families of geometries, where AL chooses simulation geometries, remains an open and practically relevant challenge.

Chapter 7

Generative-based Active Learning

As discussed in Section 5.2.2, the Breed method can be viewed as an importance sampling (IS) method for a non-stationary target distribution defined by a chosen acquisition function. However, IS methods, being direct approaches to generative modelling, are known to struggle in high-dimensional settings with complex, multimodal target distributions [107]. This mirrors the scalability and expressivity limitations we observed for adaptive Gaussian mixtures in the previous chapter.

In contrast, modern generative neural models have demonstrated stronger performance on such distributions, particularly in challenging computer vision tasks with natural images and videos, where capturing rich multimodal structure is essential [107, 33, 73]. Their high expressive power and flexible architectural design enable both efficient sampling and accurate density estimation, making them attractive candidates for modelling complex distributions in AL for PDE surrogates. Diffusion-based approaches, normalising flows, and related architectures provide complementary trade-offs between expressivity, stability, and computational cost.

Given these considerations, a natural direction for advancing AL for surrogates is to replace adaptive Gaussian mixtures with generative neural models. In our context, the dimensionality of solver parameters is typically much smaller than in vision tasks, which allows us to use compact generative models. These can adapt rapidly to evolving surrogate performance while remaining compatible with online training constraints and not competing excessively for computational resources, in line with pillars I and III from Section 6.1.1. Moreover, NNs offer additional flexibility, for example, to incorporate heterogeneous parameter types or to condition on auxiliary physical information.

These ideas motivated the master internship project of Pierre Cesar, resulting in a paper submitted to ICML on January 28, 2026, entitled "Learning Where to Simulate: Generative Active Sampling for Online PDE Surrogate Training" and co-authored with Pierre Cesar, Abhishek Purandare, and Bruno Raffin. As the paper and the manuscript were developed concurrently, this chapter includes the submitted version of the paper with adaptations. We provide the contributor roles following the CRediT taxonomy[1] below.

Contributor note. Conceptualisation, Methodology, Investigation: *Pierre Cesar* (realisation, subsequent concepts), *Sofya Dymchenko* (initial concept, further adaptation); Software, Validation: *Pierre Cesar* (main) and *Abhishek Purandare* (support) – novel contribution, *Sofya Dymchenko* – via prior contributions; Supervision: *Sofya Dymchenko, Bruno Raffin*; Visualisation: *Pierre Cesar* (main), *Abhishek Purandare* (support); Writing: *Pierre Cesar, Sofya Dymchenko, Bruno Raffin.*

[1] https://credit.niso.org

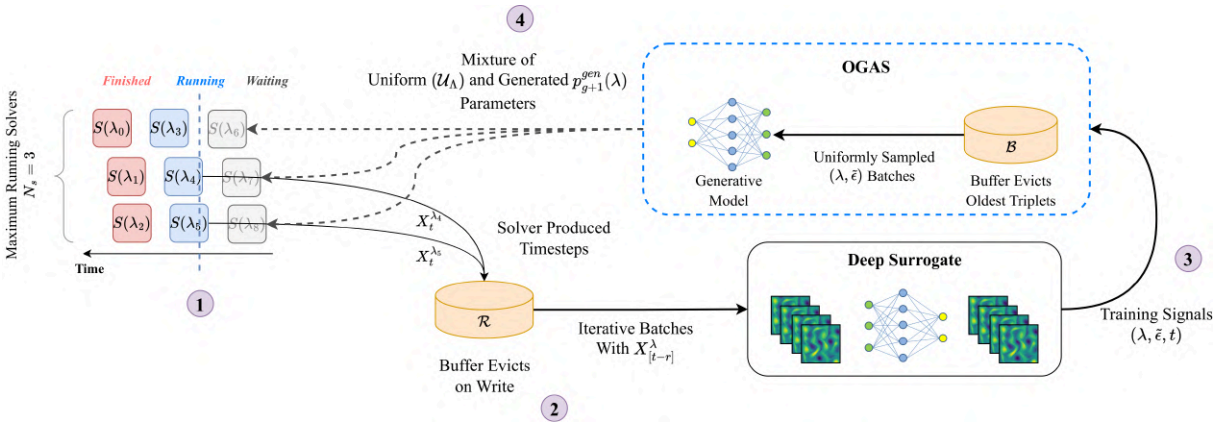


Figure 7.1: Online surrogate training workflow with OGAS (blue box). All components are running in parallel.

7.1 Online Generative Active Sampling

In this chapter, we introduce *Online Generative Active Sampling* (OGAS), a generative AL approach designed for online surrogate training.

As with Breed, OGAS operates within the Melissa framework and employs the improved active sampling mechanism described in Section 6.4.1: an AL method steers data generation by adapting the sampling distribution over solver parameters $\boldsymbol{\lambda} \in \Lambda$ and accordingly updating the subsequent solver instances periodically.

However, instead of relying on adaptive Gaussian mixtures, OGAS relies on a generative model, called the generator, to capture the evolving conditional distribution of the configuration parameters $\boldsymbol{\lambda}$ given training signals (e.g., loss or uncertainty estimates) computed for training batches. Similarly to Breed, the sampling distribution is a mixture of concentrated generator-guided and uniform samples to maintain global coverage of Λ.

To facilitate training of the generator, the Resampler component of Melissa is (1) run on a separate GPU, to speed up training and allow for more frequent updates, and (2) modified to maintain a buffer that contains training signals and corresponding solver parameters from recent surrogate updates, which are used to train the generator in parallel with the surrogate. Additionally, OGAS incorporates a correction mechanism into the generator training to mitigate bias towards previously sampled parameter regions – an effect that we formalise theoretically.

We evaluate two variants of OGAS (using two variants of training signals) across three challenging 2D PDEs with distinct dynamics (chaotic Kuramoto-Sivashinsky, turbulent Navier-Stokes Kolmogorov flow, unsteady Gray-Scott reaction–diffusion) in parameter spaces up to ≈ 300 dimensions, three surrogate architectures (UNet [248], FNO [127], scOT [75]), and five baselines including uniform sampling, pool-based AL (SBAL, Top-K [158]), and the online Breed. This experimental setup allows us to assess Breed in higher-dimensional, more realistic scenarios, and to compare it directly with OGAS as a generative alternative.

Compared to uniform sampling, OGAS consistently reduces 99th quantile errors by 28% and dispersion by 38% on average across all experiments, demonstrating improved robustness and performance uniformity of the trained surrogates, without significantly deteriorating average performance (-4%) and only $\approx 0.2\%$ training time overhead.

We detail specific to this work surrogate training formulation and notations[2] in Section 7.2 and present the OGAS method in Section 7.3. The experimental setup is described in Section 7.4, followed by results and discussion in Section 7.5. We conclude with limitations and future directions in Section 7.6.

7.2 Problem Setting

Parametrised PDE and numerical solver. Consider a parametrised PDE defined on a periodic bounded spatial domain $\mathcal{X} \subset \mathbb{R}^d$ over a time interval $[0, T]$:

$$\begin{aligned} \partial_t u &= \mathcal{N}[u](t, \boldsymbol{x}; \boldsymbol{\lambda}_{\mathrm{D}}), \quad (t, \boldsymbol{x}) \in [0, T] \times \mathcal{X}; \\ u(0, \boldsymbol{x}) &= u_0(\boldsymbol{x}; \boldsymbol{\lambda}_{\mathrm{IC}}), \quad \boldsymbol{x} \in \mathcal{X}; \end{aligned} \tag{7.1}$$

where $u(t, \boldsymbol{x}) \colon [0, T] \times \mathcal{X} \to \mathbb{R}^{N_{\mathrm{c}}}$ is the solution with N_{c} channels, $\mathcal{N}[\cdot]$ is a (possibly nonlinear) differential operator parametrised by physical coefficients $\boldsymbol{\lambda}_{\mathrm{D}}$, and $u_0(\cdot; \boldsymbol{\lambda}_{\mathrm{IC}})$ is a parametric initial condition (IC). We group physical and IC parameters into a single solver parameters vector $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_{\mathrm{D}}, \boldsymbol{\lambda}_{\mathrm{IC}}) \in \Lambda \subset \mathbb{R}^{N_{\mathrm{D}}+N_{\mathrm{IC}}}$.

For given $\boldsymbol{\lambda}$, a spatial grid of size N_{x}, and a timestep size Δ_{t}, a numerical solver S produces an approximate solution discretised in space and time in an autoregressive manner. Let $\boldsymbol{X}_t^{\boldsymbol{\lambda}} \in \mathbb{R}^{N_{\mathrm{x}} \times N_{\mathrm{c}}}$ denote the discrete solution state at timestep t. A sequence of N_{t} states over time, i.e., a trajectory, is produced by evaluating $u_0(\cdot; \boldsymbol{\lambda}_{\mathrm{IC}})$ at grid points to obtain $\boldsymbol{X}_0^{\boldsymbol{\lambda}}$ and then applying the solver for $t = 1, \dots, N_{\mathrm{t}}$:

$$\boldsymbol{X}_t^{\boldsymbol{\lambda}} = S(\boldsymbol{X}_{t-1}^{\boldsymbol{\lambda}}, \boldsymbol{\lambda}_{\mathrm{D}}), \quad \text{where } \boldsymbol{X}_t^{\boldsymbol{\lambda}} \approx u(t \cdot \Delta_{\mathrm{t}}, \cdot). \tag{7.2}$$

Online surrogate training. We aim to approximate the solver S with a $\boldsymbol{\lambda}_{\mathrm{D}}$ conditioned autoregressive surrogate S_θ parametrised by $\boldsymbol{\theta}$. We adopt an r-step training strategy, following Musekamp et al., 2025 [158]. Let $\boldsymbol{X}_{[t-r]}^{\boldsymbol{\lambda}}$ denote a training sample consisting of states $\{\boldsymbol{X}_{t-r}^{\boldsymbol{\lambda}}, \dots, \boldsymbol{X}_t^{\boldsymbol{\lambda}}\}$ and vector $\boldsymbol{\lambda}_{\mathrm{D}}$, $S_\theta^{(i)}$ be the i-th autoregressive application, then the i-th step MSE loss is:

$$\ell_i(\boldsymbol{X}_{[t-r]}^{\boldsymbol{\lambda}}; \boldsymbol{\theta}) = \frac{1}{N_{\mathrm{x}} N_{\mathrm{c}}} \left\| S_\theta^{(i)}(\boldsymbol{X}_{t-r}^{\boldsymbol{\lambda}}, \boldsymbol{\lambda}_{\mathrm{D}}) - \boldsymbol{X}_{t-r+i}^{\boldsymbol{\lambda}} \right\|_2^2. \tag{7.3}$$

For training, we use the average over r steps loss, i.e., $\ell(\boldsymbol{X}_{[t-r]}^{\boldsymbol{\lambda}}; \boldsymbol{\theta}) = \frac{1}{r} \sum_{i=1}^{r} \ell_i(\boldsymbol{X}_{[t-r]}^{\boldsymbol{\lambda}}; \boldsymbol{\theta})$, and aim to minimise expectation of $\ell(\boldsymbol{X}_{[t-r]}^{\boldsymbol{\lambda}}; \boldsymbol{\theta})$ over $(\boldsymbol{\lambda}, t)$.

Training is performed online within the Melissa framework, using a Reservoir buffer $\mathcal{R}$ described in Section 3.3.2, which ensures near-uniform distribution of timesteps in training batches. We illustrate the workflow in Figure 7.1.

7.3 Methodology

At the core of OGAS is the generator p_ϕ with weights $\boldsymbol{\phi}$, a denoising diffusion probabilistic model (DDPM) [78] equipped with classifier-free guidance [77]. It is trained asynchronously alongside the surrogate to capture the evolving distribution of $\boldsymbol{\lambda}$ conditioned on a training signal that quantifies trajectory informativeness. The signal is computed

[2] The notations used in this work slightly differ from the ones that are used throughout the thesis.

Algorithm 5 OGAS: Online Generative Active Sampling

Require: uniform-mixture $\alpha \in (0,1)$; admissible signal set $\mathcal{A} = [\tilde{\varepsilon}_{\min}, \tilde{\varepsilon}_{\max}]$; proposal $k(\tilde{\varepsilon}) \propto \tilde{\varepsilon}\,\mathbf{1}_{\mathcal{A}}(\tilde{\varepsilon})$; EMA decay γ; generator updates per cycle K; weight clip $[w_l, w_r]$.

Processes: generator $p_\phi(\lambda \mid \tilde{\varepsilon})$; discriminator $D_\psi(\lambda) \in (0,1)$; buffer $\mathcal{B}$ storing $(\lambda, \tilde{\varepsilon})$; $\mu \leftarrow 0, \sigma \leftarrow 1$.

while training **do**
  (1) Log standardized signals:
  Receive batch $\{(\lambda_i, \varepsilon_i)\}_{i=1}^n$.
  Compute minibatch $(\mu_{\text{mb}}, \sigma_{\text{mb}})$ of $\{\varepsilon_i\}$ and update $\mu \leftarrow \gamma\mu + (1-\gamma)\mu_{\text{mb}}, \quad \sigma \leftarrow \gamma\sigma + (1-\gamma)\sigma_{\text{mb}}$.
  Standardize $\tilde{\varepsilon}_i \leftarrow (\varepsilon_i - \mu)/\sigma$.
  Push $\{(\lambda_i, \tilde{\varepsilon}_i)\}_{i=1}^n$ to $\mathcal{B}$.
  (2) Train the generator:
  for $s = 1, \ldots, K$ **do**
    Sample $\{(\lambda_j, \tilde{\varepsilon}_j)\}_{j=1}^m \sim \mathcal{B}$. ▷ $y = 1$ *(buffer) class*
    Sample $\{\bar{\lambda}_j\}_{j=1}^m \sim \mathcal{U}_\Lambda$. ▷ $y = 0$ *(uniform) class*
    Update D_ψ on $\{(\bar{\lambda}_j, 0)\}_{j=1}^m \cup \{(\lambda_j, 1)\}_{j=1}^m$, where $D_\psi(\lambda) = P_\psi(y = 0 \mid \lambda)$.
    Set $\widehat{w}_{\mathcal{B}}(\lambda_j) \leftarrow \dfrac{D_\psi(\lambda_j)}{1 - D_\psi(\lambda_j)}$ and clamp to $[w_l, w_r]$.
    Update ϕ with weighted loss Eq. (7.14).
  (3) If resampling request:
  for all new solver run **do**
    if $\xi \sim \mathcal{U}(0,1) < \alpha$ **then**
      $\lambda \sim \mathcal{U}_\Lambda$
    else
      $\tilde{\varepsilon} \sim k(\tilde{\varepsilon})$, then $\lambda \sim p_\phi(\lambda \mid \tilde{\varepsilon})$

with low overhead during surrogate updates and streamed to the generator. We apply corrective updates to the generator training to mitigate a bias induced by historically sampled regions of Λ. Subsequent solver instances are parametrised by sampling from a mixture distribution consisting of a uniform component, ensuring global coverage of Λ, and the generator-guided component, targeting challenging regimes. In Algorithm 5, we summarise the full OGAS procedure, and we detail its components in the following.

7.3.1 Training Signal

For each sample $\boldsymbol{X}^\lambda_{[t-r]}$ in a training batch, we compute a *training signal* with a function:

$$\varepsilon = A(\boldsymbol{X}^\lambda_{[t-r]}; \boldsymbol{\theta}) \in \mathbb{R}_+, \tag{7.4}$$

where A is designed to quantify the informativeness of the sample for the surrogate at current parameters $\boldsymbol{\theta}$. We consider two variants of A: *loss-based* and *uncertainty-based*. The first is a last-step MSE loss divided by the L^2 norm of the target state:

$$A_\ell(\boldsymbol{X}^\lambda_{[t-r]}; \boldsymbol{\theta}) = \frac{\ell_r(\boldsymbol{X}^\lambda_{[t-r]}; \boldsymbol{\theta})}{\|\boldsymbol{X}^\lambda_t\|_2 + \delta}, \tag{7.5}$$

where $\delta > 0$ is added to avoid division by near-zero values. The second is based on an estimated uncertainty using an ensemble of M identical models with independently initialised and trained weights [194]. Let $\{\boldsymbol{\theta}_m\}_{m=1}^{M}$ denote the parameters of the M surrogates of the ensemble. Then, A is defined as the total variance of last-step predictions across the ensemble, averaged over all channels and spatial locations:

$$A_u(\boldsymbol{X}^{\lambda}_{[t-r]}; \boldsymbol{\theta}) = \mathrm{Var}_m \left[S^{(r)}_{\theta_m}(\boldsymbol{X}^{\lambda}_{t-r}, \boldsymbol{\lambda}_{\mathrm{D}}) \right]. \tag{7.6}$$

To obtain training signals with a consistent scale throughout online training, we z-standardise the raw values using running batch statistics. For each surrogate update, we compute the batch mean and standard deviation (μ, σ) of the raw signals ε, update their running estimates, and define the normalised training signal as $\tilde{\varepsilon} = (\varepsilon - \mu)/\sigma$. We log pairs $(\boldsymbol{\lambda}, \tilde{\varepsilon})$ in a dedicated buffer $\mathcal{B}$, which the generator uses for training by uniformly sampling batches from it. The signal computation and transmission are asynchronous with surrogate training to avoid overhead.

The buffer $\mathcal{B}$ has a fixed capacity and stores only recent pairs – incoming pairs overwrite the oldest ones (FIFO strategy). Because the surrogate evolves throughout training, older signals can become irrelevant; restricting $\mathcal{B}$ to recent samples reduces staleness and improves the match between the generator's conditioning variable and the current loss landscape.

7.3.2 Signal-conditional Generator

The generator is an MLP-based conditional generative DDPM that takes the training signal $\tilde{\varepsilon}$ as input condition and outputs a distribution over $\boldsymbol{\lambda}$. The model approximates a conditional distribution induced by the buffer $\mathcal{B}$:

$$p_\phi(\boldsymbol{\lambda} \,|\, \tilde{\varepsilon}) \approx p_{\mathcal{B}}(\boldsymbol{\lambda} \,|\, \tilde{\varepsilon}) \tag{7.7}$$

where $\boldsymbol{\phi}$ are the generator parameters. The generator is trained by minimizing the negative log-likelihood of the training pairs $(\boldsymbol{\lambda}, \tilde{\varepsilon})$ sampled from $\mathcal{B}$:

$$\mathbb{E}_{(\boldsymbol{\lambda}, \tilde{\varepsilon}) \sim p_{\mathcal{B}}} \Big[-\log p_\phi(\boldsymbol{\lambda} \mid \tilde{\varepsilon}) \Big]. \tag{7.8}$$

Although $\tilde{\varepsilon}$ is computed for a training sample associated with a specific timestep t, we cannot generate simulation data at arbitrary timesteps. We therefore want the generator to capture the typical difficulty of a trajectory as a whole. We treat t as a latent variable and learn the conditional distribution induced by the timestep-marginal joint law $p_{\mathcal{B}}(\boldsymbol{\lambda}, \tilde{\varepsilon})$ arising from sampling uniformly from $\mathcal{B}$. This marginalisation is supported by the near-uniform distribution of timesteps in training batches, ensured by the Reservoir $\mathcal{R}$.

The DDPM is intentionally faster than the surrogate, enabling (1) rapid adaptation of the sampling distribution to the surrogate's needs, as its loss landscape evolves, and (2) fast inference and frequent updates of the upcoming simulation parameters, avoiding the generator becoming a bottleneck. However, we limit the number of generator updates per surrogate update to prevent overfitting to the buffer $\mathcal{B}$. Specifically, when a new batch of training pairs $(\boldsymbol{\lambda}, \tilde{\varepsilon})$ arrives, i.e., each surrogate update, we run a bounded number $K_{\max}$ of training steps and otherwise idle. Generator updates never block the surrogate training. We found $K_{\max} = 50$ to be sufficient in our experiments.

7.3.3 Parameters Sampling Distribution

OGAS is periodically called to sample new parameters for upcoming solver runs. The gth *generation* of parameters is obtained by sampling from a mixture distribution p_g that combines a uniform distribution $\mathcal{U}_\Lambda$ over the parameter space Λ and the generator-guided distribution p_g^{gen}:

$$p_g(\boldsymbol{\lambda}) = \alpha\, \mathcal{U}_\Lambda + (1-\alpha)\, p_g^{\text{gen}}(\boldsymbol{\lambda}), \tag{7.9}$$

where $\alpha \in [0,1]$ is a user-defined mixture coefficient[3]. The generator-guided component p_g^{gen} is formed by (1) sampling a training signal from a user-defined proposal distribution $k(\tilde{\varepsilon})$ and (2) using it to draw from the current generator conditional p_{ϕ_g}. The resulting distribution is the marginal of the generator conditional over the proposal distribution:

$$p_g^{\text{gen}}(\boldsymbol{\lambda}) = \int k(\tilde{\varepsilon})\, p_{\phi_g}(\boldsymbol{\lambda} \mid \tilde{\varepsilon})\, d\tilde{\varepsilon}. \tag{7.10}$$

We design $k(\tilde{\varepsilon})$ to prioritise sampling parameters associated with challenging training signals, which are expected to be more informative for the surrogate's learning progress. Since $\tilde{\varepsilon}$ is z-normalised, it has a mean of zero and a standard deviation of one: average difficulty corresponds to $\tilde{\varepsilon} \approx 0$, while higher values indicate more challenging samples. Then we set k to be proportional to $\tilde{\varepsilon}$ within an admissible range $\mathcal{A} = [0, \tilde{\varepsilon}_{\max}]$:

$$k(\tilde{\varepsilon}) \propto \tilde{\varepsilon}\, \mathbf{1}_{\mathcal{A}}(\tilde{\varepsilon}). \tag{7.11}$$

We choose $\tilde{\varepsilon}_{\max} = 5$ to prioritise challenging cases while avoiding extreme outliers.

7.3.4 History-bias Correction

We formalise the bias induced by sampling from a finite buffer $\mathcal{B}$ and propose a correction method based on importance weighting.

