

UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Hybrid Gaussian Process and Deep Learning Frameworks for Computational Science and Engineering

Permalink

<https://escholarship.org/uc/item/4zf6z83g>

ISBN

9798263338350

Author

Mora Sardina, Carlos

Publication Date

2025-11-20

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Hybrid Gaussian Process and Deep Learning Frameworks
for Computational Science and Engineering

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mechanical and Aerospace Engineering

by

Carlos Mora Sardina

Committee:

Associate Professor Ramin Bostanabad, Chair
Associate Professor Aparna Chandramowlishwaran
Professor Haithem Taha

2025

DEDICATION

To my family.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	v
LIST OF TABLES	ix
ACKNOWLEDGMENTS	xi
VITA	xii
ABSTRACT OF THE DISSERTATION	xiv
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Contributions and Outline	2
2 Solving Forward and Inverse Problems Involving PDEs	5
2.1 Introduction	5
2.2 Neural Networks with Kernel-Weighted Corrective Residuals	8
2.2.1 Theoretical Rationale	9
2.2.2 Proposed Framework	11
2.2.3 Model Characteristics and Extensions	13
2.3 Results and Discussion	16
2.3.1 Description of the Benchmark Problems	17
2.3.2 Implementation Details in Our Comparative Studies	19
2.3.3 Summary of Comparative Studies	22
2.3.4 Sensitivity Analyses	26
2.3.5 Loss Behavior	31
2.3.6 Inverse Problems	33
2.4 Concluding Remarks	35
3 Operator Learning in Parametric PDEs	37
3.1 Introduction	37
3.1.1 Description of the Operator Learning Problem	38
3.1.2 Summary of the Proposed Approach	38
3.1.3 Illustrative Example	40
3.1.4 Review of Related Literature	41

3.1.5	Contributions and Chapter Outline	45
3.2	Proposed Framework	46
3.2.1	Data-driven Operator Learning	46
3.2.2	Physics-Informed Operator Learning	56
3.2.3	Parameter Initialization and Stability	58
3.2.4	Comparison to Optimal Recovery-Based Operator Learning	65
3.2.5	Inference Complexity	67
3.3	Results and Discussion	68
3.3.1	Data-driven Operator Learning: Single-Output Operators	68
3.3.2	Data-driven Operator Learning: Multi-Output Operators	72
3.3.3	Physics-Informed Operator Learning	76
3.3.4	Training Cost	79
3.4	Concluding Remarks	80
4	Learning Nonstationary Kernels	82
4.1	Introduction	82
4.2	Methods	85
4.2.1	Gaussian Processes (GPs)	85
4.2.2	Validity of Kernels for GPs	87
4.2.3	Some Closure Properties on Kernels	88
4.2.4	SEEK: Our Proposed Kernel	90
4.2.5	Explainability of SEEK: Illustrative Example	92
4.3	Results and Discussion	94
4.3.1	Benchmark Problems and Metrics	95
4.3.2	Sensitivity Studies	99
4.3.3	Comparative Studies	104
4.3.4	Engineering Problems	106
4.3.5	Bayesian Optimization	108
4.4	Concluding Remarks	110
5	Conclusions	111
	Bibliography	114
	Appendix A Solving Forward and Inverse Problems Involving PDEs	130
	Appendix B Operator Learning in Parametric PDEs	145
	Appendix C Learning Nonstationary Kernels	148

LIST OF FIGURES

	Page
2.1 Flowchart of the proposed framework for solving the 1D Burgers' equation: We endow the solution $u(\mathbf{x})$ with a GP prior whose mean and covariance functions are parameterized via a deep NN and the Gaussian kernel in Equation (2.2), respectively. In module 1, we fix θ to some random values and estimate the kernel parameters via MLE or heuristics such that the posterior GP conditioned on the BC/IC data faithfully reproduces $\mathbf{u}$. Then, in Module 2, we estimate θ by minimizing a loss function which only depends on the PDE since BC/IC are automatically satisfied.	12
2.2 Solving the 2D incompressible Navier-Stokes equations for the lid-driven cavity problem: With minor architectural changes on module two with respect to Figure 2.1, our framework can also solve coupled PDE systems. Specifically, we endow each dependent variable with a GP prior. These GPs have independent kernels but a shared mean function that is parameterized via a deep neural network. Similar to Figure 2.1, the loss function only depends on the PDE residuals and excludes data loss terms on BC/IC.	14
2.3 Model features: (a) In addition to improving the optimization of θ in module two, kernel-weighted CoRes contribute to the model predictions and ensure strict satisfaction of the BCs. (b) Kernel-weighted CoRes automatically adapt to the domain geometry and are applicable to coupled PDE systems such as the Navier-Stokes equations. Here, we solve the unsteady LDC problem for $t \in [0, 5]$ and visualize the flow at $t = 0.5$ and $t = 5$	15
2.4 Histograms of PDE loss gradients: NN-CoRes is in general more effective in minimizing its loss function as a larger portion of its gradients satisfy the first order optimality condition. While PINN _{DW} has more near-zero gradients in the Eikonal problem, it does so at the expense of violating the BC loss term. All models in this figure have a $4 \otimes 20$ architecture.	23
2.5 Error analysis: NN-CoRes achieve smaller errors and increasing the size of their networks provides more improvement compared to methods such as PINN _{DW}	24
2.6 Reference solutions and error maps: NN-CoRes provide much lower errors compared to other methods (note the scales of the error bars).	25

2.7	Sensitivity studies: We analyze the sensitivity of our results to factors such as the roughness parameters in the kernel, optimization settings, network architecture, and initialization. Based on these experiments, NN-CoRes provide a more robust machine learning-based approach for solving different nonlinear PDEs.	27
2.8	Reference and predicted solutions with noisy boundary data: We corrupt the samples obtained from boundary and initial conditions by either 0.5% or 1% of the solution range. In all cases, the performance of NN-CoRes is insignificantly affected by the noise.	30
2.9	Loss convergence and error decomposition: NN-CoRes typically converge faster than PINNs and consistently provide more accurate solutions. The NN part of NN-CoRes benefits from the kernel-weighted CoRes not only on the boundaries, but also inside the domain.	31
2.10	Loss convergence and error decomposition for LDC: The NN part of NN-CoRes benefits from the kernel-weighted CoRes not only on the boundaries, but also inside the domain. In the case of pressure, kernel-weighted CoRes do not contribute to the model's predictions inside the domain as $p(\mathbf{x})$ is only known at a single point on the boundary (for this reason, the blue and green curves are indistinguishable).	32
2.11	Inverse problems via NN-CoRes: We modify the flowchart of module two in Figure 2.1 in two ways to solve a PDE system whose one or more parameters may be unknown. NN-CoRes treat observations (i.e., labeled solution data inside the domain) identically to boundary data and are very effective in using them in estimating the unknown PDE parameters.	34
3.1	Diagram of our framework for operator learning: We convert the operator learning problem to a regression one which can be solved via GPs. For multi-output operator learning, we use multi-response or multi-task GPs. . .	40
3.2	Data-driven and physics-informed operator learning for the Burgers' problem with Dirichlet BCs: The training dataset has 400 pairs of $\{u_i, v_i\}$ that satisfy Equation (3.5). The observation operators ϕ and ψ sample u_i and v_i at $p = 100$ and $q = 12^2$ collocation points, respectively. It can be observed that leveraging the physics reduces the prediction error. A DeepONet is used as the mean function of the GP in both data-driven and physics-informed cases.	42
3.3	Effect of β_y on test accuracy with dense observations: Very small and large β values provide poor performance as they cause numerical issues and cannot capture spatial correlations, respectively. However, a relatively large range of values such as 10^3 provide high accuracy and numerical stability. . .	59
3.4	Interaction between β and feature space dimensionality: Smaller β values are needed to achieve a desired correlation value in high-dimensional feature spaces.	61

3.5	Loss and error profiles as functions of kernel parameters in zero-mean GPs: A wide range of parameter combinations provide optimal loss and error values, indicating that proper initialization is relatively easy and results in good performance even without training. $\beta_y = 10^3$ is used for the two loss profiles on the left-hand side of (a) and (b). Note that we do not show the dependency of the test relative $L2$ error with respect to σ_ϕ^2 since this does not affect Equation (3.25). For Darcy, β_y has two components and so for plotting the error map we presume that these components are equal (a similar assumption is made for β_ϕ).	63
3.6	Parameter optimization in GPs: We consider three scenarios with a zero-mean GP in (a) and NN-mean GPs in (b) and (c). The differences in training settings across these three cases are detailed in the text. The relative $L2$ errors for cases (a), (b) and (c) are 64.77%, 60.38%, and 0.09%, respectively.	65
3.7	Example test inputs, outputs, and predictions: We provide one example from each problem where the predicted outputs correspond to the best version of our approach in Table 3.1, i.e., a zero-mean GP in the case of Advection problem and an FNO-mean GP for the other three problems.	73
3.8	Train and test data generation in the LDC problem: (a) The $[0, 1]^2$ domain is swept with a smaller window to collect data. (b) If the center of the window is above the dashed line, the sample is used to assess the extrapolation error. Otherwise, the sample is used either for training or obtaining the interpolation error.	74
3.9	Example input, output, and error maps on the velocity magnitudes in the LDC problem: The test samples in extrapolation look substantially different than those in interpolation (compare the scales of the velocity magnitude profiles in the two test outputs). Our GP-based approach provides smaller errors in both scenarios especially in the interpolation case.	76
3.10	Convergence studies: We study the effect of data set size N and augmenting the data with physics on the errors for various models in the case of Burgers' (left) and LDC (right) problems.	78
4.1	Schematic illustration of our kernel: SEEK has a set of weighted base kernels with learnable hyperparameters where the weights are learnable functions. These weighted kernels, along with a learnable bias term, are summed and then passed through an appropriate activation function to produce the final nonstationary covariance function.	91
4.2	Prediction of a GP using SEEK on the Analytic I problem, along with two of the learned weighted base kernels and the resulting covariance function evaluated at two reference points.	93
4.3	Sensitivity studies: We conduct four sensitivity studies on two benchmark problems to evaluate the impact of key design choices on the performance of SEEK. The results are based on 16 repetitions.	101

4.4	Our proposed kernel SEEK with (a) 1 and (b) 6 Gaussian kernels as base kernels on the Volcano dataset: Increasing the number of base kernels reduces error and improves confidence, as reflected in the colorbars (same ranges are used in (a) and (b) to facilitate direct comparison). Black dots represent the training data points.	103
4.5	Comparative studies: We compare SEEK against other nonstationary kernels on four benchmark problems with varying degrees of nonlinearity and dimensionality.	106
4.6	Engineering problems: We compare the performance of GP using SEEK versus a standard Gaussian kernel on three high-dimensional engineering problems.	107
4.7	Bayesian optimization: We compare the performance of GPs using SEEK or the standard Gaussian kernel on two of the benchmarks.	108

LIST OF TABLES

	Page
2.1 Summary of comparative studies: We report $L_{2,e}$ of different methods as a function of model capacity and PDE parameter. The symbol $\otimes$ indicates the network architecture (e.g., $4 \otimes 10$ is an NN which has four 10–neuron hidden layers). Unlike NN-based methods, GP _{OR} ’s accuracy relies on the number of interior nodes which we set to 1,000 or 2,000. GP _{OR} is not applied to LDC as it relies on manual derivation of the equivalent variational problem which, unlike the first three PDEs, is not done by the developers [1].	22
3.1 Summary results on four benchmark problems: We report the $L2$ relative test error for four different operator learning methods. The errors in the first three rows are directly taken from the literature [2, 3, 4]. In our approach, we set the mean function to either (1) zero in which case we also provide the error metrics without training and after one optimization iteration, or (2) DeepONet and FNO whose architectures are exactly the same as those in rows 1 and 2. In the case of Darcy, we follow the references and apply PCA to the discretized input function and keep the first 200 PCs as the features for the kernel part of our GP-based methods.	71
3.2 Summary of Results for Section 3.3.2: We use DeepONet and FNO with two architectures that differ significantly in their parameter counts, referred to as <i>small</i> and <i>large</i> , for comparison in both interpolation and extrapolation tests. In our approach, we either employ these neural operators as the mean function or utilize a zero-mean GP. For the former case, we train the GPs using maximum likelihood estimation (MLE) and mean squared error (MSE) to assess the impact of the loss function on model performance. For the zero-mean GP, we also report errors before parameter optimization (denoted as 0-shot) and after one training epoch (denoted as 1-shot). All errors are reported for the velocity magnitude.	75

3.3 Training time in seconds per epoch: We report the costs for DeepONet, FNO, and our method with three different mean functions. The training time of each approach varies across the different problems based on the dimensionality of the input/output functions and their respective resolutions. For example, the zero-mean GP appears to be the slowest in problems where the input and output functions are one-dimensional, such as in the Burgers’ or Advection equations, but becomes the fastest in the Structural Mechanics problem, where the input and output functions are one- and two-dimensional, respectively. Incorporating neural operators within our GP framework increases the training time per epoch with respect to their vanilla implementations due to the additional computations involved in the loss function (see Equation (3.15) versus MSE). However, this results in higher overall accuracy, as shown in Table 3.1 and 3.2. 79

ACKNOWLEDGMENTS

I would like to begin by expressing my deepest gratitude to my advisor, Professor Ramin Bostanabad, for his mentorship and guidance throughout my PhD. I am especially grateful to him for introducing me to the world of Gaussian processes, which has become the foundation of much of my research work. I am also profoundly grateful to my colleagues in the PMACS lab and to my other collaborators, whose insightful discussions and teamwork have greatly contributed to my academic journey since the very beginning.

My heartfelt thanks go to my family for their unconditional love and support in every aspect of life. They have given me the tools, values and mindset to chart my own path and pursue my dreams. I also want to thank my friends from Barcelona, who have stood by me from afar, and the new friends I have made in California, with whom I have shared countless memories that made my time at the University of California, Irvine truly unforgettable.

I would like to express my sincere appreciation to the Balsells fellowship for their financial support, which provided me with the unique opportunity to pursue graduate studies in the United States. This transformative experience has contributed immensely to both my academic and personal growth. For this, I will remain forever grateful.

Finally, I would like to acknowledge the support of the Office of Naval Research (grant number *N000142312485*), the National Science Foundation (grant numbers 2238038 and 2211908) and NASA's Space Technology Research Grants Program (grant number 80*NSSC21K1809*) throughout my PhD.

VITA

Carlos Mora Sardina

EDUCATION

Doctor of Philosophy in Mechanical and Aerospace Engineering	2025
University of California, Irvine	<i>Irvine, CA</i>
Master of Science in Mechanical and Aerospace Engineering	2022
University of California, Irvine	<i>Irvine, CA</i>
Master of Science in Aerospace Engineering	2022
Polytechnic University of Catalonia	<i>Barcelona, Spain</i>
Bachelor of Science in Aerospace Engineering	2020
Polytechnic University of Catalonia	<i>Barcelona, Spain</i>

RESEARCH EXPERIENCE

Graduate Research Assistant	2021–2025
University of California, Irvine	<i>Irvine, California</i>

TEACHING EXPERIENCE

Teaching Assistant	2025
University of California, Irvine	<i>Irvine, California</i>

REFEREED JOURNAL PUBLICATIONS

- Operator Learning with Gaussian Processes** 2025
Computer Methods in Applied Mechanics and Engineering
- Simultaneous and Meshfree Topology Optimization with Physics-informed Gaussian Processes** 2025
Computer Methods in Applied Mechanics and Engineering
- A gaussian process framework for solving forward and inverse problems involving nonlinear partial differential equations** 2025
Computational Mechanics
- GP+: A Python library for kernel-based learning via Gaussian processes** 2024
Advances in Engineering Software
- Probabilistic neural data fusion for learning from an arbitrary number of multi-fidelity data sets** 2023
Computer Methods in Applied Mechanics and Engineering
- Data-Driven Calibration of Multifidelity Multiscale Fracture Models Via Latent Map Gaussian Process** 2023
Journal of Mechanical Design

REFEREED CONFERENCE PUBLICATIONS

- SEEK: Self-adaptive Explainable Kernel For Nonstationary Gaussian Processes** Aug 2025
International Design Engineering Technical Conferences and Computers and Information in Engineering Conference
- Integrating Kernel Methods and Deep Neural Networks for Solving PDEs** Mar 2024
ICLR Workshop on AI4DifferentialEquations in Science
- Multi-Fidelity Reduced-Order Models for Multiscale Damage Analyses With Automatic Calibration** Nov 2022
International Design Engineering Technical Conferences and Computers and Information in Engineering Conference

ABSTRACT OF THE DISSERTATION

Hybrid Gaussian Process and Deep Learning Frameworks
for Computational Science and Engineering

By

Carlos Mora Sardina

Doctor of Philosophy in Mechanical and Aerospace Engineering

University of California, Irvine, 2025

Associate Professor Ramin Bostanabad, Chair

The convergence of computational science and engineering with machine learning has opened unprecedented opportunities for modeling, prediction and scientific discovery in complex physical systems. Central to these advances has been the ability to integrate rigorous physical models, multi-fidelity data, and domain knowledge within flexible computational frameworks. Much of the recent progress in this direction has been driven by deep neural networks (NNs), which have shown remarkable representational power but often at the expense of interpretability and reliable uncertainty quantification. By contrast, kernel methods—and Gaussian processes (GPs) in particular—are limited in scalability and expressivity, but provide uncertainty estimates, mathematical tractability, and interpretability. These contrasting features motivate the development of novel hybrid approaches that enjoy the complementary strengths of NNs and GPs for advancing computational science and engineering.

This dissertation makes three major contributions toward this goal. First, we introduce a hybrid framework for solving forward and inverse problems involving partial differential equations (PDEs). The framework leverages the closure property of GPs under linear transformations and Bayesian conditioning to enforce boundary and initial conditions, while leveraging NNs through automatic differentiation to handle nonlinear constraints. Second, we

propose a general GP-based operator learning framework by reformulating operator learning as an equivalent function learning problem. This framework can incorporate prior knowledge of the underlying physics during training and optionally integrate neural operators into the mean function. To the best of our knowledge, it represents the first zero-shot operator learning method in the literature. Third, we develop SEEK, a novel class of kernels for modeling nonstationary phenomena via GPs. SEEK draws inspiration from artificial neurons and is grounded in first principles to ensure kernel validity through symmetry and positive semi-definiteness. The proposed method learns input-dependent compositions of base kernels where the weights are parameterized by NNs, enabling the resulting GP to capture spatially and temporally varying dynamics with principled uncertainty quantification.

Chapter 1

Introduction

1.1 Motivation

The emergence of digital computers in the mid-20th century opened the door to a multitude of computational methods for approximating solutions to previously intractable mathematical problems. Soon after, the rapid development of electronic computers dramatically increased both the scale and complexity of problems that could be addressed, paving the way for the modern field of computational science and engineering. This progress enabled the development of advanced numerical techniques such as the finite element method (FEM) for structural analysis and computational fluid dynamics (CFD) for fluid flow, which remain central to engineering today.

In parallel, the increase of computing power and the growing availability of large datasets fueled the resurgence of machine learning. Although historically distinct from computational science and engineering, machine learning pursues a closely aligned goal: to model and understand complex real-world phenomena through computational methods. Machine learning achieves this goal by leveraging statistical learning principles and optimization tech-

niques to construct predictive models from data, which is especially useful in applications where explicit physical models are incomplete or unavailable. In contrast, classical computational sciences rely on well-defined physical models and precise mathematical techniques to characterize system behavior.

In recent years, machine learning methods have been increasingly applied to problems in the context of computational science and engineering, offering frameworks capable of integrating physical models, multi-fidelity data, and prior domain knowledge. Much of the progress in this direction has been driven by deep neural networks (NNs), which have shown remarkable representational capacity but often at the expense of interpretability and reliable uncertainty quantification. By contrast, kernel methods —and Gaussian processes (GPs) in particular— are limited in scalability and expressivity, but offer uncertainty estimates, mathematical tractability, and interpretability. These contrasting features motivate the development of novel hybrid approaches that enjoy the complementary strengths of NNs and GPs for advancing computational science and engineering.

This dissertation pursues this vision through three novel contributions¹ that combine deep learning architectures with GPs for (1) solving forward and inverse problems involving partial differential equations (PDEs), (2) learning operators arising from parametric PDEs, and (3) modeling nonstationary functions.

1.2 Thesis Contributions and Outline

This thesis introduces three novel frameworks that leverage the complementary strengths of deep NNs and GPs to address diverse problems in computational science and engineering:

¹Each contribution corresponds to peer-reviewed work completed during the author’s PhD, though not all of the author’s published research is included in the thesis.

- **Solving Forward and Inverse Problems Involving PDEs.** Chapter 2 introduces a hybrid method that combines deep NNs and GPs for solving forward and inverse problems involving PDEs. This approach leverages the strengths of both models based on the following observations. First, GPs are naturally suited to handle linear constraints—which typically include Dirichlet, Neumann, and initial conditions (BCs/IC)—as GPs are closed under linear transformations and Bayesian conditioning. Second, deep NNs are more effective than GPs for handling nonlinear constraints, as the latter requires linearization of the PDE, but the former can directly solve nonlinear differential equations through automatic differentiation without prior linearization. By integrating these two components, the proposed model inherently satisfies linear constraints such as BCs/IC through the GP kernels, while the NN component provides the flexibility and generalization power needed to handle nonlinear constraints.
- **Operator Learning in Parametric PDEs.** Chapter 3 presents a general GP framework for operator learning that can optionally integrate neural operators into the mean function and incorporate prior knowledge of the underlying physics into its training process. This is achieved by reformulating the original operator learning problem as an equivalent function learning problem, which can be solved using well-established regression techniques based on NNs and GPs. The proposed framework naturally extends to multi-output operators and benefits from computational speed-ups achieved through product kernel structures and Kronecker product matrix representations of the underlying kernel matrices. Moreover, we demonstrate that GPs with common kernels can mitigate the so-called curse of dimensionality through proper initialization of their kernel parameters. To the best of our knowledge, this framework provides the first zero-shot operator learning method in the literature.
- **Learning Nonstationary Kernels.** GPs often rely on stationary kernels, which can lead to suboptimal predictions and miscalibrated uncertainty estimates, especially

when modeling nonstationary phenomena. Chapter 4 introduces SEEK, a novel class of learnable kernels to model nonstationary functions via GPs. SEEK draws inspiration from artificial neurons and is grounded in first principles to guarantee symmetry and positive semi-definiteness, key properties of valid kernels. The proposed method effectively models nonstationarity by learning input-dependent compositions of base kernels where the weights are parameterized using deep neural networks. This self-adaptive mechanism enables the GP to capture spatially and temporally varying dynamics with principled uncertainty estimates.

We note that Chapters 2, 3, and 4 may employ different notation systems tailored to their specific problem setting, as no single notation can naturally accommodate the full breadth of work presented in this dissertation.

Finally, Chapter 5 closes the thesis with a summary of the main findings and future research directions. Supplementary materials, including detailed derivations, additional experiments, and extended descriptions, are provided in Chapters A, B, and C.

Chapter 2

Solving Forward and Inverse Problems Involving PDEs

2.1 Introduction

Partial differential equations (PDEs) elegantly explain the behavior of many engineered and natural systems such as power grids [5, 6], advanced materials [7], tectonic cracks [8], weather and climate [9, 10], and biological agents [5, 11]. Since the solutions to most PDEs cannot be derived analytically, numerical approaches such as the finite element method are frequently used to solve them. Recently, a new class of methods known as physics-informed machine learning (PIML) has been developed and successfully used in studying fundamental phenomena such as turbulence [12], diffusion [13], shock waves [14], interatomic bonds [15], and cell signaling [16]. While PIML models have fueled a renaissance in modeling complex systems, their performance heavily depends on optimizing the model’s training mechanism (e.g., choice of optimizer), loss function, and architecture [17]. To reduce the complexity inherent in developing and training PIML models, we re-envision solving PDEs

via machine learning (ML) and introduce a framework based on Gaussian processes (GPs) to simultaneously use the strengths of deep neural networks (NNs) and kernel methods. Specifically, our resulting PIML model augments NNs with what we call kernel-weighted Corrective Residuals (CoRes) to improve NNs’ accuracy, robustness, and efficiency in solving PDEs.

Remarkable successes have been achieved via ML in many areas such as protein modeling [18, 19], designing new materials [20, 21, 22, 23, 24, 25], automated demographics monitoring [26], and expert language models [27]. Availability of big data is a common feature across these applications which, in turn, enables building large ML models that can distill highly complex relations from the data. However, in the context of solving PDEs, there is a particular interest in building ML models whose training does not rely on any sample solutions inside the domain. Indeed, the key enabler in this application is PIML which systematically infuses our physical and mathematical knowledge into the structure and/or training mechanism of ML models. Compared to classical computational tools, PIML promises a unified platform for solving inverse problems [28], obtaining meshfree solutions [29, 29, 30], assimilating experiments with simulations [31], and uncertainty quantification [32].

We can broadly classify PIML models into two categories. The first group of methods relies on variants of neural networks (NNs) and can be traced back to [33, 34]. Physics-informed neural networks (PINNs) [35, 36] and their various extensions [37, 38, 39, 40, 41, 42] are perhaps the most widely adopted member of these classes of methods and their basic idea is to parameterize the PDE solution via a deep NN. As detailed in Section A.2, the parameters of this NN are optimized by minimizing a multi-component loss function which encourages the NN to satisfy the PDE as well as the initial and/or boundary conditions (ICs and BCs), see Figure A.2. This minimization relies on automatic differentiation [43] and is known to be very sensitive to the optimizer, loss function formulation, and NN’s architecture. To decrease this sensitivity, recent works have developed adaptive loss functions [44, 45] and tailored

architectures that improve gradient flows [46] or automatically satisfy BCs [47, 34, 48, 49, 50] (see Section A.2 for more details). These advancements, however, fail to generalize to a diverse set of PDEs and substantially increase the cost and complexity of training [51]. While such works primarily focus on solving specific instances of a PDE system, NNs have also been used for operator learning [52, 53] where the goal is to approximate infinite-dimensional mappings arising in PDE systems, e.g., mapping a BC to its corresponding solution field.

The second group of PIML models leverage kernel methods. The key idea behind kernel methods is to implicitly map the original input data into a higher dimensional space where it is easier to quantify similarities among the data points. Support vector machines (SVMs) are a popular kernel method that have been successfully applied to supervised learning, clustering, dimensionality reduction, and anomaly detection [54]. GPs are also kernel methods and can be traced back to Poincaré’s course in probability theory [55]. GPs have long been used in emulation and Bayesian optimization but their application in solving PDEs is relatively new and remains largely unexplored. The few existing works [56, 57, 58] exclusively employ zero-mean GPs which are completely characterized by their parametric kernel or covariance function. With this choice, solving the PDE amounts to designing the GP’s kernel whose parameters are obtained via either maximum likelihood estimation (MLE) or a regularized MLE where the penalty term quantifies the GP’s error in satisfying the PDE system.

In a recent novel work [1], solving PDEs via a zero-mean GP is cast as an optimal recovery problem whose loss function is derived based on the PDE system and aims to estimate the solution at a finite number of interior nodes in the domain. Once these values are estimated, the PDE solution is approximated anywhere in the domain via kernel regression (see Section A.2). Hereafter, we denote this method as GP_{OR} and note that it has been recently extended to learn operators [59] and to handle large datasets using the concept

of inducing points [58]. Additionally, while GPs with non-zero means have been employed for solving PDEs in a data-driven manner [60], they have not been explored in a purely physics-informed setting with a robust training mechanism.

The rest of this chapter is organized as follows. We introduce our approach in Section 2.2 where we first provide a theoretical rationale for it in Section 2.2.1 and then in Section 2.2.2 introduce the modular and robust framework that we have developed for efficiently implementing it. We comment on the most prominent features of our approach in Section 2.2.3 where we also introduce its extensions for solving inverse problems or handling PDE systems with multiple outputs. We rigorously study the performance of our approach in Section 2.3 and conclude the chapter with some final remarks and future research directions in Section 2.4.

2.2 Neural Networks with Kernel-Weighted Corrective Residuals

Kernel methods, particularly GPs, have less extrapolation and scalability powers compared to deep NNs. They also struggle to approximate PDE solutions that have large gradients or involve coupled dependent variables such as the Navier-Stokes equations. However, GPs locally generalize better than NNs [57], are mathematically tractable, interpretable, and easy to train (see Section A.1 for discussions and examples). Grounded on these properties, we introduce deep architectures with kernel-weighted CoRes that integrate the attractive features of NNs and GPs for solving PDEs.

2.2.1 Theoretical Rationale

GPs provide a tractable and robust framework for function approximation [61, 1, 62] and their kernels are extensively studied to accommodate learning functions of varying degrees of complexity [63, 64, 65, 66, 67, 68, 69]. However, we argue that the sole reliance on the kernel serves as a double-edged sword when solving PDEs via GPs. To demonstrate, we consider the task of emulating the function $u(\mathbf{x})$ given the n samples $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with corresponding outputs $\mathbf{u} = \{u_1, \dots, u_n\}$ where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{d_x}$ with boundary $\partial\mathcal{X}$ and $u_i = u(\mathbf{x}_i) \in \mathbb{R}$. If we endow $u(\mathbf{x})$ with a GP prior with the mean function $m(\mathbf{x}; \boldsymbol{\theta})$ and kernel $c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\phi})$, the conditional process is also a GP whose expected value at $\mathbf{x}^*$ is:

$$\eta(\mathbf{x}^*; \boldsymbol{\theta}, \boldsymbol{\phi}) := \mathbb{E}[u^* | \mathbf{u}, \mathbf{X}] = m(\mathbf{x}^*; \boldsymbol{\theta}) + \mathbf{w}^T \mathbf{r}, \quad (2.1a)$$

$$\mathbf{w} := w(\mathbf{x}^*, \mathbf{X}; \boldsymbol{\phi}) = c^{-1}(\mathbf{X}, \mathbf{X}; \boldsymbol{\phi}) c(\mathbf{X}, \mathbf{x}^*; \boldsymbol{\phi}) \quad (2.1b)$$

$$\mathbf{r} := r(\mathbf{X}, \mathbf{u}; \boldsymbol{\theta}) = \mathbf{u} - m(\mathbf{X}; \boldsymbol{\theta}). \quad (2.1c)$$

Here, $\boldsymbol{\theta}$ are the parameters of the mean function, $\boldsymbol{\phi}$ are the so-called length-scale or roughness parameters of the kernel, $c(\mathbf{X}, \mathbf{x}^*; \boldsymbol{\phi}) = [c(\mathbf{x}_1, \mathbf{x}^*; \boldsymbol{\phi}), \dots, c(\mathbf{x}_n, \mathbf{x}^*; \boldsymbol{\phi})]^T$, $\mathbf{r}$ denotes the *residuals* of the mean function on the training data, $\mathbf{w}$ are the kernel-induced weights, and $\mathbf{C} = c(\mathbf{X}, \mathbf{X}; \boldsymbol{\phi})$ is the covariance matrix with ij^{th} entry $c(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\phi})$. The covariance function can be a deep NN [70] or the *simple* Gaussian covariance function:

$$c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\phi}) = \exp\{-(\mathbf{x} - \mathbf{x}')^T \text{diag}(\boldsymbol{\phi})(\mathbf{x} - \mathbf{x}')\}, \quad (2.2)$$

which can be augmented to account for noisy observations:

$$c(\mathbf{x}, \mathbf{x}'; \sigma^2, \boldsymbol{\phi}, \delta) = \sigma^2 \exp\{-(\mathbf{x} - \mathbf{x}')^T \text{diag}(\boldsymbol{\phi})(\mathbf{x} - \mathbf{x}')\} + \mathbb{1}\{\mathbf{x} == \mathbf{x}'\} \delta, \quad (2.3)$$

where σ^2 is the process variance, $\mathbb{1}\{\cdot\}$ returns 1/0 if the enclosed statement is true/false, and δ is the so-called nugget parameter that is typically used to model noise or improve the numerical stability of the covariance matrix. For simplicity, hereafter we consider the Gaussian covariance function in Equation (2.2) in our descriptions.

The optimum model parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are generally unknown and hence estimated via MLE:

$$\left[\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}\right] = \underset{\boldsymbol{\theta}, \boldsymbol{\phi}}{\operatorname{argmax}} |2\pi\mathbf{C}|^{-\frac{1}{2}} \exp \left\{ \frac{-1}{2} \mathbf{r}^T \mathbf{C}^{-1} \mathbf{r} \right\}, \quad (2.4)$$

which can be an expensive and/or numerically unstable process if $\mathbf{C}$ is large or ill-conditioned (this can happen if the training dataset is large or has samples that are very close in the input space).

Since many kernels can approximate an arbitrary continuous function [61], zero-mean GPs are used in many regression problems as eliminating $m(\mathbf{x}; \boldsymbol{\theta})$ reduces the number of trainable parameters while increasing numerical stability. As discussed in Section A.1 and shown in Figure A.1, the latter improvement stems from the fact that an over-parameterized $m(\mathbf{x}; \boldsymbol{\theta})$, while needed for learning hidden complex relations, can easily interpolate $\mathbf{u}$ and, in turn, drive the residuals in Equation (2.1c) to $\mathbf{0}$. Such residuals require $\boldsymbol{\phi} \rightarrow \mathbf{0}$ which diminishes the contributions of the kernel in Equation (2.1a) and renders $\mathbf{C}$ ill-conditioned.

Unlike regression, PDEs with IC/BCs cannot be accurately solved via zero-mean GPs without any in-domain samples since the posterior process in Equation (2.1) predicts zero for any point that is sufficiently far from the boundaries. This reversion to the mean behavior is due to the exponential decay of the correlations as the distance between two points increases, see Equation (2.2). While deep kernels can delay this decay, they cannot prevent it.

Following the above discussions, we make two important observations on the posterior distri-

bution in Equation (2.1): it heavily relies on $m(\mathbf{x}; \boldsymbol{\theta})$ in data scarce regions and it regresses $\mathbf{u}$ regardless of the values of $m(\mathbf{X}; \boldsymbol{\theta})$. These observations suggest that a GP whose mean function is parameterized with a deep NN provides an attractive *prior* for solving PDE systems. Functions of the form given in Equation (2.1a) can easily satisfy BCs/IC as well as other linear constraints if applicable¹, while their smoothness can be controlled through the mean and covariance functions. This approach, however, presents two major challenges. First, the posterior distribution in this case should be obtained by conditioning the prior on BCs/IC while constraining it to satisfy the PDE in the domain. Since most practical PDEs are nonlinear, the posterior will not be Gaussian upon the constraining and hence there are no closed form formulas available for its likelihood (to train the model) or expected value (to easily predict with the model). Second, jointly optimizing $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ is a computationally expensive and unstable process due to the repeated need for constructing and inverting $\mathbf{C}$.

2.2.2 Proposed Framework

We address the above challenges via modularization and formulating the training process based on maximum a posteriori (MAP) instead of MLE. As illustrated in Figure 2.1, our framework consists of two sequential modules that aim to solve PDE systems with deep NNs that substantially benefit from kernel-weighted CoRes. These modules seamlessly integrate the best of two worlds: (1) the local generalization power of kernels close to the domain boundaries where IC/BCs are specified, and (2) the substantial capacity of deep NNs in learning multiple levels of distributed representations in the interior regions where there are no labeled training data.

In the first module, we endow the PDE solution with a GP prior whose mean and covariance functions are a deep NN and the Gaussian kernel in Equation (2.2), respectively (note that

¹This property arises from the fact that GPs are closed under Bayesian conditioning and linear transformations, and from their inherent interpolation capabilities.

the assumption on having a unit variance does not affect our method as the variance cancels out in Equation (2.1b) due to the matrix inversion). Conditioned on the data (i.e., $\mathbf{u}$ which is obtained by sampling from the BCs/IC), the posterior distribution of the solution is again a GP and follows Equation (2.1a) where $\mathbf{r}$ and $\mathbf{w}$ denote the residuals and kernel-induced weights, respectively. Importantly, in this module we fix $\boldsymbol{\theta}$ to some random values and choose $\hat{\boldsymbol{\phi}}$ such that the GP can faithfully reproduce $\mathbf{u}$. As demonstrated in Section A.1 and Section 2.3.4, our results are not affected by the fact that we do not leverage MLE for parameter estimation in module one. Hence, we manually select $\hat{\boldsymbol{\phi}}$, fix the process variance to unity, and choose the nugget parameter to ensure the covariance matrix is not ill-conditioned (see Section A.1 for more details).

In the second module, we obtain the final model by conditioning the GP on the (nonlinear) constraints that require the predictions at arbitrary points in the domain to satisfy the PDE system. We achieve this conditioning by fixing $\hat{\boldsymbol{\phi}}$ from module 1 and optimizing $\boldsymbol{\theta}$ to ensure

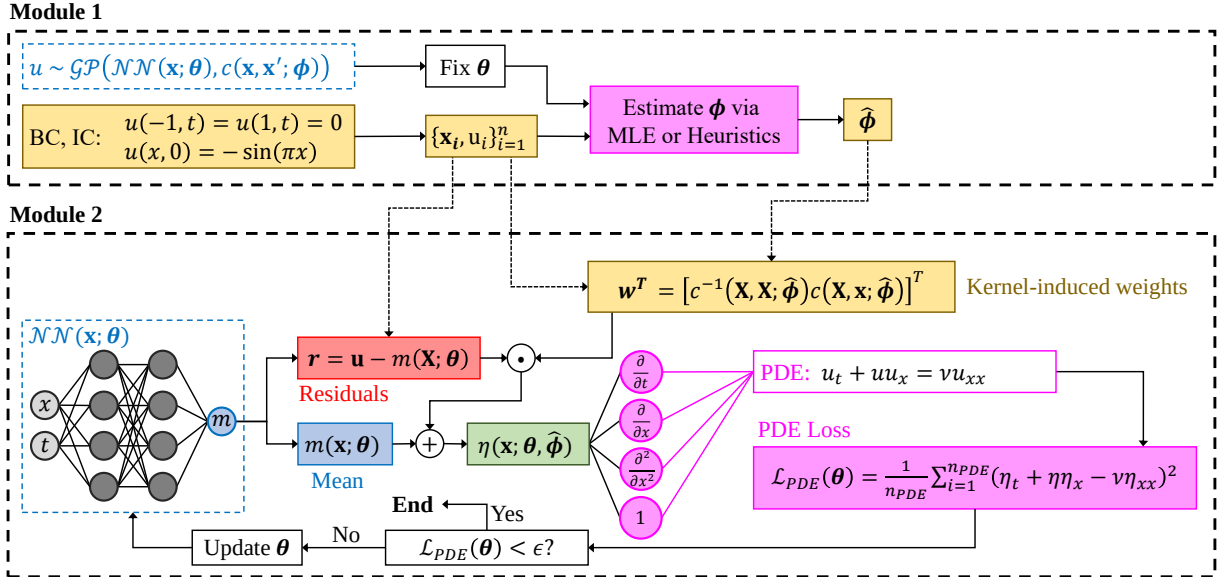


Figure 2.1 Flowchart of the proposed framework for solving the 1D Burgers' equation: We endow the solution $u(\mathbf{x})$ with a GP prior whose mean and covariance functions are parameterized via a deep NN and the Gaussian kernel in Equation (2.2), respectively. In module 1, we fix $\boldsymbol{\theta}$ to some random values and estimate the kernel parameters via MLE or heuristics such that the posterior GP conditioned on the BC/IC data faithfully reproduces $\mathbf{u}$. Then, in Module 2, we estimate $\boldsymbol{\theta}$ by minimizing a loss function which only depends on the PDE since BC/IC are automatically satisfied.

that the model in Equation (2.1a) satisfies the PDE at n_{PDE} randomly selected collocation points (CPs) in the domain.

2.2.3 Model Characteristics and Extensions

In this section, we elaborate on four unique features of our approach for solving PDE systems and discuss two major extensions that enable (1) data fusion for solving inverse problems, and (2) efficiently solving PDEs with multiple dependent variables. We highlight that combining GPs and NNs was first introduced in [57] to improve the uncertainty quantification power of NNs in supervised learning tasks. However, in sharp contrast to our work, the proposed approach in [57] relies on big data in the entire domain (we do not use any labeled data in the domain), aims to improve prediction uncertainties, leverages MLE for parameter optimization, and hinges on sparse GPs for scalability.

Feature one: The overall training cost of our approach almost entirely depends on the second module since selecting $\hat{\phi}$ does not rely on MLE and is a computationally inexpensive process. Additionally, our experiments consistently indicate that the performance of the final model across a broad range of problems is quite robust to $\hat{\phi}$ as long as the BCs/IC are sufficiently sampled (see Section 2.3.4 for sample results). This robustness is independent of the random values assigned to θ in module 1. Based on these two observations, in all of our experiments we simply assign 10^2 to all the length-scale parameters of the kernel and sample 40 points at each boundary. We highlight that while these values are certainly not the optimum, they have consistently enabled us to achieve highly competitive results.

Feature two: The computational cost of coupling GPs and deep NNs in our framework is negligible during both training and testing since $\mathbf{C}$ does not change in the second module and its size only depends on $\mathbf{u}$ which is a relatively small vector. When solving PDEs such as the Navier-Stokes equations that have multiple dependent variables, the size of $\mathbf{u}$

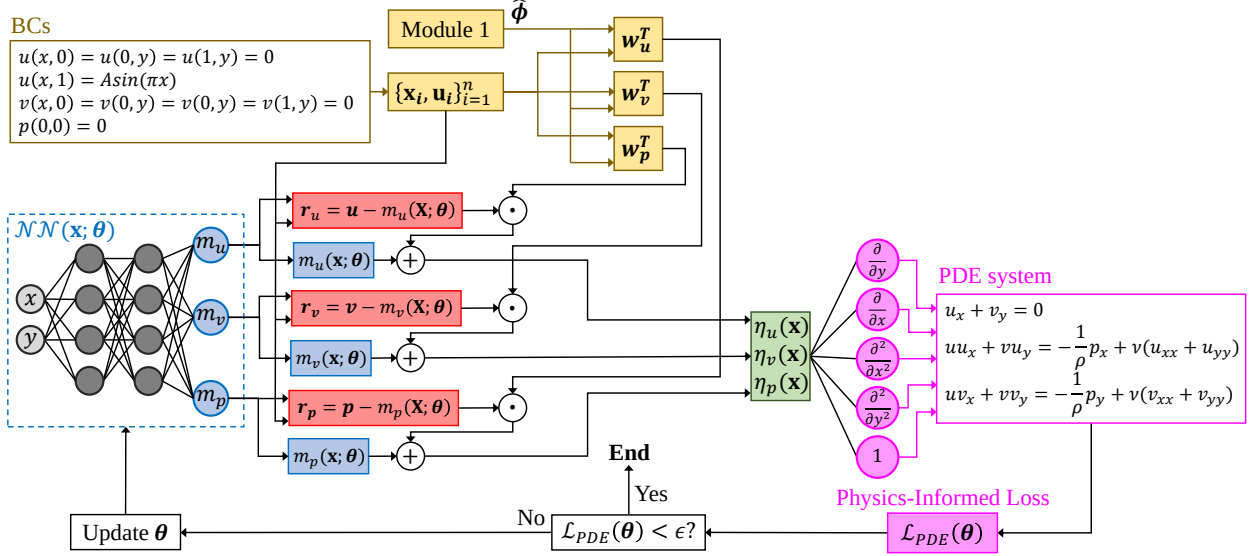


Figure 2.2 Solving the 2D incompressible Navier-Stokes equations for the lid-driven cavity problem: With minor architectural changes on module two with respect to Figure 2.1, our framework can also solve coupled PDE systems. Specifically, we endow each dependent variable with a GP prior. These GPs have independent kernels but a shared mean function that is parameterized via a deep neural network. Similar to Figure 2.1, the loss function only depends on the PDE residuals and excludes data loss terms on BC/IC.

can grow rapidly since it will store boundary and initial data on multiple outputs. Hence, for such PDEs we decouple the kernel-weighted CoRes of the outputs to keep the size of $\mathbf{C}$ and $\mathbf{u}$ small. As shown in Figure 2.2, we formulate this decoupling by endowing the dependent variables with a collection of GP priors which share the same mean function but have independent kernels (i.e., the predictive formula in Equation (2.1a) is used for as many outputs as the PDE has). While a single kernel can help in learning the inter-variable relations, we avoid this formulation for the following two main reasons. Firstly, it increases the size and condition number of the covariance matrix especially if the BCs on these variables are significantly different. For instance, on the top edge ($y = 1$) in the lid-driven cavity (LDC) benchmark problem formulated in Equation (2.8), pressure is unknown while the vertical and horizontal velocity components are equal to, respectively, zero and $A \sin(\pi x)$. Secondly, our empirical findings indicate that the shared mean function is able to adequately learn the hidden interactions between these dependent variables.

Feature three: As proven in Section A.3, our model can exactly satisfy the BCs/IC as the number of sampled boundary points increases. Due to this feature, the loss function in Figure 2.1 or Figure 2.2 only minimizes the error in satisfying the PDE and excludes data loss terms that encourage the NN to reproduce the BCs/IC. This exclusion indicates that our framework does not need weight balancing which is an expensive process that ensures each component of a composite loss is appropriately minimized during training. We highlight that per Equation (2.1) the contribution of kernel-weighted CoRes to the model’s predictions decreases as the distance with the boundaries (where the data is available) increases. As shown in Figure 2.3a this decrease is not sudden and depends on the quality of the learnt mean function.

The proof in Section A.3 indicates that the convergence as $n \rightarrow \infty$ is independent of the domain geometry and the potential noise that may corrupt the boundary data. The former feature is especially useful when solving PDEs over irregular domains such as the unsteady LDC problem in Figure 2.3b.

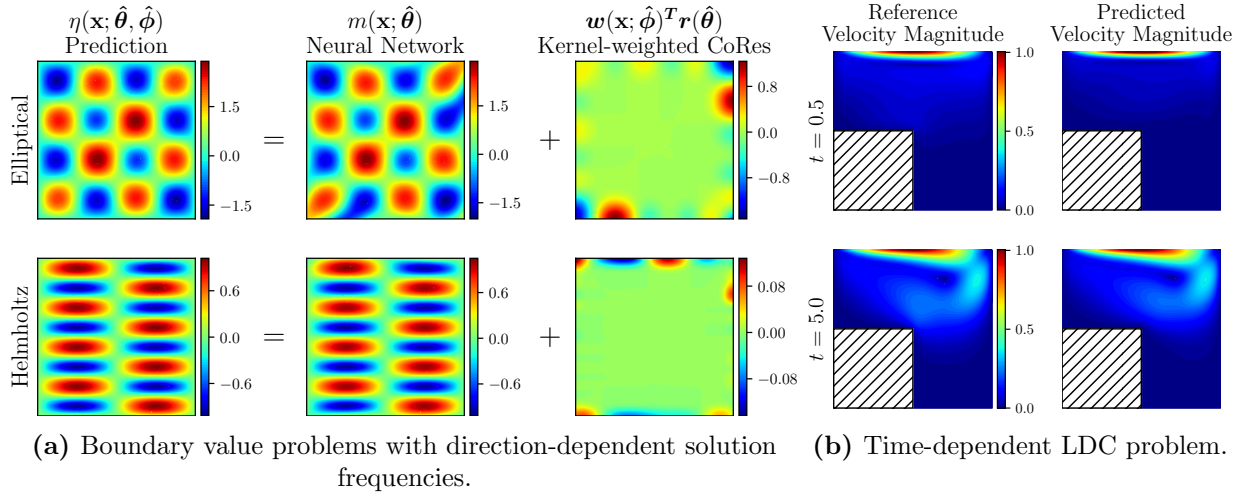


Figure 2.3 Model features: (a) In addition to improving the optimization of θ in module two, kernel-weighted CoRes contribute to the model predictions and ensure strict satisfaction of the BCs. (b) Kernel-weighted CoRes automatically adapt to the domain geometry and are applicable to coupled PDE systems such as the Navier-Stokes equations. Here, we solve the unsteady LDC problem for $t \in [0, 5]$ and visualize the flow at $t = 0.5$ and $t = 5$.

The above three features imply that training a PINN (or any of its extensions) costs similarly to the case where the same network is used as $m(\mathbf{x}; \boldsymbol{\theta})$ in our framework. This behavior is very attractive since our approach consistently and substantially improves the performance of existing NN-based methods while also simplifying the training process by dispensing with the need for balancing the loss terms or fine-tuning the optimization settings.

Feature four: Our framework allows to perform data fusion and system identification by incorporating the additional measurements or observations in the kernel structure in exactly the same way that BC/IC data are handled, see Section 2.3.6 for a graphical flowchart and some examples (note that since we handle observations similarly to BC/IC data, the reproducibility proof in Section A.3 applies to them as well). This extension provides fast convergence rates for solving inverse problems and enables combining multiple data sets (e.g., experiments and simulations) to discover missing physics or unknown PDE parameters.

2.3 Results and Discussion

We compare the performance of our approach against four baseline PIML methods on four different PDE systems. We describe these PDE systems in Section 2.3.1 and then provide some details on implementation and training/testing of the five PIML models in Section 2.3.2 where we also comment on our rationale for choosing the baselines. We summarize the results of our comparative studies in Section 2.3.3 and then conduct extensive sensitivity analyses in Section 2.3.4 to assess the impact of factors such as random initialization, noise, network architecture, and optimization settings on the summary results reported in Section 2.3.3. We conclude this section by comparing the training loss trajectory of our model to those of PINNs in Section 2.3.5. Additional experiments supporting our claims are provided in Section A.4.

2.3.1 Description of the Benchmark Problems

The four PDE systems used in our studies are described below. Each problem is solved under two settings to understand the effect of PDE complexity on the performance of the PIML models.

Throughout each problem, we specify the collection of independent variables $\mathbf{x}$, e.g., $\mathbf{x} := [x, t]$ in Burgers' equation. In the case of Navier-Stokes equation, the collection of dependent variables is denoted by $\mathbf{u}(\mathbf{x}) := [u(\mathbf{x}), v(\mathbf{x}), p(\mathbf{x})]^T$.

Burgers' Equation: We consider a viscous system subject to IC and Dirichlet BC in one space dimension:

$$\begin{aligned} u_t + uu_x - \nu u_{xx} &= 0, \quad \forall x \in (-1, 1), t \in (0, 1] \\ u(-1, t) = u(1, t) &= 0, \quad \forall t \in [0, 1] \\ u(x, 0) &= -\sin(\pi x), \quad \forall x \in [-1, 1] \end{aligned} \tag{2.5}$$

where $\mathbf{x} := [x, t]$ and ν is the kinematic viscosity. Equation (2.5) frequently arises in fluid mechanics and nonlinear acoustics. In our studies, we investigate the performance of different PIML models in solving Equation (2.5) for $\nu \in \left\{\frac{0.01}{\pi}, \frac{0.02}{\pi}\right\}$ which controls the solution smoothness at $x = 0$ where a shock wave forms as ν approaches zero.

Nonlinear Elliptic PDE: To assess the ability of our approach in learning high-frequency solutions, we study the boundary value problem developed in [1]:

$$\begin{aligned} u_{xx} + u_{yy} - \alpha u^3 &= f(x, y), \quad \forall x, y \in (0, 1)^2 \\ u(x, 0) = u(x, 1) &= 0, \quad \forall x \in [0, 1] \\ u(0, y) = u(1, y) &= 0, \quad \forall y \in [0, 1] \end{aligned} \tag{2.6}$$

where $\mathbf{x} := [x, y]$ and $\alpha \in \{20, 30\}$ is a constant that controls the nonlinearity degree. $f(x, y)$ is designed such that the solution is $u(x, y) = \sin(\pi x) \sin(\pi y) + 2 \sin(4\pi x) \sin(4\pi y)$.

Eikonal Equation: We consider the two-dimensional regularized Eikonal equation [1] which is typically encountered in the context of wave propagation:

$$\begin{aligned}
u_x^2 + u_y^2 - \epsilon(u_{xx} + u_{yy}) &= 1, \quad \forall x, y \in (0, 1)^2 \\
u(x, 0) = u(x, 1) &= 0, \quad \forall x \in [0, 1] \\
u(0, y) = u(1, y) &= 0, \quad \forall y \in [0, 1]
\end{aligned} \tag{2.7}$$

where $\mathbf{x} := [x, y]$ and $\epsilon \in \{0.01, 0.05\}$ is a constant that controls the smoothing effect of the regularization term.

Lid-Driven Cavity (LDC): The two-dimensional steady state LDC problem has become a gold standard for evaluating the ability of PIML models in solving coupled PDEs. This problem is governed by the incompressible Navier-Stokes equations:

$$\begin{aligned}
u_x + v_y &= 0, \quad \forall \mathbf{x} \in (0, 1)^2 \\
uu_x + vv_y &= -\frac{1}{\rho}p_x + \nu(u_{xx} + u_{yy}), \quad \forall \mathbf{x} \in (0, 1)^2 \\
uv_x + vv_y &= -\frac{1}{\rho}p_y + \nu(v_{xx} + v_{yy}), \quad \forall \mathbf{x} \in (0, 1)^2 \\
v(x, 0) = v(x, 1) &= v(0, y) = v(1, y) = 0, \quad \forall x, y \in [0, 1] \\
u(x, 0) = u(0, y) &= u(1, y) = 0, \quad \forall x, y \in [0, 1] \\
u(x, 1) &= A \sin(\pi x), \quad \forall x \in [0, 1] \\
p(0, 0) &= 0
\end{aligned} \tag{2.8}$$

where $\mathbf{x} := [x, y]$, $\nu = 0.01$ is the kinematic viscosity, $\rho = 1.0$ denotes the density, and $A \in \{3, 5\}$ is a scaling constant. The Reynolds number for this LDC problem can be computed via $Re = \frac{\rho \bar{u} L}{\nu}$ where $\bar{u} = \int_0^1 A \sin(\pi x) dx$ is the characteristic speed of the flow and $L = 1$ is the characteristic length. For the two cases $A \in \{3, 5\}$, we obtain $Re \in \{191, 318\}$.