History bias. From Equations (7.7) and (7.10), and applying Bayes' rule to the buffer conditional $p_{\mathcal{B}}(\boldsymbol{\lambda} \mid \tilde{\varepsilon})$, the generator-guided component of the sampling distribution at generation $g+1$ is:

$$p_{g+1}^{\text{gen}}(\boldsymbol{\lambda}) \propto p_{\mathcal{B}}(\boldsymbol{\lambda}) \int k(\tilde{\varepsilon})\, \frac{p_{\mathcal{B}}(\tilde{\varepsilon} \mid \boldsymbol{\lambda})}{p_{\mathcal{B}}(\tilde{\varepsilon})}\, d\tilde{\varepsilon}, \tag{7.12}$$

so the learned conditional inherits the generally non-uniform buffer prior from precedent generations $p_{\mathcal{B}}(\boldsymbol{\lambda})$. Consequently, reusing $p_\phi(\boldsymbol{\lambda} \mid \tilde{\varepsilon})$ to propose future parameters can induce a *history bias* that persistently favours historically over-sampled regions of Λ and potentially keeps the generator focused on regions that are no longer relevant as the surrogate evolves.

To debias the generator, we train it *under a uniform reference prior* by targeting the joint distribution:

$$p_{\mathcal{B}}^{\star}(\boldsymbol{\lambda}, \tilde{\varepsilon}) = \mathcal{U}_\Lambda(\boldsymbol{\lambda})\, p_{\mathcal{B}}(\tilde{\varepsilon} \mid \boldsymbol{\lambda}), \tag{7.13}$$

which enforces a uniform reference prior on Λ while preserving the (time-marginal) signal mechanism encoded by $p_{\mathcal{B}}(\tilde{\varepsilon} \mid \boldsymbol{\lambda})$.

[3] In principle, this is the concentrate-explore control value $1 - \varepsilon^{(i)}$ as in the original Breed algorithm, described in subsection 5.2.1, which similarly follows a warm-up schedule defined in Equation (5.7).

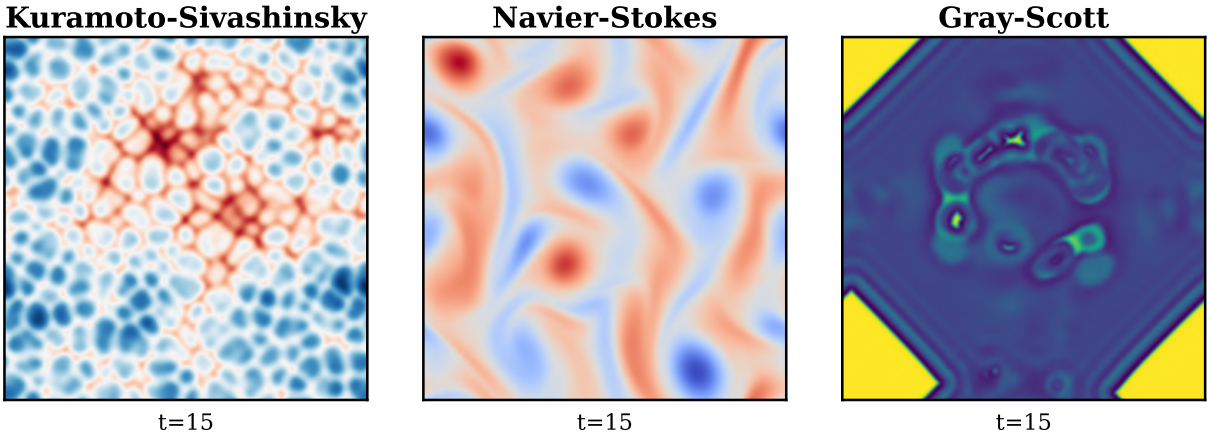


Figure 7.2: 2D PDE states at t=15 for, left to right, KS, NS, and GS. KS exhibits chaotic wave patterns, NS shows turbulent vortices, and GS displays complex reaction–diffusion shapes.

We minimise the corresponding importance-weighted log-likelihood. Using samples from $\mathcal{B}$, we implement this objective by importance weighting [24]:

$$\mathcal{L}_{\text{unif}}(\boldsymbol{\phi}) = \mathbb{E}_{(\boldsymbol{\lambda},\tilde{\varepsilon})\sim p_{\mathcal{B}}}[-w_{\mathcal{B}}(\boldsymbol{\lambda}) \log p_{\phi}(\boldsymbol{\lambda} \mid \tilde{\varepsilon})] \,, \tag{7.14}$$

with $w_{\mathcal{B}}(\boldsymbol{\lambda}) = \mathcal{U}_{\Lambda}(\boldsymbol{\lambda})/p_{\mathcal{B}}(\boldsymbol{\lambda})$ the corresponding density-ratio.

The weight $w_{\mathcal{B}}(\boldsymbol{\lambda})$ is not available in closed form since the buffer marginal $p_{\mathcal{B}}(\boldsymbol{\lambda})$ is induced by a history-dependent sampling policy. We therefore *estimate the density-ratio* via binary classification. We train a lightweight discriminator $D_{\psi}(\boldsymbol{\lambda}) \in (0,1)$ (two-layer MLP: $d \to 64 \to 1$) to distinguish samples from the uniform reference $\mathcal{U}_{\Lambda}$ (y=0) and from the buffer marginal $p_{\mathcal{B}}$ (y=1), i.e., $D_{\psi}(\boldsymbol{\lambda}) = P(y{=}0 \mid \boldsymbol{\lambda})$. The implied density-ratio estimator is

$$\widehat{w}_{\mathcal{B}}(\boldsymbol{\lambda}) = \frac{\pi_1}{\pi_0} \cdot \frac{D_{\psi}(\boldsymbol{\lambda})}{1 - D_{\psi}(\boldsymbol{\lambda})}, \tag{7.15}$$

where $\pi_c = P(y{=}c)$ are the class priors [65]. In practice, we use balanced classes ($\pi_0 = \pi_1$), so the multiplicative factor cancels. We clip $\widehat{w}_{\mathcal{B}}(\boldsymbol{\lambda})$ to $[w_l, w_r]$ for numerical stability.

A simpler alternative is to train the generator only on pairs originating from the uniform component of the sampling distribution. But we found it underperforms when α is small, as it significantly reduces the number of available training pairs.

7.4 Experimental Setup

7.4.1 Data Generation

2D PDEs. We use the pseudo-spectral solvers from APEBench [108] for three high-order nonlinear PDEs: Kuramoto-Sivashinsky (KS), Navier-Stokes Kolmogorov Flow (NS), and unsteady Gray-Scott-β (GS). The first two model chaotic and turbulent fluid dynamics, while the third captures continually evolving reaction–diffusion pattern formation in a chemical system ($N_c = 2$). We show representative states of each PDE at $t = 15$ in Figure 7.2 to illustrate the diversity of spatial patterns in the benchmark suite.

Each PDE is defined on a 2D periodic domain $\mathcal{X} = [0, L]^2$ discretised into $N_{\text{x}} = 256^2$ grid points. To ensure numerically stable and accurate trajectories, solvers run

at a fine time resolution Δt_{eff}, but we expose only one state every Δt timesteps for training or validation. We also discard the first N_{t_w} states to avoid initial transients. Training uses trajectories of $N_t = 32$ timesteps, while validation uses $N_t = 51$ timesteps to assess generalization beyond the training horizon. Specifically, we set $(\Delta t, \Delta t_{\text{eff}}, N_{t_w})$ to $(0.5, 0.1, 200)$ for KS, $(0.2, 0.002, 50)$ for NS, and $(30.0, 1.0, 50)$ for GS. These choices closely follow the established setup of Holzschuh et al., 2025 [82].

Solver input parameters. Each solver takes an input parameter $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_{\text{IC}}, \boldsymbol{\lambda}_{\text{D}})$ as described in Sec. 7.2. To capture diverse physical regimes, KS and GS domain sizes $\boldsymbol{\lambda}_{\text{D}} = L$ vary within the $[10.0, 130.0]$ and $[0.5, 10.0]$ intervals, respectively; NS viscosity coefficient $\boldsymbol{\lambda}_{\text{D}} = \nu$ is defined on $[0.003, 0.008]$, corresponding to Reynolds numbers $Re \in [125, 330]$.

We use two families of ICs, each designed to produce diverse spatial structures while remaining smooth enough to avoid numerical instabilities. For uniform baselines, the defined intervals are sampled uniformly, while for AL methods, the sampling is guided by the method's strategy. Out-of-bounds parameters are clipped to the valid intervals.

For the KS and NS equations, we sample ICs from a *truncated Fourier series* with up to $K = 8$ modes, with varied amplitudes and phases per mode, yielding up to $N_{\text{IC}} = 308$ parameters. For the GS system, we construct ICs by superimposing $B = 8$ anisotropic *Gaussian blobs* for the first channel, with varied locations, variances, and intensities, accounting for $N_{\text{IC}} = 48$ parameters. As GS models two interacting chemical species, the second channel IC is set as the complement of the first.

7.4.2 Compared Methods

We evaluate two variants of the OGAS method: *OGAS-L* uses the training signal A_ℓ from Eq. (7.5), while *OGAS-U* uses A_u from Eq. (7.6). We compare them against two standard baselines, namely *Uniform* and quasi-uniform sampling with *Sobol* sequences, two adapted pool-based AL methods with uncertainty estimates, which we present in detail below, and *Breed*, which was introduced in Section 6.2.3. To isolate the effect of the sampling strategy, i.e., generative model vs. Gaussian mixture, for *Breed* we use the same update period, mixture rate α, and loss-based training signal as in *OGAS-L*. Other hyperparameters match broad Breed regime as in Table 6.2.

Pool-based AL with uncertainty. We take two pool-based batch AL approaches, namely *SBAL* and *Top-K*, with ensemble-based uncertainty estimates from AL4PDE [158] and adapt them to the online setting. The online SBAL adaptation is detailed in Algorithm 6 and described below.

In offline settings, the pool-based AL proceeds in rounds: (1) train a surrogate for fixed number of epochs, (2) measure acquisition scores for each candidate solver input parameter in the pool $\mathcal{P}$, (3) select a batch of candidates based on the scores (and remove them from $\mathcal{P}$), (4) obtain solver trajectories for the selected candidates, and (5) add the new trajectories to the surrogate training set. The candidate pool $\mathcal{P}$ is initially populated with a large number of parameters sampled uniformly from Λ, and is updated by removing selected candidates at each round. The initial training set is selected uniformly from $\mathcal{P}$. Often, the surrogate is retrained from scratch at each round.

Musekamp et al., 2025 [158] employs uncertainty estimates as acquisition scores. Specifically, uncertainty is measured via ensemble output variance, similar to training

Algorithm 6 SBAL adaptation for online training.

Require: Pool size N_P, batch size K, rollout fraction $\rho \in (0, 1]$, sharpness $\beta \geq 0$, ensemble $\{f_\theta^{(m)}\}_{m=1}^{M}$

Construct candidate pool: Sample $\{\lambda_i\}_{i=1}^{N_P} \sim \mathcal{U}_\Lambda$ uniformly
for $i = 1, \ldots, N_P$ **do**
  Sample initial condition $u_0 \sim p(\cdot \mid \lambda_i)$
  Set rollout horizon $T' \leftarrow \max(1, \lfloor \rho T \rfloor)$
  for $t = 0, \ldots, T' - 1$ **do**
    Compute ensemble predictions $\{\hat{u}_{t+1}^{(m)}\}_{m=1}^{M}$ from u_t
    Record disagreement $d_t \leftarrow \mathrm{Var}_m\left(\hat{u}_{t+1}^{(m)}\right)$ or other metric
    Update state $u_{t+1} \leftarrow \bar{u}_{t+1}$ (ensemble mean)
  Aggregate uncertainty: $u_i \leftarrow \frac{1}{T'} \sum_{t=0}^{T'-1} d_t$
Compute selection probabilities: $q_i \leftarrow \dfrac{u_i^\beta}{\sum_{j=1}^{N_P} u_j^\beta}$ for $i = 1, \ldots, N_P$
Sample batch: Draw K candidates $\{\lambda_{s_1}, \ldots, \lambda_{s_K}\}$ from $\{q_i\}$ without replacement
Return: Selected parameters $\{\lambda_{s_k}\}_{k=1}^{K}$ to simulate

signal in Equation (7.6), but computed over the full trajectory rollout instead of the last timestep. To select candidates, a batch is drawn without replacement from the pool using a softened sampling distribution over uncertainties:

$$q_i = \frac{u_i^\beta}{\sum_j u_j^\beta}, \tag{7.16}$$

where u_i is the uncertainty score for candidate i [158, 106], and a sharpness parameter $\beta \geq 0$ controls the degree of exploitation of the uncertainty scores. Specifically, SBAL corresponds to $\beta = 2$, while Top-K corresponds to $\beta \to \infty$ (selecting K candidates with the highest uncertainties).

SBAL adaptation. We adapt the pool-based procedure to our online setting, where at each resampling trigger: (1) data generation and surrogate training are paused, (2) a candidate pool $\mathcal{P}$ is constructed by uniformly sampling parameters from Λ, (3) the AL selection is performed on the candidate pool, (4) the selected parameters update the upcoming solver runs, and (5) data generation and surrogate training are resumed. We trigger resampling every 1,000 completed simulations, construct a candidate pool of size 5,000, and select 1,000 candidates for the next simulations. The first resampling is triggered after an initial warm-up phase that allows the surrogate to stabilise.

In principle, it is equivalent to the active sampling mechanism described in Section 6.4.1, except that the data generation and training are blocked during the AL procedure. In the case of pool-based AL, where the AL selection is computationally expensive, the pause is necessary to avoid divergence between the surrogate state at the time of AL procedure and the surrogate state at the time the new data is available for training, which would make the AL selection less effective.

This adaptation differs from the original in [158] in three main aspects to balance quality of selection and computational efficiency: we use a smaller but always new candidate pool (5,000 vs. 10,000), we do not retrain the surrogate from scratch at each round, but rather continue training from the current surrogate state, and we measure the uncertainty over only a half of trajectory rollout ($\rho = 0.5$).

7.4.3 Surrogates Training and Validation

Training protocol. For each experiment, we train an ensemble of two surrogates ($M = 2$) that differ only by their weight initialisations. Both models receive the same training batches from $\mathcal{R}$ and compute 2-step MSE loss ($r = 2$). This ensemble is required to compute uncertainty estimates for the *OGAS-U*, *SBAL* and *Top-K* methods. An ensemble of two members has been shown to provide sufficient uncertainty estimates [158]. The loss-based AL methods (*OGAS-L*, *Breed*) are also evaluated from the same surrogate ensemble, with the training signals computed using the average loss of the two members. This enables to compare the different methods at equivalent surrogate training conditions. However, as the loss-based signal does not require an ensemble in principle, we additionally evaluate an *OGAS-L1* variant using only one surrogate, and complement the comparison with single-member validation errors for all other methods to ensure a consistent evaluation protocol.

Surrogate architectures. We adapt two conditional architectures from AL4PDE [158], namely modern UNet [248] and FNO [127], which were originally designed for 64×64 resolution in 2D. The UNet architecture has 4 times fewer hidden channels to compensate for the increased resolution, resulting in a total parameter count of 2.3M. The FNO model has 3 times more Fourier modes retained to capture the higher resolution, resulting in a total parameter count of 29.5M. Optimisation and hyperparameters follow AL4PDE.

We adapt the scOT-T architecture from Herde et al., 2024 [75], originally designed for 128×128 resolution and all-to-all temporal prediction, by doubling the patch size, tripling the embedding size, and replacing time-conditioning with physics-conditioning to accommodate our autoregressive task. This results in a total parameter count of 184.3M.

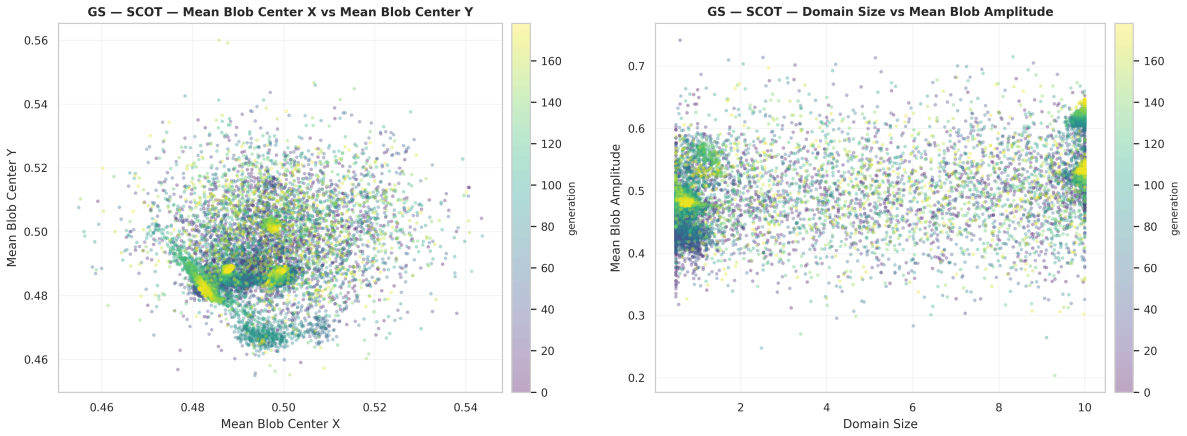
Online setup and speed limiter. Experiments use the Melissa framework with the active sampling mechanism. The simulation budget is $N = 10^4$ with up to $N_S = 56$ concurrent clients (2 CPU cores each), generating approximately 80GB of data on-the-fly per run per PDE channel. Each surrogate performs 18,000 gradient updates (≈3–4.5 hours on 2 A100 GPUs, one per ensemble member). OGAS runs its generative AL process on a dedicated A100 GPU, although the computational load of the generator training is relatively small, and it can run on a fraction[4] of the surrogate's GPU. Breed runs on a separate CPU core. All experiments are repeated over 3 random seeds.

In an asynchronous client–server setup, cluster variability (preemption, stragglers) can shift the relative speeds of data generation and training mid-run, making results irreproducible even with fixed seeds. To ensure comparability across runs, we standardise the training protocol by implementing a *speed limiter*. We control relative speeds with a view ratio: the average number of surrogate gradient updates performed per simulated trajectory. A speed limiter throttles either simulation intake or surrogate execution so that this ratio remains stable throughout training. We set a target of 1.8 updates per trajectory, consistent with available cluster throughput, yielding 18,000 surrogate updates for the 10^4-simulation budget.

Validation metrics. For validation, we use average ensemble predictions, i.e.,

$$S_\theta(\cdot) = \frac{1}{M} \sum_{m=1}^{M} S_{\theta_m}(\cdot), \tag{7.17}$$

[4]This is a feature that was not available on the machines we had access to for the experiments.



(a) Mean blobs centres coordinates.

(b) Domain size vs mean blobs amplitudes.

Figure 7.3: Evolution of the sampled parameter distribution with OGAS for unsteady Gray-Scott. The distribution progressively focuses on challenging parameter regions.

using the surrogates states after the final training step (≈18k). We then evaluate the performance using the normalised root-square 1-step MSE, given in Equation (7.3), i.e.:

$$\mathrm{RMSE}(\boldsymbol{\lambda}, t; \boldsymbol{\theta}) = \frac{\sqrt{\ell_1(\boldsymbol{X}^{\lambda}_{[t-1]}; \boldsymbol{\theta})}}{\|\boldsymbol{X}^{\lambda}_{t}\|_2 + \delta}, \tag{7.18}$$

which is computed per sample of the validation set $\mathcal{D}_v$, composed of 750 trajectories with parameters $\boldsymbol{\lambda}$ sampled using Halton sequences and $t \in \{1, \dots, 51\}$, for a total of 37,500 samples. For these per-sample RMSEs, we compute: mean (*RMSE-mean*), to quantify global generalization error, 99th percentile (*RMSE-p99*), to capture robustness, and standard deviation (*RMSE-std*), to reflect performance consistency.

7.5 Experimental Results

We plot *RMSE-mean*, *RMSE-p99*, and *RMSE-std* for every PDE–architecture–method–seed run in Figure 7.4. Tables 7.3–7.5 provide complete tabular results, and we summarise each method's relative improvement w.r.t. the *Uniform* baseline, averaged over all experiments, in Table 7.1. Additionally, we visualise example trajectories, comparing the predictions of surrogates trained with *OGAS-L* versus *Uniform* sampling against the ground truth. See Figures 7.6–7.8. We discuss the key findings below.

Significantly improved robustness and uniformity. Across PDEs, architectures, and seeds, both OGAS variants improve tail behaviour and reduce error dispersion, yielding more uniform errors. *OGAS-L* yields the strongest gains, reducing *RMSE-p99* and *RMSE-std* by 28% and 38%, while *OGAS-U* achieves 21% and 29% reductions. *OGAS-L* also shows smaller seed-to-seed variation, suggesting these improvements are robust. This matches the OGAS mechanism: concentrating samples on regimes where the surrogate struggles primarily suppresses rare high-error events. In Figure 7.3, we illustrate this for GS with scOT, where the sampled distribution progressively concentrates on specific Blob Center regions, demonstrating how *OGAS-L* autonomously focuses on harder parameter regimes to improve robustness. *Breed* also improves tail metrics but to a lesser extent and with higher variance across experiments.