2.3.2 Implementation Details in Our Comparative Studies

Below, we first describe the architecture and training procedure of the PIML models used throughout this chapter and then comment on how the reference solutions are obtained for each PDE system.

Architecture and Training

We use a fully connected feed-forward NN as the mean function in our framework and design its input and output dimensionality based on the PDE system. We denote our model via NN-CoRes and compare it against (1) GP_{OR} which is the optimal recovery approach of [1] that leverages zero-mean GPs, (2) PINNs whose architectures are exactly the same as our NNs in the mean function $m(\mathbf{x}; \boldsymbol{\theta})$, (3) PINN_{DW} which is a variation of PINNs that balances loss components with dynamic weights [39], and (4) PINN_{HC} which is a PINN whose output is designed to strictly satisfy the BCs/IC [50]. More detailed information about these four models is provided in Section A.2.

Our rationale for comparing our method against GP_{OR} , PINNs, PINN_{DW} , and PINN_{HC} are as follows. Evaluation against GP_{OR} assesses our arguments on the limitations of using zero-mean GPs for solving PDEs when labeled solution data are only available as IC/BC. Comparisons against vanilla PINNs directly show the impact of kernel-weighted CoRes as the same network architecture is used as the mean function in NN-CoRes. Lastly, comparisons against PINN_{DW} and PINN_{HC} highlight the benefits of NN-CoRes in automatically satisfying the BC/IC.

The NN-based approaches (i.e., NN-CoRes, PINN, PINN_{DW} , and PINN_{HC}) are all implemented in PyTorch [71] and use hyperbolic tangent activation functions in all their layers except the output one where a linear activation function is used. The number and size of the

hidden layers are exactly the same across these methods to enable a fair and straightforward comparison. For NN-CoRes we use the Gaussian kernel in Equation (2.3) with $\sigma^2 = 1$ and $\phi = 10^\omega = 10^2$ in all the simulations in Section 2.3.3 (note that we study the effect of other values of the length-scale parameter in Section 2.3.4). The nugget or jitter parameter of the kernel in Equation (2.3) is chosen such that the covariance matrix is numerically stable. We ensure this stability by imposing an upper bound of approximately $\kappa_{max} \approx 10^6$ on the condition number of the covariance matrix, i.e., $\kappa < \kappa_{max}$. This constraint typically results in a nugget value of around 10^{-5} or 10^{-4} . We have not optimized the performance of NN-CoRes with respect to κ_{max} as we have found our current results to be sufficiently accurate.

To optimize NN-CoRes, PINNs, and PINN_{HC} we leverage L-BFGS with a learning rate of 10^{-2} while PINN_{DW} is optimized using Adam with a learning rate of 10^{-3} (note that the performance of L-BFGS deteriorates if dynamic weights are used in the loss function). To ensure these NN-based methods produce optimum models, we use a very large number of epochs during training. Specifically, we employ 1,000 and 2,000 epochs for single- and multi-output problems, respectively. Since Adam typically requires more epochs for convergence, we train PINN_{DW} for 40,000 epochs across all problems. To evaluate the loss function, we use 10,000 collocation points within the domain in all cases. For PINN and PINN_{DW} we uniformly sample boundary and/or initial conditions at 1,000 locations while we only sample 40 points for NN-CoRes. This significant difference is due to the fact that we observed that NN-CoRes with just 40 boundary points can outperform other methods. Leveraging more boundary data improves the performance of NN-CoRes in solving PDE systems especially in satisfying the IC/BC.

We fit GP_{OR} based on the code and specifications provided by [1] which leverages a variant of the Gauss–Newton algorithm for optimization. The performance of GP_{OR} depends on the kernel parameters and the number of interior nodes n_{PDE} where $\mathbf{z}$ needs to be estimated. For the former, we use the recommended values in [1] and for the latter we choose two values

(1,000 and 2,000) in our experiments.

NN-CoRes, PINN, PINN_{DW}, and PINN_{HC} are trained on an NVIDIA GeForce RTX 3060 with 64 GB of RAM whereas GP_{OR} is trained on a CPU equipped with a 11th Gen Intel-Core i7-11700K running at a base clock speed of 3.6 GHz. The training cost of NN-CoRes compared to PINN for each problem is reported in Table A.2 and discussed in Section A.4.3.

Reference Solutions and Accuracy Metric

We obtain the reference solutions for the PDE systems as follows:

- Burgers' Equation: The reference solution is obtained from the code provided in [1] which employs the Cole-Hopf transformation [72] together with the numerical quadrature.
- Elliptic PDE: The analytical solution for this problem is $u(x, y) = \sin(\pi x) \sin(\pi y) + 2 \sin(4\pi x) \sin(\pi y)$.
- Eikonal Equation: We leverage the solution method provided by [1] which applies the transformation $u(x, y) = -\epsilon \log g(x, y)$ leading to the linear PDE $g - \epsilon^2 \Delta g = 0$ that can be solved via the finite difference method.
- Lid-Driven Cavity: we use the finite element method implemented in the commercial software package COMSOL [73].

To quantify the accuracy of the PIML models, we calculate the Euclidean norm of the error between the reference and predicted solutions at $n_t = 10^4$ randomly chosen points. We denote this error metric via $L_{2,e}$ and calculate it as:

$$L_{2,e} = \sqrt{\frac{1}{n_t} \sum_{i=1}^{n_t} \left(u(\mathbf{x}_i) - \eta(\mathbf{x}_i; \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}) \right)^2}. \quad (2.9)$$

2.3.3 Summary of Comparative Studies

The results of our studies are summarized in Table 2.1 and indicate that our approach consistently outperforms other methods by relatively large margins. Interestingly, in most cases even the small NN-CoRes with four 10–neuron hidden layers achieve lower errors than the high capacity version of the competing methods; indicating NN-CoRes more effectively use their networks’ capacity to learn the PDE solution. To visually compare the efficiency in capacity utilization across different NN-based models, in Figure 2.4 we provide the histogram of the PDE loss gradients with respect to θ at the end of training. For any model, most gradients in such a histogram should be ideally close to zero following the first-order necessary conditions of the Karush–Kuhn–Tucker theorem [74]. We observe that NN-CoRes achieve the most near-zero gradients *while* satisfying the BCs/IC. In contrast, PINN_{HC}, which is also designed to automatically satisfy the BCs/IC, struggles to minimize the PDE loss (note that models such as PINNs and PINN_{DW} which do not strictly satisfy BCs/IC, can achieve lower overall loss values and perhaps show better first-order necessary conditions at the expense of violating the BCs/IC).

We observe in Table 2.1 that the performance of all the methods drops as either the problem complexity increases (e.g., Burgers’ vs. LDC) or PDE parameters are changed to introduce

Problem	Capacity	NN-CoRes		GP _{OR}		PINN		PINN _{DW}		PINN _{HC}	
		4 ⊗ 10	4 ⊗ 20	1,000	2,000	4 ⊗ 10	4 ⊗ 20	4 ⊗ 10	4 ⊗ 20	4 ⊗ 10	4 ⊗ 20
Burgers’	$\nu = 0.02/\pi$	0.80e−3	0.89e−3	2.24e−1	1.69e−1	2.42e−3	1.50e−3	2.86e−3	3.18e−3	3.41e−1	3.36e−1
	$\nu = 0.01/\pi$	1.91e−3	1.29e−3	1.69e−1	2.08e−1	4.26e−3	4.38e−3	1.93e−2	5.79e−3	3.65e−1	3.52e−1
Elliptic	$\alpha = 20$	4.50e−3	2.04e−3	2.44e−3	4.06e−3	6.55e−1	4.68e−1	2.97e−1	1.26e−1	6.35e−1	5.95e−1
	$\alpha = 30$	4.38e−3	1.24e−3	7.08e−3	6.55e−3	8.45e−1	5.55e−1	1.69e−1	1.19e−1	2.89e−1	6.53e−1
Eikonal	$\epsilon = 0.05$	0.52e−3	0.37e−3	1.76e−1	1.25e−1	2.76e−3	2.19e−3	2.01e−3	1.54e−3	2.89e−1	2.88e−1
	$\epsilon = 0.01$	4.60e−3	4.99e−3	2.18e−1	2.06e−1	6.41e−3	6.38e−3	5.03e−3	4.97e−3	2.91e−1	3.42e−1
LDC	$A = 3$	1.86e−1	8.67e−2	—	—	2.72e−1	1.28e−1	3.01e−1	1.25e−1	4.32e−1	5.29e−1
	$A = 5$	3.11e−1	2.79e−1	—	—	7.17e−1	6.77e−1	7.16e−1	6.23e−1	1.03e0	9.11e−1

Table 2.1 Summary of comparative studies: We report $L_{2,e}$ of different methods as a function of model capacity and PDE parameter. The symbol $\otimes$ indicates the network architecture (e.g., $4\otimes 10$ is an NN which has four 10–neuron hidden layers). Unlike NN-based methods, GP_{OR}’s accuracy relies on the number of interior nodes which we set to 1,000 or 2,000. GP_{OR} is not applied to LDC as it relies on manual derivation of the equivalent variational problem which, unlike the first three PDEs, is not done by the developers [1].

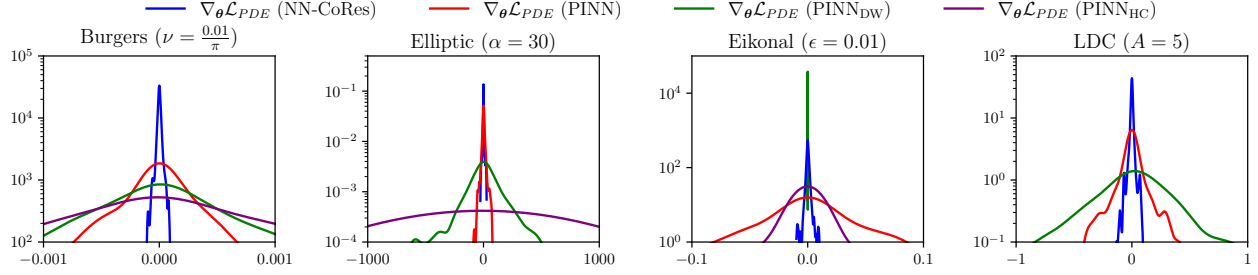
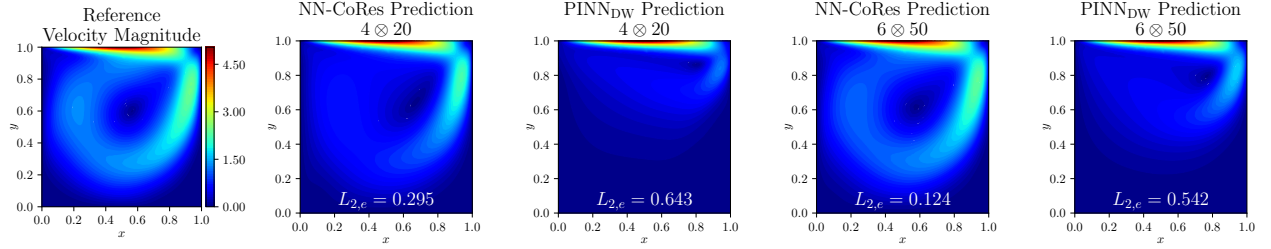
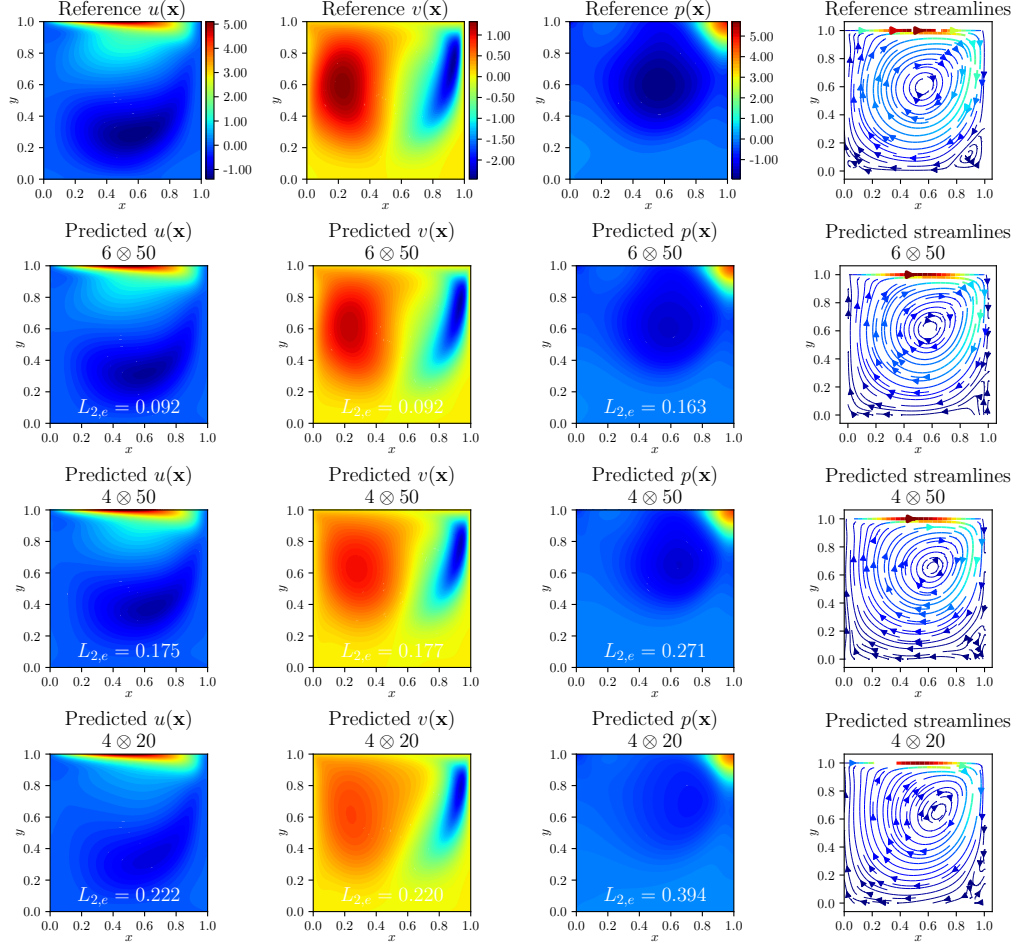


Figure 2.4 Histograms of PDE loss gradients: NN-CoRes is in general more effective in minimizing its loss function as a larger portion of its gradients satisfy the first order optimality condition. While PINN_{DW} has more near-zero gradients in the Eikonal problem, it does so at the expense of violating the BC loss term. All models in this figure have a $4 \otimes 20$ architecture.

more nonlinearity (e.g., $A = 3$ vs $A = 5$ in LDC). This trend is expected since we do not change the architecture and training settings across our experiments. That is, we can increase the accuracy of all methods by increasing their capacity or improving the training process. We demonstrate this improvement for the LDC problem in Figure 2.5a where the errors of PINN_{DW} and especially NN-CoRes are decreased by merely increasing the size of their networks. To assess the convergence behavior of NN-CoRes for this problem, we keep increasing the size of its mean function and observe in Figure 2.5b that its performance improves (note that the training mechanism is not altered in these studies and merely the size of the models is increased). We also note that in the case of LDC problem pressure is only known at a single point on the boundary (rather than everywhere on the boundary) which indicates that the kernel-weighted CoRes insignificantly help the model in learning pressure. We study this behavior further in Section 2.3.5.



(a) **Reference vs predictions (LDC with $A = 5$):** Performance improves as the network sizes increase. The small NN-CoRes is more accurate than the large PINN_{DW}.



(b) **Effect of network size on the accuracy in the LDC problem:** The accuracy of NN-CoRes consistently increases in predicting u, v , and p as the network size (either the number of hidden layers or their size) increases.

Figure 2.5 Error analysis: NN-CoRes achieve smaller errors and increasing the size of their networks provides more improvement compared to methods such as PINN_{DW}.

To gain more insight into the performance of each method, we visualize the error maps for some of our simulations in Figure 2.6. We observe that GP_{OR} is least accurate either

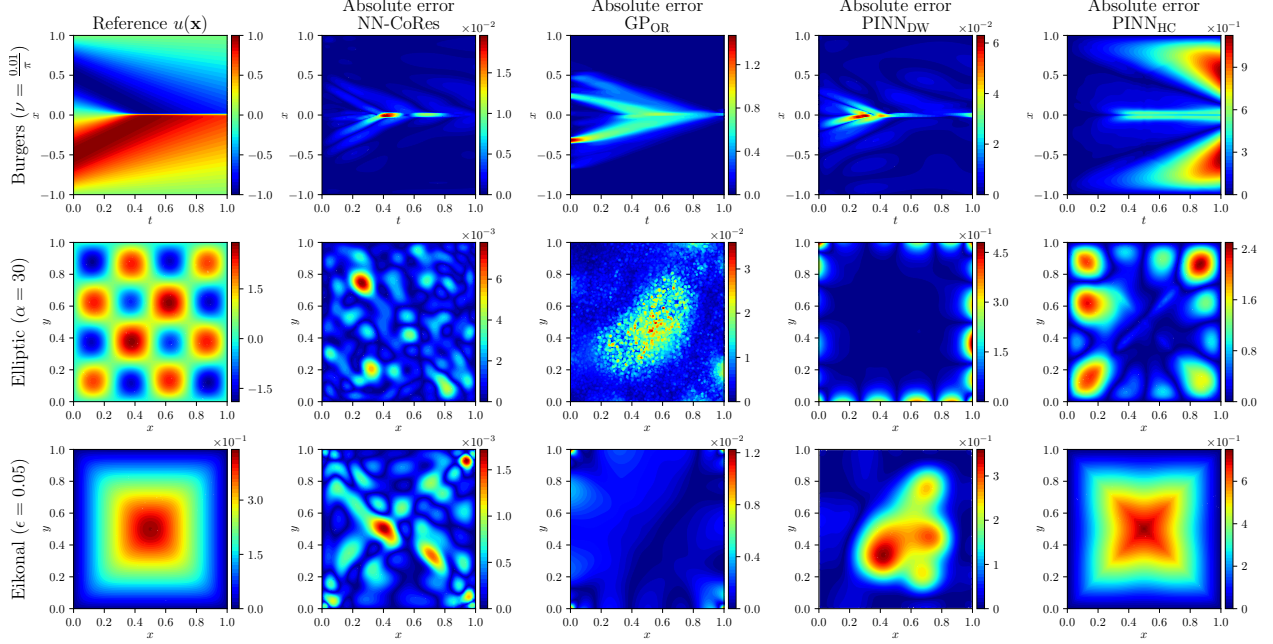


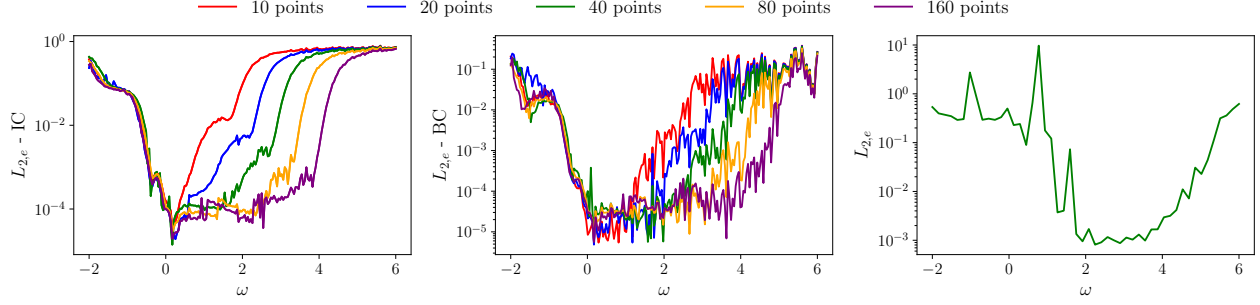
Figure 2.6 Reference solutions and error maps: NN-CoRes provide much lower errors compared to other methods (note the scales of the error bars).

in regions with sharp solution gradients or inside the domain where boundary information is not effectively propagated inward by the zero-mean GP. For PINN_{DW} , the errors are predominantly close to either the boundaries or where the PDE solution has large gradients. PINNs' errors in reproducing the BCs/IC are eliminated in PINN_{HC} but at the expense of significant loss of accuracy elsewhere in the domain. These issues are largely addressed by NN-CoRes which reproduce BCs/IC and approximate solutions with high gradients quite well. Specifically, we observe that in the case of Elliptic and Eikonal PDEs NN-CoRes' errors are quite evenly distributed in the domain and almost vanish close to the boundaries. In the case of Burgers' equation, NN-CoRes' errors are mostly around the $x = 0$ line where the shock develops for an inviscid flow when $\nu = 0$ (this error behavior is similar to other methods but the magnitude of the errors is much smaller for NN-CoRes).

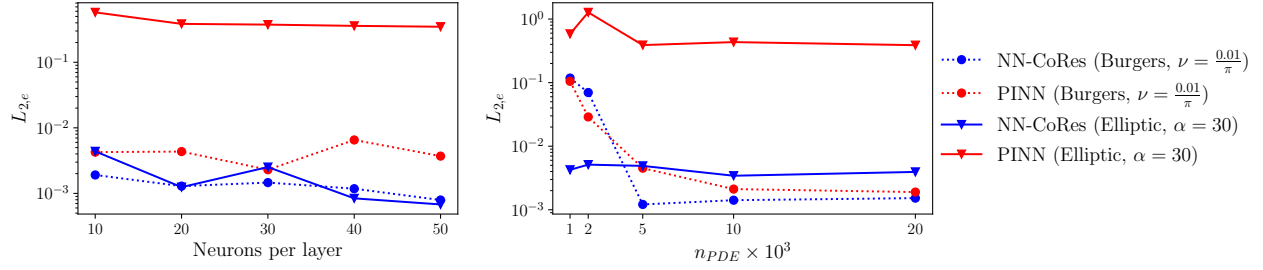
2.3.4 Sensitivity Analyses

In this section, we conduct a wide range of sensitivity studies to assess the impact of factors such as random initialization, noise, network architecture, and optimization settings on the results reported in Section 2.3.3.

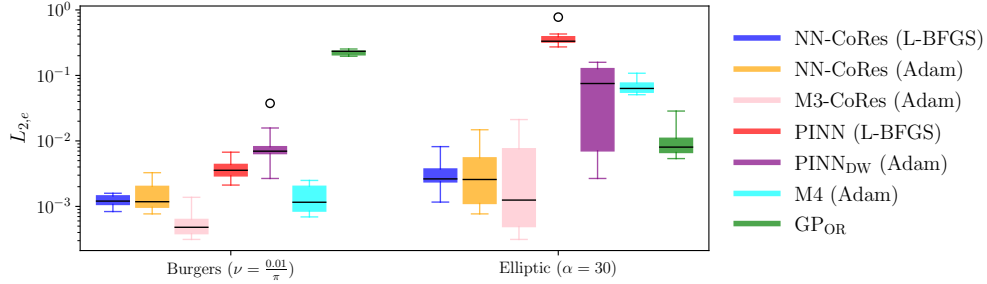
As stated in Section 2.2, the performance of NN-CoRes is quite robust to the values chosen for $\phi = 10^\omega$ as long as they lie within a certain range. To obtain this range, we conduct the following inexpensive experiment using the Burgers' problem in Equation (2.5) and $c(\mathbf{x}, \mathbf{x}'; \phi, \delta, \sigma^2 = 1) = \exp\{-\phi(x - x')^2 - \phi(t - t')^2\} + \mathbb{1}\{\mathbf{x} == \mathbf{x}'\}\delta$. We first sample n_{train} equally spaced boundary samples using the provided analytic IC and BCs. To quantify the effect of data size on the results, we consider 5 scenarios where $n_{train} \in \{10, 20, 40, 80, 160\}$. For each of these five cases, we build 200 independent GPs whose only difference is the value that we assign to $\phi = 10^\omega$. Specifically, we consider 200 equally spaced values in the $[-2, 6]$ range for ω and use each of these values in one of the GPs which all have a non-zero mean function (we use a deep NN whose parameters are randomly initialized and frozen as the mean function). Once these GPs are built, we use them to predict on $n_{test} = 10^4$ boundary points (see Equation (2.1) for the prediction formula). The results of this study are shown in the left and middle plots in Figure 2.7a and indicate that as more training data are sampled on the boundaries a wider range of values for ω result in small test errors. We highlight that this study is computationally very fast since none of the GPs are optimized; rather their parameters are either chosen by us (i.e., ω), or fixed (i.e., δ, σ^2 , and parameters of the NN mean).



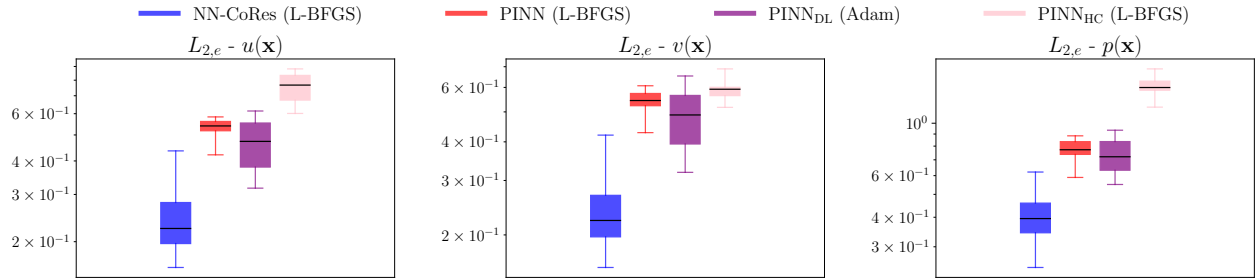
(a) Effect of ω on GP's interpolation power (left and middle plots) and NN-CoRes (right plot). Burgers' equation is used in this study.



(b) Effect of network size (left, $n_{PDE} = 10^4$) and n_{PDE} (right, $4 \otimes 20$ architecture) on the accuracy of PINNs and NN-CoRes.



(c) Effect of optimizer, random initialization, and architecture type on errors for the Burgers' and Elliptic problems.



(d) Effect of random initialization and optimizer on errors for the LDC problem ($A = 5$). All models have $4 \otimes 20$ architecture and use $n_{PDE} = 10^4$.

Figure 2.7 Sensitivity studies: We analyze the sensitivity of our results to factors such as the roughness parameters in the kernel, optimization settings, network architecture, and initialization. Based on these experiments, NN-CoRes provide a more robust machine learning-based approach for solving different nonlinear PDEs.