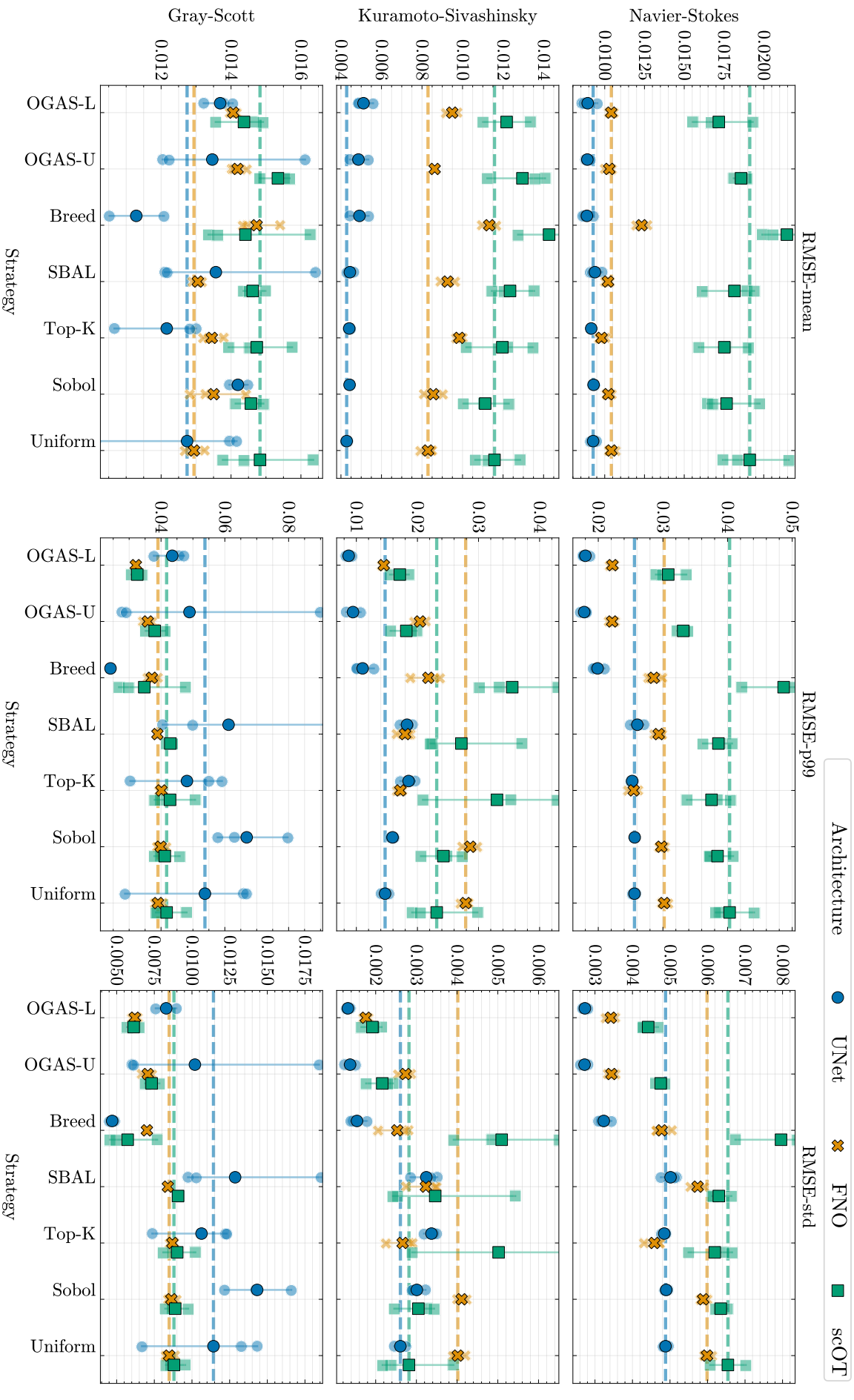


Figure 7.4: Distributional statistics of *RMSE* errors (y-axis): mean (left column), 99th percentile (middle column), and standard deviation (right column), for each PDE (rows), surrogate architecture (marker shape and colour), and compared method (x-axis). Each vertically connected set of points corresponds to one of the three random seeds, with a bold point indicating their average. Horizontal dashed lines indicate the uniform baseline for each architecture (corresponding colour), facilitating relative comparisons.

Table 7.1: Relative improvement (%) w.r.t *Uniform*, averaged over all architecture and PDEs (mean ± std). Higher is better.

Strategy	RMSE-mean	RMSE-p99	RMSE-std
OGAS-L	−4.22±10.51	**28.15±10.44**	**37.95±10.72**
OGAS-U	−4.46±9.30	20.56±21.24	29.02±22.03
Breed	−11.15±16.01	8.59±32.11	15.62±43.48
SBAL	−2.44±8.46	−1.88±24.44	−4.14±25.11
Top-K	−0.89±9.54	−0.15±27.98	−4.68±38.71
Sobol	**−0.82±7.00**	−3.88±11.50	−5.54±12.15

Consistency across architectures and PDEs. Differences across settings mainly appear in *RMSE-mean*. For lower-capacity surrogates (UNet, FNO), steering toward harder regimes can slightly increase the mean on KS/GS, where domain-size variation induces large-scale shifts and makes it harder to fit both typical and challenging behaviours within a fixed budget. With NS, whose parameter range is tighter, all backbones improve in *RMSE-mean* (avg. $\approx 4.65\%$). The highest-capacity surrogate (scOT) is also the most stable across PDEs: it yields a smaller but consistently positive mean improvement (avg. $\approx 2.7\%$) while preserving the robustness gains seen in tail metrics. This supports the intuition that larger models can absorb the harder-regime emphasis induced by OGAS with less impact on typical-case accuracy. *Breed* and *Top-K* seem to be more sensitive to the backbone architecture and can underperform *Uniform* with scOT, whereas both OGAS variants remain comparatively consistent across architectures.

Average performance. For NS, all methods except *Breed* improve average errors compared to *Uniform*, with *OGAS-L* achieving up to 10.17% reduction for scOT and 3.69% for UNet. For GS and KS, AL strategies can slightly increase *RMSE-mean* relative to *Uniform*, but these changes remain limited compared to the improvements obtained on tail metrics. This behaviour is expected when prioritizing robustness: allocating more samples to rare or difficult regimes reduces extreme failures, and for lower-capacity surrogates, it comes with a modest shift in average-case accuracy, similar to findings in [47, 158].

Loss vs. uncertainty. Across PDEs, *OGAS-L* tends to be the most reliable choice, with noticeably lower variance of tail gains. This is particularly clear on GS, where uncertainty-based methods can be noisy with small ensembles: for GS–UNet, *OGAS-U* exhibits very large seed-to-seed variability in p99 (4.89 ± 3.55) compared to *OGAS-L* (4.35 ± 0.50), and similarly inflated variance on p95/p99. *Breed* shows a significant improvement for GS–UNet. At the same time, *OGAS-U* can match or slightly outperform *OGAS-L* on smoother settings (notably NS), suggesting uncertainty signals are effective when they remain well-calibrated.

Minimal training delays. We computed cumulative wall-clock training overheads of each AL method and present them in Figure 7.5 as average across all experiments and normalised by the total surrogate training time to facilitate comparisons. OGAS incurs negligible overheads (23 seconds, ≈0.2% of total training time) thanks to its asynchronous design. *Breed* adds slightly more (70 seconds, ≈0.7%). In contrast, *SBAL* introduces

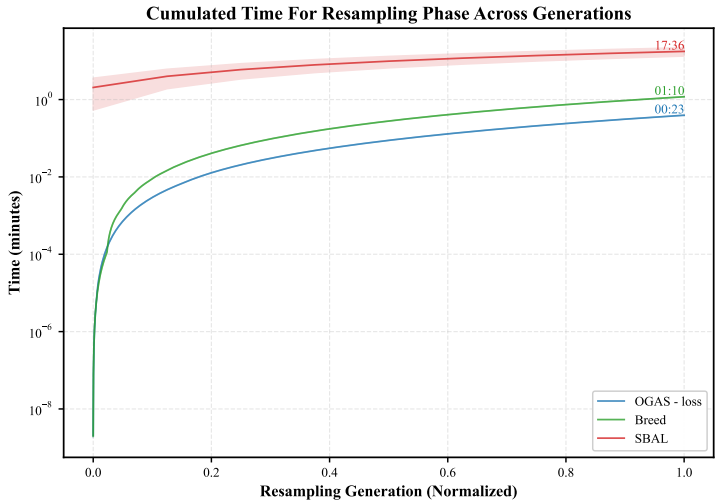


Figure 7.5: Cumulative training overheads of AL methods, normalised by the total surrogate training time.

significant delays for pool management (17.6 minutes, ≈10%), making it impractical for online training even though the pool size was moderate on purpose.

Ensemble-free loss-based steering. Table 7.2 reports aggregated results under a single-member evaluation protocol, where validation errors are computed using only the first ensemble member. For the *OGAS-L1* variant, a single surrogate is trained, making loss-based training signal ensemble-free. On robustness metrics, *OGAS-L1* marginally exceeds the ensemble-informed *OGAS-L* – 24.8% vs. 23.9% on *RMSE-p99*, 35.3% vs. 33.8% on *RMSE-std*, respectively, however, average error is degraded more (−10.8% vs. −8.8%). This confirms that the loss-based signal is effective without the overhead of ensemble training, but suggests that the ensemble-informed signal provides a regularising effect on mean performance. More broadly, the general drop in metrics across all methods under this protocol indicates that ensemble prediction is itself beneficial, independently of the training signal.

7.5.1 Additional Experiments

Motivated by the limited improvements in average error achieved by AL methods in our main experiments, we conduct additional small-scale experiments on 1D PDEs. These experiments provide a fast, controlled testbed where training is computationally tractable, allowing us to assess potential AL gains more cleanly.

We train a UNet surrogate without conditioning on three PDEs, namely default KS, KS-c, and KdV from 6.4.3.1, and we compare loss-based *OGAS* (a slight variant of *OGAS-L*) against *Uniform* and *Breed*. We include two variants of *OGAS* to study the impact of bias-correction: *OGAS-NC* (no correction) and *OGAS-R* (only uniform component samples are used for generator training). Additionally, for *OGAS* on KS, we investigate how the surrogate buffer $\mathcal{R}$ size and resampling frequency affect learning dynamics. The results are provided in Tables 7.6 and 7.7.

Stronger gains on average error. In this 1D setting, *OGAS* consistently improves all metrics across KS variants, achieving up to 4× and 2× reductions in average error

Table 7.2: Single-model evaluation. Relative improvement (%) w.r.t *Uniform*, averaged over all architecture and PDEs (mean ± std). Higher is better.

Strategy	RMSE-mean	RMSE-p99	RMSE-std
OGAS-L1	−10.77±14.94	**24.81±16.45**	**35.31±16.00**
OGAS-L	−8.84±12.19	23.93±18.59	33.78±20.20
OGAS-U	−7.69±14.81	11.29±50.61	23.11±40.29
Breed	−14.93±16.50	5.58±29.91	13.12±44.76
SBAL	−1.37±11.77	−0.87±27.91	−3.98±27.23
Top-K	−2.48±14.14	0.45±30.83	−5.81±45.58
Sobol	**0.80±10.55**	−1.84±16.12	−4.08±17.59

compared to *Uniform* sampling and *Breed*, respectively. For KdV, *OGAS* improves worst-case and dispersion by one order of magnitude each, compared to both baselines, while *Breed* achieves the best average error. These results suggest that for simpler tasks (fewer than 10 parameters, a smaller model without conditioning, and 1D trajectories), *OGAS* steering can be highly effective.

Importance of bias-correction. Relative to no correction, bias-correction decreases error dispersion by 20%, average by 17%, and worst-case by 16%. Relying only on uniform samples for training (*OGAS-R*) performs better than *Uniform*, but worse than *OGAS-NC*. This confirms that the bias induced by the sampling distribution shift is non-negligible, and that correcting it with importance weights is beneficial for surrogate performance.

Reactivity vs. stability. We find that smaller buffers with more frequent resampling outperform larger configurations. The best configuration uses 10× smaller buffer and 10× higher resampling rate, achieves 1.8× lower worst-case error, 2.4× lower dispersion, and 2.5× lower average error, compared to the worst configuration. This highlights the trade-off between rapid adaptation to evolving target conditionals and the lower-variance gradient estimates provided by larger buffers.

7.6 Conclusion

This chapter introduced OGAS as a generative alternative to online Breed, replacing adaptive Gaussian mixtures with a more expressive neural generator to overcome parameter-scale limitations. OGAS complies with every pillar of online AL design: it runs asynchronously with negligible delay, remains architecture-agnostic, and reacts fast enough to track evolving surrogate dynamics. The history-bias correction stabilises training under non-stationary sampling distribution and contributes significantly to the gains. Overall, OGAS performs consistently well across the considered PDEs and architectures, despite their distinctly different dynamics and capacities.

Among the observations, loss-based signals appear as a stable and convenient choice for measuring informativeness, whereas uncertainty from small ensembles can be noisy and would need much larger ensembles to become competitive, directly increasing compute. Additionally, the reactivity of the AL loop emerge as a critical factor for effective steering

in online settings, as the surrogate's evolving state can quickly make the AL selection less effective if it cannot be updated frequently enough.

The evaluation of Breed is extended beyond 1D experiments with a simpler surrogate from Section 6.4. In this larger-scale testbed, Breed performs reasonably well and improves over uniform sampling and pool-based AL in many cases. However, it struggles with the largest surrogate, suggesting that its Gaussian mixture proposal can be too rigid or too easily biased for higher-capacity models and may require further tuning to perform well across different settings.

Limitations and Outlook. The current OGAS design leaves several aspects open to improvement. First, the loss-based training signal is normalised by the norm of the target state rather than its square, which may be less effective at prioritising high-error samples. Second, the proposal range with $\tilde{\varepsilon}_{\max} = 5$ may also be counterproductive, as it covers a large portion of the right tail, leading to overly difficult trajectories and surrogate destabilisation. Third, out-of-bounds parameters sampled by the generator are clipped to Λ, which can reduce sampling diversity; notably, Breed applies a softer approach (described in 6.2.3.3) that may partially explain its relative advantage in GS, where boundary regimes are particularly atypical and may destabilise training.

On the practical side, OGAS uses an extra GPU for generator training. Running it on the surrogate's device is possible but, in typical multi-GPU setup, it risks some interference; allocating a dedicated device is thus more feasible, assuming the AL gains justify the extra resource cost. As for pool-based methods adaptation, similarly using an additional GPU could avoid pausing the online workflow, yet the lengthy AL procedure would still introduce lag between surrogate updates and acquisition decisions, essentially limiting the reactivity we consider critical for AL efficiency. Such experiments would nonetheless help isolate the contribution of reactivity from that of the acquisition strategy. Finally, the 2D experiments require longer training to reveal AL's full potential, as the surrogate likely requires more time to converge and adapt to harder regimes; the 1D experiments indicate that sufficient convergence yields stronger gains in average error.

Several promising directions are open to further improve OGAS and explore the limits of online AL for surrogate training. The current setup fixes the IC type; a natural extension is to let OGAS also select among IC families, further diversifying the training distribution; in offline setting, it is a common practice to mix IC types [82, 75]. Combining OGAS with timestep-level acquisition [101] or adopting information-based acquisition functions [229] is a complementary axis of improvement that remains unexplored. More ambitiously, conditioning a generative model to produce initial states directly, rather than sampling from a fixed parametric family, could greatly expand the expressivity of the data generation process.

Finally, our results in parameter spaces up to $\approx$300 dimensions suggest that generative AL makes scaling in parameter dimensionality more approachable. This opens a natural question: how far can this scale, and what happens when the target space is no longer a continuous parameter vector but something structurally richer, such as varying geometries? A related and perhaps more fundamental question is at what point training the generator itself becomes as demanding as training the surrogate – and what that would imply for the design of online AL pipelines.

Table 7.3: **2D Navier-Stokes Results.** Active learning strategy comparison on Kolmogorov flow. Values show mean ± std across seeds. Best per architecture in **bold**.

(a) U-Net

Strategy	Mean ($\times 10^{-3}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	8.93 ± 0.53	2.73 ± 0.07	4.16 ± 0.79	1.80 ± 0.06	1.44 ± 0.07
OGAS-U	8.91 ± 0.16	$\mathbf{2.72 \pm 0.10}$	$\mathbf{3.72 \pm 0.49}$	$\mathbf{1.78 \pm 0.05}$	$\mathbf{1.42 \pm 0.02}$
Breed	$\mathbf{8.88 \pm 0.36}$	3.24 ± 0.18	4.82 ± 0.10	1.99 ± 0.09	1.52 ± 0.08
SBAL	9.38 ± 0.39	5.01 ± 0.22	6.12 ± 0.37	2.60 ± 0.11	1.93 ± 0.08
Top-K	9.15 ± 0.02	4.85 ± 0.06	6.59 ± 0.07	2.52 ± 0.02	1.87 ± 0.02
Sobol	9.30 ± 0.06	4.90 ± 0.04	6.09 ± 0.24	2.56 ± 0.01	1.90 ± 0.00
Uniform	9.27 ± 0.20	4.88 ± 0.07	6.16 ± 0.10	2.56 ± 0.03	1.89 ± 0.03

(b) FNO

Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	1.04 ± 0.01	$\mathbf{3.42 \pm 0.12}$	4.19 ± 0.14	2.22 ± 0.04	1.71 ± 0.03
OGAS-U	1.03 ± 0.02	3.44 ± 0.10	$\mathbf{4.09 \pm 0.18}$	$\mathbf{2.22 \pm 0.05}$	$\mathbf{1.70 \pm 0.04}$
Breed	1.23 ± 0.03	4.78 ± 0.24	5.60 ± 0.30	2.86 ± 0.11	2.17 ± 0.08
SBAL	1.02 ± 0.01	5.74 ± 0.18	5.70 ± 0.13	2.93 ± 0.06	2.17 ± 0.05
Top-K	$\mathbf{0.98 \pm 0.02}$	4.58 ± 0.24	5.03 ± 0.12	2.55 ± 0.08	1.89 ± 0.06
Sobol	1.02 ± 0.02	5.88 ± 0.10	5.66 ± 0.10	2.98 ± 0.05	2.20 ± 0.03
Uniform	1.04 ± 0.02	5.99 ± 0.12	5.84 ± 0.11	3.02 ± 0.05	2.24 ± 0.05

(c) scOT

Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	$\mathbf{1.72 \pm 0.19}$	$\mathbf{4.42 \pm 0.23}$	$\mathbf{5.45 \pm 0.34}$	$\mathbf{3.08 \pm 0.25}$	$\mathbf{2.55 \pm 0.24}$
OGAS-U	1.86 ± 0.04	4.75 ± 0.13	5.75 ± 0.14	3.31 ± 0.08	2.76 ± 0.06
Breed	2.15 ± 0.21	7.96 ± 1.07	10.10 ± 1.44	4.87 ± 0.58	3.64 ± 0.40
SBAL	1.82 ± 0.18	6.30 ± 0.29	8.32 ± 0.34	3.86 ± 0.23	3.01 ± 0.22
Top-K	1.75 ± 0.16	6.19 ± 0.61	7.97 ± 0.76	3.75 ± 0.34	2.92 ± 0.27
Sobol	1.77 ± 0.18	6.35 ± 0.16	8.27 ± 0.26	3.85 ± 0.21	2.96 ± 0.21
Uniform	1.91 ± 0.22	6.54 ± 0.47	8.83 ± 0.99	4.03 ± 0.33	3.14 ± 0.29

Table 7.4: **2D Kuramoto-Sivashinsky Results.** Active learning strategy comparison. Values show mean ± std across seeds. Best per architecture in **bold**.

(a) U-Net

Strategy	Mean ($\times 10^{-3}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-3}$)
OGAS-L	5.11 ± 0.41	$\mathbf{1.32 \pm 0.05}$	5.08 ± 0.41	$\mathbf{0.87 \pm 0.05}$	6.99 ± 0.42
OGAS-U	4.86 ± 0.46	1.37 ± 0.13	6.07 ± 0.81	0.94 ± 0.12	$\mathbf{6.70 \pm 0.59}$
Breed	4.91 ± 0.46	1.54 ± 0.21	$\mathbf{4.50 \pm 0.95}$	1.10 ± 0.16	7.07 ± 1.00
SBAL	4.44 ± 0.17	3.24 ± 0.34	10.56 ± 0.15	1.83 ± 0.10	8.57 ± 0.60
Top-K	4.41 ± 0.06	3.37 ± 0.16	10.15 ± 0.66	1.86 ± 0.12	8.87 ± 0.03
Sobol	4.43 ± 0.02	3.01 ± 0.18	11.14 ± 1.68	1.59 ± 0.03	7.79 ± 0.24
Uniform	$\mathbf{4.28 \pm 0.06}$	2.61 ± 0.14	9.53 ± 0.07	1.47 ± 0.06	7.51 ± 0.32

(b) FNO

Strategy	Mean ($\times 10^{-3}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	9.49 ± 0.28	$\mathbf{1.76 \pm 0.06}$	$\mathbf{3.26 \pm 0.18}$	$\mathbf{1.45 \pm 0.03}$	$\mathbf{1.22 \pm 0.04}$
OGAS-U	8.63 ± 0.02	2.73 ± 0.16	4.88 ± 0.28	2.04 ± 0.09	1.29 ± 0.04
Breed	11.32 ± 0.35	2.53 ± 0.41	5.02 ± 0.58	2.18 ± 0.26	1.61 ± 0.07
SBAL	9.26 ± 0.36	3.23 ± 0.42	3.34 ± 0.11	1.80 ± 0.12	1.57 ± 0.12
Top-K	9.84 ± 0.16	2.66 ± 0.36	3.49 ± 0.23	1.72 ± 0.05	1.50 ± 0.04
Sobol	8.56 ± 0.47	4.11 ± 0.12	6.46 ± 0.16	2.86 ± 0.12	1.56 ± 0.05
Uniform	$\mathbf{8.30 \pm 0.32}$	4.02 ± 0.16	5.95 ± 0.12	2.79 ± 0.07	1.53 ± 0.06

(c) scOT

Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	1.22 ± 0.12	$\mathbf{1.92 \pm 0.25}$	3.20 ± 0.23	$\mathbf{1.71 \pm 0.15}$	$\mathbf{1.51 \pm 0.16}$
OGAS-U	1.30 ± 0.15	2.16 ± 0.35	$\mathbf{3.19 \pm 0.28}$	1.82 ± 0.23	1.63 ± 0.22
Breed	1.43 ± 0.15	5.09 ± 1.31	8.86 ± 1.20	3.55 ± 0.67	2.37 ± 0.53
SBAL	1.23 ± 0.11	3.46 ± 1.70	5.78 ± 1.98	2.72 ± 0.86	1.85 ± 0.46
Top-K	1.20 ± 0.17	5.02 ± 2.12	7.47 ± 2.19	3.30 ± 1.12	2.09 ± 0.61
Sobol	$\mathbf{1.11 \pm 0.11}$	3.05 ± 0.52	5.17 ± 0.82	2.42 ± 0.34	1.64 ± 0.24
Uniform	1.16 ± 0.11	2.82 ± 0.95	4.80 ± 0.39	2.32 ± 0.58	1.66 ± 0.35

Table 7.5: **2D Gray-Scott Results.** Active learning strategy comparison. Values show mean ± std across seeds. Best per architecture in **bold**.