Following the above study, we have decided to use 40 boundary points in NN-CoRes. Based on the left and middle plots in Figure 2.7a, $\omega = 2$ seems to be a good choice (but not the optimum one) for minimizing the error in reproducing the IC and BCs. To see the effect of this choice on the performance of a trained NN-CoRes, we again vary ω (50 equally spaced values in the $[-2, 6]$ range) but this time we train an NN-CoRes model for each value of ω . We evaluate the performance of these models in solving the Burgers' equation by reporting the Euclidean norm of the error $L_{2,e}$ at $n_{test} = 10^4$ points randomly located in the domain. The results are shown in the right plot in Figure 2.7a and indicate that although $\omega = 2$ is not the optimum choice, it yields a model whose performance is close to optimal (the optimum model is achieved via an ω close to 3).

We now conduct a few extensive experiments to study the effect of network size and optimization settings on the performance of various NN-based models. First, we fix everything and increase the number of neurons in each hidden layer from 10 to 50 (at increments of 10) and solve the Burgers' and Elliptic PDEs via both NN-CoRes and PINNs. We then repeat this experiment but this time we fix the architecture to $4 \otimes 20$ and incrementally increase n_{PDE} from 10^3 to 10^4 . The results of these two experiments are summarized in Figure 2.7b and indicate that NN-CoRes is much less sensitive to the problem than PINNs which perform quite well on Burgers' but fail at accurately solving the Elliptic PDE that has direction-dependent frequency. We also observe that NN-CoRes provide lower errors than PINNs in almost all simulations.

In our next experiment, we study the effects of optimizer (L-BFGS vs Adam), random initialization, and architecture type on the performance of various models. To this end, we again consider the Burgers' and Elliptic PDE systems and solve them with six NN-based methods and GP_{OR}. For each case we repeat the training process of each model 10 times to quantify the effect of random initialization on the models' solution accuracy. For these experiments, we also consider a new network architecture that we denote by M3

which is introduced in [39] and aims to improve gradient flows by designing feed-forward networks with connections that resemble transformers [75]. In our framework, we replace the architecture that is used in all of our studies (which is a feed-forward neural network or an FFNN) with M3 and train the model with Adam (the resulting model is denoted by M3-CoRes). We also train another NN-based model denoted by M4 [39] whose architecture is the same as M3 but leverages dynamic weights in its loss function. We highlight that the simulations that leverage M3 as their architecture have more parameters (and hence learning capacity) than cases where FFNNs are used so we expect M3-based simulations to provide lower errors.

The results of these simulations are summarized in Figure 2.7c and indicate that (1) NN-CoRes and GP_{OR} are less sensitive to random initializations compared to PINNs and their variations, (2) unlike other models, NN-CoRes performs well in both PDE systems, i.e., our framework provides a more transferable method for solving PDEs via machine learning, and (3) architectures besides simple FFNNs (such as M3) can also be used in our framework to achieve higher accuracy.

The above experiments are based on the Burgers' and Elliptic PDE problems but our studies indicate that similar trends appear in other problems. To demonstrate this, we solve the LDC problem via four NN-based models that either use L-BFGS or Adam as their optimizer. We repeat the training process of each model 10 times to assess the effect of random parameter initialization on each model's performance. The results are summarized with the boxplots in Figure 2.7d and agree with our previous findings that indicate NN-CoRes consistently outperform other methods.

Finally, we investigate the effect of noisy boundary data on our results. We corrupt the solution values sampled from the IC and/or BCs before using them in our approach. We use a zero-mean normal distribution to model the noise and set the standard deviation to either 0.5% or 1% of the solution range. As shown in Figure 2.8, the solution accuracy decreases

as the noise variance increases (this trend is expected) but in all cases NN-CoRes are able to quite effectively eliminate the noise and solve the Burgers' and Elliptic PDE systems.

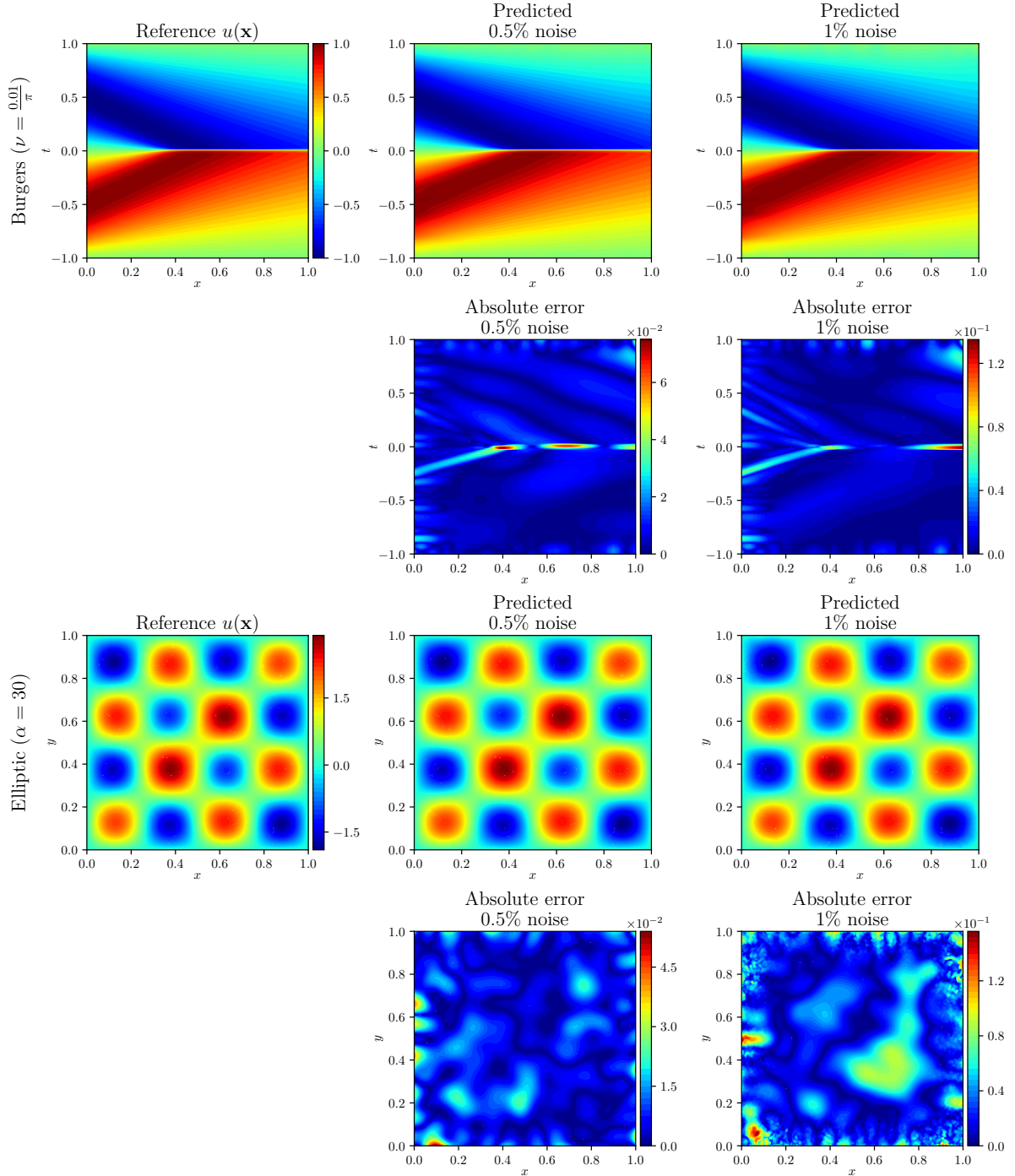


Figure 2.8 Reference and predicted solutions with noisy boundary data: We corrupt the samples obtained from boundary and initial conditions by either 0.5% or 1% of the solution range. In all cases, the performance of NN-CoRes is insignificantly affected by the noise.

2.3.5 Loss Behavior

To gain more insights into the training dynamics of our approach, we visualize the loss and accuracy during the training process in Figure 2.9 and Figure 2.10 where in the latter figure we track the errors individually for each output for the LDC problem. We provide these plots for both PINNs and NN-CoRes where the loss function of the former is based on Equa-

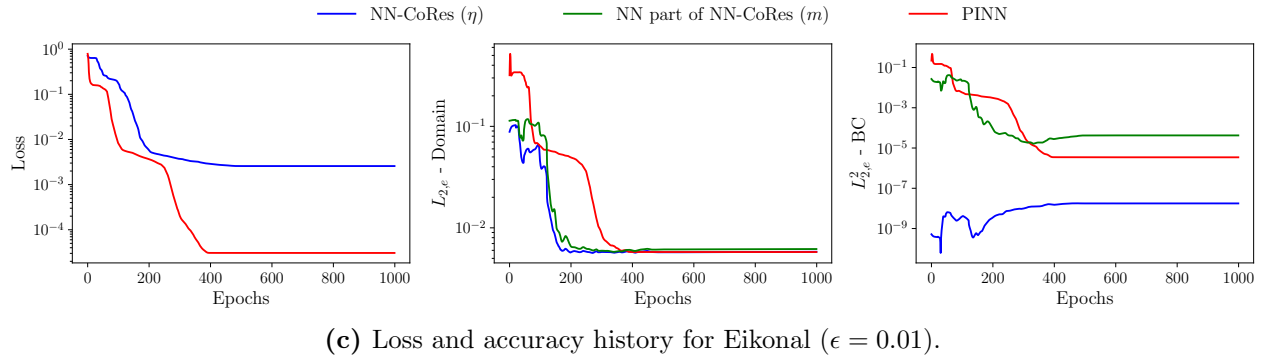
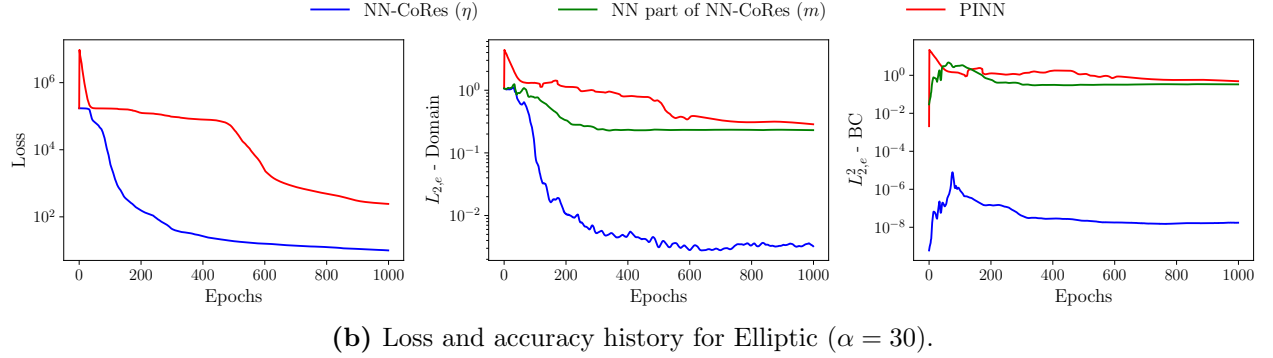
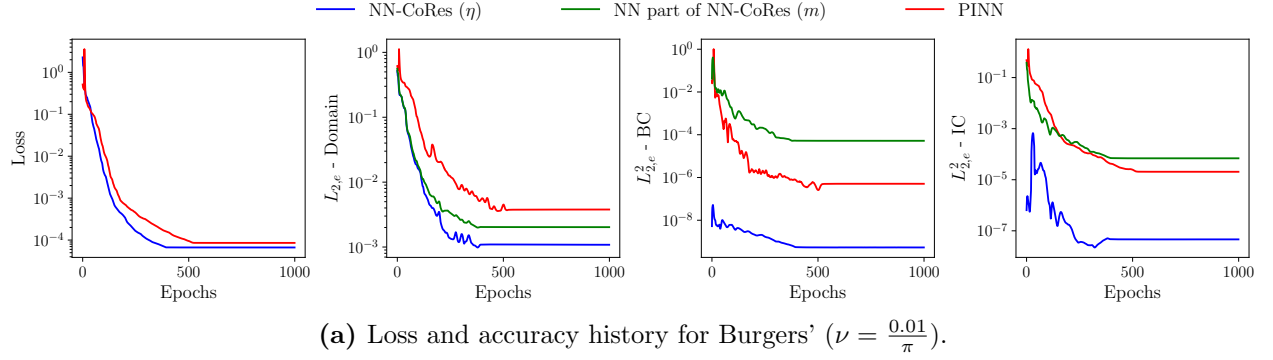


Figure 2.9 Loss convergence and error decomposition: NN-CoRes typically converge faster than PINNs and consistently provide more accurate solutions. The NN part of NN-CoRes benefits from the kernel-weighted CoRes not only on the boundaries, but also inside the domain.

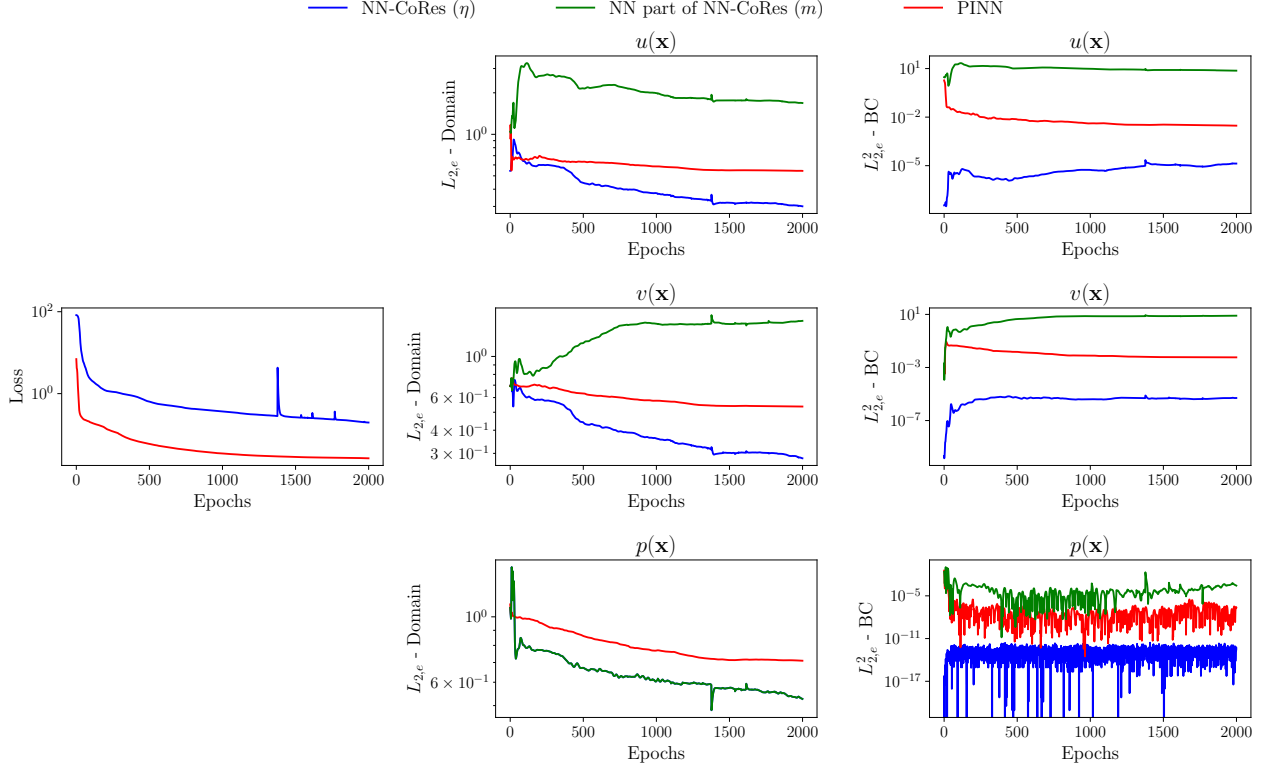


Figure 2.10 Loss convergence and error decomposition for LDC: The NN part of NN-CoRes benefits from the kernel-weighted CoRes not only on the boundaries, but also inside the domain. In the case of pressure, kernel-weighted CoRes do not contribute to the model’s predictions inside the domain as $p(\mathbf{x})$ is only known at a single point on the boundary (for this reason, the blue and green curves are indistinguishable).

tion (A.2-3) while NN-CoRes only use $\mathcal{L}_{PDE}(\theta)$ in their loss function. The solution accuracy is measured based on $L_{2,e}$ and $(L_{2,e})^2$ for points inside the domain and on its boundaries. Note that we square $L_{2,e}$ on the boundaries to be able to directly see its contribution to PINNs’ loss, see $\mathcal{L}_{BC}(\theta)$ and $\mathcal{L}_{IC}(\theta)$ in Equation (A.2-3). In the case of NN-CoRes, we also report the accuracy of its NN part on predicting the PDE solution to quantify the contributions of kernel-weighted CoRes towards the model’s predictions.

As it can be observed in Figures 2.9 and 2.10, NN-CoRes typically converge faster than PINNs, see the plots whose y -axis title is $L_{2,e}$ - Domain. We attribute this trend to the fact that, unlike in PINNs, the initial and boundary conditions are automatically satisfied in our models thanks to the kernel-weighted CoRes which are smooth functions. This feature

enables NN-CoRes to focus on satisfying the PDE system in module two of our framework while PINNs have to struggle with both the differential equations as well as the initial and boundary conditions.

An interesting trend in Figures 2.9 and 2.10 is that the errors of NN-CoRes are consistently lower than their NN components both in the domain and on the boundaries. That is, the kernel-weighted CoRes positively contribute to the model’s predictions both on the boundaries and inside the domain. This behavior is in sharp contrast to most approaches such as PINN_{HC} that satisfy the boundary conditions at the expense of complicating the training process.

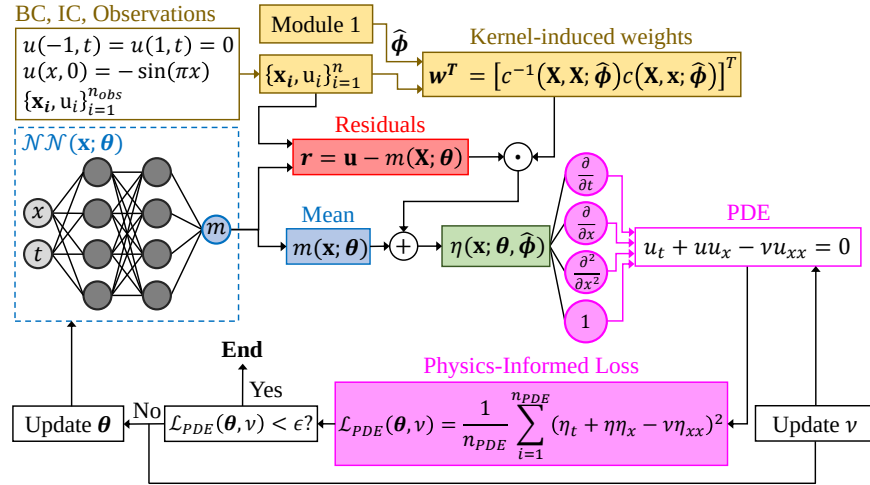
Another interesting trend that we observe in Figures 2.9 and 2.10 is that PINNs achieve lower loss values than NN-CoRes in the case of Eikonal and LDC problems. While lower loss values are desirable, in these cases the observed trends are misleading. To explain this behavior, we note that the loss function of NN-CoRes is simply $\mathcal{L}_{PDE}(\boldsymbol{\theta})$ as the boundary and initial conditions are automatically satisfied. However, the loss function of PINNs minimizes $\mathcal{L}_{IC}(\boldsymbol{\theta})$ and/or $\mathcal{L}_{BC}(\boldsymbol{\theta})$ in addition to $\mathcal{L}_{PDE}(\boldsymbol{\theta})$. That is, since PINNs do not strictly satisfy the BCs/IC, they are less regularized and hence can minimize $\mathcal{L}_{PDE}(\boldsymbol{\theta})$ (which dominates the overall loss) in a more flexible manner. However, this behavior provides less accuracy since the boundary conditions are not learnt sufficiently well.

2.3.6 Inverse Problems

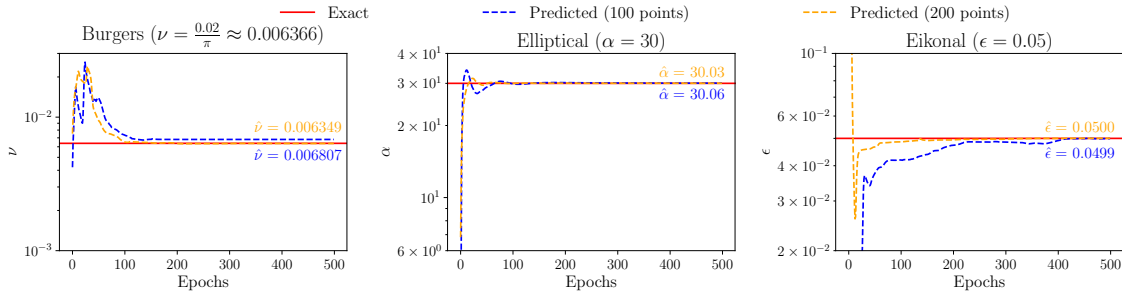
In the previous experiments we have only used the differential equations along with the IC and/or BCs in building NN-CoRes. In this section, we introduce an extension of our framework for solving inverse problems where (1) there are some (possibly noisy) labeled data available inside the domain (we refer to these samples as observations to distinguish them from the boundary data), and (2) one or more parameters in the differential equations

are unknown. Our goal in such applications is to solve the PDE system while estimating the unknown parameters.

As shown in Figure 2.11a, we modify our framework in two ways to solve the PDE system in Equation (A.2-2c) assuming ν is unknown but $u(\mathbf{x})$ is known at n_{obs} random points in the domain. Specifically, we (1) use the n_{obs} observations in the kernel of NN-CoRes in exactly the same way that the $n_{BC} + n_{IC}$ boundary data are handled by the kernel, and (2) treat ν as one additional parameter that must be optimized along with the weights and biases of the NN.



(a) Flowchart of module two of our framework for solving inverse problems. The flowchart is tailored to the PDE system in Equation (2.5).



(b) Convergence rates are fast and improve as more data are infused into our model.

Figure 2.11 Inverse problems via NN-CoRes: We modify the flowchart of module two in Figure 2.1 in two ways to solve a PDE system whose one or more parameters may be unknown. NN-CoRes treat observations (i.e., labeled solution data inside the domain) identically to boundary data and are very effective in using them in estimating the unknown PDE parameters.

To evaluate the performance of our approach in solving inverse problems, we consider the Burgers', Elliptic, and Eikonal PDE systems introduced in Section 2.3.1. We solve each problem in two scenarios where there are either $n_{obs} = 100$ or $n_{obs} = 200$ observations available in the domain. As shown in Figure 2.11b, in all cases NN-CoRes can estimate the unknown PDE parameter quite accurately. The convergence rate in all cases is quite fast and insignificantly reduces as n_{obs} is halved from 200 to 100.

2.4 Concluding Remarks

We introduce a general framework based on Gaussian processes to solve forward and inverse problems that involve nonlinear PDEs. Our framework adds the strengths of kernels to deep NNs via kernel-weighted corrective residuals that ensure the combined model reproduces labeled data *by construction*, i.e., we eliminate data loss terms from the training process of neural PDE solvers.

We design a modular, robust, and efficient framework to build NNs with kernel-weighted corrective residuals (or NN-CoRes for short) and show that the resulting model consistently outperforms competing methods on a broad range of experiments. This performance improvement is particularly impactful as our approach simplifies the training process of deep NNs while negligibly increasing the inference costs. As we extensively study in Section 2.3, our findings show that NN-CoRes are not only very robust to the choice of optimizer and initial parameter values, but also applicable to a wide range of neural architectures other than FFNNs. We also show in Section 2.3.6 that NN-CoRes can solve inverse problems with fast convergence rates. As proved in Section A.3 our approach strictly satisfies the boundary and initial conditions (and reproduce the in-domain data in the case of inverse problems) as the number of sampled data tends to infinity, regardless of the geometry and the variance of the noise that may be corrupting the data.

Although we have mainly used a simple FFNN as the mean function of NN-CoRes in our studies, we emphasize that our framework can easily accommodate any differentiable function approximator. This flexibility is demonstrated in Figure 2.7, where we achieve a superior performance by leveraging a more expressive architecture as the mean function, specifically the M3 model developed in [39]. Hence, the key takeaway is that our framework can incorporate a wide variety of parametric mean functions. One such example is utilizing a recurrent neural network as the mean function to predict the evolution of dynamic physical systems. As demonstrated in [76], our approach is also useful in addressing some of the limitations of conventional PDE solvers (e.g., the FEM) in causing local convergence in topology optimization.

The current limitation of our approach is that the contributions of the kernel-weighted CoRes decrease in the absence of boundary data. This behavior is also observed in the LDC problem where pressure is known only at a single point on the boundary. We believe devising periodic kernels is a promising direction for addressing this limitation which will be particularly useful in multiscale simulations where PDEs with periodic BCs frequently arise in the fine-scale analyses.

Chapter 3

Operator Learning in Parametric PDEs

3.1 Introduction

Operator learning naturally arises in many applications such as solving partial differential equation (PDE) systems [77], statistical description and modeling of random functions [78, 79], mechanistic reduced order modeling [80], speech inversion and sound recognition [81], or emulation of expensive simulations [82, 83, 84]. Recent advancements in machine learning (ML) have pushed operator learning to the forefront of research in both academia and industry. While the vast majority of the developments on this topic leverage variants of deep neural networks (NNs) [85, 52, 53, 86], kernel-based methods [4] and feature-map-based methods [87] have also recently been shown to be competitive. In this chapter, we propose a hybrid GP/NN-based framework for approximating mappings between infinite-dimensional function spaces. Our framework provides a first of its kind mechanism for simultaneously leveraging the strengths of both deep NNs and kernels for operator learning.

3.1.1 Description of the Operator Learning Problem

Let $\mathcal{U}$ and $\mathcal{V}$ denote two separable infinite-dimensional Banach spaces of continuous functions and define the nonlinear operator $\mathcal{G}^\dagger$ as:

$$\mathcal{G}^\dagger : \mathcal{U} \rightarrow \mathcal{V}. \quad (3.1)$$

Furthermore, let $\phi : \mathcal{U} \rightarrow \mathbb{R}^p$ and $\psi : \mathcal{V} \rightarrow \mathbb{R}^q$ be two bounded linear observation operators, that is:

$$\phi : u \mapsto u^p := [u(x_1), \dots, u(x_p)]^T \quad \text{and} \quad \psi : v \mapsto v^q := [v(y_1), \dots, v(y_q)]^T \quad (3.2)$$

where $\{x_j\}_{j=1}^p$ and $\{y_j\}_{j=1}^q$ denote the location of the points where the functions u and v are observed in their respective domains Ω_u and Ω_v . Our goal is to approximate $\mathcal{G}^\dagger$ using N pairs of input-output observations, i.e., we aim to approximate $\mathcal{G}^\dagger$ given $\mathcal{D} := \{\phi(u_i), \psi(v_i)\}_{i=1}^N$ where $\{u_i, v_i\}_{i=1}^N$ are elements from $\mathcal{U} \times \mathcal{V}$.