(a) U-Net

Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-2}$)	Max ($\times 10^{-1}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	1.37 ± 0.04	0.83 ± 0.07	$\mathbf{1.00 \pm 0.39}$	4.35 ± 0.50	3.16 ± 0.09
OGAS-U	1.35 ± 0.23	1.02 ± 0.71	1.08 ± 0.06	4.89 ± 3.55	4.18 ± 2.98
Breed	$\mathbf{1.13 \pm 0.08}$	$\mathbf{0.47 \pm 0.02}$	1.52 ± 0.51	$\mathbf{2.42 \pm 0.00}$	$\mathbf{1.89 \pm 0.12}$
SBAL	1.36 ± 0.25	1.29 ± 0.49	1.77 ± 0.19	6.12 ± 2.81	4.50 ± 1.86
Top-K	1.22 ± 0.13	1.06 ± 0.28	1.67 ± 0.07	4.81 ± 1.56	3.81 ± 1.08
Sobol	1.42 ± 0.03	1.43 ± 0.22	2.02 ± 0.39	6.69 ± 1.15	5.21 ± 0.91
Uniform	1.27 ± 0.23	1.14 ± 0.41	1.78 ± 0.26	5.38 ± 2.17	4.05 ± 1.49

(b) FNO

Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	1.41 ± 0.01	$\mathbf{6.21 \pm 0.17}$	$\mathbf{7.36 \pm 0.43}$	$\mathbf{3.21 \pm 0.03}$	$\mathbf{2.71 \pm 0.03}$
OGAS-U	1.42 ± 0.02	7.06 ± 0.35	8.11 ± 0.26	3.58 ± 0.14	2.99 ± 0.12
Breed	1.47 ± 0.06	7.01 ± 0.03	9.07 ± 0.24	3.71 ± 0.17	2.81 ± 0.05
SBAL	1.31 ± 0.01	8.40 ± 0.02	9.74 ± 1.76	3.89 ± 0.01	3.21 ± 0.01
Top-K	1.35 ± 0.03	8.70 ± 0.15	10.08 ± 0.75	4.01 ± 0.06	3.32 ± 0.05
Sobol	1.35 ± 0.08	8.63 ± 0.30	9.24 ± 0.82	3.99 ± 0.15	3.29 ± 0.13
Uniform	$\mathbf{1.29 \pm 0.03}$	8.49 ± 0.33	10.22 ± 0.70	3.90 ± 0.13	3.23 ± 0.11

(c) scOT

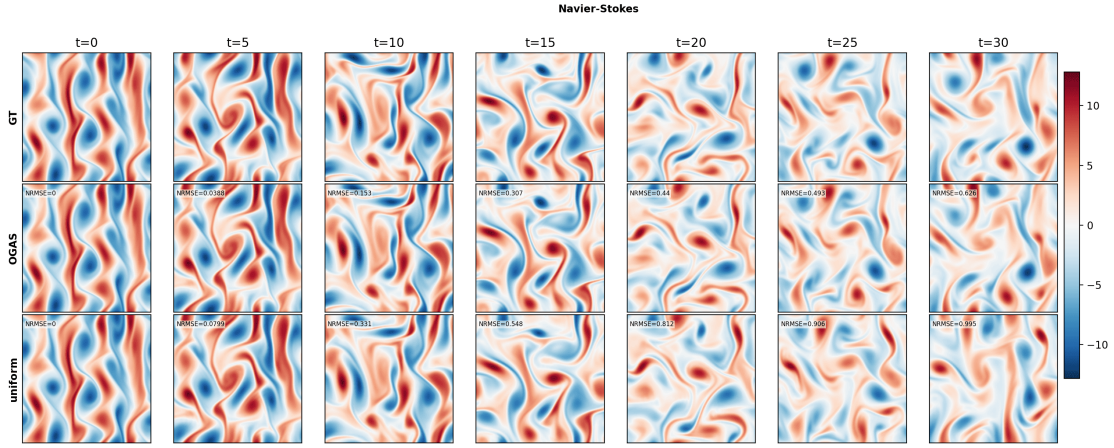
Strategy	Mean ($\times 10^{-2}$)	Std ($\times 10^{-3}$)	Max ($\times 10^{-2}$)	P99 ($\times 10^{-2}$)	P95 ($\times 10^{-2}$)
OGAS-L	$\mathbf{1.44 \pm 0.07}$	6.13 ± 0.40	$\mathbf{6.41 \pm 1.57}$	$\mathbf{3.25 \pm 0.17}$	2.77 ± 0.13
OGAS-U	1.53 ± 0.05	7.32 ± 0.47	7.47 ± 1.04	3.80 ± 0.31	3.10 ± 0.10
Breed	1.44 ± 0.16	$\mathbf{5.73 \pm 1.71}$	8.84 ± 2.41	3.47 ± 1.13	$\mathbf{2.54 \pm 0.53}$
SBAL	1.46 ± 0.03	9.07 ± 0.05	10.13 ± 1.59	4.29 ± 0.11	3.31 ± 0.03
Top-K	1.47 ± 0.09	9.02 ± 1.09	10.71 ± 2.03	4.28 ± 0.69	3.30 ± 0.18
Sobol	1.46 ± 0.04	8.89 ± 0.78	9.63 ± 1.44	4.12 ± 0.41	3.29 ± 0.12
Uniform	1.48 ± 0.14	8.80 ± 0.66	9.98 ± 2.53	4.18 ± 0.54	3.28 ± 0.18

Table 7.6: Additional 1D experiments. Summary across PDEs and sampling strategies (reported as mean ± half-range). OGAS = with correction, OGAS-NC = no correction, OGAS-R = trained only on uniform component samples. Best (lowest) per PDE and metric is in **bold**.

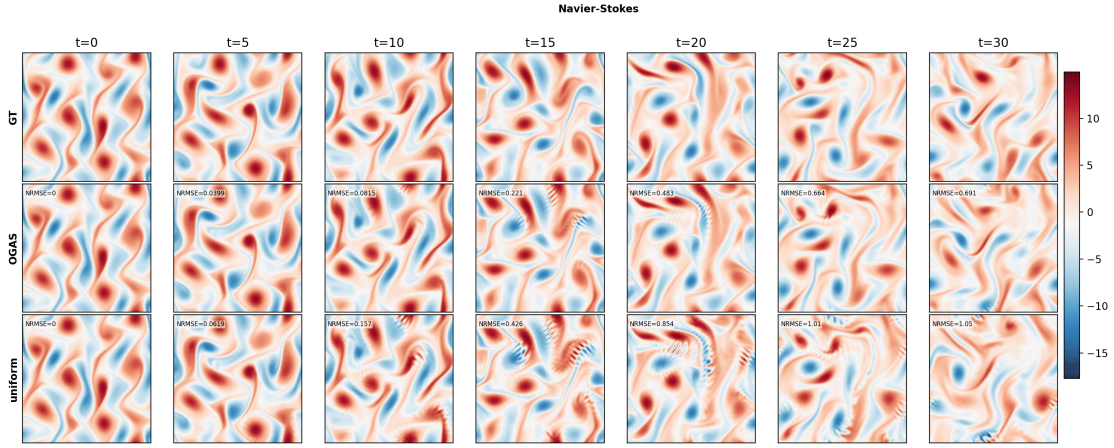
PDE	*Strategy*	*Mean* ($\times 10^{-5}$)	*Max* ($\times 10^{-2}$)	*Std* ($\times 10^{-4}$)
KS	Uniform	4.30 ± 1.46	17.8 ± 5.72	15.3 ± 7.59
KS	Breed	2.37 ± 1.00	14.2 ± 5.99	11.3 ± 6.43
KS	OGAS-NC	1.41 ± 0.58	9.65 ± 3.73	7.44 ± 3.57
KS	OGAS-R	2.64 ± 0.40	13.6 ± 3.21	10.2 ± 4.16
KS	OGAS	**1.15 ± 0.38**	**8.08 ± 2.17**	**5.88 ± 2.43**
PDE	*Strategy*	*Mean* ($\times 10^{-7}$)	*Max* ($\times 10^{-6}$)	*Std* ($\times 10^{-7}$)
KdV	Uniform	2.42 ± 1.15	97.0 ± 217.0	15.0 ± 31.8
KdV	Breed	**2.09 ± 0.70**	1.67 ± 0.70	1.34 ± 0.35
KdV	OGAS-NC	2.42 ± 1.63	**1.53 ± 0.66**	1.38 ± 0.58
KdV	OGAS-R	2.43 ± 1.37	1.62 ± 0.71	**1.23 ± 0.44**
KdV	OGAS	2.37 ± 1.16	1.83 ± 0.75	1.45 ± 0.40
PDE	*Strategy*	*Mean* ($\times 10^{-6}$)	*Max* ($\times 10^{-3}$)	*Std* ($\times 10^{-5}$)
KS-c	Uniform	6.08 ± 1.29	5.02 ± 0.11	5.16 ± 0.16
KS-c	Breed	5.18 ± 1.20	4.19 ± 0.13	4.05 ± 0.15
KS-c	OGAS-NC	4.88 ± 1.15	2.96 ± 0.80	2.24 ± 0.78
KS-c	OGAS-R	5.89 ± 1.42	4.91 ± 0.31	4.88 ± 0.47
KS-c	OGAS	**4.87 ± 0.90**	**2.11 ± 1.44**	**1.71 ± 1.07**

Table 7.7: Additional 1D experiments. Buffer size and resampling cadence ablation on 1D KS. SR=Slow Refresh (3× buffer size), NR=Normal Refresh (1×), FR=Fast Refresh (0.3×). Best (lowest) metric in **bold**.

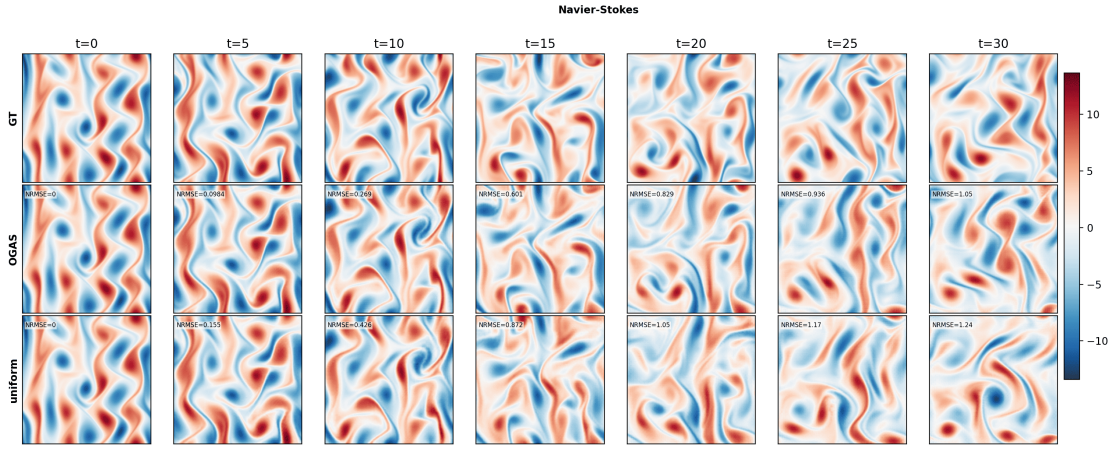
Configuration	*Max* ($\times 10^{-1}$)	*Std* ($\times 10^{-4}$)	*Mean* ($\times 10^{-5}$)
Large buffer (1,500 trajectories = 5% of total budget)			
+ SR	1.20 ± 0.57	9.57	2.03
+ NR	1.01 ± 0.24	7.48	1.47
+ FR	0.88 ± 0.27	6.17	1.20
Medium buffer (750 trajectories = 2.5%)			
+ SR	1.16 ± 0.49	9.09	1.67
+ NR	0.89 ± 0.29	6.75	1.21
+ FR	0.72 ± 0.23	4.94	0.98
Small buffer (150 trajectories, ≈ 0.5%)			
+ SR	0.89 ± 0.29	6.48	1.45
+ NR	0.79 ± 0.28	5.39	1.20
+ FR	**0.66 ± 0.15**	**3.98**	**0.80**



(a) U-Net

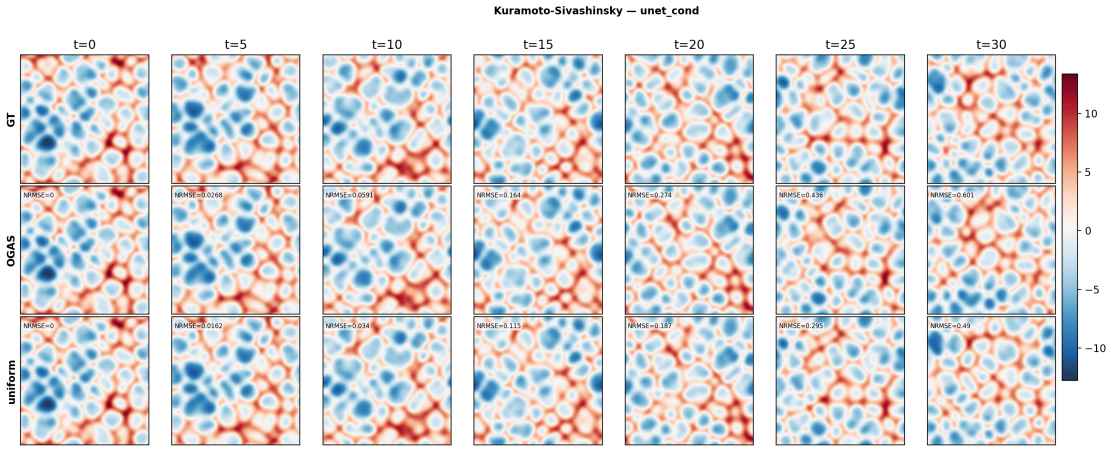


(b) FNO

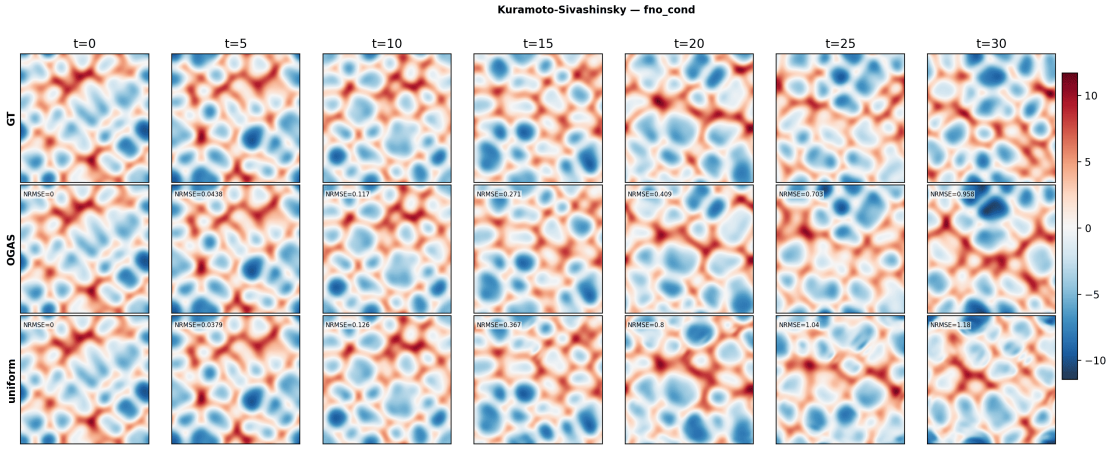


(c) scOT

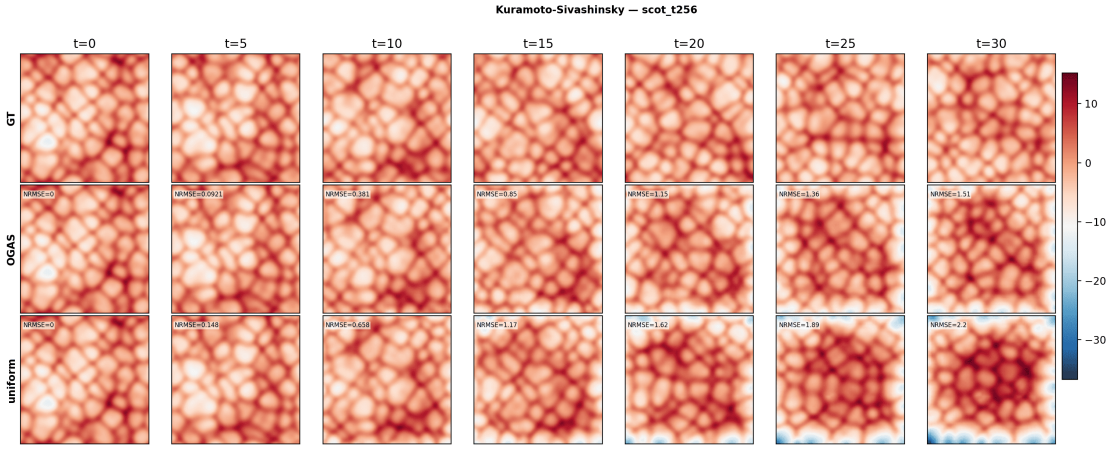
Figure 7.6: **Navier-Stokes Trajectory Samples.** Comparison of rollout predictions on difficult trajectories. We compare the ground truth (GT) against surrogates trained with Uniform sampling and OGAS.



(a) U-Net

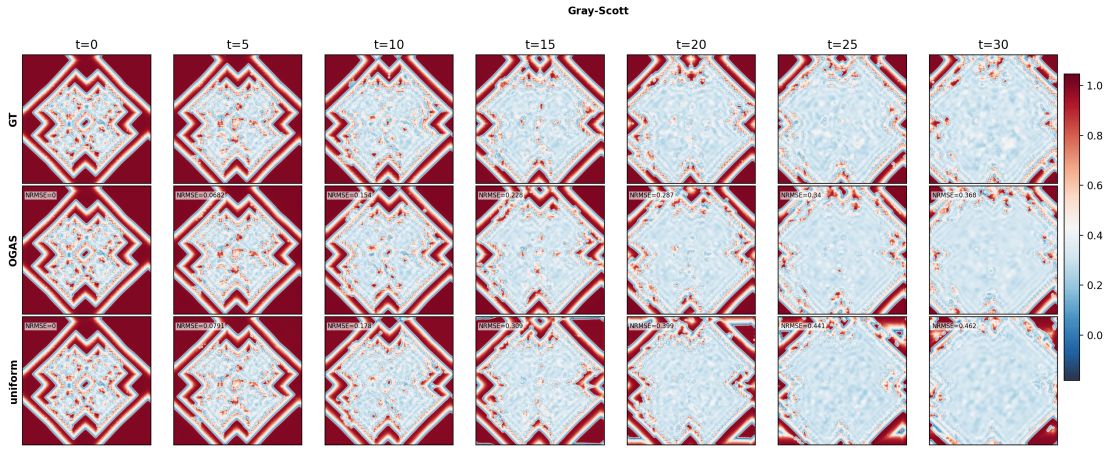


(b) FNO

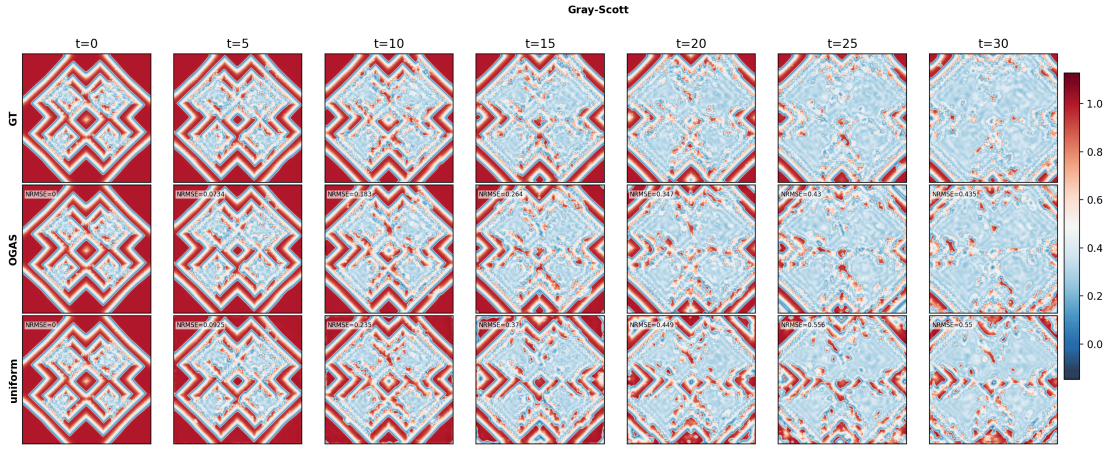


(c) scOT

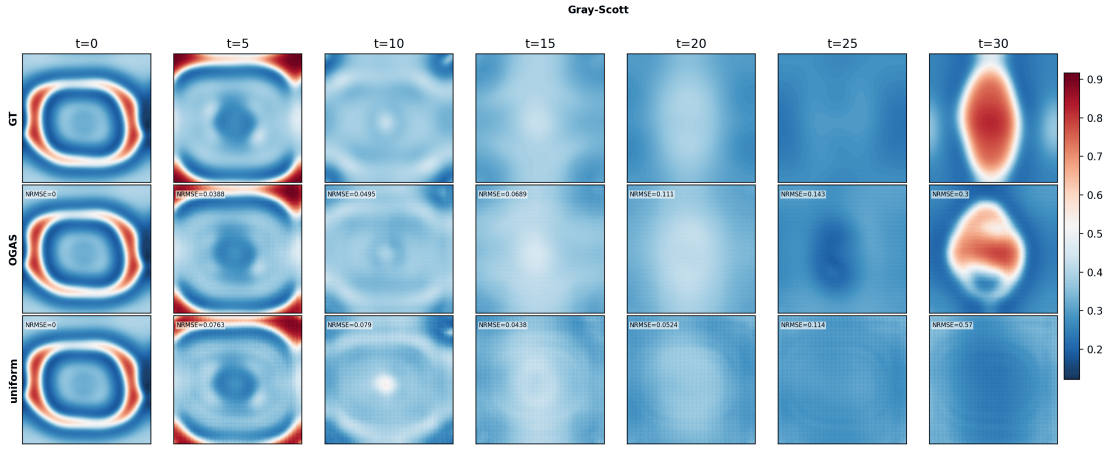
Figure 7.7: **Kuramoto-Sivashinsky Trajectory Samples.** Comparison of rollout predictions on difficult trajectories. We compare the ground truth (GT) against surrogates trained with Uniform sampling and OGAS.



(a) U-Net



(b) FNO



(c) scOT

Figure 7.8: **Gray-Scott Trajectory Samples.** Comparison of rollout predictions on difficult trajectories. We compare the ground truth (GT) against surrogates trained with Uniform sampling and OGAS.

Conclusion

When developing DL surrogates, the data generation stage is often overlooked in the overall process. This largely reflects the practice inherited from observation-based DL, where a finite, pre-collected dataset is used for training. For data-driven surrogates, an analogous finite dataset is typically created synthetically by running many numerical solver instances with input parameters sampled uniformly from the defined domain and storing the resulting trajectory solutions on disk.

This approach, however, **ignores the unique opportunity to participate in data creation**, which does not exist in observation-based DL. As a consequence, it introduces two issues that degrade the I/O and compute efficiency of surrogates, as well as their generalisation capabilities. First, large-scale scientific computing data quickly encounters storage-related limits: non-negligible, irreducible I/O overhead slows down data generation and training, and storage budget constrains the discretisation resolution and the number of samples. Second, while a uniform prior for parameter sampling is reasonable when the true data distribution is unknown, it often results in non-uniform distributions over solutions, leading to datasets with limited diversity, dominated by trivial samples.

In this thesis, we set a goal to mitigate this synthetic data bottleneck from a data-centric perspective. To overcome storage-related limits, we adopt an online training approach, in which data creation and surrogate training are concurrent and overlapped, and samples are transferred memory-to-memory, removing the need for intermediate storage. Once I/O limits are lifted, compute becomes the primary bottleneck.

To improve the compute efficiency of training, we aim to increase data efficiency with query-synthesis active learning (AL): instead of generating data passively, data creation is actively steered by the surrogate's training dynamics towards more informative solution states, thereby reducing the number of solver calls and training iterations required to reach a target performance. In standard query-synthesis AL, an acquisition function scores the informativeness over the input data space, and an acquisition strategy, based on these scores, synthesises input queries to an oracle for corresponding outputs; the resulting input-output pairs are used for training. In surrogate training, however, queries are generated within the solver parameter space rather than directly in the high-dimensional solution space, inducing an intricate and effectively intractable mapping between parameters and informativeness. This makes adoption of AL in this setting non-trivial, particularly under concurrent data generation and training.

While online training for surrogates has been explored in prior work, namely the Melissa framework [187], no AL methods had been developed for this setting. This leads to the central research question of this thesis:

How can AL be effectively applied to online surrogate training?

Summary of Contributions

This thesis approached the research question through a series of contributions, each building on the preceding one by addressing identified limitations and moving towards more challenging scenarios.

In Chapter 5, we introduced Breed, a loss-based AL method for sampling training points, whose core is an adaptive multiple importance sampling (IS) algorithm with a Gaussian mixture (GM) proposal mixed with a uniform distribution under a controlled ratio. We applied Breed in two offline settings, comparing it to a uniform resampling baseline and the state-of-the-art R3 method [40]. First, we used Breed for adaptive sampling of collocation points in physics-informed NNs (PINNs). This setup differs from our target data-driven setting: points are sampled directly in the spatio-temporal solution domain, and no simulation data must be created, eliminating the need for solvers. Yet, this was the only setup where prior work has applied query-synthesis AL for surrogate learning, allowing us to benchmark Breed against related active collocation sampling methods [235] and validate the approach before moving to the more complex targeted setting. Second, we developed a data-driven toy-simulator benchmark where difficulty over parameter space is known by construction, enabling simple analysis and visual interpretation of AL effectiveness against a region-of-interest test set.

In Chapter 6, we first characterised the challenges of AL in the online setting, namely limited access to generated data due to buffering, deferred and non-stationary acquisition under concurrent data generation and surrogate training, and the requirement not to interrupt training. Building on these, an active sampling (AS) mechanism was designed for the Melissa framework, and Breed was adapted to this setting by introducing a delta-loss statistic, a queue-based proposal population, and an efficient out-of-bounds sample handling strategy. We evaluated online Breed in a proof-of-concept setup with a direct surrogate for a heat-transfer PDE with five input parameters, demonstrating the feasibility of steering the sampling of initial condition (IC) parameters towards regions of increased difficulty. The AS mechanism was then refined, and the experimental suite was extended to an autoregressive surrogate for nonlinear and chaotic PDEs of varying difficulty, with Breed compared to the uniform baseline.

In Chapter 7, we presented OGAS, a generative AL method designed to address limitations of Breed: the low expressivity of GMs, which hinders capturing complex distributions; the slow adaptability of IS, which struggles to keep up with evolving training dynamics; and sensitivity to numerous hyperparameters, which are tedious to tune. OGAS employs a fast diffusion model to learn a conditional distribution over solver input parameters given training signals that capture sample difficulty with minimal computational overhead. Additionally, we formalised the history-bias effect of training under a shifting sampling distribution and proposed a correction mechanism for the diffusion objective to mitigate it. The method, in two variants (loss-based and uncertainty-based signals), was compared to Breed and to adapted pool-based AL methods, guiding both IC and physics parameters sampling, and evaluated on a benchmark suite spanning three different backbone surrogate architectures of increasing size and three nonlinear 2D PDEs with distinct dynamics, with up to 308 IC parameters.

Overall, the contributions of this thesis form a unified online framework for training PDE surrogates with query-synthesis AL, and reflect a coherent progression along three axes: from offline to online training, from explicit parametric methods to a generative approach, and from simpler to more complex PDEs modelling tasks.

Cross-cutting Discussion and Limitations

The main finding of this thesis is that active data generation is better than passive, and that a relatively direct approach like Breed can be applied without incurring additional overhead or delays to online training. The effectiveness of active data generation then depends on the desired goal, the surrogate's capabilities, methodological and system-level design choices, and their interplay.

Improved reliability. The experiments repeatedly showed that our proposed AL methods reduce error variance and tail metrics, increasing surrogate reliability, which is directly valuable for safety-critical applications. In Chapter 5, Breed reduced test errors over the hard validation set for the toy-simulator benchmark by a factor of 2.5 relative to the uniform baseline. In Chapter 6, maximum error and dispersion were significantly reduced in most cases, sometimes by an order of magnitude. In Chapter 7, on average over all 18 experiments, relative to the uniform baseline, loss-based OGAS reduced 99th percentile error and dispersion by 28% and 38%, uncertainty-based OGAS by 21% and 29%, and Breed by 9% and 15%, respectively, while all incurring less than 1% of delays to training time.

Model capacity constrains average performance gains. While AL consistently improves reliability, its effect on average performance is conditioned by the surrogate's capacity. In the main experiments of Section 6.4.4, changes in average error are marginal and inconclusive. In Chapter 7, OGAS and Breed degrade average error by 4% and 11% respectively relative to the uniform baseline. However, when the model has sufficient capacity and training budget, AL notably improves average performance alongside tail metrics and variance. This is supported by the extended experiments in Section 6.4.5, small-scale experiments in Section 7.5.1, results for the Navier-Stokes equation in Chapter 7, and the offline setting experiments of Chapter 5.

We hypothesise that this finding is connected to both the bias-variance trade-off and uncertainty decomposition: AL reduces variance by exposing the model to diverse samples, but cannot reduce the bias induced by insufficient model expressivity. Similarly, AL reduces uncertainty stemming from lack of knowledge about data, but not the uncertainty stemming from the model's limited capabilities to represent the data. Therefore, to exploit AL effectively, it is important that the surrogate has sufficient capacity and is given adequate training budget for the task at hand. This hypothesis was not formalised within the scope of this thesis.

Importance of distributional analysis of test errors. Throughout this thesis, we repeatedly examined data and model errors in detail, by constructing region-of-interest datasets, investigating relations between input parameters, training data, and errors, and analysing error distributions both quantitatively and visually. We argue that average error is insufficient to assess surrogate performance, as the intricate parametrisation via solver inputs produces imbalanced data representation and highly skewed error distributions that aggregated metrics fail to capture.

Beyond the surrogate model's capabilities, other factors impact the effectiveness of AL in online surrogate training: the design of the AL method itself, how it accommodates the training data shifts it induces, and how it is integrated into the online training workflow.

Acquisition function and its approximation. The acquisition function must not only accurately represent sample informativeness, but also remain robust to indirect parametrisation via solver inputs, and incur minimal computational overhead to not hinder the online workflow. Within this thesis, we experimented with several relatively lightweight acquisition signals, progressing from raw and aggregated relative loss, to time-marginal signals based on scaled loss and output uncertainty. We found that loss-based scores can be sensitive to training noise and require proper normalisation, while uncertainty estimates, despite being more principled, shown to be less consistent and effective in the experiments of Chapter 7, which we attribute to the small ensemble size that limits the quality of uncertainty estimates in exchange for lower computational overhead. In the scope of this thesis, we did not explore representation-based or information-theoretic acquisition functions from the broader AL [193], as their adaptation to the online surrogate training setting is less straightforward, and they typically require more computational overhead.

To enable effective sampling, an approximator must not only be sufficiently expressive to capture complexity of the acquisition landscape, but also be adaptive enough to follow its evolution, while remaining computationally efficient. In this thesis, we progress from IS with a GM proposal to a fast diffusion-based NN with correction mechanism, finding that OGAS generally outperforms Breed. Nonetheless, the design space of density approximation methods is vast, and we have explored only a small corner of it.

Accommodating training data shifts. Maintaining surrogate stability under the training data distributional shifts introduced by AL is also critical, as these can otherwise lead to a compounding feedback loop of degraded AL decisions and surrogate performance. This thesis addressed it through a concentrate-explore control with linear-constant schedule that mixes proposal and uniform distributions to maintain knowledge of the whole parameter space and avoid overfitting to specific regions. We found this approach to be an essential part of AL effectiveness, as supported by ablation experiments in Section 6.3, and subsequently adopted it in OGAS. However, our approach, while relatively simple, is restricted to a fixed schedule with heuristically chosen hyperparameters. Additionally, the training data update frequency follows a passive, periodic pattern.

Efficient feedback-loop. In our setting, AL operates one step behind the surrogate: it requests new samples based on the current surrogate state, these are generated concurrently with the surrogate training, and the evolved surrogate is ultimately exposed to the new samples tailored to a past state. Minimising this latency between AL decisions and the surrogate's exposure to the corresponding data is therefore critical. It depends not only on the computational cost of acquisition evaluation and sampling, but also on the concrete implementation of the feedback loop. We sought to reduce this latency by improving the AS mechanism, and, in OGAS, by using a fast generative NN. In experiments of Section 7.5.1, we find that using a Buffer of smaller capacity substantially improves AL effectiveness, as it increases the model's exposure to recent samples during training, but with reduced mitigation of intra-simulation bias. The speed imbalance between solver and surrogate, as well as the current buffer management in Melissa, remain important limitations that influence both the latency and the data shifts discussed above.

Future Work

This thesis has laid the groundwork for active data generation in online surrogate training, but several open questions and promising directions remain.

Cross-pollination. The problem setting of this thesis presents a combination of challenges that cuts across several neighbouring to AL fields, as discussed in Chapter 4, most notably Bayesian optimisation (BO), reinforcement learning (RL), and simulation-based inference (SBI), among others; however, it is not solely addressed by any of them. While our proposed methods were designed largely in isolation from their established theory and practice, they already share certain methodological elements with each. For instance, the concentrate-explore control is directly connected to the exploration-exploitation trade-off in RL, and the use of a generative model to approximate the acquisition landscape is tightly connected to the use of surrogate models for black-box optimisation in BO and for amortised inference in SBI, conceptually similar but with different objectives and constraints. Data shifts are addressed in RL through off-policy learning and experience replay, and in continual learning through regularisation and rehearsal method. One promising direction for future work is to further explore these connections and draw inspiration from the rich literature in these fields to develop more theoretically principled and adaptive query-synthesis AL methods for non-stationary and parametrised data generation settings.

Systematic empirical investigations. More systematic empirical studies are still needed to decouple AL methodological effects from implementation-level factors of online training, as acknowledged in Section 3.3.4. A natural next step is to replicate and extend the experiments of this thesis on genuinely compute-bound simulations, such as more complex 3D or industrial-scale solvers, with careful I/O and runtime profiling and explicit accounting of the cost of regenerating data. This would enable a quantitative study of how different speed imbalances and buffer configurations affect the latency of the AL feedback loop and the training data shifts, which can be examined by varying the parallelisation level of data generation and training. Such studies should also characterise regimes in which online training and AL bring limited benefit or even become bottlenecks themselves, for example, when acquisition overhead dominates or solver and model runtimes are poorly matched resulting in stale AL decisions, clarifying when simpler passive strategies remain preferable.

The results of Chapter 7 have made scaling up and applying generative AL to more complex surrogate training scenarios more approachable, opening several interesting directions.

Foundation models. As the generalisation objective of foundation PDE surrogates [75, 145] widens the design space of pretraining data, we see future work on AL in this regime as particularly important. AL can be applied to guide the choice of IC families, PDE types for a given symbolic or latent parametrisation, time-steps in trajectory [101], and resolution [125], or even directly synthesise informative states in solution space. In settings where regenerating data online is costly or infeasible, such mechanisms could also be used to construct compact yet informative training and validation datasets for reuse across studies and models, not for the purpose of training a surrogate, but for benchmarking and evaluation of surrogate performance and generalisation capabilities.

Irregular domains and unstructured geometries. Extending AL beyond square domains and regular meshes to surrogates that generalise over complex geometries and unstructured grids remains open and is highly relevant for industrial-scale applications. A key challenge is how to parameterise the geometry and spatial space, including higher-dimensional and topological aspects, in a way that allows informative-region discovery, for example via graph-based models or space-continuous formulations akin to PINNs, without making the AL model itself as hard and expensive to train as the surrogate. An important open question is whether, in this setting, AL actually improves over uniform priors and whether online schemes still feasible once memory requirements and regeneration costs for complex geometries are taken into account.

Broader Impact and Closing Remarks

Beyond the specific methods developed in this thesis, it is worth reflecting on the broader context of AI for scientific computing and surrogate modelling. This area is still young and evolving rapidly, and many current studies are based on simplified or idealised benchmark problems, echoing the early stages of deep learning in computer vision (CV) and natural language processing (NLP). In those domains, progress was strongly driven by abundant observational data and increasing compute and model scale, culminating in today's generative models that would have seemed out of reach a decade ago.

AI for scientific computing, however, differs from these in a fundamental way: data-driven surrogate models rely on data generated by computationally expensive solvers, i.e., the same systems they aim to approximate, rather than on collected observational data. In principle, more synthetic data can always be generated, avoiding the data scarcity issues that often arise in observation-based DL, but at the cost of additional computation. This creates a tension. On one hand, scientific computing underpins many critical domains, such as engineering, climate and weather prediction, aeronautics, and energy systems, where surrogates can have substantial societal and economic impact. On the other hand, blindly scaling solver calls and model sizes risks amplifying the environmental and economic footprint of computation.

In this context, data generation itself becomes a central bottleneck and a key point of divergence from the historical trajectory of CV and NLP, motivating data-centric approaches. Instead of generating as much data as resources allow and relying solely on larger models and more compute, there is a need for more deliberate, resource-aware use of synthetic data: *generating the right data at the right time.* This thesis presents a step in that direction, treating synthetic data generation as a first-class design choice in surrogate training.

Last but not least, it is important to acknowledge that the long-term impact of AI for science will depend not only on methodological and system-level advances, but also on the way research is conducted and shared. We believe that open, collaborative, multidisciplinary, and inclusive science, grounded in careful consideration of environmental, social, and economic implications, is essential for ensuring that advances in AI for science contribute meaningfully to both scientific progress and human well-being.

Appendix A

Carbon Footprint

One of the main research goals of the thesis was to make surrogate training and data generation more computationally efficient, which is not only important for the practical applicability of the surrogates, but also for reducing the AI impact on the economy and the environment, specifically, the carbon emissions from the computational experiments.

To not be hypocritical, I have tried to be mindful of the carbon emissions from the experiments performed during this thesis, and to estimate them as accurately as possible. Here, I provide an estimate of the carbon footprint of the computational experiments performed during this thesis, as well as the emissions from academic travel and daily commute. Frankly, some numbers have surprised me, and not in a good way.

I hope that this section will motivate other researchers to be more mindful of their carbon emissions and to take steps to reduce them, such as, using more efficient algorithms, reducing unnecessary experiments, choosing HPC infrastructures with lower carbon intensity (those powered by renewable energy sources), reducing unnecessary travel and choosing more sustainable transportation options when travel is necessary.

Computational Emissions

During this thesis, computational experiments were mainly performed on the GriCAD cluster located at the University of Grenoble, France, and the Leonardo supercomputer (#10 in TOP500-list) hosted by CINECA in Bologna, Italy.

GriCAD HPC Cluster

The GriCAD cluster provides carbon footprint estimates per user based on the total CPU hours consumed, using a factor[1] of 3.57 gCO_2e/core.h and an electricity carbon intensity in France of 59.9 gCO_2e/kWh (2023 update).

The total CPU hours consumed during this thesis, specifically for the experiments reported in Chapters 5 and 6 (excluding the additional experiments in Section 6.4.5) that were performed on my account, were approximately 56,937 hours, split between the Dahu partition (13,440 hours) and the Bigfoot partition (43,496 hours). This results in an estimated total carbon footprint of **203 $kgCO_2e$**, split between Dahu (155 $kgCO_2e$) and Bigfoot (48 $kgCO_2e$).

[1] https://hal.archives-ouvertes.fr/hal-02549565v5

However, the electricity production in France decreased[2] from 59.9 gCO_2e/kWh in 2023 to 44.4 gCO_2e/kWh in 2024, and to 33.3 gCO_2e/kWh in 2025, due to the increase in the share of renewable energy sources. Therefore, the actual carbon footprint of the experiments performed on the GriCAD cluster is likely to be lower than the estimated value based on the 2023 factor.

I am grateful to have had access to the French HPC infrastructure, which is powered by a low-carbon electricity grid (#6 in the world with 97% of low-carbon electricity[3]: nuclear 69.6%, solar 5.7%, wind 9.1%, hydro 11.2%, and others), and to have been able to perform the experiments for this thesis with a relatively low carbon footprint.

Leonardo CINECA Supercomputer

The Leonardo supercomputer does not provide per-user carbon footprint estimates, so I calculated the emissions according to Lannelongue et al., 2021 [118] (excluding memory power draw), using the formula:

$$\begin{aligned} \text{Emissions } (gCO_2e) &= \text{Energy (kWh)} \times \text{Carbon Intensity } (gCO_2/kWh), \\ \text{where Energy (Wh)} &= (\text{Time (GPU.h)} \times \text{GPU Power (W)} + \\ &\quad + \text{Time (core.h)} \times \text{CPU Power (W)}) \times \text{PUE}, \end{aligned} \tag{A.1}$$

where power draw of the GPUs and CPUs is estimated based on their TDP specifications.

Hardware TDP specifications. The TDP specifications of the hardware used on the Leonardo supercomputer are as follows[4]:

- Booster partition accelerator: NVIDIA custom Ampere A100 SXM4[5]: 440W per GPU;
- Booster partition processor: Intel Ice Lake Intel Xeon Platinum 8358[6], 32 cores: 7.8125 W/core (8 cores per GPU);
- DCGP partition processor: Intel Sapphire Rapids Intel Xeon Platinum 8480+[7], 56 cores: 6.25 W/core.

Carbon intensity and PUE. The Leonardo supercomputer is located in Italy, where the electricity grid has a higher carbon intensity compared to France. The Italy's grid carbon intensity in 2025 is estimated[8] to be 324 gCO_2/kWh. The PUE (Power Usage Effectiveness) of the Leonardo is estimated[9] to be below 1.1.

[2] https://lowcarbonpower.org/region/France
[3] https://lowcarbonpower.org/region/France
[4] https://docs.hpc.cineca.it/hpc/leonardo.html#hardware-details
[5] https://jlsrf.org/index.php/lsf/article/view/186
[6] https://www.intel.com/content/www/us/en/products/sku/212282/intel-xeon-platinum-8358-processor-48m-cache-2-60-ghz/specifications.html
[7] https://www.intel.com/content/www/us/en/products/sku/231746/intel-xeon-platinum-8480-processor-105m-cache-2-00-ghz/specifications.html
[8] https://lowcarbonpower.org/region/Italy
[9] https://leonardo-supercomputer.cineca.eu/technopole

Compute budget. Our Melissa project got access to the Leonardo supercomputer in May 2025, and the experiments performed on this system are reported in Section 6.4.5 and Chapter 7. Overall, since the access to the system, the project account (3 active users including myself) has consumed 115,812 core.h on Booster partition, and with the conversion of 8 core.h to 1 GPU.h we get 14,476.5 GPU.h, and 452,056 core.h on DCGP partition.