3.1.2 Summary of the Proposed Approach

We propose to cast the operator learning problem in Equation (3.1) as the following equivalent functional learning problem:

$$\tilde{\mathcal{G}}^\dagger : \mathcal{U} \times \mathcal{V}^* \rightarrow \mathbb{R} \quad (3.3)$$

where $\mathcal{V}^*$ is the dual space of $\mathcal{V}$, consisting of all continuous linear functionals from $\mathcal{V}$ to $\mathbb{R}$. For any pair $(u, \varphi) \in \mathcal{U} \times \mathcal{V}^*$, $\tilde{\mathcal{G}}^\dagger(u, \varphi)$ returns the scalar value $[\varphi, \mathcal{G}^\dagger(u)]$ defined as the dual pairing between φ and $\mathcal{G}^\dagger(u)$. If we specifically choose $\varphi = \delta_y$, where δ_y is the delta functional which evaluates the output function at a specific point $y \in \Omega_v \subset \mathbb{R}^d$, then we can

easily retrieve the desired operator $\mathcal{G}^\dagger$ given that $\mathcal{G}^\dagger(u)(y) = \tilde{\mathcal{G}}^\dagger(u, \delta_y)$.

In practice, (1) we only have access to a discretization of u , i.e., $\phi(u) \in \mathbb{R}^p$, and (2) we are only interested in pointwise evaluations $\mathcal{G}^\dagger(u)(y)$ with $y \in \Omega_v$. Hence, the functional learning problem can be reduced to identifying an operator:

$$\tilde{\mathcal{G}}_p^\dagger : \mathbb{R}^p \times \Omega_v \rightarrow \mathbb{R}. \quad (3.4)$$

such that $\tilde{\mathcal{G}}_p^\dagger(\phi(u), y) \approx \mathcal{G}^\dagger(u)(y)$.

Since the mapping in Equation (3.4) is between finite-dimensional Euclidean spaces, it can be numerically approximated via classical NN/GP-based regression techniques. This regression-based approach differs from other kernel-based methods such as [4] that directly approximate the operator rather than its dual action on $\mathcal{V}^*$. The overall articulation of our regression-based approach for operator learning is schematically shown in Figure 3.1 which also applies to other well-known techniques such as DeepONet [52]. In this chapter, we approximate the mapping $\tilde{\mathcal{G}}_p^\dagger$ in Equation (3.4) via GPs whose parameters are estimated either in a purely data-driven manner, or via both data and physics. In the former case, we use maximum likelihood estimation (MLE) for parameter optimization and set the mean function to be either zero or a deep NN. In the latter case, we rely on GPs whose mean functions are represented via neural operators (e.g., FNO or DeepONet) whose parameters are estimated via a weighted combination of MLE and mean squared error (MSE).

As in [4], the diagram in Figure 3.1 can be made commutative by introducing two reconstruction (optimal recovery) operators associated with ϕ and ψ that map the observations in $\mathbb{R}^p$ and $\mathbb{R}^q$ back to $\mathcal{U}$ and $\mathcal{V}$, respectively. Since ϕ and ψ are bounded and linear, their elements are in $\mathcal{U}^*$ and $\mathcal{V}^*$ which are the dual spaces of $\mathcal{U}$ and $\mathcal{V}$, respectively. Assuming $\mathcal{U}$ and $\mathcal{V}$ are reproducing kernel Hilbert spaces (RKHSs) and the elements of the observation operators are linearly independent, the reconstruction (optimal recovery) operators can be

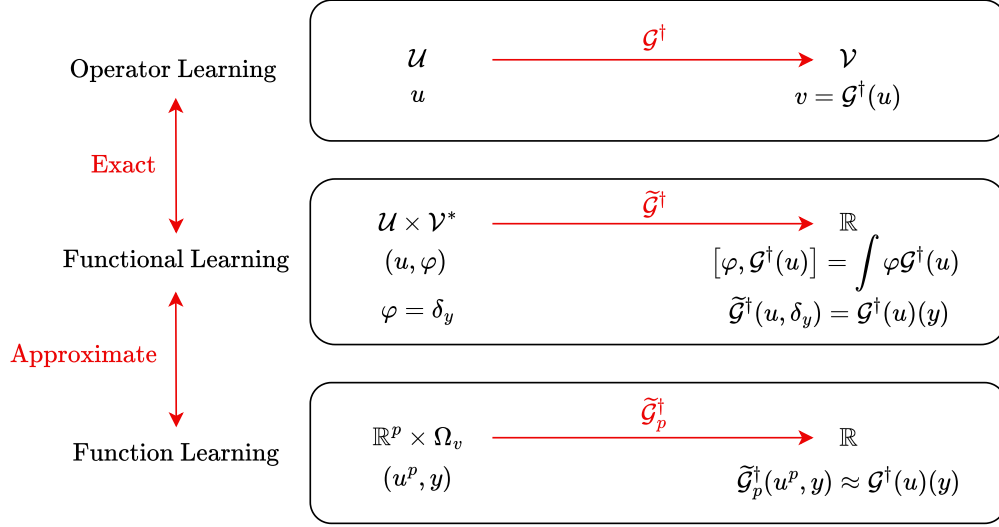


Figure 3.1 Diagram of our framework for operator learning: We convert the operator learning problem to a regression one which can be solved via GPs. For multi-output operator learning, we use multi-response or multi-task GPs.

obtained via the usual representer theorem as detailed in [4].

3.1.3 Illustrative Example

To contextualize the above abstract definitions, we consider the Burgers' equation with Dirichlet boundary conditions (BCs):

$$\begin{aligned}
 v_t + vv_x - \zeta v_{xx} &= 0, & \forall x \in (0, 1), & \quad t \in (0, 1] \\
 v &= 0, & \forall x = 0, & \quad t \in [0, 1] \\
 v &= 1, & \forall x = 1, & \quad t \in [0, 1] \\
 v &= u, & \forall x \in [0, 1], & \quad t = 0,
 \end{aligned} \tag{3.5}$$

where x and t denote space and time, v is the PDE solution, u denotes the initial condition (IC), and $\zeta = 0.1$ is the kinematic viscosity. We denote the $(0, 1)^2$ domain and its boundaries where IC and BCs are specified by Ω , $\partial\Omega_t$, and $\partial\Omega_x$, respectively. The goal in this problem

is to learn the nonlinear operator $\mathcal{G}^\dagger$ that maps u to v , i.e., to the solution field in Ω . To approximate $\mathcal{G}^\dagger$, we consider two scenarios:

- **Data-driven operator learning:** We take the linear operators ϕ and ψ defined in Section 3.1.1 to be the function evaluations at a set of p and q collocation points (CPs) on $\partial\Omega_t$ and in Ω , respectively. This choice provides the training dataset $\mathcal{D} := \{\phi(u_i), \psi(v_i)\}_{i=1}^N$ where $\phi(u_i)$ and $\psi(v_i)$ are of size p and q , respectively, and the pair $\{u_i, v_i\}$ satisfies the PDE system in Equation (3.5). In this scenario, we follow the diagram in Figure 3.1 and approximate $\mathcal{G}^\dagger$ via $\tilde{\mathcal{G}}_p^\dagger$ using $\mathcal{D}$.
- **Physics-informed operator learning:** In this scenario, we leverage the PDE system in Equation (3.5) as well as $\mathcal{D}$ to approximate $\mathcal{G}^\dagger$. That is, we require the N_{pi} samples $\mathcal{D}_{pi} := \{u_i, \hat{v}_i\}_{i=1}^{N_{pi}}$, where the elements of $\hat{v}_i$ are predicted by $\tilde{\mathcal{G}}_p^\dagger$, to satisfy the PDE system in Equation (3.5).

We set $N = 400, p = 100, q = 12^2, N_{pi} = 50$, and consider ϕ and ψ to be pointwise function evaluations. The GPs' training mechanisms and the experimental setup for both scenarios are detailed in Sections 3.2 and 3.3, respectively. We provide example train and test samples along with our predictions in Figure 3.2. It is observed that (1) the predictions achieve small relative $L2$ errors which are 7.35% and 1.65% in the data and physics-informed cases, respectively, and (2) leveraging the physics increases the approximation accuracy.

3.1.4 Review of Related Literature

Operator learning aims at approximating mappings between infinite-dimensional function spaces and is particularly well suited for solving parametric nonlinear PDEs. Extensive research over the past few decades has explored operator learning in the context of methods such as polynomial chaos expansion [88, 89, 90] or the stochastic finite element method

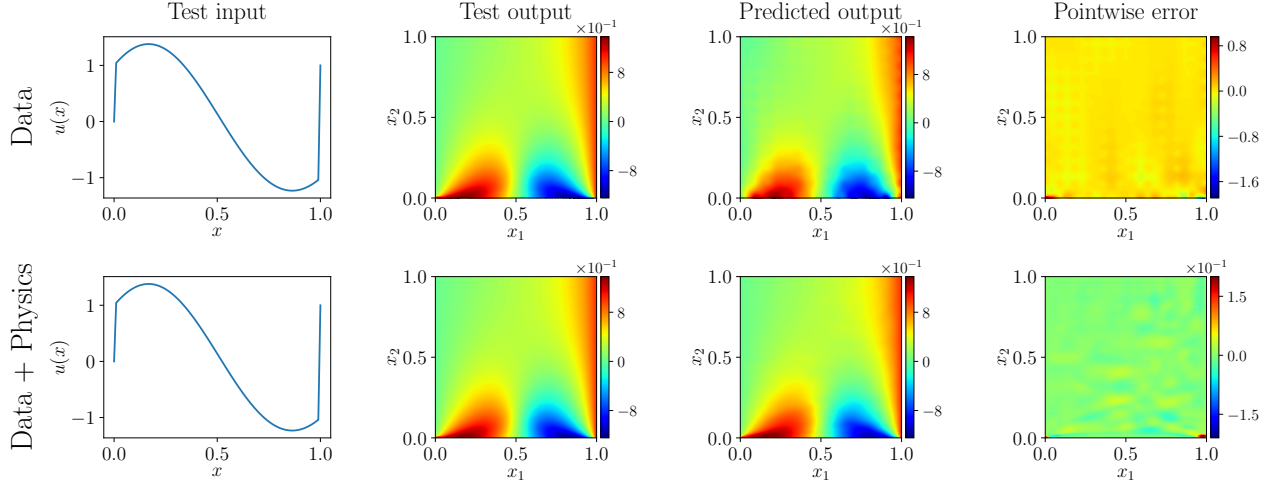


Figure 3.2 Data-driven and physics-informed operator learning for the Burgers' problem with Dirichlet BCs: The training dataset has 400 pairs of $\{u_i, v_i\}$ that satisfy Equation (3.5). The observation operators ϕ and ψ sample u_i and v_i at $p = 100$ and $q = 12^2$ collocation points, respectively. It can be observed that leveraging the physics reduces the prediction error. A DeepONet is used as the mean function of the GP in both data-driven and physics-informed cases.

(FEM) [91, 92, 93]. These methods have been widely applied to solving stochastic and parametric PDEs. In essence, they represent both the input function and the PDE solution via a set of suitable basis functions, and then learn a high-dimensional mapping between the coefficients of these two sets of bases. Similarly, GP and numerical homogenization methods have been shown to be data efficient and achieve competitive performance for learning/compressing the solution operator of linear elliptic PDEs with rough coefficients [94, 95, 96].

Neural Operators: Fueled by their success in many scientific applications and the advancements in hardware and software, deep NNs are increasingly used for operator approximation. One of the main distinguishing factors among different methods is the network architecture which may be designed via multi-layer perceptrons, convolution layers, attention mechanism, skip connections, graphs, or many other alternatives [85, 52, 86, 97]. We review some of the most prominent methods below and refer the interested reader to [98, 99, 100] for more information.

Following the universal approximation theorem for operators [101], a standard DeepONet

[102] approximates an operator via a deep NN that combines two sub-networks that encode the input function (branch net) and the spatiotemporal coordinates (trunk net). Since their inception, DeepONets have been extended in a few directions and a particular one that is related to our work is physics-informed DeepONet [103] which aim to reduce the reliance of the network on large training datasets by regularizing the training process based on the known PDE system.

Integral neural operators, first introduced in [104, 105], express the solution operator as an integral operator of Green’s function. Their network primarily consist of lifting, iterative kernel integration, and projection layers. To accelerate the kernel evaluations and increase efficiency, FNO [106] was developed where the integral kernel is parameterized in the Fourier space where the fast Fourier transform (FFT) and its inverse are used to accelerate the computations. Similar to DeepONets, FNO has been extended in multiple directions to better capture spatial signal behaviors [107] or to build a physics-informed neural operator (PINO) that leverages the physics [108] to reduce the reliance on training data.

To build more flexible neural operators that accommodate irregular geometries and sampling grids, [109] introduces a coordinate-based model that leverages implicit neural representations to encode functions into low-dimensional latent spaces and, in turn, infer mappings between function representations in the latent space. Another example is GNOT [110] which is a transformer-based model that leverages attention and geometric gating mechanisms to handle irregular data grids and multiple input functions. Multi-grid techniques which are ubiquitously used in numerical solvers have also been combined with deep NNs to increase the accuracy and flexibility of neural operators [111, 112, 113].

Operator-valued Kernels: Functional data analysis (FDA) [114] also deals with learning mappings between function spaces and has been extensively studied in the statistics literature as a natural extension of multi-response (aka multi-task or multi-output function) learning problems [115, 116, 117, 118, 119]. In this context, the focus has primary been on the use

of RKHS theory which naturally enables the construction of optimal recovery maps between function spaces via the representer formula [120, 121, 122]. In the context of learning PDE operators, recently, [4] introduced a framework based on the theory of operator-valued RKHS and GPs where the reconstruction maps (corresponding to the observation operators) as well as the finite-dimensional map connecting u^p and v^q are all formulated as optimal recovery problems whose solution can be identified by the representer formula. Motivated by this work and FNO, kernel neural operators [123] are recently developed that use parameterized, closed-form, finitely-smooth, and compactly-supported kernels with trainable sparsity parameters in integral operators. A more technical review of [43] and its relation to this article is provided in Section 3.2.4.

GP Regression: The mapping in Equation (3.4) is between finite-dimensional Euclidean spaces and hence can be numerically approximated via GPs which have long been used for probabilistic regression [124, 125, 126, 127, 128, 62]. Over the past few years, GPs have attracted some attention for solving nonlinear PDE systems. The majority of existing works [56, 57, 58] employ zero-mean GPs and with this choice, solving a PDE system amounts to designing the GP’s kernel whose hyperparameters are obtained via either MLE or a regularized MLE where the penalty term quantifies the GP’s error in satisfying the PDE system. Using zero-mean GPs, [1] casts solving nonlinear PDEs as an optimal recovery problem whose loss function is derived based on the PDE system and aims to estimate the solution at a finite number of interior nodes in the domain. Once these values are estimated, the PDE solution is approximated anywhere in the domain via kernel regression. The scalability of this approach can be increased using sparse GPs [129, 130, 58]. GPs with non-zero means have been employed for solving PDEs in a data-driven manner [60] but [131] introduced the first robust mechanism for solving nonlinear PDE systems via NN-mean GPs for both forward and inverse problems. This latter work serves as a foundation for the current chapter.

3.1.5 Contributions and Chapter Outline

We summarize our main contributions as follows:

- We introduce a hybrid GP/NN-based operator learning framework. Our approach incorporates both zero-mean and NN-mean GPs both of which provide very competitive performance. The latter option, in particular, leverages the strengths of both kernel methods and neural operators such as DeepONet and FNO.
- We consider both data-driven and physics-informed operator learning scenarios. In the former case, we use MLE for parameter optimization and set the mean function to be either zero or a neural operator. In the latter case, we only use NN-mean GPs whose parameters are estimated via a weighted combination of MLE and PDE residuals.
- We address the computational costs and ill-conditioning issues associated with the covariance matrix of GPs by (1) leveraging specific kernel structures and (2) proper initialization of the kernel parameters that are either trained (for zero-mean GPs) or frozen (for NN-mean GPs).
- We introduce the first *zero-shot* mechanism for operator learning based on zero-mean GPs whose kernel parameters are properly initialized.
- We test our approach on single- and multi-output operator learning problems and demonstrate how the prediction accuracy is affected by (1) replacing MSE via MLE in data-driven operator learning, (2) augmenting MLE with physics in the case of NN-mean GPs, and (3) using the trained models for extrapolation.

The rest of the chapter is organized as follows. We introduce our framework in Section 3.2 where we consider both data-driven and physics-informed scenarios in Sections 3.2.1 and 3.2.2, respectively. Details on the initialization of the kernel parameters, stability of

the covariance matrix, and inference costs are also included in Section 3.2. We compare our approach against competing methods in Section 3.3 using single- and multi-output operators with and without incorporating physics into the operator learning problem. Concluding remarks and future research directions are provided in Section 3.4.

3.2 Proposed Framework

We first introduce our approach in the context of data-driven operator learning in Section 3.2.1 and then consider the physics-informed version in Section 3.2.2 to enable augmenting the training data with physical laws. The data-driven scenario in Section 3.2.1 can leverage either zero-mean or non-zero mean GPs while the physics-informed one in Section 3.2.2 relies on non-zero mean GPs. To ensure numerical stability and computational efficiency, in Section 3.2.3 we study the kernel parameters and provide general guidance for properly initializing them and potentially updating them in zero-mean and NN-mean GPs. We conclude this section by comparing our approach to the optimal recovery-based approach of [4] and then commenting on its inference complexity in Sections 3.2.4 and 3.2.5, respectively.

3.2.1 Data-driven Operator Learning

Single-output Operators

In Section 3.1.2 we transformed the operator learning problem into an equivalent regression one with the goal of learning $\tilde{\mathcal{G}}_p^\dagger : \mathbb{R}^n \times \Omega_v \rightarrow \mathbb{R}$. To approximate $\tilde{\mathcal{G}}_p^\dagger$ we start by placing a GP prior on it:

$$\tilde{\mathcal{G}}_p^\dagger \sim GP(m(\phi(u), y; \theta), c([\phi(u), y], [\phi(u'), y']; \beta, \sigma^2)) \quad (3.6)$$

where $\phi(u)$ is the linear observation operator defined in Equation (3.2), $m(\phi(u), y; \theta)$ is the prior mean function parameterized by θ , and $c([\phi(u), y], [\phi(u'), y']; \beta, \sigma^2)$ is a kernel with hyperparameters β and σ^2 . We elaborate on the choice of the mean function later in this section. As for the kernel, there are many choices available but in this work we employ the Gaussian or Matérn covariance functions defined as:

$$c(x, x'; \beta, \sigma^2) = \sigma^2 \exp\{-(x - x')^T \text{diag}(\beta)(x - x')\} = \sigma^2 \exp\left\{-\sum_{i=1}^{d_x} \beta_i (x_i - x'_i)^2\right\} \quad (3.7a)$$

$$c(x, x'; \beta, \sigma^2) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu(x - x')^T \text{diag}(\beta)(x - x')} \right)^\nu K_\nu \left(\sqrt{2\nu(x - x')^T \text{diag}(\beta)(x - x')} \right) \quad (3.7b)$$

where x denotes the feature vector with length d_x , β are the length-scale parameters, σ^2 is the process variance, K_ν is the modified Bessel function of the second kind, and Γ is the gamma function. It is common practice [132, 133] to restrict $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$ since these values lead to simplified expressions of K_ν and hence a substantial decrease in the computational cost. In addition, ν is not typically optimized but rather chosen a priori to avoid evaluating repeatedly K_ν .

The inductive bias that the kernels in Equation (3.7b) encode into the learning process is that close-by inputs x and x' have similar (i.e., correlated) output values. The degree of this correlation also depends on β whose magnitude quantifies the pace at which the correlations die out as the distance between x and x' increases.

Predicting $\mathcal{G}^\dagger(u^*)(y^*)$ for an unseen input function u^* at unseen output location y^* requires the estimation of θ , σ^2 , and β . To this end, we combine the training data $\mathcal{D} := \{\phi(u_i), \psi(v_i)\}_{i=1}^N$ with maximum likelihood estimation. To accommodate our regression problem, we reorganize $\mathcal{D}$ as $\{(\phi(u_i), y_j), [\psi(v_i)]_j\}_{i,j=1}^{N,q}$ where $\phi(u_i) \in \mathbb{R}^p$ and $y_j \in \Omega_v$ denote an input pair whose corresponding scalar output is $[\psi(v_i)]_j \in \mathbb{R}$. For notational convenience, we denote the inputs and outputs of this dataset by $\mathbf{X} = \{\phi(u_i), y_j\}_{i,j=1}^{N,q}$ and $\mathbf{v} = \{[\psi(v_i)]_j\}_{i,j=1}^{N,q}$,

respectively. The MLE process involves the following maximization problem:

$$\left[\hat{\theta}, \hat{\beta}, \hat{\sigma}^2\right] = \underset{\theta, \beta, \sigma^2}{\operatorname{argmax}} (2\pi)^{-Nq/2} |\mathbf{C}|^{-\frac{1}{2}} \exp \left\{ \frac{-1}{2} (\mathbf{v} - \mathbf{m})^T \mathbf{C}^{-1} (\mathbf{v} - \mathbf{m}) \right\}, \quad (3.8)$$

or equivalently:

$$\left[\hat{\theta}, \hat{\beta}, \hat{\sigma}^2\right] = \underset{\theta, \beta, \sigma^2}{\operatorname{argmin}} \frac{1}{2} \log |\mathbf{C}| + \frac{1}{2} (\mathbf{v} - \mathbf{m})^T \mathbf{C}^{-1} (\mathbf{v} - \mathbf{m}), \quad (3.9)$$

where $\mathbf{C} = c(\mathbf{X}, \mathbf{X}; \beta, \sigma^2)$ is the $Nq \times Nq$ symmetric positive-definite covariance matrix, $\mathbf{m} = m(\mathbf{X}; \theta)$ is the $Nq \times 1$ vector of mean function evaluations at $\mathbf{X}$, and $\log |\mathbf{C}|$ denotes the log-determinant of $\mathbf{C}$. The $\mathbf{v} - \mathbf{m}$ vector quantifies the residuals of the mean function in reproducing the training outputs and is discussed further in Section 3.2.1.

The minimization problem in Equation (3.9) is almost always solved with a gradient-based optimization technique where the parameters are first initialized and then iteratively updated until a convergence metric (e.g., maximum number of iterations or function evaluations) is met. The overall computational cost of this iterative process is dominated by the repeated inversion of the covariance matrix $\mathbf{C}$. Specifically, the computational complexity associated with $\mathbf{C}^{-1}$ is $\mathcal{O}((Nq)^3)$ which poses a major challenge in operator learning problems where N is typically in the order of thousands. Even with relatively small N , inverting $\mathbf{C}$ can still be very costly for high-resolution data where q is large.

To reduce the computational costs of inverting $\mathbf{C}$, we exploit the structure of the data where we assume that the discretization of u and the locations where v is observed are fixed across the N training samples. This assumption is quite reasonable in many operator learning problems and allows us to leverage the Kronecker product properties to bypass the construction and inversion of $\mathbf{C}$. To this end, we adopt a separable kernel of the form:

$$c([\phi(u), y], [\phi(u'), y']; \beta, \sigma^2) = c_\phi(\phi(u), \phi(u'); \beta_\phi, \sigma_\phi^2) c_y(y, y'; \beta_y, \sigma_y^2) \quad (3.10)$$

where we use Equation (3.7b) for both kernels on the right-hand side. Without loss of generality, we assume $\sigma_y^2 = 1$ hereafter¹. Using this formulation and the assumption on the data structure we rewrite the covariance matrix in Equation (3.9) as:

$$\mathbf{C} = \mathbf{C}_\phi \otimes \mathbf{C}_y, \quad (3.11)$$

where $\mathbf{C}_\phi = c_\phi(\mathbf{U}, \mathbf{U}; \beta_\phi, \sigma_\phi^2)$ and $\mathbf{C}_y = c_y(\mathbf{Y}, \mathbf{Y}; \beta_y)$ with $\mathbf{U} = \{\phi(u_1), \dots, \phi(u_N)\}$ and $\mathbf{Y} = \{y_1, \dots, y_q\}$, $\beta = [\beta_\phi, \beta_y]$. In this equation, $\mathbf{C}_\phi$ and $\mathbf{C}_y$ model the correlations among the discretized input functions and the locations where the output function is observed, respectively. Using the Kronecker product in Equation (3.11) we can simplify three of the computationally expensive operations in Equation (3.9):

$$\mathbf{C}^{-1} = \mathbf{C}_\phi^{-1} \otimes \mathbf{C}_y^{-1}, \quad (3.12a)$$

$$\mathbf{C}^{-1}(\mathbf{v} - \mathbf{m}) = \text{vec}(\mathbf{C}_y^{-1}(\mathbf{V} - \mathbf{M})\mathbf{C}_\phi^{-1}) \quad (3.12b)$$

$$|\mathbf{C}| = |\mathbf{C}_\phi|^q |\mathbf{C}_y|^N, \quad (3.12c)$$

where $\mathbf{V}$ and $\mathbf{M}$ are formed by reshaping $\mathbf{v}$ and $\mathbf{m}$ to $q \times N$ matrices, respectively, and $\text{vec}(\cdot)$ denotes the vectorization operation. Using Equation (3.12c) we now rewrite Equation (3.9) as:

$$\begin{aligned} [\hat{\theta}, \hat{\beta}, \hat{\sigma}_\phi^2] &= \underset{\theta, \beta, \sigma_\phi^2}{\text{argmin}} \mathcal{L}_{\text{MLE}}(\theta, \beta, \sigma_\phi^2) \\ &= \underset{\theta, \beta, \sigma_\phi^2}{\text{argmin}} \log \left(|\mathbf{C}_\phi|^q |\mathbf{C}_y|^N \right) + (\mathbf{v} - \mathbf{m})^T \text{vec}(\mathbf{C}_y^{-1}(\mathbf{V} - \mathbf{M})\mathbf{C}_\phi^{-1}), \end{aligned} \quad (3.13)$$

which can be used to simultaneously estimate the parameters of the mean and covariance functions of the GP.

We highlight that the optimization problems in Equations (3.9) and (3.13) both correspond

¹Alternatively, we could have also assumed $\sigma_\phi^2 = 1$ and estimate σ_y^2 instead, leading to the same result.

to our generic GP-based operator learning framework except that the latter one exploits the data structure and the kernel separability assumption in Equation (3.10) to decrease the computational costs and memory demands. If the locations where the output function is observed or the discretization of the input function vary across the samples, Equation (3.13) can still be used by either pre-processing the data (which may introduce some additional errors) or sparse GPs.