Estimated emissions. Based on the above specifications and estimates, the total emissions from the Melissa experiments performed on the Leonardo are:

$$(6369.7 + 904.8 + 2825.4)\ \text{kWh} \times 1.1 \times 324\ \text{gCO}_2/\text{kWh} \approx \mathbf{3.6\ tCO_2e}, \tag{A.2}$$

which is equivalent[10] to approximately 1.75 round-trip flight Paris-New York, or 39.6% of the average annual carbon footprint of a French citizen.

The DCGP partition was used for generating the surrogate training data and contributes to about 28% of the total emissions, while the other 72% of the emissions are from the Booster partition, which was used for the surrogate training.

I attribute third of the emissions to my personal carbon footprint, which is approximately **1.2 $\mathbf{tCO_2e}$**.

Transport Emissions

Transportation is considered one of the largest contributors to carbon emissions, and academic travel can significantly contribute to a researcher's carbon footprint. In this section, I provide an estimate of the carbon emissions from the academic travel and daily commute during the 4 years of this thesis.

Academic travel

During the 4 years of this thesis, I attended several academic events in person. Here is a list of the events I attended, along with the mode of transportation used for each trip:

1. Three project meetings and three schools in Paris, France (train);
2. One project meeting in Bordeaux, France (train via Paris);
3. JLESC 2023 workshop in Bordeaux, France (train via Paris);
4. ICML 2024 in Vienna, Austria (flight from/to Lyon);
5. SC 2024 in Atlanta, USA, contributing to three trips:
   - 2 trips to Bern, Switzerland, in order to get US visa (train via Geneva),
   - 1 trip to Atlanta, USA (train to/from Paris, transatlantic flight from/to Paris);
6. TUM research visit in Munich, Germany (train via Paris);

These result in:

- 10 round-trip TGV train journeys Grenoble–Paris (960 km round-trip);

[10] https://impactco2.fr/outils/comparateur

- 2 round-trip TGV train journeys Paris–Bordeaux (1,168 km round-trip);
- 1 round-trip TGV train journey Paris–Munich (1,366 km round-trip);
- 2 round-trip TER train journeys Grenoble–Geneva (314 km round-trip);
- 2 round-trip train journeys Geneva–Bern (260 km round-trip);
- 1 round-trip transatlantic flight Paris–Atlanta (14,100 km round-trip);
- 1 round-trip flight Lyon–Vienna (1,800 km round-trip).

French trains' carbon emissions[11] (2025): 3.5 gCO_2e/pkm for TGV and 23.8 gCO_2e/pkm for TER. The difference is mainly due to higher occupancy rates and more efficient energy use of the TGV trains compared to the TER trains, which are regional trains that often have lower occupancy rates and less efficient energy use. In Switzerland, the carbon emissions of trains are even lower due to the high share of renewable energy in the electricity mix. The SBB regional trains' carbon emissions[12] (2024): 0.213 gCO_2e/pkm.

Thus, the estimated emissions from the train journeys are approximately:

- 13303 km by TGV train $\times$ 3.5 gCO_2e/pkm = 46.6 $kgCO_2e$;
- 628 km by TER train $\times$ 23.8 gCO_2e/pkm = 14.9 $kgCO_2e$;
- 520 km by Swiss SBB train $\times$ 0.213 gCO_2e/pkm = 0.11 $kgCO_2e$;

which totals to approximately **61.6 $kgCO_2e$**.

Flights' carbon emission estimates[13]: 2.4 tCO_2e for Paris–Atlanta flight, and 0.5 tCO_2e for Lyon–Vienna flight, which totals to approximately **2.9 tCO_2e**.

These estimates only include the direct emissions from A-to-B transportation, and do not include so many more factors, such as, hotel stays and local transportation. However, nothing of these factors would change the overall conclusion that the carbon emissions from the flights are much higher than those from the train journeys, and that the total emissions from academic travel are significant.

I am grateful to have had the opportunity to attend these events in person, which allowed me to connect with the research community and learn from my peers, probably, impossible to achieve through virtual attendance. Although, I feel that it is not fair that some of top-tier conferences in ML/DL are held exclusively in the US, forcing many European researchers to take long-haul flights to attend them, which significantly increases their carbon footprint. I hope that in the future, more conferences will be held in different locations around the world – this trend is already happening, with more conferences being held or dual-hosted in Europe and Asia, which is a positive step towards reducing the carbon footprint of academic travel.

Daily commute

Little steps add up, and the carbon emissions from daily commute can also be significant over time. Grenoble is a great city for sustainable transportation, with a good network of bike lanes and public transport. I have been commuting to the university by bike and

[11] https://www.sncf-voyageurs.com/en/discover-our-company/csr-and-transitions/calculating-the-carbon-footprint-of-transport

[12] https://reporting.sbb.ch/en/sustainability

[13] https://travel-footprint-calculator.irap.omp.eu/estimate.html

public transport for the past 4 years, which has significantly reduced my carbon footprint compared to commuting by car or diesel bus.

Commuting emissions were estimated based on approximately 5 days per week over 4 years (minus 7 weeks annual holiday), totalling 900 days, with a round-trip distance of approximately 7 km. Transportation was approximately split 75% by bicycle (0.17gCO_2/km) and 25% by public transport, specifically, biogas-fuelled buses (26 gCO_2/km) and electric trams (4.28 gCO_2/km). The numbers are per person and taken from https://impactco2.fr/outils/transport/, which is provided by the French Environment and Energy Management Agency (ADEME). Note: for gas-fuelled buses, it provides a figure of 122 gCO_2/km, but in Grenoble, the CHRONO buses (the line I use) are fuelled by biogas since 2021, which has 79% lower carbon intensity based on https://www.mobiogaz.fr/bilan-co2-du-gnv-ou-biognv/.

Total emissions then:

$$\begin{aligned}
\text{Total distance} &= 900 \text{ days} \times 7 \text{ km/day} = 6300 \text{ km} \\
\text{Bike} &\approx 0.75 \times 6300 \text{ km} \times 0.17 \text{ gCO}_2\text{/km} \approx 803 \text{ gCO}_2 \\
\text{Bus} &\approx 0.25 \times 0.5 \times 6300 \text{ km} \times 26 \text{ gCO}_2\text{/km} = 20475 \text{ gCO}_2 \\
\text{Tram} &\approx 0.25 \times 0.5 \times 6300 \text{ km} \times 4.28 \text{ gCO}_2\text{/km} = 3370.5 \text{ gCO}_2 \\
\text{Total} &\approx 803 + 20475 + 3370 \approx 24648 \approx \mathbf{25\ kgCO_2}
\end{aligned} \tag{A.3}$$

If I commuted only by diesel bus (122 gCO_2/km) it would be instead 768 $kgCO_2$ (30×), by car (218 gCO_2/km) it would be 1373 $kgCO_2$ (55×). I am thankful to have the option to commute by bike and carbon-neutral public transport of Grenoble, which is a great example of how cities can reduce the carbon footprint of their inhabitants.

Overall Carbon Footprint

In total, over 4 years, the estimated carbon footprint of this thesis is **at least 4.39 tCO_2e**, which is around **1.1 tCO_2e per year**. With the goal "well below 2 tCO_2e per year" in mind, this thesis has taken more than half of the budget, which is a significant contribution. Moreover, 55% of the emissions are from one single flight, which is a clear example of how a single decision can have a large impact on the overall carbon footprint.

However, I am glad that I was able to keep the carbon footprint of this thesis relatively low, and I hope that it can serve as an example for other researchers to be mindful of their carbon emissions and to take steps to reduce them.

Bibliography

*

[1] Fujitsu and Yokohama National University achieve world's first real-time prediction of tornadoes associated with typhoons using supercomputer Fugaku. URL https://www.fujitsu.com/global/about/resources/news/press-releases/2025/0212-01.html.

[2] GPU computational performance per dollar. URL https://ourworldindata.org/grapher/gpu-price-performance.

[3] Modern Experimental Design and Active Learning in the Real World. URL https://realworldml.github.io/.

[4] PhysicsNeMo Contributors . NVIDIA PhysicsNeMo: An open-source framework for physics-based deep learning in science and engineering, February 2023. URL https://github.com/NVIDIA/physicsnemo.

[5] V. Adetola and M. Guay. Integration of real-time optimization and model predictive control. *Journal of Process Control*, 20(2):125–133, February 2010. ISSN 0959-1524. doi: 10.1016/j.jprocont.2009.09.001. URL https://www.sciencedirect.com/science/article/pii/S0959152409001632.

[6] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), February 2019. URL http://arxiv.org/abs/1803.08375.

[7] Yuri Aikawa, Naonori Ueda, and Toshiyuki Tanaka. Improving the Efficiency of Training Physics-Informed Neural Networks Using Active Learning. *New Gener. Comput.*, 42(4):739–760, November 2024. ISSN 1882-7055. doi: 10.1007/s00354-024-00253-6. URL https://doi.org/10.1007/s00354-024-00253-6.

[8] Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance Reduction in SGD by Distributed Importance Sampling, April 2016. URL http://arxiv.org/abs/1511.06481.

[9] Benedikt Alkin, Maurits Bleeker, Richard Kurle, Tobias Kronlachner, Reinhard Sonnleitner, Matthias Dorfer, and Johannes Brandstetter. AB-UPT: Scaling Neural CFD Surrogates for High-Fidelity Automotive Aerodynamics Simulations via Anchored-Branched Universal Physics Transformers, October 2025. URL http://arxiv.org/abs/2502.09692.

*We acknowledge that the bibliography contains incomplete metadata and preprint versions of some publications. These will be corrected in the final manuscript version prior the publication.

[10] Sokratis J. Anagnostopoulos, Juan Diego Toscano, Nikolaos Stergiopulos, and George Em Karniadakis. Residual-based attention and connection to information bottleneck theory in PINNs, July 2023. URL https://arxiv.org/abs/2307.00379v1.

[11] Sokratis J. Anagnostopoulos, Juan Diego Toscano, Nikolaos Stergiopulos, and George Em Karniadakis. Learning in PINNs: Phase transition, total diffusion, and generalization, March 2024. URL http://arxiv.org/abs/2403.18494.

[12] Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep Batch Active Learning by Diverse, Uncertain Gradient Lower Bounds, February 2020. URL http://arxiv.org/abs/1906.03671.

[13] Neil Ashton, Johannes Brandstetter, and Siddhartha Mishra. Fluid Intelligence: A Forward Look on AI Foundation Models in Computational Fluid Dynamics, November 2025. URL http://arxiv.org/abs/2511.20455.

[14] Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nat Rev Phys*, 6(5):320–328, May 2024. ISSN 2522-5820. doi: 10.1038/s42254-024-00712-5. URL https://www.nature.com/articles/s42254-024-00712-5.

[15] Pradeep Bajracharya, Javier Quetzalcóatl Toledo-Marín, Geoffrey Fox, Shantenu Jha, and Linwei Wang. Feasibility Study on Active Learning of Smart Surrogates for Scientific Simulations, July 2024. URL http://arxiv.org/abs/2407.07674.

[16] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data driven discretizations for partial differential equations. *Proc. Natl. Acad. Sci. U.S.A.*, 116(31):15344–15349, July 2019. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1814058116. URL http://arxiv.org/abs/1808.04930.

[17] Brian R. Bartoldson, Bhavya Kailkhura, and Davis Blalock. Compute-Efficient Deep Learning: Algorithmic Trends and Opportunities, October 2022. URL http://arxiv.org/abs/2210.06640.

[18] Brian R. Bartoldson, Bhavya Kailkhura, and Davis Blalock. Compute-Efficient Deep Learning: Algorithmic Trends and Opportunities, March 2023. URL http://arxiv.org/abs/2210.06640.

[19] Francesca Bartolucci, Emmanuel de Bézenac, Bogdan Raonić, Roberto Molinaro, Siddhartha Mishra, and Rima Alaifari. Representation Equivalent Neural Operators: A Framework for Alias-free Operator Learning, November 2023. URL http://arxiv.org/abs/2305.19913.

[20] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, and E. W. Bethel. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Comput. Graph. Forum*, 35(3):577–597, June 2016. ISSN 0167-7055.

[21] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey, February 2018. URL http://arxiv.org/abs/1502.05767.

[22] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, June 2009. Association for Computing Machinery. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

[23] Peter Benner, Serkan Gugercin, and Karen Willcox. A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems. *SIAM Rev.*, 57(4): 483–531, January 2015. ISSN 0036-1445. doi: 10.1137/130932715. URL https://epubs.siam.org/doi/10.1137/130932715.

[24] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative Learning Under Covariate Shift. *Journal of Machine Learning Research*, 10(75):2137–2155, 2009. ISSN 1533-7928. URL http://jmlr.org/papers/v10/bickel09a.html.

[25] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical Fourier Neural Operators: Learning Stable Dynamics on the Sphere, June 2023. URL http://arxiv.org/abs/2306.03838.

[26] L. Bottou. Stochastic Gradient Learning in Neural Networks. 1991. URL https://www.semanticscholar.org/paper/Stochastic-Gradient-Learning-in-Neural-Networks-Bottou/82eec4af1475de9a7e876bcbaddb4a0c4a1dc187.

[27] Lise Le Boudec, Emmanuel de Bezenac, Louis Serrano, Ramon Daniel Regueiro-Espino, Yuan Yin, and Patrick Gallinari. Learning a Neural Solver for Parametric PDE to Enhance Physics-Informed Methods, June 2025. URL http://arxiv.org/abs/2410.06820.

[28] Alexander Brace, Igor Yakushin, Heng Ma, Anda Trifan, Todd Munson, Ian Foster, Arvind Ramanathan, Hyungro Lee, Matteo Turilli, and Shantenu Jha. Coupling streaming AI and HPC ensembles to achieve 100-1000x faster biomolecular simulations. April 2021. URL http://arxiv.org/abs/2104.04797.

[29] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message Passing Neural PDE Solvers, March 2023. URL http://arxiv.org/abs/2202.03376.

[30] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. URL http://arxiv.org/abs/2005.14165.

[31] Monica F. Bugallo, Victor Elvira, Luca Martino, David Luengo, Joaquin Miguez, and Petar M. Djuric. Adaptive Importance Sampling: The past, the present, and the future. *IEEE Signal Process. Mag.*, 34(4):60–79, July 2017. ISSN 1053-5888, 1558-0792. doi: 10.1109/MSP.2017.2699226. URL https://ieeexplore.ieee.org/document/7974876/.

[32] Davide Cacciarelli and Murat Kulahci. Active learning for data streams: A survey. *Mach Learn*, 113(1):185–239, January 2024. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-023-06454-2. URL http://arxiv.org/abs/2302.08893.

[33] Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z. Li. A Survey on Generative Diffusion Model, December 2023. URL http://arxiv.org/abs/2209.02646.

[34] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components, June 2005. URL http://arxiv.org/abs/cs/0506006.

[35] O Cappé, A Guillin, J. M Marin, and C. P Robert. Population Monte Carlo. *Journal of Computational and Graphical Statistics*, 13(4):907–929, December 2004. ISSN 1061-8600, 1537-2715. doi: 10.1198/106186004X12803. URL http://www.tandfonline.com/doi/abs/10.1198/106186004X12803.

[36] Kathryn Chaloner and Isabella Verdinelli. Bayesian Experimental Design: A Review. *Statistical Science*, 10(3):273–304, August 1995. ISSN 0883-4237, 2168-8745. doi: 10.1214/ss/1177009939. URL https://projecteuclid.org/journals/statistical-science/volume-10/issue-3/Bayesian-Experimental-Design-A-Review/10.1214/ss/1177009939.full.

[37] Wuyang Chen, Jialin Song, Pu Ren, Shashank Subramanian, Dmitriy Morozov, and Michael W. Mahoney. Data-Efficient Operator Learning via Unsupervised Pretraining and In-Context Learning, June 2025. URL http://arxiv.org/abs/2402.15734.

[38] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. 2022. doi: 10.1073/pnas.1912789117. URL www.pnas.org/cgi/doi/10.1073/pnas.1912789117.

[39] Aymen Danoun, Etienne Prulière, and Yves Chemisky. FE-LSTM: A hybrid approach to accelerate multiscale simulations of architectured materials using Recurrent Neural Networks and Finite Element Analysis. *Computer Methods in Applied Mechanics and Engineering*, 429:117192, September 2024. ISSN 0045-7825. doi: 10.1016/j.cma.2024.117192. URL https://www.sciencedirect.com/science/article/pii/S0045782524004481.

[40] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating Propagation Failures in Physics-informed Neural Networks using Retain-Resample-Release (R3) Sampling. In *Proceedings of the 40th International Conference on Machine Learning*, pages 7264–7302. PMLR, July 2023. URL https://proceedings.mlr.press/v202/daw23a.html.

[41] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge, January 2018. URL http://arxiv.org/abs/1711.07970.

[42] Michael Deistler, Jan Boelts, Peter Steinbach, Guy Moss, Thomas Moreau, Manuel Gloeckler, Pedro L. C. Rodrigues, Julia Linhart, Janne K. Lappalainen, Benjamin Kurt Miller, Pedro J. Gonçalves, Jan-Matthis Lueckmann, Cornelius

Schröder, and Jakob H. Macke. Simulation-Based Inference: A Practical Guide, August 2025. URL http://arxiv.org/abs/2508.12939.

[43] Waleed Diab and Mohammed Al Kobaisi. Temporal neural operator for modeling time-dependent physical phenomena. *Sci Rep*, 15:32791, September 2025. ISSN 2045-2322. doi: 10.1038/s41598-025-16922-5. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC12464229/.

[44] Alexander Diel, Tania Lalgi, Isabel Carolin Schröter, Karl F. MacDorman, Martin Teufel, and Alexander Bäuerle. Human performance in detecting deepfakes: A systematic review and meta-analysis of 56 papers. *Computers in Human Behavior Reports*, 16:100538, December 2024. ISSN 2451-9588. doi: 10.1016/j.chbr.2024.100538. URL https://www.sciencedirect.com/science/article/pii/S2451958824001714.

[45] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. URL http://arxiv.org/abs/2010.11929.

[46] Sofya Dymchenko and Bruno Raffin. Loss-driven sampling within hard-to-learn areas for simulation-based neural network training. In *MLPS 2023 - Machine Learning and the Physical Sciences Workshop at NeurIPS 2023 - 37th Conference on Neural Information Processing Systems*, pages 1–5, New Orleans, United States, December 2023. URL https://hal.science/hal-04305233.

[47] Sofya Dymchenko, Abhishek Purandare, and Bruno Raffin. MelissaDL x Breed: Towards Data-Efficient On-line Supervised Training of Multi-parametric Surrogates with Active Learning, October 2024. URL http://arxiv.org/abs/2410.05860.

[48] Víctor Elvira and Émilie Chouzenoux. Optimized Population Monte Carlo. *IEEE Trans. Signal Process.*, 70:2489–2501, 2022. ISSN 1053-587X, 1941-0476. doi: 10.1109/TSP.2022.3172619. URL http://arxiv.org/abs/2204.06891.

[49] Mohammad Sadegh Eshaghi, Cosmin Anitescu, Navid Valizadeh, Yizheng Wang, Xiaoying Zhuang, and Timon Rabczuk. NOWS: Neural Operator Warm Starts for Accelerating Iterative Solvers, November 2025. URL http://arxiv.org/abs/2511.02481.

[50] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, second edition, 2010.

[51] Geir Evensen. The Ensemble Kalman Filter: Theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, November 2003. ISSN 1616-7341. doi: 10.1007/s10236-003-0036-9. URL https://ui.adsabs.harvard.edu/abs/2003OcDyn..53..343E.

[52] V. S. Fanaskov and I. V. Oseledets. Spectral Neural Operators. *Dokl. Math.*, 108 (2):S226–S232, December 2023. ISSN 1531-8362. doi: 10.1134/S1064562423701107. URL https://doi.org/10.1134/S1064562423701107.

[53] Mohsen Fayyaz, Ehsan Aghazadeh, Ali Modarressi, Mohammad Taher Pilehvar, Yadollah Yaghoobzadeh, and Samira Ebrahimi Kahou. BERT on a Data Diet: Finding Important Examples by Gradient-Based Pruning. In *NeurIPS*, 2022.

[54] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting Fundamentals of Experience Replay. 2020.

[55] Joel H. Ferziger, Milovan Perić, and Robert L. Street. *Computational Methods for Fluid Dynamics*. Springer International Publishing, Cham, 2020. ISBN 978-3-319-99691-2 978-3-319-99693-6. doi: 10.1007/978-3-319-99693-6. URL http://link.springer.com/10.1007/978-3-319-99693-6.

[56] R. Fisher. *The Design Of Experiments*. 1935. URL http://archive.org/details/in.ernet.dli.2015.502684.

[57] Adam Foster, Martin Jankowiak, Eli Bingham, Paul Horsfall, Yee Whye Teh, Tom Rainforth, and Noah Goodman. Variational Bayesian Optimal Experimental Design, January 2020. URL http://arxiv.org/abs/1903.05480.

[58] Adam Foster, Desi R. Ivanova, Ilyas Malik, and Tom Rainforth. Deep Adaptive Design: Amortizing Sequential Bayesian Experimental Design. March 2021. URL http://arxiv.org/abs/2103.02438.

[59] Sebastian Friedemann and Bruno Raffin. An elastic framework for ensemble-based large-scale data assimilation, November 2020. URL http://arxiv.org/abs/2011.11635.

[60] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling Machine Learning Programs via High-Level Tracing. In *SysML Conference 2018*, Stanford, United States, March 2019. URL https://hal.science/hal-05188750.

[61] Qiang Gao, Zhipeng Luo, and Diego Klabjan. Efficient Architecture Search for Continual Learning, June 2020. URL http://arxiv.org/abs/2006.04027.

[62] Wenhan Gao and Chunmei Wang. Active Learning Based Sampling for High-Dimensional Nonlinear Partial Differential Equations, December 2021. URL http://arxiv.org/abs/2112.13988.

[63] Zhiwei Gao, Tao Tang, Liang Yan, and Tao Zhou. Failure-informed adaptive sampling for PINNs, Part II: Combining with re-sampling and subset simulation, February 2023. URL http://arxiv.org/abs/2302.01529.

[64] Manuel Gloeckler, Michael Deistler, Christian Weilbach, Frank Wood, and Jakob H. Macke. All-in-one simulation-based inference, July 2024. URL http://arxiv.org/abs/2404.09636.

[65] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, October 2016. ISBN 978-0-262-03561-3.

[66] Bill Goodwine. *Engineering Differential Equations: Theory and Applications*. Springer New York, New York, NY, 2011. ISBN 978-1-4419-7918-6 978-1-4419-7919-3. doi: 10.1007/978-1-4419-7919-3. URL https://link.springer.com/10.1007/978-1-4419-7919-3.

[67] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-Informed Deep Neural Operator Networks, July 2022. URL http://arxiv.org/abs/2207.05748.

[68] Sebastian G. Gruber and Florian Buettner. Uncertainty Estimates of Predictions via a General Bias-Variance Decomposition, April 2023. URL http://arxiv.org/abs/2210.12256.

[69] Quentin Guilloteau, Raphaël Bleuse, Sophie Cerf, Bogdan Robu, Rosa Pagano, and Éric Rutten. Autonomic Resource Harvesting in HPC: Control Methods and their Reusability. *ACM Trans. Auton. Adapt. Syst.*, September 2025. ISSN 1556-4665. doi: 10.1145/3764590. URL https://dl.acm.org/doi/10.1145/3764590.

[70] Yulin Guo, Paromita Nath, Sankaran Mahadevan, and Paul Witherell. Active learning for adaptive surrogate model improvement in high-dimensional problems. *Struct Multidisc Optim*, 67(7):122, July 2024. ISSN 1615-1488. doi: 10.1007/s00158-024-03816-9. URL https://doi.org/10.1007/s00158-024-03816-9.

[71] Jayesh K. Gupta and Johannes Brandstetter. Towards Multi-spatiotemporal-scale Generalized PDE Modeling, November 2022. URL http://arxiv.org/abs/2209.15616.

[72] Jan Hagnberger, Daniel Musekamp, and Mathias Niepert. CALM-PDE: Continuous and Adaptive Convolutions for Latent Space Modeling of Time-dependent PDEs, May 2025. URL http://arxiv.org/abs/2505.12944.

[73] Ali Hatamizadeh, Jiaming Song, Guilin Liu, Jan Kautz, and Arash Vahdat. DiffiT: Diffusion Vision Transformers for Image Generation, August 2024. URL http://arxiv.org/abs/2312.02139.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. URL http://arxiv.org/abs/1512.03385.

[75] Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient Foundation Models for PDEs, November 2024. URL http://arxiv.org/abs/2405.19101.

[76] Pieter Hintjens. ZeroMQ: Messaging for Many Applications. March 2013.

[77] Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance, July 2022. URL http://arxiv.org/abs/2207.12598.

[78] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020. URL http://arxiv.org/abs/2006.11239.

[79] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models, March 2022. URL http://arxiv.org/abs/2203.15556.

[80] David Holzmüller, Viktor Zaverkin, Johannes Kästner, and Ingo Steinwart. A Framework and Benchmark for Deep Batch Active Learning for Regression, August 2023. URL http://arxiv.org/abs/2203.09410.

[81] Benjamin Holzschuh, Georg Kohl, Florian Redinger, and Nils Thuerey. P3D: Scalable Neural Surrogates for High-Resolution 3D Physics Simulations with Global Context, October 2025. URL http://arxiv.org/abs/2509.10186.

[82] Benjamin Holzschuh, Qiang Liu, Georg Kohl, and Nils Thuerey. PDE-Transformer: Efficient and Versatile Transformers for Physics Simulations, May 2025. URL http://arxiv.org/abs/2505.24717.

[83] Zheyuan Hu, Qianying Cao, Kenji Kawaguchi, and George Em Karniadakis. DeepOMamba: State-space model for spatio-temporal PDE neural operator learning. *Journal of Computational Physics*, 540:114272, November 2025. ISSN 0021-9991. doi: 10.1016/j.jcp.2025.114272. URL https://www.sciencedirect.com/science/article/pii/S0021999125005558.

[84] Zheyuan Hu, Nazanin Ahmadi Daryakenari, Qianli Shen, Kenji Kawaguchi, and George Em Karniadakis. State-space models are accurate and efficient neural operators for dynamical systems, January 2025. URL http://arxiv.org/abs/2409.03231.

[85] Siyu Huang, Tianyang Wang, Haoyi Xiong, Jun Huan, and Dejing Dou. Semi-Supervised Active Learning with Temporal Output Discrepancy, July 2021. URL http://arxiv.org/abs/2107.14153.

[86] Dominik Huber, Martin Schreiber, Martin Schulz, Howard Pritchard, and Daniel Holmes. Design Principles of Dynamic Resource Management for High-Performance Parallel Programming Models, March 2024. URL http://arxiv.org/abs/2403.17107.

[87] Desi R. Ivanova, Adam Foster, Steven Kleinegesse, Michael U. Gutmann, and Tom Rainforth. Implicit Deep Adaptive Design: Policy-Based Experimental Design without Likelihoods, November 2021. URL http://arxiv.org/abs/2111.02329.

[88] A. Jameson. Aerodynamic Shape Optimization Using the Adjoint Method. 2003. URL https://www.semanticscholar.org/paper/Aerodynamic-Shape-Optimization-Using-the-Adjoint-Jameson/2b66da3968c41e5f6984a1fc98329274c5d7aa5e.

[89] Md Hasibul Jamil, MD S. Q. Zulkar Nine, and Tevfik Kosar. EMLIO: Minimizing I/O Latency and Energy Consumption for Large-Scale AI Training. In *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC Workshops '25, pages 2022–2031, New York, NY, USA, November 2025. Association for Computing Machinery. ISBN 979-8-4007-1871-7. doi: 10.1145/3731599.3767566. URL https://dl.acm.org/doi/10.1145/3731599.3767566.

[90] Edwin T. Jaynes. Foundations of Probability Theory and Statistical Mechanics. In Mario Bunge, editor, *Delaware Seminar in the Foundations of Physics*, pages

77–101, Berlin, Heidelberg, 1967. Springer. ISBN 978-3-642-86102-4. doi: 10.1007/978-3-642-86102-4_6.

[91] Yuling Jiao, Di Li, Xiliang Lu, Jerry Zhijian Yang, and Cheng Yuan. GAS: A Gaussian Mixture Distribution-Based Adaptive Sampling Method for PINNs, April 2023. URL http://arxiv.org/abs/2303.15849.

[92] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N. Cohen, and Adrian Weller. Synthetic Data – what, why and how?, May 2022. URL http://arxiv.org/abs/2205.03257.

[93] Guillaume Jouvet. Inversion of a Stokes glacier flow model emulated by deep learning. *Journal of Glaciology*, 69(273):13–26, February 2023. ISSN 0022-1430, 1727-5652. doi: 10.1017/jog.2022.41. URL https://www.cambridge.org/core/journals/journal-of-glaciology/article/inversion-of-a-stokes-glacier-flow-model-emulated-by-deep-learning/40A3BBE3E1747E1D93B37C57268748DF.

[94] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2. URL https://www.nature.com/articles/s41586-021-03819-2.

[95] A. Kalogirou, E. E. Keaveny, and D. T. Papageorgiou. An in-depth numerical study of the two-dimensional Kuramoto–Sivashinsky equation. *Proc Math Phys Eng Sci*, 471(2179):20140932, July 2015. ISSN 1364-5021. doi: 10.1098/rspa.2014.0932. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC4528647/.

[96] Richard M. Karp. On-Line Algorithms Versus Off-Line Algorithms: How Much is it Worth to Know the Future? In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*, pages 416–429, NLD, September 1992. North-Holland Publishing Co. ISBN 978-0-444-89747-3.

[97] Angelos Katharopoulos and François Fleuret. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. 2018.

[98] Kenji Kawaguchi and Haihao Lu. Ordered SGD: A New Stochastic Optimization Framework for Empirical Risk Minimization, February 2020. URL http://arxiv.org/abs/1907.04371.

[99] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring Catastrophic Forgetting in Neural Networks, November 2017. URL http://arxiv.org/abs/1708.02072.

[100] Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum*, 38(2):59–70, May 2019. ISSN 0167-7055, 1467-8659. doi: 10.1111/cgf.13619. URL http://arxiv.org/abs/1806.02071.

[101] Yegon Kim, Hyunsu Kim, Gyeonghoon Ko, and Juho Lee. Active Learning with Selective Time-Step Acquisition for PDEs. In *Forty-Second International Conference on Machine Learning*, June 2025.

[102] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980.

[103] Matthieu Kirchmeyer, Yuan Yin, Jérémie Donà, Nicolas Baskiotis, Alain Rakotomamonjy, and Patrick Gallinari. Generalizing to New Physical Systems via Context-Informed Dynamics Model, June 2022. URL http://arxiv.org/abs/2202.01889.

[104] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. U.S.A.*, 114(13):3521–3526, March 2017. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1611835114. URL http://arxiv.org/abs/1612.00796.

[105] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning, October 2019. URL http://arxiv.org/abs/1906.08158.

[106] Andreas Kirsch, Sebastian Farquhar, Parmida Atighehchian, Andrew Jesson, Frederic Branchaud-Charron, and Yarin Gal. Stochastic Batch Acquisition: A Simple Baseline for Deep Active Learning, September 2023. URL http://arxiv.org/abs/2106.12059.

[107] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, August 2019. ISSN 19393539. doi: 10.1109/tpami.2020.2992934. URL https://arxiv.org/abs/1908.09257v4.

[108] Felix Koehler, Simon Niedermayr, Rüdiger Westermann, and Nils Thuerey. APEBench: A Benchmark for Autoregressive Neural Emulators of PDEs, October 2024. URL http://arxiv.org/abs/2411.00180.

[109] Ron Kohavi and D. Wolpert. Bias Plus Variance Decomposition for Zero-One Loss Functions. In *International Conference on Machine Learning*, July 1996.

[110] D. J. Korteweg and G. De Vries. XLI. *On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 39(240): 422–443, May 1895. ISSN 1941-5982, 1941-5990. doi: 10.1080/14786449508620739. URL https://www.tandfonline.com/doi/full/10.1080/14786449508620739.

[111] Armand Kassaï Koupaï, Jorge Mifsut Benet, Yuan Yin, Jean-Noël Vittaut, and Patrick Gallinari. GEPS: Boosting generalization in parametric PDE neural solvers through adaptive conditioning, November 2024. URL http://arxiv.org/abs/2410.23889.

[112] Nikola B. Kovachki, Zong-Yi Li, Burigede Liu, K. Azizzadenesheli, K. Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces. *arXiv.org*, 2021.

[113] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks, November 2021. URL http://arxiv.org/abs/2109.01050.

[114] Punit Kumar and Atul Gupta. Active Learning Query Strategies for Classification, Regression, and Clustering: A Survey. *jcst*, 35(4):913–945, July 2020. ISSN 1000-9000(Print) /1860-4749(Online). doi: 10.1007/s11390-020-9487-4. URL https://jcst.ict.ac.cn/en/article/doi/10.1007/s11390-020-9487-4.pdf.

[115] Seong Min Kye, Kwanghee Choi, and Buru Chang. TiDAL: Learning Training Dynamics for Active Learning. 2022. doi: 10.48550/ARXIV.2210.06788. URL https://arxiv.org/abs/2210.06788.

[116] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles, November 2017. URL http://arxiv.org/abs/1612.01474.

[117] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. GraphCast: Learning skillful medium-range global weather forecasting, August 2023. URL http://arxiv.org/abs/2212.12794.

[118] Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green algorithms: quantifying the carbon footprint of computation. *Advanced science*, 8(12):2100707, 2021.

[119] Gregory Kang Ruey Lau, Apivich Hemachandra, See-Kiong Ng, and Bryan Kian Hsiang Low. PINNACLE: PINN Adaptive ColLocation and Experimental points selection, April 2024. URL http://arxiv.org/abs/2404.07662.

[120] Alexander Lavin, Hector Zenil, Brooks Paige, David Krakauer, Justin Gottschlich, Tim Mattson, Anima Anandkumar, Sanjay Choudry, Kamil Rocki, Atılım Güneş Baydin, Carina Prunkl, Brooks Paige, Olexandr Isayev, Erik Peterson, Peter L. McMahon, Jakob Macke, Kyle Cranmer, Jiaxin Zhang, Haruko Wainwright, Adi Hanuka, Manuela Veloso, Samuel Assefa, Stephan Zheng, and Avi Pfeffer. Simulation Intelligence: Towards a New Generation of Scientific Methods. December 2021. URL http://arxiv.org/abs/2112.03235.

[121] JunKyu Lee, Lev Mukhanov, Amir Sabbagh Molahosseini, Umar Minhas, Yang Hua, Jesus Martinez del Rincon, Kiril Dichev, Cheol-Ho Hong, and Hans Vandierendonck. Resource-Efficient Deep Learning: A Survey on Model-, Arithmetic-,

and Implementation-Level Techniques. *ACM Comput. Surv.*, 55(13s):1–36, December 2023. ISSN 0360-0300, 1557-7341. doi: 10.1145/3587095. URL http://arxiv.org/abs/2112.15131.

[122] Noah Lewis, Jean Luca Bez, and Suren Byna. I/O in Machine Learning Applications on HPC Systems: A 360-degree Survey. *ACM Comput. Surv.*, 57(10):256:1–256:41, May 2025. ISSN 0360-0300. doi: 10.1145/3722215. URL https://dl.acm.org/doi/10.1145/3722215.

[123] Dongyuan Li, Zhen Wang, Yankai Chen, Renhe Jiang, Weiping Ding, and Manabu Okumura. A Survey on Deep Active Learning: Recent Advances and New Frontiers, July 2024. URL http://arxiv.org/abs/2405.00334.

[124] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670, August 2014. doi: 10.1145/2623330.2623612. URL https://dl.acm.org/doi/10.1145/2623330.2623612.

[125] Shibo Li, Xin Yu, Wei Xing, Mike Kirby, Akil Narayan, and Shandian Zhe. Multi-Resolution Active Learning of Fourier Neural Operators, March 2024. URL http://arxiv.org/abs/2309.16971.

[126] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Graph Kernel Network for Partial Differential Equations, March 2020. URL http://arxiv.org/abs/2003.03485.

[127] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, May 2021. URL http://arxiv.org/abs/2010.08895.

[128] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for PDEs on general geometries. *J. Mach. Learn. Res.*, 24(1):388:18593–388:18618, January 2023. ISSN 1532-4435.

[129] Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-Informed Neural Operator for Large-Scale 3D PDEs, September 2023. URL http://arxiv.org/abs/2309.00583.

[130] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations, July 2023. URL http://arxiv.org/abs/2111.03794.

[131] Julia Linhart, Alexandre Gramfort, and Pedro L. C. Rodrigues. Validation Diagnostics for SBI algorithms based on Normalizing Flows, November 2022. URL http://arxiv.org/abs/2211.09602.

[132] Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E. Turner, and Johannes Brandstetter. PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers, October 2023. URL http://arxiv.org/abs/2308.05732.

[133] Chengyuan Liu, Yangyang Kang, Fubang Zhao, Kun Kuang, Zhuoren Jiang, Changlong Sun, and Fei Wu. Evolving Knowledge Distillation with Large Language Models and Active Learning, March 2024. URL http://arxiv.org/abs/2403.06414.

[134] Qiang Liu, Mengyu Chu, and Nils Thuerey. ConFIG: Towards Conflict-free Training of Physics Informed Neural Networks, March 2025. URL http://arxiv.org/abs/2408.11104.

[135] Ilya Loshchilov and Frank Hutter. Online Batch Selection for Faster Training of Neural Networks, April 2016. URL http://arxiv.org/abs/1511.06343.

[136] Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nat Mach Intell*, 3(3):218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://arxiv.org/abs/1910.03193.

[137] Lu Lu, Xuhui Meng, Zhiping Mao, and George E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Rev.*, 63(1):208–228, January 2021. ISSN 0036-1445, 1095-7200. doi: 10.1137/19M1274067. URL http://arxiv.org/abs/1907.04502.

[138] Lu Lu, Raphael Pestourie, Steven G. Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport, April 2022. URL http://arxiv.org/abs/2204.06684.

[139] Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H. Macke. Likelihood-free inference with emulator networks, May 2019. URL http://arxiv.org/abs/1805.09294.

[140] J. L. Lumley. The structure of inhomogeneous turbulent flows. *Atmospheric turbulence and radio propagation*, pages 166–178, 1967.

[141] Scott A. Martin, Georgy E. Manucharyan, and Patrice Klein. Generative Data Assimilation for Surface Ocean State Estimation From Multi-Modal Satellite Observations. *Journal of Advances in Modeling Earth Systems*, 17(8):e2025MS005063, 2025. ISSN 1942-2466. doi: 10.1029/2025MS005063. URL https://onlinelibrary.wiley.com/doi/abs/10.1029/2025MS005063.

[142] Simon Martina-Perez, Matthew J. Simpson, and Ruth E. Baker. Bayesian uncertainty quantification for data-driven equation learning. *Proc. A*, 477(2254): 20210426, October 2021. ISSN 1364-5021. doi: 10.1098/rspa.2021.0426. URL https://doi.org/10.1098/rspa.2021.0426.

[143] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Vanessa Parli, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, and Jack Clark. Artificial Intelligence Index Report 2024, May 2024. URL http://arxiv.org/abs/2405.19522.

[144] Michael McCabe, Bruno Régaldo-Saint Blancard, Liam Holden Parker, Ruben Ohana, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Siavash Golkar, Geraud Krawezik, Francois Lanusse, Mariel Pettee, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. Multiple Physics Pretraining for Physical Surrogate Models, December 2024. URL http://arxiv.org/abs/2310.02994.

[145] Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Regaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze W. K. Wong, Hadi Sotoudeh, Alberto Bietti, Irina Espejo, Rio Fear, Siavash Golkar, Tom Hehir, Keiya Hirashima, Geraud Krawezik, Francois Lanusse, Rudy Morel, Ruben Ohana, Liam Parker, Mariel Pettee, Jeff Shen, Kyunghyun Cho, Miles Cranmer, and Shirley Ho. Walrus: A Cross-Domain Foundation Model for Continuum Dynamics, November 2025. URL http://arxiv.org/abs/2511.15684.

[146] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

[147] Lucas Meyer. *Online Deep Learning for Numerical Simulations at Scale*. Theses, Université Grenoble Alpes [2020-....], March 2024. URL https://theses.hal.science/tel-04606849.

[148] Lucas Thibaut Meyer, Marc Schouler, Robert Alexander Caulk, Alejandro Ribes, and Bruno Raffin. High Throughput Training of Deep Surrogates from Large Ensemble Runs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, pages 1–16, New York, NY, USA, November 2023. Association for Computing Machinery. ISBN 979-8-4007-0109-2. doi: 10.1145/3581784.3607083. URL https://dl.acm.org/doi/10.1145/3581784.3607083.

[149] Lucas Thibaut Meyer, Marc Schouler, Robert Alexander Caulk, Alejandro Ribes, and Bruno Raffin. Training Deep Surrogate Models with Large Scale Online Learning. In *Proceedings of the 40th International Conference on Machine Learning*, pages 24614–24630. PMLR, July 2023. URL https://proceedings.mlr.press/v202/meyer23b.html.

[150] Sören Mindermann, Jan Brauner, Muhammed Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N. Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized Training on Points that are Learnable, Worth Learning, and Not Yet Learnt, September 2022. URL http://arxiv.org/abs/2206.07137.

[151] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, February 1997. ISBN 978-0-07-042807-2.

[152] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. Analyzing and Mitigating Data Stalls in DNN Training, January 2021. URL http://arxiv.org/abs/2007.06775.

[153] Julien Monniot, François Tessier, Henri Casanova, and Gabriel Antoniu. Simulation of Large-Scale HPC Storage Systems: Challenges and Methodologies. In *2024 IEEE 31st International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 232–242, December 2024. doi: 10.1109/HiPC62374.2024.00031. URL https://ieeexplore.ieee.org/document/10884205.

[154] Rudy Morel, Jiequn Han, and Edouard Oyallon. DISCO: Learning to DISCover an evolution Operator for multi-physics-agnostic prediction, April 2025. URL http://arxiv.org/abs/2504.19496.

[155] Sepehr Mousavi, Shizheng Wen, Levi Lingsch, Maximilian Herde, Bogdan Raonić, and Siddhartha Mishra. RIGNO: A Graph-based framework for robust and accurate operator learning for PDEs on arbitrary domains, October 2025. URL http://arxiv.org/abs/2501.19205.

[156] Johannes Müller and Marius Zeinhofer. Achieving High Accuracy with PINNs via Energy Natural Gradient Descent. In *Proceedings of the 40th International Conference on Machine Learning*, pages 25471–25485. PMLR, July 2023. URL https://proceedings.mlr.press/v202/muller23b.html.

[157] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, July 2012. ISBN 978-0-262-01802-9.

[158] Daniel Musekamp, Marimuthu Kalimuthu, David Holzmüller, Makoto Takamoto, and Mathias Niepert. Active Learning for Neural PDE Solvers, March 2025. URL http://arxiv.org/abs/2408.01536.

[159] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *arXiv: Learning*, April 2021.