For a zero-mean GP, Equation (3.13) simplifies to:

$$\begin{aligned} \left[\widehat{\beta}, \widehat{\sigma}_{\phi}^2 \right] &= \underset{\beta, \sigma_{\phi}^2}{\operatorname{argmin}} \mathcal{L}_0(\beta, \sigma_{\phi}^2) \\ &= \underset{\beta, \sigma_{\phi}^2}{\operatorname{argmin}} q \log(|\mathbf{C}_{\phi}|) + N \log(|\mathbf{C}_y|) + \mathbf{v}^T \operatorname{vec}(\mathbf{C}_y^{-1} \mathbf{V} \mathbf{C}_{\phi}^{-1}), \end{aligned} \quad (3.14)$$

which can be solved via a first-order gradient-based optimizer (e.g., Adam) that leverages automatic differentiation to obtain the gradients of $\mathcal{L}_0(\beta, \sigma_{\phi}^2)$ with respect to β and σ_{ϕ}^2 . In typical GP regression problems, it is common practice to initialize gradient-based optimizers multiple times to avoid local optimality. This process substantially increases the cost of minimizing $\mathcal{L}_0(\beta, \sigma_{\phi}^2)$ due to the high costs of repeatedly constructing and inverting $\mathbf{C}_{\phi}$ and $\mathbf{C}_y$ to calculate $\mathcal{L}_0(\beta, \sigma_{\phi}^2)$ and its gradients. We address this computational issue in Section 3.2.3 where we show that β_y and β_{ϕ} can be, respectively, fixed and initialized to some judiciously chosen values to, in turn, optimize $\mathcal{L}_0(\beta_{\phi}, \sigma_{\phi}^2)$ only once.

For a GP whose mean function is parameterized with a deep NN (e.g., FNO or DeepONet), minimizing $\mathcal{L}(\theta, \beta, \sigma_{\phi}^2)$ can be prohibitively costly and memory intensive (even if β are initialized well and the optimization problem in Equation (3.13) is solved only once) since differentiating $\mathcal{L}(\theta, \beta, \sigma_{\phi}^2)$ with respect to θ involves large matrices. Moreover, we show in Section 3.2.3 that the simultaneous estimation of θ and β renders the covariance matrices ill-conditioned. As detailed in Section 3.2.3, we address these computational issues by fixing β and σ_{ϕ}^2 to some values that ensure (1) the covariance matrices are numerically stable, and

(2) the posterior distributions in Section 3.2.1 regress the training data. Fixing β and σ_ϕ^2 makes the first term in Equation (3.13) a constant and hence we can estimate θ via:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}_{\text{nn}}(\theta) = \underset{\theta}{\operatorname{argmin}} (\mathbf{v} - \mathbf{m})^T \operatorname{vec}(\mathbf{C}_y^{-1}(\mathbf{V} - \mathbf{M})\mathbf{C}_\phi^{-1}). \quad (3.15)$$

where $\mathbf{m}$ and $\mathbf{M}$ are the only terms that depend on θ , i.e., we can construct $\mathbf{C}_y^{-1}$ and $\mathbf{C}_\phi^{-1}$ once and store them so that evaluating $\mathcal{L}_{\text{nn}}(\theta)$ and its gradients only relies on matrix multiplications.

We notice in Equation (3.15) that if $\mathbf{C}_y$ and $\mathbf{C}_\phi$ are identity matrices (i.e., if the correlations among different u^p and different query points y are eliminated), $\mathcal{L}_{\text{nn}}(\theta)$ becomes the mean squared error (MSE) loss function that is typically used in operator learning. We demonstrate the benefits of using $\mathcal{L}_{\text{nn}}(\theta)$ instead of MSE in Section 3.3.

Multi-output Operators

Consider now the following multi-output operator

$$\mathcal{G}^\dagger : \mathcal{U} \rightarrow \prod_{s=1}^S \mathcal{V}^s \quad (3.16)$$

where S is the total number of (possibly related) output functions. In this case, our goal is to approximate $\mathcal{G}^\dagger$ given the training dataset $\mathcal{D}^S := \{s, \phi(u_i), \psi(v_i^s)\}_{i=1}^N$ where $\{u_i, v_i^s\}_{i=1}^N$ are elements from $\mathcal{U} \times \mathcal{V}^s$ with $s = \{1, \dots, S\}$. Analogously to Section 3.1.2, we cast the operator learning problem in Equation (3.16) as the following regression problem:

$$\tilde{\mathcal{G}}_p^\dagger : \mathbb{R}^p \times \mathcal{V}^* \rightarrow \mathbb{R}^S. \quad (3.17)$$

where now $\tilde{\mathcal{G}}_p^\dagger$ is a *multi-output* function and $\mathcal{V}^*$ denotes the dual of the product space $\prod_{s=1}^S \mathcal{V}^s$. To approximate $\tilde{\mathcal{G}}_p^\dagger$, we start by placing a multi-task or S –dimensional GP prior [134, 135] on it:

$$\tilde{\mathcal{G}}_p^\dagger \sim GP(m(\phi(u), y; \theta)(s), c([s, \phi(u), y], [s, \phi(u), y]'; \beta, \sigma^2)) \quad (3.18)$$

where $m(\phi(u), y; \theta)$ is the S –dimensional mean function of the GP.

The covariance matrix of the training data cannot be used in training unless $c([s, \phi(u), y], [s, \phi(u), y]'; \beta, \sigma^2)$ possesses certain features that eliminate the need to build and invert the $NSq \times NSq$ covariance matrix. Hence, we invoke the assumptions made on the data structure in Section 3.2.1 and also presume that all the outputs are observed at the same collocation points (CPs). Following these assumptions, we formulate the kernel in Equation (3.18) as:

$$c([s, \phi(u), y], [s, \phi(u), y]'; \beta, \sigma^2) = c_s(s, s'; \beta_s, \sigma_s^2) c_y(y, y'; \beta_y, \sigma_y^2) c_\phi(\phi(u), \phi(u'); \beta_\phi, \sigma_\phi^2), \quad (3.19)$$

where $c_y(y, y'; \beta_y, \sigma_y^2)$ and $c_\phi(\phi(u), \phi(u'); \beta_\phi, \sigma_\phi^2)$ are standard kernels such as the Gaussian or Matérn defined in Equation (3.7b), $\beta = [\beta_s, \beta_y, \beta_\phi]$, and $c_s(s, s'; \beta_s, \sigma_s^2)$ captures the average correlations among the responses. Once again, and without loss of generality, we assume $\sigma_y^2 = 1$ and hence only consider σ_ϕ^2 and σ_s^2 . The kernel in Equation (3.19) has a separable structure and is stationary since all the kernels on the right-hand side are stationary.

To further simplify the computations we set $c_s(s, s'; \beta_s)$ to a diagonal matrix, that is:

$$c_s(s, s') = \mathbb{1}\{s == s'\} \sigma_s^2 \quad (3.20)$$

where $\mathbb{1}\{\cdot\}$ returns 1/0 if the enclosed statement is true/false and σ_s^2 is the kernel variance for the s –th response. With this choice, the correlations between the different responses

are either neglected or captured by the mean function in zero-mean and NN-mean GPs, respectively.

To estimate θ , β , and σ^2 we use the training dataset $\mathcal{D}^S$ to maximize the likelihood function. We rearrange $\mathcal{D}^S$ as $\{(s, \phi(u_i), y_j), [\psi(v_i^s)]_j\}_{i,j,s=1}^{N,q,S}$ and denote its inputs and outputs via $\mathbf{X} = \{\phi(u_i), y_j\}_{i,j=1}^{N,q}$ and $\mathbf{v} = \{[\psi(v_i^s)]_j\}_{i,j,s=1}^{N,q,S}$, respectively, where we have used the assumptions that $\mathbf{X}$ do not depend on s . The MLE process for our multi-output regression task involves the following minimization problem:

$$\begin{aligned} [\hat{\theta}, \hat{\beta}, \hat{\sigma}^2] &= \underset{\theta, \beta, \sigma^2}{\operatorname{argmin}} \frac{1}{2} \log |\Sigma \otimes \mathbf{C}| + \frac{1}{2} (\mathbf{v} - \mathbf{m})^T (\Sigma \otimes \mathbf{C})^{-1} (\mathbf{v} - \mathbf{m}) \\ &= \underset{\theta, \beta, \sigma^2}{\operatorname{argmin}} \frac{1}{2} \log |\Sigma|^{Nq} + \frac{1}{2} \log |\mathbf{C}|^S + \frac{1}{2} (\mathbf{v} - \mathbf{m})^T (\Sigma^{-1} \otimes \mathbf{C}^{-1}) (\mathbf{v} - \mathbf{m}), \end{aligned} \quad (3.21)$$

where $\sigma^2 = [\sigma_\phi^2, \sigma_1^2, \dots, \sigma_S^2]$, $\Sigma = \operatorname{diag}\{\sigma_1^2, \dots, \sigma_S^2\}$, $\mathbf{C} = \mathbf{C}_\phi \otimes \mathbf{C}_y$ is the $Nq \times Nq$ symmetric positive-definite covariance matrix, and $\mathbf{m}$ is a column vector of length NqS that corresponds to the evaluations of the mean function for all the outputs, samples, and query points.

Similar to the previous section, we leverage the properties of the Kronecker product in Equation (3.12c) to simplify Equation (3.21) to:

$$\begin{aligned} [\hat{\theta}, \hat{\beta}, \hat{\sigma}^2] &= \underset{\theta, \beta, \sigma^2}{\operatorname{argmin}} \frac{1}{2} Nq \sum_{s=1}^S \log \sigma_s^2 + \frac{1}{2} \log (|\mathbf{C}_\phi|^{qS} |\mathbf{C}_y|^{NS}) \\ &\quad + \frac{1}{2} \sum_{s=1}^S \sigma_s^{-2} (\mathbf{v}^s - \mathbf{m}^s)^T \operatorname{vec}(\mathbf{C}_y^{-1} (\mathbf{V}^s - \mathbf{M}^s) \mathbf{C}_\phi^{-1}), \end{aligned} \quad (3.22)$$

where $\mathbf{V}^s$ and $\mathbf{M}^s$ are formed by reshaping $\mathbf{v}^s$ and $\mathbf{m}^s$ to $q \times N$ matrices, respectively, $\mathbf{v}^s = \{[\psi(v_i^s)]_j\}_{i,j}^{N,q}$, and $\mathbf{m}^s = m(s, \mathbf{X}; \theta)$ denotes the evaluations of the mean function for response s .

Parameter estimation now follows as in Section 3.2.1. Specifically, for a zero-mean GP,

Equation (3.22) simplifies to:

$$\begin{aligned}
\left[\widehat{\beta}, \widehat{\sigma}^2\right] &= \underset{\beta, \sigma^2}{\operatorname{argmin}} \mathcal{L}_{0v}(\beta, \sigma^2) \\
&= \underset{\beta, \sigma^2}{\operatorname{argmin}} \frac{1}{2} Nq \sum_{s=1}^S \log(\sigma_s^2) + \frac{1}{2} \log\left(|\mathbf{C}_\phi|^{qS} |\mathbf{C}_y|^{NS}\right) + \frac{1}{2} \sum_{s=1}^S \sigma_s^{-2} (\mathbf{v}^s)^T \operatorname{vec}(\mathbf{C}_y^{-1} \mathbf{V}^s \mathbf{C}_\phi^{-1}).
\end{aligned} \tag{3.23}$$

where we can fix β_ϕ similar to the scalar case in Section 3.2.1. For an NN-mean GP, we fix the kernel hyperparameters which makes the first two terms on the right-hand side of Equation (3.22) to become constants. Hence, we can estimate θ via:

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}_{\text{nnv}}(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{2} \sum_{s=1}^S (\mathbf{v}^s - \mathbf{m}^s)^T \operatorname{vec}(\mathbf{C}_y^{-1} (\mathbf{V}^s - \mathbf{M}^s) \mathbf{C}_\phi^{-1}). \tag{3.24}$$

Inference

Once the parameters of the mean and covariance functions are estimated, we obtain the predictive response for a test input function u^* at the query point y^* by computing the expected value of the posterior distribution of the GP conditioned on the training data. That is:

$$\begin{aligned}
\eta\left(\phi(u^*), y^*; \widehat{\theta}, \widehat{\beta}, \widehat{\sigma}_\phi^2\right) &:= \mathbb{E}[\widetilde{\mathcal{G}}_p^\dagger(\phi(u^*), \delta_{y^*}) | \mathcal{D}] \\
&= m(\phi(u^*), y^*; \widehat{\theta}) + c\left([\phi(u^*), y^*], \mathbf{X}; \widehat{\beta}, \widehat{\sigma}_\phi^2\right) \mathbf{C}^{-1}(\mathbf{v} - \mathbf{m}) \\
&= m(\phi(u^*), y^*; \widehat{\theta}) + \\
&\quad \left[c_\phi\left(\phi(u^*), \mathbf{U}; \widehat{\beta}_\phi, \widehat{\sigma}_\phi^2\right) \otimes c_y\left(y^*, \mathbf{Y}; \widehat{\beta}_y\right)\right] \operatorname{vec}(\mathbf{C}_y^{-1} (\mathbf{V} - \mathbf{M}) \mathbf{C}_\phi^{-1}) \\
&= m(\phi(u^*), y^*; \widehat{\theta}) + \\
&\quad \operatorname{vec}\left(c_y\left(y^*, \mathbf{Y}; \widehat{\beta}_y\right) \mathbf{C}_y^{-1} (\mathbf{V} - \mathbf{M}) \mathbf{C}_\phi^{-1} c_\phi\left(\mathbf{U}, \phi(u^*); \widehat{\beta}_\phi, \widehat{\sigma}_\phi^2\right)\right),
\end{aligned} \tag{3.25}$$

where we have leveraged the Kronecker product properties in Equation (3.12c) to simplify the matrix multiplications. Equation (3.25) accommodates batch computations for accelerated prediction of the output field at the m points $\mathbf{Y}^* = \{y_1^*, \dots, y_m^*\}$:

$$\begin{aligned} \eta\left(\phi(u^*), \mathbf{Y}^*; \hat{\theta}, \hat{\beta}, \hat{\sigma}_\phi^2\right) &= m(\phi(u^*), \mathbf{Y}^*; \hat{\theta}) + \\ &\quad \text{vec}\left(c_y\left(\mathbf{Y}^*, \mathbf{Y}; \hat{\beta}_y\right) \mathbf{C}_y^{-1} (\mathbf{V} - \mathbf{M}) \mathbf{C}_\phi^{-1} c_\phi\left(\mathbf{U}, \phi(u^*); \hat{\beta}_\phi, \hat{\sigma}_\phi^2\right)\right) \end{aligned} \quad (3.26)$$

where $\mathbf{U} = \{\phi(u_1), \dots, \phi(u_N)\}$ and $\mathbf{Y} = \{y_1, \dots, y_q\}$ as defined previously. In the case of multi-output operators, our predictor for output s is:

$$\begin{aligned} \eta\left(s, \phi(u^*), \mathbf{Y}^*; \hat{\theta}, \hat{\beta}, \hat{\sigma}_\phi^2\right) &= m(\phi(u^*), \mathbf{Y}^*; \hat{\theta})(s) \\ &\quad + \text{vec}\left(c_y(\mathbf{Y}^*, \mathbf{Y}; \hat{\beta}_y) \mathbf{C}_y^{-1} (\mathbf{V}^s - \mathbf{M}^s) \mathbf{C}_\phi^{-1} c_\phi(\mathbf{U}, \phi(u^*); \hat{\beta}_\phi, \hat{\sigma}_\phi^2)\right). \end{aligned} \quad (3.27)$$

For a zero-mean GP, Equations (3.25) to (3.27) can be simplified by removing the mean function and the associated matrices $\mathbf{M}$ and $\mathbf{M}^s$.

The predictor in Equation (3.25) is a differentiable function with respect to y^* . We use this property in Section 3.2.2 for physics-informed operator learning where the training process leverages the known differential operators that govern the problem. This predictor also interpolates the training data which is a feature that most neural operators do not provide. To see this, we use Equation (3.26) to predict the output function at $\mathbf{Y}$ for the N input functions $\mathbf{U}$ that are observed during training:

$$\begin{aligned} \eta\left(\mathbf{U}, \mathbf{Y}; \hat{\theta}, \hat{\beta}, \hat{\sigma}_\phi^2\right) &= m(\mathbf{U}, \mathbf{Y}; \hat{\theta}) + \text{vec}\left(c_y(\mathbf{Y}, \mathbf{Y}; \hat{\beta}_y) \mathbf{C}_y^{-1} (\mathbf{V} - \mathbf{M}) \mathbf{C}_\phi^{-1} c_\phi(\mathbf{U}, \mathbf{U}; \hat{\beta}_\phi, \hat{\sigma}_\phi^2)\right) \\ &= \mathbf{m} + \text{vec}(I_y(\mathbf{V} - \mathbf{M}) I_\phi) = \mathbf{m} + \mathbf{v} - \mathbf{m} = \mathbf{v}. \end{aligned} \quad (3.28)$$

Lastly, we highlight that both terms on the right-hand side of Equation (3.25) capture the correlations not only in $\mathcal{U}$, but also in the support of $\mathcal{V}$, i.e., Ω_v . That is, our predictor leverages the strengths of both neural operators and kernel methods for operator learning.

3.2.2 Physics-Informed Operator Learning

Consider the general differential operator $\mathcal{T}$ acting upon $u \in \mathcal{U}$ and $v \in \mathcal{V}$ such that $\mathcal{T} : \mathcal{U} \times \mathcal{V} \rightarrow \mathcal{O}$, where $\mathcal{O}$ is the null space, u is the input function, and v denotes the unknown solution. Then, a general family of (possibly nonlinear) time-dependent PDEs can be expressed as:

$$\mathcal{T}(u, v) = 0 \quad \text{in } \Omega_v \times (0, \infty), \quad (3.29a)$$

$$v = f \quad \text{on } \partial\Omega_v \times (0, \infty), \quad (3.29b)$$

$$v = g \quad \text{in } \bar{\Omega}_v \times \{0\}. \quad (3.29c)$$

Given Equation (3.29) and N pairs of $\{\phi(u_i), \psi(v_i)\}$ that satisfy it, our goal is to approximate the operator $\mathcal{G}^\dagger : \mathcal{U} \rightarrow \mathcal{V}$. In this context, $\mathcal{U}$ may represent a family of PDE coefficients $a(y, t) : \Omega_v \times (0, \infty) \rightarrow \mathbb{R}$ where t denotes time, boundary conditions $v(y, t) : \partial\Omega_v \times (0, \infty) \rightarrow \mathbb{R}$, or initial conditions $v(y, t) : \bar{\Omega}_v \times \{0\} \rightarrow \mathbb{R}$.

To improve the approximation accuracy of the predictor in Equation (3.25), we regularize the parameter estimation process via Equation (3.29). To this end, we use a batch of N_{pi} samples with inputs $\{\phi(u_i)\}_{i=1}^{N_{pi}}$ which may or may not be unique from the N training samples used throughout Section 3.2.1. For each of the N_{pi} samples, we consider n_{PDE} , n_{BC} , and n_{IC} collocation points in the domain, on the boundaries, and at $t = 0$, respectively, to obtain the following losses:

$$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_{pi}n_{\text{PDE}}} \sum_{j=1}^{N_{pi}} \sum_{i=1}^{n_{\text{PDE}}} (\mathcal{T}(u_j, \eta(\phi(u_j), \delta_{y_i})))^2, \quad (3.30a)$$

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N_{pi}n_{\text{BC}}} \sum_{j=1}^{N_{pi}} \sum_{i=1}^{n_{\text{BC}}} (\eta(\phi(u_j), \delta_{y_i}) - f(y_i))^2, \quad (3.30b)$$

$$\mathcal{L}_{\text{IC}}(\theta) = \frac{1}{N_{pi}n_{\text{IC}}} \sum_{j=1}^{N_{pi}} \sum_{i=1}^{n_{\text{IC}}} (\eta(\phi(u_j), \delta_{y_i}) - g(y_i))^2, \quad (3.30c)$$

which quantify the errors of Equation (3.25) in satisfying Equation (3.29). Note that we calculate $\mathcal{L}_{\text{BC}}(\theta)$ and $\mathcal{L}_{\text{IC}}(\theta)$ since the N_{pi} samples may be different from the N labeled training samples. But if these two sets of samples have identical ICs/BCs, $\mathcal{L}_{\text{BC}}(\theta)$ and $\mathcal{L}_{\text{IC}}(\theta)$ in Equation (3.30c) will be zero due to the reproducing property of the predictor, see Equation (3.28).

To estimate θ , we combine the loss terms in Equation (3.30c) with Equation (3.15):

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PDE}}(\theta) + \alpha_{\text{BC}}\mathcal{L}_{\text{BC}}(\theta) + \alpha_{\text{IC}}\mathcal{L}_{\text{IC}}(\theta) + \alpha_{\text{MLE}}\mathcal{L}_{\text{nn}}(\theta), \quad (3.31)$$

where α_{BC} , α_{IC} , and α_{MLE} are weights that ensure none of the loss terms in Equation (3.31) dominates the optimization process. Specifically, we choose $\mathcal{L}_{\text{PDE}}$ as a reference and assign the unit weight to it. Then, we determine the weights of the other loss terms via an Adam-inspired approach [136, 39] such that their gradients with respect to θ have comparable magnitudes at all optimization iterations. For example, we obtain α_{BC} at epoch $k + 1$ via:

$$\alpha_{\text{BC}}^{k+1} = (1 - \lambda)\alpha_{\text{BC}}^k + \lambda \frac{\max |\nabla_{\theta} \mathcal{L}_{\text{PDE}}(\theta)|}{\text{mean} |\nabla_{\theta} \mathcal{L}_{\text{BC}}(\theta)|}, \quad (3.32)$$

where $\mathcal{L}_{\text{PDE}}(\theta)$ is chosen as the reference term, $\nabla_{\theta} \mathcal{L}_{\text{BC}}(\theta)$ denotes the gradient of $\mathcal{L}_{\text{BC}}(\theta)$ with respect to θ , and λ is typically set to 0.9.

We conclude this section by highlighting three points. First, if the regularized loss function in Equation (3.31) involves nonlinear functions of the predictor (i.e., if the PDE system in Equation (3.29) is nonlinear), then the posterior in Equation (3.25) is not a Gaussian and the predictor is merely a maximum a posteriori (MAP) estimator. Second, for physics-informed operator learning we only use NN-mean GPs with fixed kernel parameters because (1) optimizing the kernel parameters requires the repeated inversion of the covariance matrices which is computationally expensive and prone to ill-conditioning, and (2) our kernels are quite simple and have insufficient parameters to accommodate residual minimization.

Third, for multi-output operators we replace $\mathcal{L}_{\text{nn}}(\theta)$ by $\mathcal{L}_{\text{nnv}}(\theta)$ in Equation (3.24).

3.2.3 Parameter Initialization and Stability

For a zero-mean GP we fix β_y to some *large* values and only optimize β_ϕ and σ_ϕ^2 in Equation (3.14) or Equation (3.23). We rationalize this decision in Section 3.2.3 and then in Section 3.2.3 we elaborate on our logic for initializing β_ϕ to some *small* values before they are optimized. For an NN-mean GP, we optimize θ while $\beta = [\beta_y, \beta_\phi]$ and σ_ϕ^2 are all fixed to their initial values which are the same as those for a zero-mean GP. As detailed in Section 3.2.3 this approach dramatically accelerates the computations while reducing the numerical instabilities.

Effect of Observation Operator

In many operator learning problems $\psi(v)$ provides dense observations, i.e., q in Equation (3.2) is a relatively large number. To demonstrate the effect of dense observations on the optimum kernel parameters of a zero-mean GP, we consider the simple task of interpolating $u(x) = \sin(2\pi x)$, $x \in [0, 1]$ using a zero-mean GP with the Gaussian kernel $c(x, x'; \beta, \sigma^2)$. We set $\sigma^2 = 1$ without loss of generality and sample $N \gg 1$ points from $u(x)$. Instead of optimizing β via MLE, we vary it in the $[0, 10^6]$ range and for each value we calculate the error of the corresponding conditional predictor in Equation (3.28) on a large set of test samples.

The results of this study are summarized in Figure 3.3 which clearly shows two pathological regimes for $\beta \rightarrow 0$ and $\beta \rightarrow \infty$ (i.e., not all values of β result in accurate interpolation of the test samples, even when abundant data is available). This trend is expected since $1/\beta$ is proportional to the correlation length. For this particular example, $\beta < 1$ renders the covariance matrix ill-conditioned while $\beta > 10^4$ prevents the GP from modeling the spatial

correlations. For instance, consider points A and B on the curve which mark $\beta = 10^{-1}$ and $\beta = 10^3$, respectively, and the corresponding insets illustrate the predictions of the resulting GPs. We observe in inset A that the prediction curve with $\beta = 10^{-1}$ is quite rough which is due to the ill-conditioning of the GP’s covariance matrix. In contrast, for $\beta \in [10^1, 3.50 \times 10^3]$ the resulting GP interpolates unseen data very accurately and provides a smooth prediction curve. Further increasing β reduces the prediction accuracy as the posterior mean cannot leverage the spatial correlations at all.

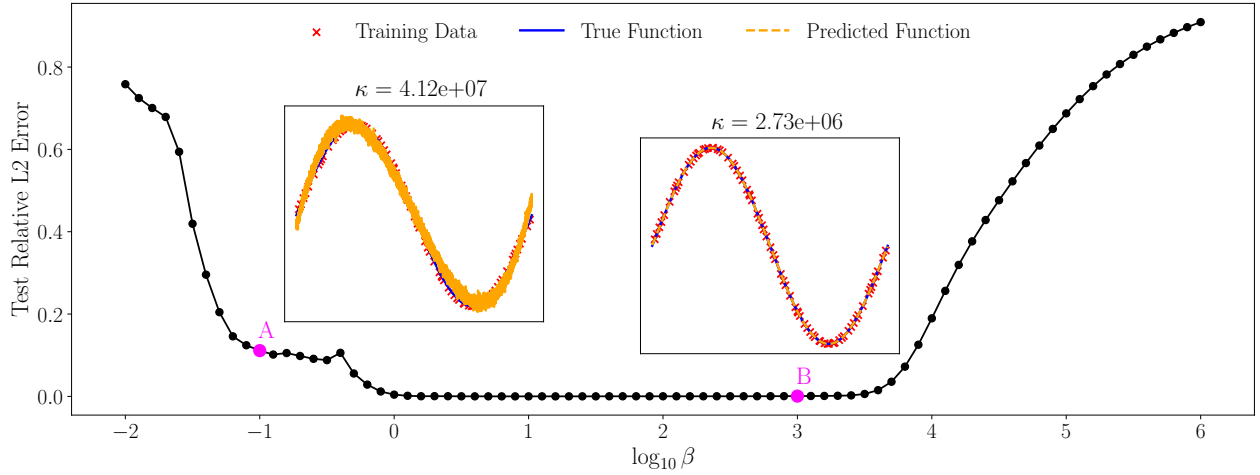


Figure 3.3 Effect of β_y on test accuracy with dense observations: Very small and large β values provide poor performance as they cause numerical issues and cannot capture spatial correlations, respectively. However, a relatively large range of values such as 10^3 provide high accuracy and numerical stability.