[160] James C. Newman, Arthur C. Taylor, Richard W. Barnwell, Perry A. Newman, and Gene J.-W. Hou. Overview of Sensitivity Analysis and Shape Optimization for Complex Aerodynamic Configurations. *Journal of Aircraft*, 36(1):87–96, January 1999. ISSN 0021-8669, 1533-3868. doi: 10.2514/2.2416. URL https://arc.aiaa.org/doi/10.2514/2.2416.

[161] Ben Noordijk, Monica L. Garcia Gomez, Kirsten H. W. J. ten Tusscher, Dick de Ridder, Aalt D. J. van Dijk, and Robert W. Smith. The rise of scientific machine learning: A perspective on combining mechanistic modelling with machine learning for systems biology. *Front. Syst. Biol.*, 4, August 2024. ISSN 2674-0702. doi: 10.3389/fsysb.2024.1407994. URL https://www.frontiersin.org/journals/systems-biology/articles/10.3389/fsysb.2024.1407994/full.

[162] Ruben Ohana, Michael McCabe, Lucas Meyer, Rudy Morel, Fruzsina J. Agocs, Miguel Beneitez, Marsha Berger, Blakesley Burkhart, Keaton Burns, Stuart B. Dalziel, Drummond B. Fielding, Daniel Fortunato, Jared A. Goldberg, Keiya Hirashima, Yan-Fei Jiang, Rich R. Kerswell, Suryanarayana Maddu, Jonah Miller, Payel Mukhopadhyay, Stefan S. Nixon, Jeff Shen, Romain Watteaux, Bruno Régaldo-Saint Blancard, François Rozet, Liam H. Parker, Miles Cranmer, and Shirley Ho. The Well: A Large-Scale Collection of Diverse Physics Simulations for Machine Learning, February 2025. URL http://arxiv.org/abs/2412.00568.

[163] Soumyasundar Pal, Antonios Valkanas, and Mark Coates. Population Monte Carlo with Normalizing Flow, December 2023. URL http://arxiv.org/abs/2312.03857.

[164] George Papamakarios, David C Sterratt, and Iain Murray. Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows. 2019.

[165] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *Journal of Machine Learning Research*, 22:1–64, 2021. URL http://jmlr.org/papers/v22/19-1028.html.

[166] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019. URL http://arxiv.org/abs/1912.01703.

[167] Wei Peng, Weien Zhou, Xiaoya Zhang, Wen Yao, and Zheliang Liu. RANG: A Residual-based Adaptive Node Generation Method for Physics-Informed Neural Networks, May 2022. URL http://arxiv.org/abs/2205.01051.

[168] Raphaël Pestourie, Youssef Mroueh, Thanh V. Nguyen, Payel Das, and Steven G. Johnson. Active learning of deep surrogates for PDEs: Application to metasurface design. *npj Comput Mater*, 6(1), October 2020. ISSN 2057-3960. doi: 10.1038/s41524-020-00431-2. URL https://www.nature.com/articles/s41524-020-00431-2.

[169] Raphaël Pestourie, Youssef Mroueh, Chris Rackauckas, Payel Das, and Steven G. Johnson. Physics-enhanced deep surrogates for partial differential equations. *Nat Mach Intell*, 5(12):1458–1465, December 2023. ISSN 2522-5839. doi: 10.1038/s42256-023-00761-y. URL http://arxiv.org/abs/2111.05841.

[170] Konstantin F. Pilz, James Sanders, Robi Rahman, and Lennart Heim. Trends in AI Supercomputers, April 2025. URL http://arxiv.org/abs/2504.16026.

[171] Sarunya Pumma, Min Si, Wu-Chun Feng, and Pavan Balaji. Scalable Deep Learning via I/O Analysis and Optimization. *ACM Trans. Parallel Comput.*, 6(2):6:1–6:34, July 2019. ISSN 2329-4949. doi: 10.1145/3331526. URL https://dl.acm.org/doi/10.1145/3331526.

[172] Christopher Rackauckas and Qing Nie. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Discrete and continuous dynamical systems. Series B*, 22(7):2731, 2017.

[173] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[174] Md Ashiqur Rahman, Zachary E. Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped Neural Operators, May 2023. URL http://arxiv.org/abs/2204.11127.

[175] Md Ashiqur Rahman, Robert Joseph George, Mogab Elleithy, Daniel Leibovici, Zongyi Li, Boris Bonev, Colin White, Julius Berner, Raymond A. Yeh, Jean Kossaifi, Kamyar Azizzadenesheli, and Anima Anandkumar. Pretraining Codomain Attention Neural Operators for Solving Multiphysics PDEs, November 2024. URL http://arxiv.org/abs/2403.12553.

[176] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[177] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation, February 2021. URL http://arxiv.org/abs/2102.12092.

[178] Bogdan Raonić, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bézenac. Convolutional Neural Operators for robust and accurate learning of PDEs, December 2023. URL http://arxiv.org/abs/2302.01178.

[179] Stephan Rasp, Michael S. Pritchard, and Pierre Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, September 2018. doi: 10.1073/pnas.1810286115. URL https://www.pnas.org/doi/full/10.1073/pnas.1810286115.

[180] Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in Training PINNs: A Loss Landscape Perspective, June 2024. URL http://arxiv.org/abs/2402.01868.

[181] Andrin Rehmann, Nolan Black, Josiah Bjorgaard, Alessandro Angioi, Andrei Paleyes, Niklas Heim, Dion Häfner, and Alexander Lavin. Surrogate-Based Differentiable Pipeline for Shape Optimization, November 2025. URL http://arxiv.org/abs/2511.10761.

[182] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer, New York, NY, 2004. ISBN 978-1-4419-1939-7 978-1-4757-4145-2. doi: 10.1007/978-1-4757-4145-2. URL http://link.springer.com/10.1007/978-1-4757-4145-2.

[183] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning Internal Representations by Error Propagation. In *Readings in Cognitive Science*, pages 399–421. Elsevier, 1988. ISBN 978-1-4832-1446-7. doi: 10.1016/B978-1-4832-1446-7.50035-2. URL https://linkinghub.elsevier.com/retrieve/pii/B9781483214467500352.

[184] Elizabeth G. Ryan, Christopher C. Drovandi, James M. McGree, and Anthony N. Pettitt. A Review of Modern Computational Algorithms for Bayesian Optimal Design. *International Statistical Review / Revue Internationale de Statistique*, 84(1): 128–154, 2016. ISSN 0306-7734. URL https://www.jstor.org/stable/44162464.

[185] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *ICML'20*, pages 8459–8468. JMLR.org, July 2020.

[186] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay, February 2016. URL http://arxiv.org/abs/1511.05952.

[187] Marc Schouler, Robert Alexander Caulk, Lucas Meyer, Théophile Terraz, Christoph Conrads, Sebastian Friedemann, Achal Agarwal, Juan Manuel Baldonado, Bartłomiej Pogodziński, Anna Sekuła, Alejandro Ribes, and Bruno Raffin. Melissa: Coordinating large-scale ensemble runs for deep learning and sensitivity analyses. *Journal of Open Source Software*, 8(86):5291, June 2023. ISSN 2475-9066. doi: 10.21105/joss.05291. URL https://joss.theoj.org/papers/10.21105/joss.05291.

[188] Nilo Schwencke and Cyril Furtlehner. ANaGRAM: A Natural Gradient Relative to Adapted Model for efficient PINNs learning, March 2025. URL http://arxiv.org/abs/2412.10782.

[189] Nilo Schwencke, Cyriaque Rousselot, Alena Shilova, and Cyril Furtlehner. AMStraMGRAM: Adaptive Multi-cutoff Strategy Modification for ANaGRAM, October 2025. URL http://arxiv.org/abs/2510.15998.

[190] Louis Serrano, Lise Le Boudec, Armand Kassaï Koupaï, Thomas X. Wang, Yuan Yin, Jean-Noël Vittaut, and Patrick Gallinari. Operator Learning with Neural Fields: Tackling PDEs on General Geometries, November 2023. URL http://arxiv.org/abs/2306.07266.

[191] Louis Serrano, Thomas X. Wang, Etienne Le Naour, Jean-Noël Vittaut, and Patrick Gallinari. AROMA: Preserving Spatial Structure for Latent PDE Modeling with Local Neural Fields, October 2024. URL http://arxiv.org/abs/2406.02176.

[192] Louis Serrano, Armand Kassaï Koupaï, Thomas X. Wang, Pierre Erbacher, and Patrick Gallinari. Zebra: In-Context Generative Pretraining for Solving Parametric PDEs, June 2025. URL http://arxiv.org/abs/2410.03437.

[193] Burr Settles. Active Learning Literature Survey. Technical Report, University of Wisconsin-Madison Department of Computer Sciences, 2009. URL https://minds.wisconsin.edu/handle/1793/60660.

[194] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 287–294. Publ by ACM, 1992. doi: 10.1145/130385.130417. URL https://collaborate.princeton.edu/en/publications/query-by-committee/.

[195] Jaime Sevilla and Edu Roldán. Training compute of frontier ai models grows by 4-5x per year, 2024. URL https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year. Accessed: 2026-02-24.

[196] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute Trends Across Three Eras of Machine Learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2022. doi: 10.1109/IJCNN55064.2022.9891914. URL http://arxiv.org/abs/2202.05924.

[197] Jingyi Shen, Yuhan Duan, and Han-Wei Shen. SurroFlow: A Flow-Based Surrogate Model for Parameter Space Exploration and Uncertainty Quantification. *IEEE Transactions on Visualization and Computer Graphics*, 31(1):635–644, January 2025. ISSN 1077-2626. doi: 10.1109/TVCG.2024.3456372. URL https://doi.org/10.1109/TVCG.2024.3456372.

[198] Cristina Silvano, Daniele Ielmini, Fabrizio Ferrandi, Leandro Fiorin, Serena Curzel, Luca Benini, Francesco Conti, Angelo Garofalo, Cristian Zambelli, Enrico Calore, Sebastiano Schifano, Maurizio Palesi, Giuseppe Ascia, Davide Patti, Nicola Petra, Davide De Caro, Luciano Lavagno, Teodoro Urso, Valeria Cardellini, Gian Carlo Cardarilli, Robert Birke, and Stefania Perri. A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms. *ACM Comput. Surv.*, 57(11):286:1–286:39, June 2025. ISSN 0360-0300. doi: 10.1145/3729215. URL https://dl.acm.org/doi/10.1145/3729215.

[199] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961.

[200] Tvesha Sippy, Florence Enock, Jonathan Bright, and Helen Z. Margetts. Behind the Deepfake: 8% Create; 90% Concerned. Surveying public exposure to and perceptions of deepfakes in the UK, July 2024. URL http://arxiv.org/abs/2407.05529.

[201] Erich Strohmaier, Hans W. Meuer, Jack Dongarra, and Horst D. Simon. The TOP500 List and Progress in High-Performance Computing. *Computer*, 48(11):42–49, November 2015. ISSN 0018-9162, 1558-0814. doi: 10.1109/MC.2015.338. URL https://ieeexplore.ieee.org/document/7328648/.

[202] Unique Subedi and Ambuj Tewari. On the Benefits of Active Data Collection in Operator Learning, February 2025. URL http://arxiv.org/abs/2410.19725.

[203] Qun Sui and Sujit K. Ghosh. Similarity-based active learning methods. *Expert Systems with Applications*, 251:123849, October 2024. ISSN 0957-4174. doi: 10.1016/j.eswa.2024.123849. URL https://www.sciencedirect.com/science/article/pii/S0957417424007152.

[204] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, USA, 2 edition, November 2018. ISBN 978-0-262-03924-6. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

[205] Makoto Takamoto, Francesco Alesiani, and Mathias Niepert. Learning Neural PDE Solvers with Parameter-Guided Channel Attention, July 2023. URL http://arxiv.org/abs/2304.14118.

[206] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBENCH: An Extensive Benchmark for Scientific Machine Learning, August 2024. URL http://arxiv.org/abs/2210.07182.

[207] Ronak Tali, Ali Rabeh, Cheng-Hau Yang, Mehdi Shadkhah, Samundra Karki, Abhisek Upadhyaya, Suriya Dhakshinamoorthy, Marjan Saadati, Soumik Sarkar, Adarsh Krishnamurthy, Chinmay Hegde, Aditya Balu, and Baskar Ganapathysubramanian. FlowBench: A Large Scale Benchmark for Flow Simulation over Complex Geometries, September 2024. URL http://arxiv.org/abs/2409.18032.

[208] Hewei Tang, Qingkai Kong, and Joseph P. Morris. Multi-fidelity Fourier Neural Operator for Fast Modeling of Large-Scale Geological Carbon Storage, January 2024. URL http://arxiv.org/abs/2308.09113.

[209] Kejun Tang, Xiaoliang Wan, and Chao Yang. DAS-PINNs: A deep adaptive sampling method for solving high-dimensional partial differential equations, July 2022. URL http://arxiv.org/abs/2112.14038.

[210] Meng Tang, Yimin Liu, and Louis J. Durlofsky. A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. *Journal of Computational Physics*, 413:109456, July 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109456. URL https://www.sciencedirect.com/science/article/pii/S0021999120302308.

[211] Théophile Terraz, Alejandro Ribes, Yvan Fournier, Bertrand Iooss, and Bruno Raffin. Melissa: Large Scale In Transit Sensitivity Analysis Avoiding Intermediate Files. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing)*, page 1, November 2017. URL https://inria.hal.science/hal-01607479.

[212] The HDF Group. Hierarchical Data Format, version 5. URL https://github.com/HDFGroup/hdf5.

[213] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The Computational Limits of Deep Learning, July 2022. URL http://arxiv.org/abs/2007.05558.

[214] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian Fluid Simulation With Convolutional Networks, November 2022. URL http://arxiv.org/abs/1607.03597.

[215] Edgar Torres, Jonathan Schiefer, and Mathias Niepert. Adaptive Physics-informed Neural Networks: A Survey, March 2025. URL http://arxiv.org/abs/2503.18181.

[216] Juan Diego Toscano, Vivek Oommen, Alan John Varghese, Zongren Zou, Nazanin Ahmadi Daryakenari, Chenxi Wu, and George Em Karniadakis. From PINNs to PIKANs: Recent Advances in Physics-Informed Machine Learning, October 2024. URL http://arxiv.org/abs/2410.13228.

[217] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized Fourier Neural Operators, March 2023. URL http://arxiv.org/abs/2111.13802.

[218] Jendrik-Alexander Tröger, Hamidreza Eivazi, Stefan Hartmann, Stefan Wittek, and Andreas Rausch. Efficient integration of deep neural networks in sequential multiscale simulations. *PAMM*, 23(4):e202300052, 2023. ISSN 1617-7061. doi: 10.1002/pamm.202300052. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.202300052.

[219] Kiwon Um, Robert Brand, Yun, Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers, January 2021. URL http://arxiv.org/abs/2007.00016.

[220] Harsh Vardhan, Umesh Timalsina, Peter Volgyesi, and Janos Sztipanovits. Data efficient surrogate modeling for engineering design: Ensemble-free batch mode deep active learning for regression, November 2022. URL http://arxiv.org/abs/2211.10360.

[221] Bruno Veloso, João Gama, Benedita Malheiro, and João Vinagre. Hyperparameter self-tuning for data streams. *Information Fusion*, 76:75–86, December 2021. ISSN 1566-2535. doi: 10.1016/j.inffus.2021.04.011. URL https://www.sciencedirect.com/science/article/pii/S1566253521000841.

[222] Tianjiao Wan, Kele Xu, Ting Yu, Xu Wang, Dawei Feng, Bo Ding, and Huaimin Wang. A Survey of Deep Active Learning for Foundation Models. *Intelligent Computing*, 2:0058, November 2023. doi: 10.34133/icomputing.0058. URL https://spj.science.org/doi/10.34133/icomputing.0058.

[223] Haonan Wang, Wei Huang, Ziwei Wu, Hanghang Tong, Andrew J. Margenot, and Jingrui He. Deep Active Learning by Leveraging Training Dynamics. In *Advances in Neural Information Processing Systems*, October 2022. URL https://openreview.net/forum?id=aJ5xc1QB7EX.

[224] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A Comprehensive Survey of Continual Learning: Theory, Method and Application, February 2024. URL http://arxiv.org/abs/2302.00487.

[225] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective, July 2020. URL http://arxiv.org/abs/2007.14527.

[226] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An Expert's Guide to Training Physics-informed Neural Networks, August 2023. URL http://arxiv.org/abs/2308.08468.

[227] Wei Wang, Maryam Hakimzadeh, Haihui Ruan, and Somdatta Goswami. Time-marching neural operator–FE coupling: AI-accelerated physics modeling. *Computer Methods in Applied Mechanics and Engineering*, 446:118319, November 2025. ISSN 0045-7825. doi: 10.1016/j.cma.2025.118319. URL https://www.sciencedirect.com/science/article/pii/S0045782525005912.

[228] Yongji Wang and Ching-Yao Lai. DIFFICE-jax: Differentiable neural-network solver for data assimilation of ice shelves in JAX. *Journal of Open Source Software*, 10(109):7254, May 2025. ISSN 2475-9066. doi: 10.21105/joss.07254. URL https://joss.theoj.org/papers/10.21105/joss.07254.

[229] Zhiyuan Wang, Jinwoo Go, Byung-Jun Yoon, Nathan Urban, and Xiaoning Qian. A Plug-and-Play Query Synthesis Active Learning Framework for Neural PDE Solvers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, October 2025. URL https://openreview.net/forum?id=uyJcF4cwMc.

[230] Logan Ward, Ganesh Sivaraman, J Gregory Pauloski, Yadu Babuji, Ryan Chard, Naveen Dandu, Paul C Redfern, Rajeev S Assary, Kyle Chard, Larry A Curtiss, Rajeev Thakur, and Ian Foster. Colmena: Scalable Machine-Learning-Based Steering of Ensemble Simulations for High Performance Computing. 2021.

[231] Theodore Weber, Austin Corotan, Brian Hutchinson, Ben Kravitz, and Robert Link. Technical note: Deep learning for creating surrogate models of precipitation in Earth system models. *Atmospheric Chemistry and Physics*, 20(4):2303–2317, February 2020. ISSN 1680-7316. doi: 10.5194/acp-20-2303-2020. URL https://acp.copernicus.org/articles/20/2303/2020/.

[232] Hao Wei, Aleksandra Franz, Bjoern List, and Nils Thuerey. INC: An Indirect Neural Corrector for Auto-Regressive Hybrid PDE Solvers, November 2025. URL http://arxiv.org/abs/2511.12764.

[233] Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry Aware Operator Transformer as an Efficient and Accurate Neural Surrogate for PDEs on Arbitrary Domains, October 2025. URL http://arxiv.org/abs/2505.18781.

[234] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems. *ACM Comput. Surv.*, 55(4):66:1–66:37, November 2022. ISSN 0360-0300. doi: 10.1145/3514228. URL https://dl.acm.org/doi/10.1145/3514228.

[235] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, July 2022. URL http://arxiv.org/abs/2207.10289.

[236] Dongxia Wu, Ruijia Niu, Matteo Chinazzi, Alessandro Vespignani, Yi-An Ma, and Rose Yu. Deep Bayesian Active Learning for Accelerating Stochastic Simulation, June 2023. URL http://arxiv.org/abs/2106.02770.

[237] Tailin Wu, Qinchen Wang, Yinan Zhang, Rex Ying, Kaidi Cao, Rok Sosič, Ridwan Jalali, Hassan Hamam, Marko Maucec, and Jure Leskovec. Learning Large-scale Subsurface Simulations with a Hybrid Graph Network Simulator. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data*

Mining, pages 4184–4194, August 2022. doi: 10.1145/3534678.3539045. URL http://arxiv.org/abs/2206.07680.

[238] Zhanghao Wu, Eric Liang, Michael Luo, Sven Mika, Joseph E. Gonzalez, and Ion Stoica. RLlib flow: Distributed reinforcement learning is a dataflow problem. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/2bce32ed409f5ebcee2a7b417ad9beed-Paper.pdf.

[239] Zhengyu Wu, Xunkai Li, Yinlin Zhu, Zekai Chen, Guochen Yan, Yanyu Yan, Hao Zhang, Yuming Ai, Xinmo Jin, Rong-Hua Li, and Guoren Wang. A Comprehensive Data-centric Overview of Federated Graph Learning, July 2025. URL http://arxiv.org/abs/2507.16541.

[240] Zijiang Yang, Zhongwei Qiu, and Dongmei Fu. DMIS: Dynamic Mesh-based Importance Sampling for Training Physics-Informed Neural Networks, November 2022. URL http://arxiv.org/abs/2211.13944.

[241] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-39727-4. doi: 10.1007/10968987_3.

[242] Donggeun Yoo and In So Kweon. Learning Loss for Active Learning, May 2019. URL http://arxiv.org/abs/1905.03677.

[243] Yuliana Zamora, Logan Ward, Ganesh Sivaraman, Ian Foster, and Henry Hoffmann. Proxima: Accelerating the integration of machine learning in atomistic simulations. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '21, pages 242–253, New York, NY, USA, June 2021. Association for Computing Machinery. ISBN 978-1-4503-8335-6. doi: 10.1145/3447818.3460370. URL https://dl.acm.org/doi/10.1145/3447818.3460370.

[244] L. Zanisi, A. Ho, T. Madula, J. Barr, J. Citrin, S. Pamela, J. Buchanan, F. Casson, V. Gopakumar, and J. E. T. contributors. Efficient training sets for surrogate models of tokamak turbulence with Active Deep Ensembles, October 2023. URL http://arxiv.org/abs/2310.09024.

[245] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. Experience Replay Optimization. 2019.

[246] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. Data-centric Artificial Intelligence: A Survey, June 2023. URL http://arxiv.org/abs/2303.10158.

[247] Hang Zhou, Yuezhou Ma, Haixu Wu, Haowen Wang, and Mingsheng Long. Unisolver: PDE-Conditional Transformers Are Universal PDE Solvers, October 2024. URL http://arxiv.org/abs/2405.17527.

[248] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation, January 2020. URL http://arxiv.org/abs/1912.05074.