Our studies indicate that the trends in Figure 3.3 extend to not only higher dimensions, but also the profile of the likelihood function (we further elaborate on the latter point in Section 3.2.3). Following these observations, in all of our studies we set β_y to large values, e.g., 3, regardless of d (i.e., the dimensionality of the output space Ω_v) and the choice of the GP’s mean function (NN or zero). This decision ensures that C_y is numerically stable and the GP can faithfully interpolate the dense observations in Ω_v .

High-Dimensional Features in GPs

The performance of GPs deteriorates in very high feature spaces [137] due to the so-called curse of dimensionality: kernels capture the correlations between the samples based on their distance in the feature space. Most kernels such as those in Equation (3.7b) poorly characterize these correlations in high-dimensional feature spaces as the quantified distances quickly increase and, in turn, render the correlations among samples to become zero. To illustrate this, consider the kernel in Equation (3.7a) with $\sigma^2 = 1$. Assuming the same length-scale parameter β is used across all d_x dimensions, we can express $c(x, x'; \beta)$ as a function of the distance between two feature vectors x and x' :

$$c(x, x'; \beta) = \exp\left\{-\beta \|x - x'\|^2\right\}, \quad (3.33)$$

which can be solved for β :

$$\beta = -\frac{\log c(x, x')}{\|x - x'\|^2}. \quad (3.34)$$

To explore the interaction between d_x and β , we perform the following experiment. Given fixed covariance values $c(x, x') = 0.2$ and $c(x, x') = 0.8$, we generate 10,000 random samples in the d_x -dimensional unit hypercube, where the squared Euclidean distance between two samples is used as the denominator of Equation (3.34). Then, we calculate β corresponding to each pair of samples and repeat this process for various values of d_x . The corresponding histograms are provided in Figure 3.4 and demonstrate that as d_x increases, the range of β values that achieve the specified covariances not only shrinks (compare the width of the histograms for $d_x = 2$ and $d_x = 2000$), but also includes smaller values.

Following the above experiment, we initialize β_ϕ at 10^{-2} in all of our studies in this chapter since the input function is typically discretized at a large number of points. This value is obviously not an optimal choice so in the case of zero-mean GPs we associate each input

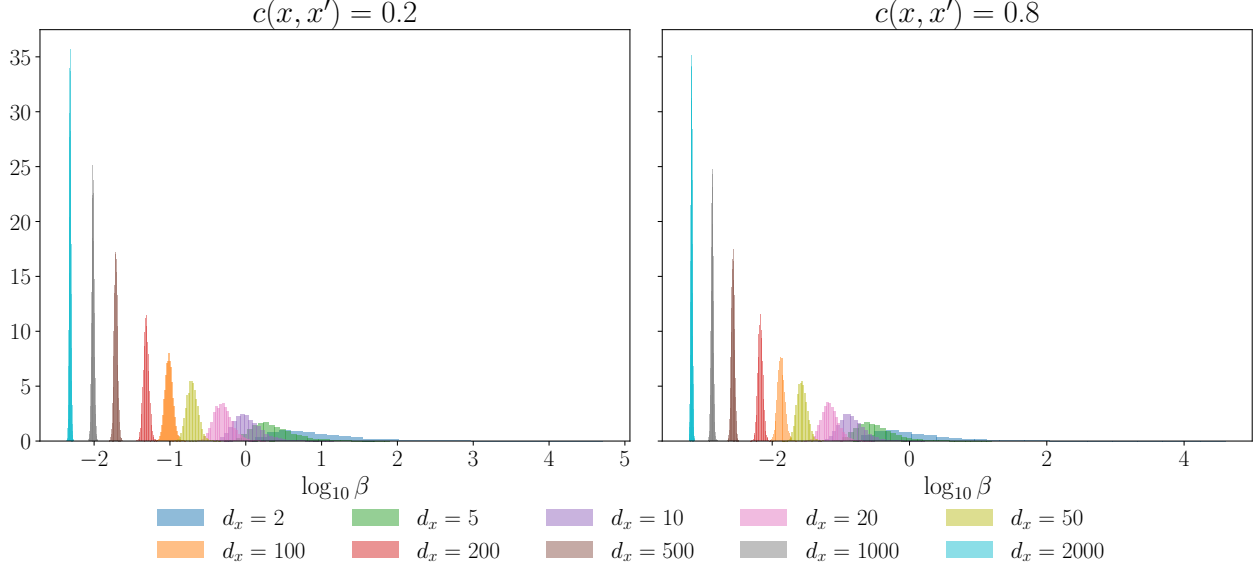


Figure 3.4 Interaction between β and feature space dimensionality: Smaller β values are needed to achieve a desired correlation value in high-dimensional feature spaces.

feature with its own β and then fine tune them via MLE. In the case of NN-mean GPs, we do not optimize these initial values (due to computational costs and numerical instabilities) and rely on the neural operator part of the GP to correct (if needed) how the correlations between $\phi(u)$ and $\phi(u')$ are quantified.

Our rationale for initializing β_ϕ at a small value, e.g., 10^{-2} , can also be explained as follows. Consider the Gaussian kernel $c(u, u'; \beta, \sigma^2 = 1)$:

$$c(u, u'; \beta, \sigma^2 = 1) = \exp\left\{-\beta \|u - u'\|^2\right\}, \quad (3.35)$$

where $u(x)$ and $u'(x)$ are one-dimensional functions and $\|\cdot\|$ is the $L2$ norm that can be approximated as:

$$\|u - u'\|^2 = \int_{\Omega} (u(x) - u'(x))^2 dx \approx \sum_{i=1}^p (u(x_i) - u'(x_i))^2 \Delta x_i, \quad (3.36)$$

where Δx_i is the discretization step. Assuming the points are equally spaced, Equation (3.35)

simplifies to:

$$c(u, u'; \beta, \sigma^2 = 1) = \exp \left\{ - \underbrace{\beta \Delta x}_{\beta_u} \sum_{i=1}^p (u(x_i) - u'(x_i))^2 \right\}. \quad (3.37)$$

We observe that for function inputs the effective length-scale parameter β_u corresponds to the product of the length-scale β and the discretization step Δx , which is generally small. So, compared to the case where the features are scalars by nature, smaller values should be chosen for the effective length-scale parameter if the features are obtained by finely discretizing a function.

Once again, these findings are also validated in Figure 3.5(a.2) and Figure 3.5(b.2), where we plot the test relative $L2$ errors as functions of β_y and β_ϕ for the Burgers' and Darcy benchmarks. Our experiments suggest that the range of values for β_ϕ (corresponding to the functional input) that minimize the test error is comparably smaller to β_y (corresponding to scalar inputs). An interesting observation from Figure 3.5(a.2) and Figure 3.5(b.2) is that the test error is minimized across a broad range of β_y and β_ϕ values. This suggests that a zero-mean GP, when properly initialized, can achieve high accuracy without requiring any training.

To examine the interaction between the initialization of β_y and β_ϕ , we consider the Burgers' and Darcy problems studied in Section 3.3. Applying zero-mean GPs to these two problems, we visualize the training loss and test error maps as a function of the kernel parameters in Figure 3.5. We observe that in all cases our suggested initial values provide minimal values for these metrics in both problems. Interestingly, we notice that many parameter combinations effectively minimize the loss and error maps in both problems, indicating that a high accuracy is obtained by a *zero-shot* zero-mean GP whose kernel parameters are properly initialized but never optimized. We examine the performance of zero-shot and one-shot zero-mean GPs in Section 3.3 which benefit from zero and one optimization iterations, respectively.

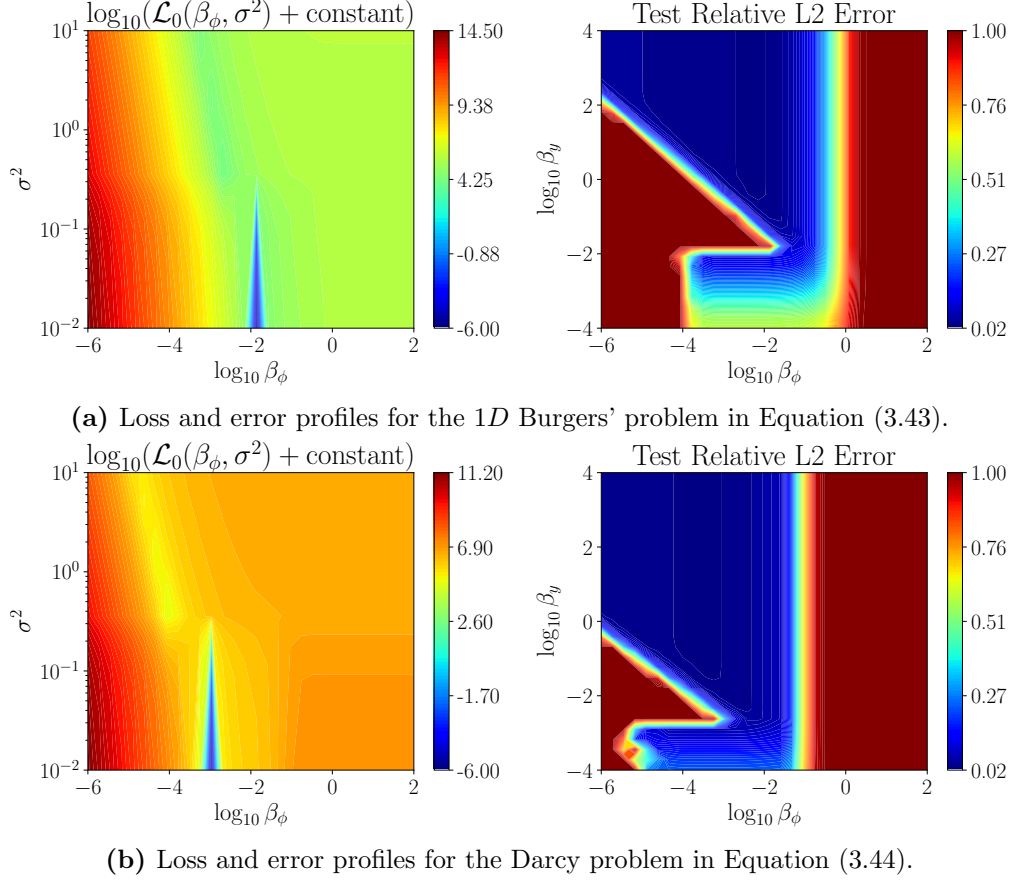


Figure 3.5 Loss and error profiles as functions of kernel parameters in zero-mean GPs: A wide range of parameter combinations provide optimal loss and error values, indicating that proper initialization is relatively easy and results in good performance even without training. $\beta_y = 10^3$ is used for the two loss profiles on the left-hand side of (a) and (b). Note that we do not show the dependency of the test relative L_2 error with respect to σ_ϕ^2 since this does not affect Equation (3.25). For Darcy, β_y has two components and so for plotting the error map we presume that these components are equal (a similar assumption is made for β_ϕ).

Parameter Optimization in NN-mean GPs

Non-zero mean functions have the potential to increase the extrapolation capabilities of GPs [138] and they have been effectively employed for materials modeling [139] or to solve forward and inverse problems in PDEs [131]. Specifically, parameterizing the mean function of a GP by a deep NN provides the means to leverage the strengths of both kernels and NNs as the former effectively learns local trends by interpolating neighboring points in regions with abundant data, while the latter improves generalization by learning complex distributed

representations for unseen inputs. However, this added model complexity increases the risk of overfitting and can lead to numerical instability during optimization [138]. We demonstrate these issues with a simple regression example which motivates our decision for fixing the kernel parameters while optimizing θ .

We again consider the function $u(x) = \sin(2\pi x)$, $x \in [0, 1]$ and sample $N = 20$ points from it such that the samples are concentrated close to the lower- and upper-bounds of x , see the red crosses in Figure 3.6. We fit three GPs to this dataset:

- Zero-mean GP: We set the kernel to $c(x, x'; \beta, \sigma^2 = 1)$ and use MLE to optimize β .
- NN-mean GP: We set the kernel and the mean function to, respectively, $c(x, x'; \beta, \sigma^2 = 1)$ and a fully-connected feed-forward NN with 4 layers of 20 neurons. We use MLE to jointly optimize the parameters of the kernel and the NN.
- Physics-informed NN-mean GP: We set the kernel to $c(x, x'; \beta = 10^3, \sigma^2 = 1)$ and optimize the NN parameters to minimize a weighted combination of MLE and MSE where the latter encourages the GP-based predictor to satisfy the ordinary differential equation $\frac{du}{dx} = 2\pi \cos(2\pi x)$.

In all cases the Adam optimizer with a learning rate scheduler is used where the initial learning rate and maximum number of epochs are set to 10^{-3} and 20,000, respectively. The results of our studies are summarized in Figure 3.6 and demonstrate that the zero-mean GP (left plot) expectedly fails to generalize to regions where there are no training samples. This issue is slightly alleviated in the case of the NN-mean GP (middle plot) at the expense of worsening the condition number, κ , of the covariance matrix. The increase in κ is due to the fact that the NN mean has sufficient parameters to interpolate the data which causes the kernel to model very small residuals and, in turn, be numerically ill-conditioned. The magnitude of κ is directly related to the number of parameters of the mean function which is typically a very large number in neural operators.

The results of the third scenario are provided in Figure 3.6(c) where we observe that the physics of the problem is effectively used to learn $u(x)$ with high accuracy without increasing the condition number of the covariance matrix. The training mechanism corresponding to this scenario is similar to the one explained in Section 3.2.2 for operator learning.

3.2.4 Comparison to Optimal Recovery-Based Operator Learning

The optimal recovery method of [4] approximates the operator $\mathcal{G}^\dagger$ via the following explicit representer formula:

$$\bar{\mathcal{G}}(u^*)(y) = c_\nu(y, \mathbf{Y}) c_\nu(\mathbf{Y}, \mathbf{Y})^{-1} \Lambda(\phi(u^*), \mathbf{U}) \Lambda(\mathbf{U}, \mathbf{U})^{-1} \mathbf{V} \quad (3.38)$$

where $\Lambda(\mathbf{U}, \mathbf{U}')$ is a matrix-valued kernel in the space of $m \times m$ matrices, $c_\nu(\cdot, \cdot)$ is the kernel associated with the RKHS of functions in $\mathcal{V}$, and y denotes a location in the output space where observations are made. [4] assumes a diagonal kernel of the form $\Lambda(\mathbf{U}, \mathbf{U}') = c_\lambda(\mathbf{U}, \mathbf{U}') I$ where I is the $m \times m$ identity matrix and $c_\lambda : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ is a real-valued kernel. Following these assumptions and using the Kronecker product, we can rewrite Equa-

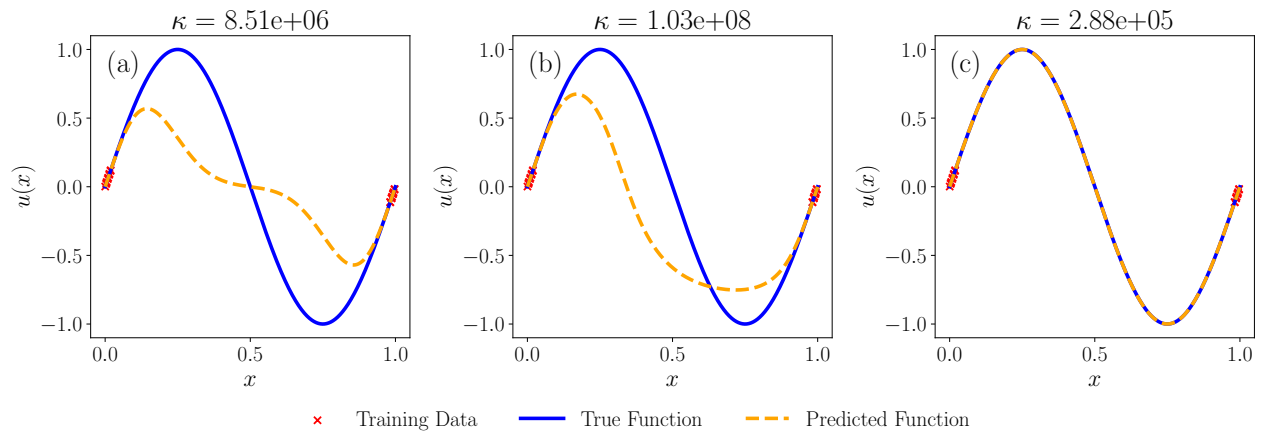


Figure 3.6 Parameter optimization in GPs: We consider three scenarios with a zero-mean GP in (a) and NN-mean GPs in (b) and (c). The differences in training settings across these three cases are detailed in the text. The relative L_2 errors for cases (a), (b) and (c) are 64.77%, 60.38%, and 0.09%, respectively.

tion (3.38) as:

$$\bar{\mathcal{G}}(u^*)(y) = c_\nu(y, \mathbf{Y})c_\nu(\mathbf{Y}, \mathbf{Y})^{-1}[c_\lambda(\phi(u^*), \mathbf{U}) \otimes I][c_\lambda(\mathbf{U}, \mathbf{U}) \otimes I]^{-1}\mathbf{V}. \quad (3.39)$$

To directly compare Equation (3.39) with our approach, we drop the mean function in our GP predictor to obtain:

$$\mathbb{E}[\tilde{\mathcal{G}}_p^\dagger(\phi(u^*), \delta_{y^*})|\mathcal{D}] = [c_\phi(\phi(u^*), \mathbf{U}) \otimes c_y(\mathbf{Y}^*, \mathbf{Y})][c_\phi(\mathbf{U}, \mathbf{U}) \otimes c_y(\mathbf{Y}, \mathbf{Y})]^{-1}\mathbf{V}. \quad (3.40)$$

Comparing Equations (3.39) and (3.40) highlights the main differences between the two approaches:

- Unlike the optimal recovery method of [4], we account for the correlations in the output function space through $c_y(\cdot, \cdot)$. Hence, we solve a single regression problem as opposed to solving m independent optimal recovery ones at the expense of (1) using all the data at once, and (2) having far fewer hyperparameters (and hence flexibility).
- Equation (3.39) directly approximates $\mathcal{G}^\dagger$ which explains the presence of the operator-valued kernels. Instead, our method approximates $\mathcal{G}^\dagger(u^*)(y^*)$, i.e., the *evaluation* of the output function at the *arbitrary* query point y^* for the given input function u^* , see Figure 3.1. This feature enables us to incorporate additional physical constraints into the approximation process as detailed in Section 3.2.2.

We also note that our approach can also use deep NNs as the mean function which has the advantage of reducing the sole reliance of the predictor on the kernel whose misspecification can substantially reduce the performance of zero-mean GPs.

3.2.5 Inference Complexity

We quantify the evaluation costs of our predictor by computing the floating point operators (FLOPs) per a test sample [2]. We assume that the product between two matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{m \times q}$ has a cost of $nq(2m-1)$ flops, vectorization has no costs, and that a nonlinear function acting upon an input vector of size n (e.g., activation function in NNs or kernel evaluation in GPs) costs n flops. Additionally, we assume the cost of constructing the $m \times n$ covariance matrix between two matrices $\mathbf{A} \in \mathbb{R}^{m \times q}$ and $\mathbf{B} \in \mathbb{R}^{n \times q}$ using the Gaussian kernel in Equation (3.7a) is $mn(4q+2)$.

We observe in Equation (3.25) that the $m \times N$ matrix $\mathbf{C}_y^{-1}(\mathbf{V} - \mathbf{M})\mathbf{C}_\phi^{-1}$ can be stored from training and therefore we do not attribute any associated cost to that operation at the inference stage. Hence, the cost of evaluating the learned operator in our framework consists of (1) evaluation cost of the mean function which is either zero for a zero-mean GP, or $\mathcal{C}_m$ for a non-zero mean GP, (2) building $c_y(\mathbf{Y}^*, \mathbf{Y}; \hat{\beta}_y) \in \mathbb{R}^{m \times m}$ and $c_\phi(\phi(u^*), \phi(\mathbf{U}); \hat{\beta}_\phi, \hat{\sigma}_\phi^2) \in \mathbb{R}^{N \times 1}$, which have costs of $m^2(4d+2)$ and $N(4n+2)$, respectively, and (3) matrix multiplication costs in $\text{vec}\left(c_y(\mathbf{Y}^*, \mathbf{Y}; \hat{\beta}_y)\mathbf{C}_y^{-1}(\mathbf{V}^s - \mathbf{M}^s)\mathbf{C}_\phi^{-1}c_\phi(\phi(u^*), \mathbf{U}; \hat{\beta}_\phi, \hat{\sigma}_\phi^2)\right)$ which sum up to $m(2N-1) + m(2m-1)$. Given these cost components, the total inference cost of predicting the output function at m points for a given test input function discretized at n points is:

$$\begin{aligned} \mathcal{C} &= \mathcal{C}_m + m^2(4d+2) + N(4n+2) + m(2N-1) + m(2m-1) \\ &= \mathcal{C}_m + 4m^2(d+1) - 2m + 2N(2n+2m+1), \end{aligned} \tag{3.41}$$

where N is the number of training samples and $\mathcal{C}_m$ can be calculated as detailed in [2] if neural operators such as DeepONet or FNO are used as the mean function in our approach.

Equation (3.41) indicates the price that one has to pay for embedding a neural operator in our framework. This additional cost scales linearly in N but incurring it, as argued in Section 3.3 based on our comparative studies, is justified since GPs with neural operators

as their mean function provide higher accuracy than the base neural operators especially in extrapolation.

3.3 Results and Discussion

We evaluate the performance of our approach in data-driven and physics-informed operator learning problems in Sections 3.3.1 to 3.3.3 where we consider both single- and multi-output operators. While we can leverage any neural operator as the mean function in our approach, herein we only consider FNO [53] and DeepONet [52] which are two of the most popular neural operators. Throughout, we use the relative $L2$ metric defined below to quantify the accuracy on test data:

$$L2 = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{\left\| \mathcal{G}^\dagger(u^i) - \tilde{\mathcal{G}}_p^\dagger(u^i) \right\|_{\mathcal{V}}}{\left\| \mathcal{G}^\dagger(u^i) \right\|_{\mathcal{V}}} \quad (3.42)$$

where N_{test} is the number of test samples, $\mathcal{G}^\dagger$ is the target operator and $\tilde{\mathcal{G}}_p^\dagger$ is the predicted operator. We conclude our studies in Section 3.3.4 by commenting on the training cost of different approaches.

3.3.1 Data-driven Operator Learning: Single-Output Operators

We consider four canonical examples that are routinely studied in the relevant literature [4, 2, 3, 53]. These examples are defined below and used to compare our approach to DeepONet [52], FNO [53], and GP-OR which is the GP-based optimal recovery approach of [4]. We take the training data from [3] and use the reported accuracy metrics in [4] for these three baselines. In our approach, we consider three cases: a zero-mean GP, DeepONet-mean GP, and FNO-mean GP. In the latter two cases, we set up the architecture of the neural

operators to match those of the baselines². We also train our models with the same data used for training the baselines. We train our GP-based models via Adam for 3000 training epochs and set the learning rate to 10^{-3} and 10^{-2} for NN-mean and zero-mean cases, respectively.

Burgers' Equation: We consider the 1D Burgers' equation with periodic BCs:

$$\begin{aligned} u_t + uu_x &= \zeta u_{xx}, \quad \forall (x, t) \in (0, 1) \times (0, 1], \\ u(x, 0) &= u_0(x), \quad \forall x \in (0, 1) \end{aligned} \tag{3.43}$$

with $\zeta = 0.1$. The operator that we aim to learn is $\mathcal{G}^\dagger : u(x, 0) \rightarrow u(x, 1)$, i.e., the mapping from the IC to the solution at time $t = 1$. The dataset is originally generated in [53] where the IC is sampled from a Gaussian random field with a Riesz kernel. As in [3], we use 1000 samples for training and 200 for testing with a spatial resolution of 128 points, i.e., $p = q = 128$.

Darcy Flow: The 2D Darcy flow problem with zero BCs is formulated as:

$$\begin{aligned} -\nabla \cdot (a(x, y) \nabla u(x, y)) &= f(x, y), \quad \forall (x, y) \in (0, 1)^2 \\ u(x, y) &= 0, \quad \forall (x, y) \in \partial(0, 1)^2 \end{aligned} \tag{3.44}$$

where $a(x, y)$ is the permeability field and $f(x, y)$ is a source term. For a fixed $f(x, y)$, we aim to learn $\mathcal{G}^\dagger : a(x, y) \rightarrow u(x, y)$, i.e., the mapping from the permeability field to the solution in the entire domain. The dataset is extracted from [3] and we use 1000 samples for training and 200 for testing where the input and output functions are available on a 29×29 grid. The coefficient field $a(x, y)$ is obtained via $a = \xi(\mu)$ where $\mu \sim \mathcal{GP}(0, (-\Delta + 9I)^{-2})$ is a Gaussian random field with zero Neumann BCs on the Laplacian and ξ is a function that maps positive values to 12 and negative values to 3 [3].

²In some cases, insufficient information about the architecture is available so we highlight the potential differences when applicable.

Advection Equation: We consider the 1D advection equation with periodic BCs:

$$\begin{aligned} u_t + u_x &= 0, & \forall (x, t) \in (0, 1) \times (0, 1], \\ u(x, 0) &= u_0(x), & \forall x \in (0, 1) \end{aligned} \tag{3.45}$$

where $\zeta = 0.1$ and our goal is to approximate $\mathcal{G}^\dagger : u(x, 0) \rightarrow u(x, 0.5)$, i.e., the mapping from the IC to the solution at time $t = 0.5$. The dataset is originally generated by [3] where the IC is designed as a square wave centered at $x = c$ with width w and height h , i.e., $u_0(x) = h\mathbf{1}_{\{c-\frac{w}{2}\}, \{c+\frac{w}{2}\}}$. The parameters c, w and h are randomly drawn from the uniform distribution $U([0.3, 0.7] \times [0.3, 0.6] \times [1, 2])$. We use 1000 samples for training and 200 for testing whose input and output functions are discretized via 40 equally-spaced grid points.

Structural Mechanics: We consider the equation governing the displacement field u of an elastic solid under small deformations:

$$\begin{aligned} \nabla \cdot \sigma &= 0, & \text{in } D, \\ u &= \bar{u}, & \text{on } \Gamma_u, \\ \nabla \cdot n &= \bar{t}, & \text{on } \Gamma_t, \end{aligned} \tag{3.46}$$

where $D = (0, 1)^2$ and the boundary ∂D is divided into $\Gamma_t = [0, 1] \times 1$ and $\Gamma_u = \partial D \setminus \Gamma_t$, i.e., Γ_u is the compliment of Γ_t . We aim to learn the operator $\mathcal{G}^\dagger : \bar{t} \rightarrow \sigma$, i.e., the mapping between the 1D load $\bar{t}$ applied on Γ_t to the von Misses stress field σ . The dataset is extracted from [2] where $\bar{t}$ is drawn from a Gaussian random field $\mathcal{GP}(100, 400^2(-\nabla + 3^2I)^{-1})$ where ∇ is the Laplacian with Neumann BCs applied to the space of functions with a spatial mean of zero. The output function σ is obtained via the FEM. We use 1250 samples for training and 20,000 for testing. The input function is sampled on 41 discretized points and the output function is interpolated on a regular 41×41 grid.

The results of our comparative studies are summarized in Table 3.1 and indicate that our

		Burgers' $N = 1000, p = 128, q = 128$		Darcy $N = 1000, p = 29^2, q = 29^2$		Advection $N = 1000, p = 40, q = 40$		Structural Mechanics $N = 1250, p = 41, q = 41^2$	
		Error	# of Parameters	Error	# of Parameters	Error	# of Parameters	Error	# of Parameters
DeepONet		2.15%	148,864	2.91%	715,776	0.22%	471,552	8.70%	137,856
FNO		1.93%	320,705	2.41%	6,304,257	0.66%	320,705	6.62%	888,001
GP-OR		2.15%	1280	2.75%	169,882	2.75e-3%	1,600	6.95%	68,921
Our GP	0-shot Zero-mean	2.90%	129	5.14%	201	0.17%	41	7.13%	42
	1-shot Zero-mean	2.90%	129	5.01%	201	0.17%	41	7.13%	42
	Zero-mean	2.99%	129	2.89%	201	4.11e-3%	41	6.74%	42
	DeepONet-mean	1.84%	148,864	3.44%	715,776	0.18%	471,552	7.12%	137,856
	FNO-mean	0.08%	320,705	2.19%	6,304,257	0.23%	320,705	6.49%	888,001

Table 3.1 Summary results on four benchmark problems: We report the $L2$ relative test error for four different operator learning methods. The errors in the first three rows are directly taken from the literature [2, 3, 4]. In our approach, we set the mean function to either (1) zero in which case we also provide the error metrics without training and after one optimization iteration, or (2) DeepONet and FNO whose architectures are exactly the same as those in rows 1 and 2. In the case of Darcy, we follow the references and apply PCA to the discretized input function and keep the first 200 PCs as the features for the kernel part of our GP-based methods.

approach provides very competitive results. We observe that except for the Darcy example with DeepONet³, the performance of FNO and DeepONet consistently improves if they are used as the mean function of a GP. We attribute this desirable property to two features. First, due to the second term on the right-hand side of Equation (3.28) our predictor naturally interpolates the data and hence (a) its mean function (i.e., the neural operator) can focus more on learning the sample-to-sample correlations that are *not* captured by the kernel, see the loss function in Equation (3.15), and (b) the irregular part of the target operator (that is hard to approximate with a smooth kernel) can be captured by the neural operator. Second, our use of kernels provides a natural regularization scheme that improves generalizability. To see this, we analyze Equation (3.9) within a regularization framework which is also used in contexts such as kernel ridge regression [61]. The second term in Equation (3.9) measures the data fit while the first term is a regularizer whose magnitude can be calculated based on the eigenvalues of $\mathbf{C}$. Since we freeze the kernel parameters in our NN-mean GPs, these eigenvalues are essentially fixed which imposes a smoothness condition on the underlying function space which is controlled by the decay rate of the sequence of eigenvalues.

We also observe in Table 3.1 that our zero-mean GP is highly efficient in that it achieves small

³This may be due to the fact that we could not exactly replicate the training settings and model architecture since the corresponding reference [3] lacks sufficient details.

errors with orders of magnitude fewer hyperparameters. Due to the proper initialization of the kernel parameters, these GPs provide quite competitive results even prior to parameter optimization or just after one training epoch (referred to as 0-shot and 1-shot, respectively, in the table).

In Figure 3.7 we visualize example test inputs, outputs, and predictions with associated error maps. In these plots we use the best version of our approach, i.e., a zero-mean GP in the case of Advection problem and an FNO-mean GP for the other three cases. We observe that the majority of the errors are concentrated at regions with sharp gradients which correspond to the jumps in the Advection example and the large traction regions at the top left corner in the Structural Mechanics problem. To further examine the performance of our framework in learning solution maps that exhibit large gradients, in Section B.1 we study the 1D Burgers' equation with $\zeta = 0.01$ and show that our approach provides accurate results in such settings as well.

3.3.2 Data-driven Operator Learning: Multi-Output Operators

To benchmark our approach for learning multi-output operators we use the 2D lid-driven cavity (LDC) problem which is governed by the incompressible Navier-Stokes equations:

$$\begin{aligned}
u_x + v_y &= 0, & \forall (x, y) \in (0, 1)^2 \\
uu_x + vu_y &= -\frac{1}{\rho}p_x + \zeta(u_{xx} + u_{yy}), & \forall (x, y) \in (0, 1)^2 \\
uv_x + vv_y &= -\frac{1}{\rho}p_y + \zeta(v_{xx} + v_{yy}), & \forall (x, y) \in (0, 1)^2 \\
v(x, 0) = v(x, 1) = v(0, y) = v(1, y) &= 0, & \forall x, y \in [0, 1] \\
u(x, 0) = u(0, y) = u(1, y) &= 0, & \forall x, y \in [0, 1] \\
u(x, 1) &= f(x), & \forall x \in [0, 1] \\
p(0, 0) &= 0
\end{aligned} \tag{3.47}$$

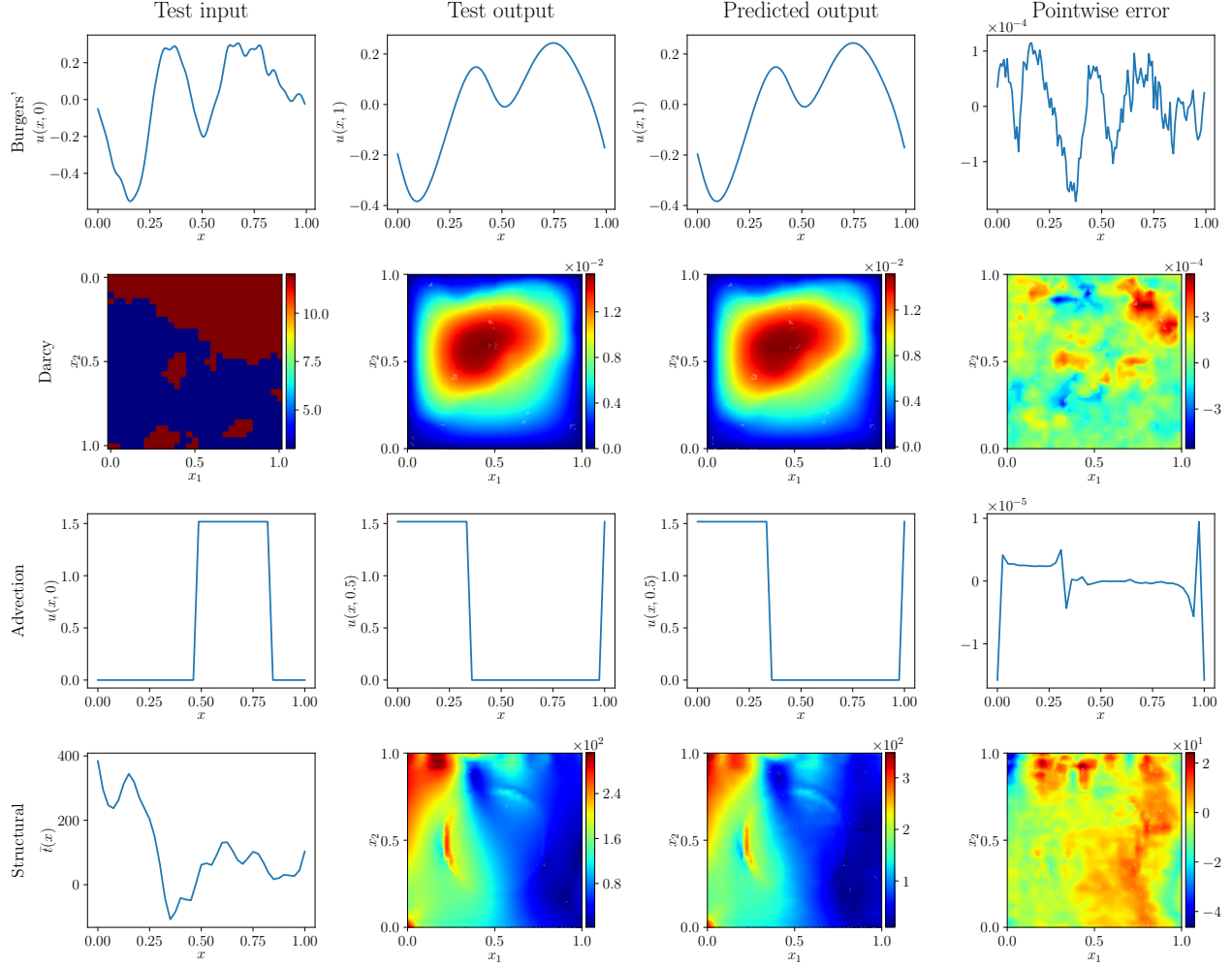


Figure 3.7 Example test inputs, outputs, and predictions: We provide one example from each problem where the predicted outputs correspond to the best version of our approach in Table 3.1, i.e., a zero-mean GP in the case of Advection problem and an FNO-mean GP for the other three problems.

where $\zeta = 0.002$ is the kinematic viscosity, $\rho = 1.0$ denotes the density, and $f(x)$ is the lid velocity profile. To generate the data, we follow [140] and first solve Equation (3.47) via the FEM for a variety of lid velocity profiles⁴ and then sweep the flow field via a window of size 0.25×0.25 to collect samples, see Figure 3.8. We divide the resulting data into two parts: if the window is above the horizontal dashed line in Figure 3.8b the corresponding sample is used to assess the extrapolation test error. Otherwise, the sample is used for either training or obtaining the interpolation test error. The samples positioned on the top part of the

⁴The velocity profiles are taken from [140] who use GPs to generate them and solve Equation (3.47) via OpenFOAM [141].

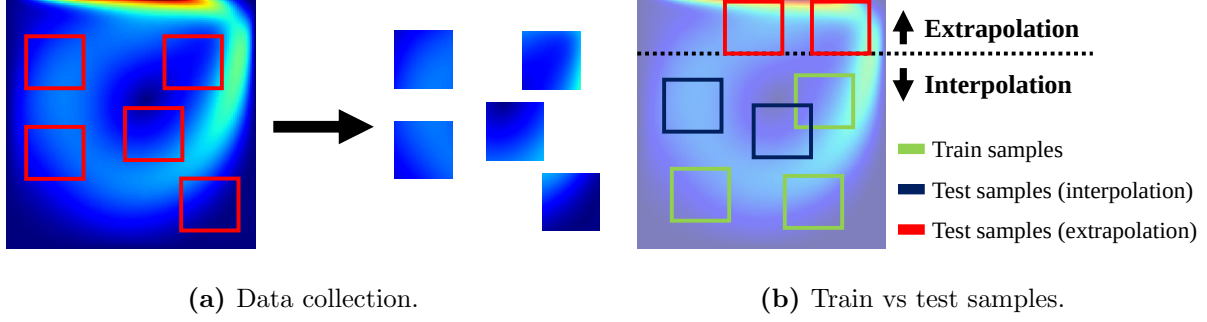


Figure 3.8 Train and test data generation in the LDC problem: (a) The $[0, 1]^2$ domain is swept with a smaller window to collect data. (b) If the center of the window is above the dashed line, the sample is used to assess the extrapolation error. Otherwise, the sample is used either for training or obtaining the interpolation error.

domain provide good candidates for evaluating the extrapolation capabilities of the models since the solution fields inside them contain much larger velocity magnitudes and gradients than those in the bottom part of the domain.

The above procedure generates a total of 961 samples with observations on a 30×30 regular grid in the 0.25×0.25 subdomain. The operator learning problems involves mapping the arbitrary set of subdomain BCs (pressure and velocity components) to the solution inside the 0.25×0.25 subdomain, i.e., $\mathcal{G}^\dagger : [u_{\text{BC}}, v_{\text{BC}}, p_{\text{BC}}] \rightarrow [u, v, p]$. We use velocity magnitude to calculate the interpolation/extrapolation errors and train all models via Adam with 2000 epochs and learning rates of 10^{-2} and 10^{-3} for the 0-mean and NN-mean GPs, respectively. We use two different architectures for all models with DeepONet and FNO that differ significantly in their parameter counts (denoted as *small* or *large*). Specifically, DeepONet-based models either have a 4-layer branch and trunk nets (each layer with 150 neurons) or a 4-layer branch and trunk nets (each layer with 24 neurons). FNO-based models either have 4 layers with a width of 4 and 8 modes, or 4 layers with a width of 24 and 8 modes. We note that we intentionally did not conduct an exhaustive search to optimize the architecture of these models because we aim to determine whether even sub-optimal architectures can benefit from being integrated within our GP framework. We use 500 samples for training while 151 and 31 samples are used to obtain the interpolation and extrapolation errors,

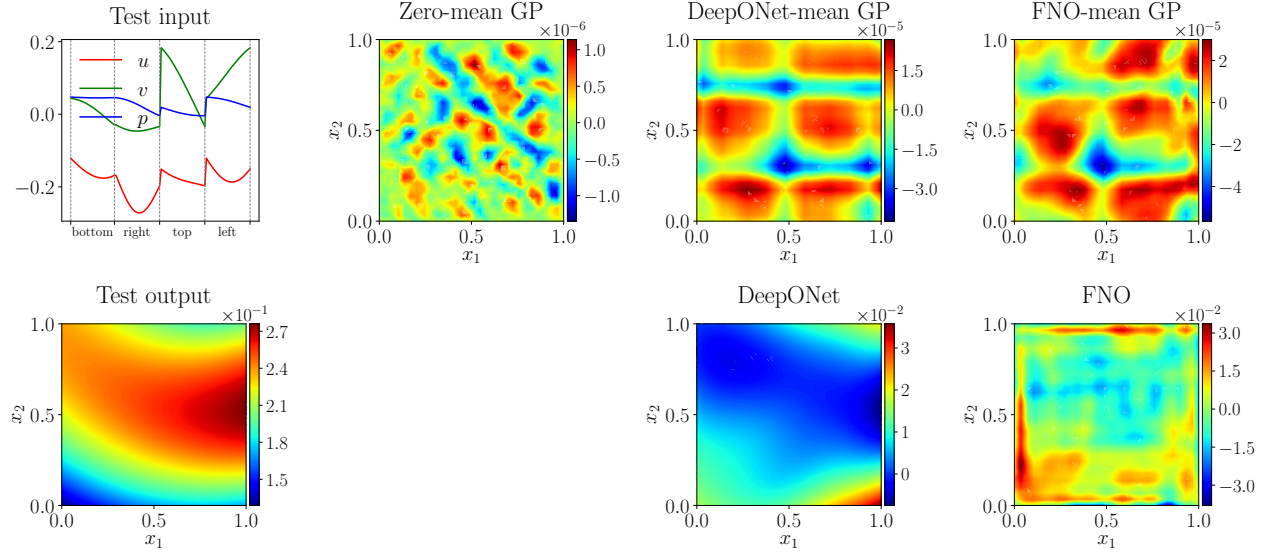
respectively.

The results of our studies are summarized in Table 3.2 and indicate that our GP-based approach consistently outperforms FNO and DeepONet in this example. The best results in interpolation are achieved with the zero-mean GP, whereas the DeepONet-mean GP trained using MLE yields the lowest errors in extrapolation. Switching to MSE from MLE reduces the accuracy in both extrapolation and interpolation regardless of whether FNO or DeepONet is used as the mean function in our approach since MSE does not take the sample-to-sample correlations into account while MLE does, see Equation (3.24). Once again, we observe that the performance of the neural operator significantly improves when it is incorporated as the mean function of our GP framework, regardless of its architecture.

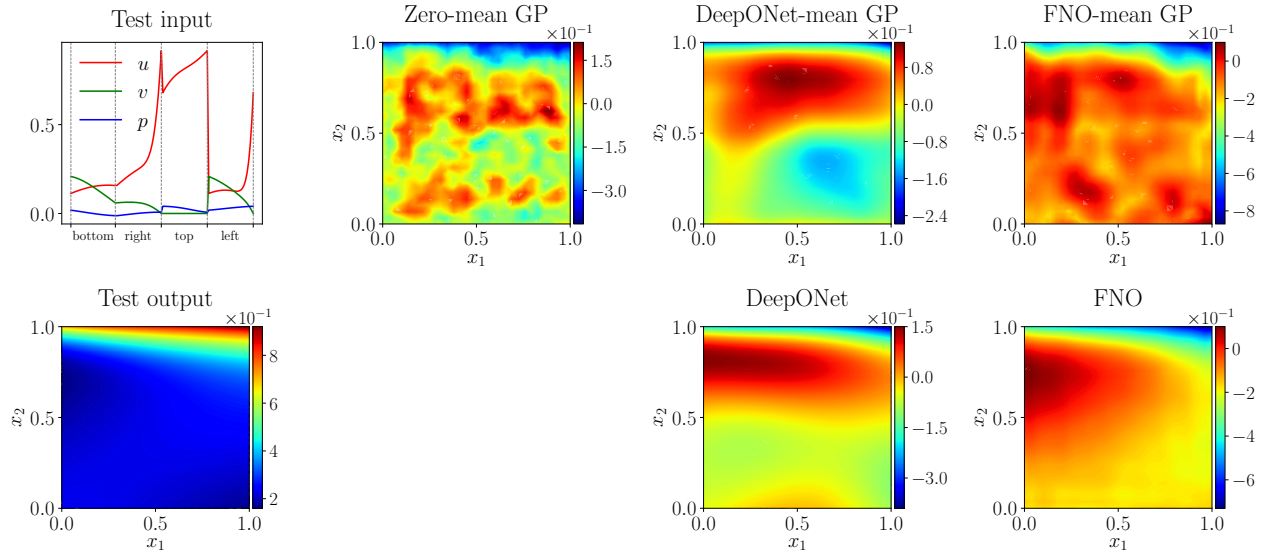
To visually compare the accuracy of different methods in extrapolation and interpolation, in Figure 3.9 we provide the error maps corresponding to the velocity magnitude for two test cases. We observe in Figure 3.9a that all methods perform quite well but our GP-based approach achieves two orders of magnitude smaller errors compared to FNO and DeepONet. However, the performance of all methods expectedly decreases in Figure 3.9b as the imposed BCs are substantially different than the ones that the models are trained on.

		Number of Parameters	Interpolation Error		Extrapolation Error	
			Training via MLE	Training via MSE	Training via MLE	Training via MSE
DeepONet (small)		13,536	N/A	4.46%	N/A	42.60%
DeepONet (large)		235,800	N/A	3.49%	N/A	35.14%
FNO (small)		6,235	N/A	7.80%	N/A	62.66%
FNO (large)		223,155	N/A	1.76%	N/A	44.20%
Ours	0-shot Zero-mean	361	7.10e−3%	N/A	24.48%	N/A
	1-shot Zero-mean	361	7.10e−3%	N/A	24.48%	N/A
	Zero-mean	361	4.37e−4%	N/A	41.00%	N/A
	DeepONet-mean (small)	13,536	5.02e−3%	1.43e−2%	31.13%	34.25%
	DeepONet-mean (large)	235,800	2.73e−3%	7.12e−3%	22.77%	23.00%
	FNO-mean (small)	6,235	3.76e−3%	9.53e−3%	26.86%	34.41%
	FNO-mean (large)	223,155	1.48e−3%	7.31e−3%	39.15%	26.70%

Table 3.2 Summary of Results for Section 3.3.2: We use DeepONet and FNO with two architectures that differ significantly in their parameter counts, referred to as *small* and *large*, for comparison in both interpolation and extrapolation tests. In our approach, we either employ these neural operators as the mean function or utilize a zero-mean GP. For the former case, we train the GPs using maximum likelihood estimation (MLE) and mean squared error (MSE) to assess the impact of the loss function on model performance. For the zero-mean GP, we also report errors before parameter optimization (denoted as 0-shot) and after one training epoch (denoted as 1-shot). All errors are reported for the velocity magnitude.



(a) Interpolation.



(b) Extrapolation.

Figure 3.9 Example input, output, and error maps on the velocity magnitudes in the LDC problem: The test samples in extrapolation look substantially different than those in interpolation (compare the scales of the velocity magnitude profiles in the two test outputs). Our GP-based approach provides smaller errors in both scenarios especially in the interpolation case.

3.3.3 Physics-Informed Operator Learning

We now consider the effect of augmenting data with physics in two examples, the Burger's equation introduced in Section 3.1.2 and the LDC problem in Section 3.2.1. In the former example, we discretize the input function via 100 points and record the output at a very

coarse grid of size 12×12 and use $N_{pi} = 50, n_{PDE} = 100^2, n_{BC} = 100$, and, $n_{IC} = 100$. In the LDC problem, each variable is discretized via 120 points on the boundaries (30 points on each edge of the square subdomain in Figure 3.8a) and the PDE solution is recorded at a fine mesh of size 30×30 , and use $N_{pi} = 100, n_{PDE} = 30^2, n_{BC} = 30$.

We solve these two problems via FNO, DeepONet, and their physics-informed versions denoted by PI-DeepONet [142] and PINO [143]. For our approach, we consider both zero-mean and NN-mean GPs where the latter is trained either with data or with both data and physics. In our NN-mean GP we only use DeepONet as we were unable to successfully integrate our code with PINO. All the models are trained via Adam with a learning rate of 10^{-3} which, similar to the previous sections, is increased to 10^{-2} for a zero-mean GP. We use 1000 training epochs in all cases except for the physics-informed models of the LDC problem where the number of epochs is reduced to 500 to decrease the computational costs. Since the density of the observations is very low in the Burgers' equation, we optimize β_y along with β_ϕ and σ^2 via MLE.

The results of our studies are summarized in Figure 3.10 where the errors on unseen data are reported for each model as N increases. For the Burgers' equation where the observations are sparse, augmenting the training data with the physics substantially helps in the case of GP and DeepONet but not FNO (compare the dashed and solid lines with the same color). This difference is much smaller in the case of LDC where adding physics slightly improves GP and DeepONet when N is very small. We also notice that even in the Burgers' problem where the output function is recorded at a coarse resolution, our purely data-driven models, i.e., without physics, achieve lower errors than the vanilla implementations of DeepONet and FNO. In the case of the zero-mean GP for the Burgers' problem, we also optimized β_y to relax the assumption on having dense observations, as opposed to other problems in Section 3.3.1. More details on this choice can be found in Section B.2. In the DeepONet-mean case, we still fix β_y as suggested in Section 3.2.3 and leverage the neural operator to

learn the irregular part of the target operator.

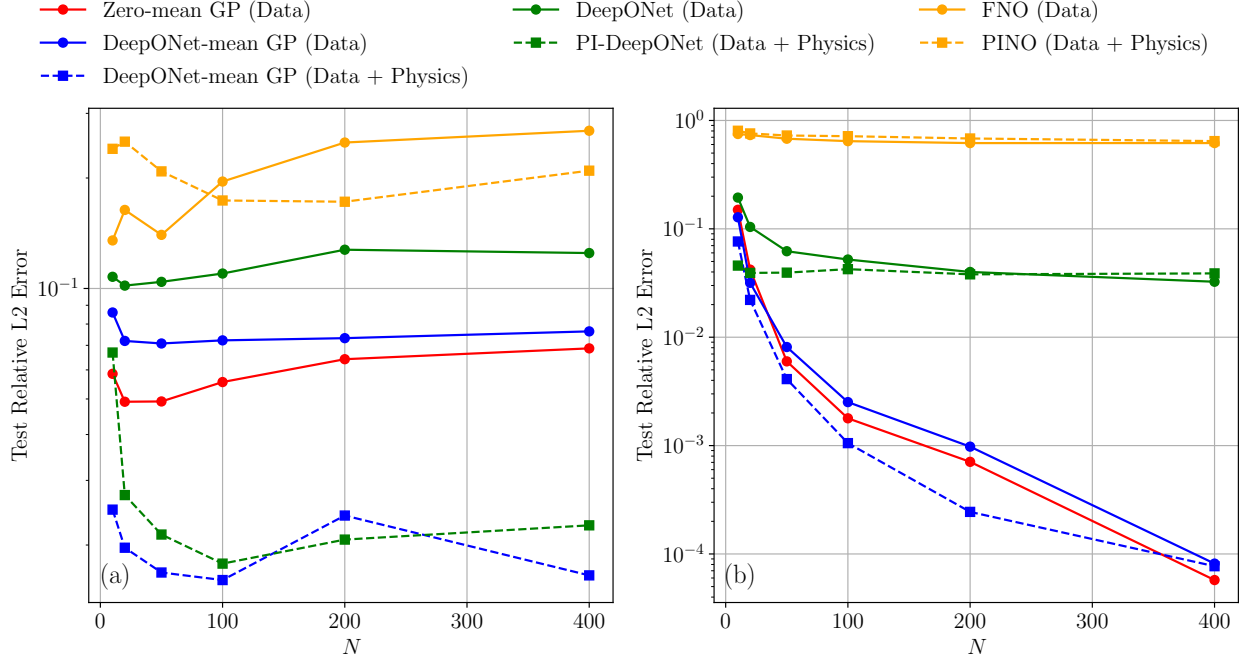


Figure 3.10 Convergence studies: We study the effect of data set size N and augmenting the data with physics on the errors for various models in the case of Burgers' (left) and LDC (right) problems.

We notice as N increases that the errors associated with the data-only models in the left panel of Figure 3.10 slightly fluctuate while those in the right panel either decrease (in our approach) or negligibly change (FNO and DeepONet). The former trend is due to the fact that we observe the PDE solution at a very coarse grid and that the architecture and training mechanisms are not altered as N increases. We attribute the latter trend to FNO and DeepONet's inability to effectively use data. However, once the exact same DeepONet is used as the mean function of a GP, the prediction accuracy dramatically improves (compare the solid green and blue lines in the right panel). See Section 3.3.1 for our reasoning behind this trend.

3.3.4 Training Cost

We compare the training costs of our approach to those of FNO and DeepONet in Table 3.3 where the costs are reported for one epoch of parameter optimization. We observe that the training cost of each approach varies depending on the dimensionality and resolution of the input/output functions in each problem. For instance, the zero-mean GP appears to have the highest training time per epoch for the Burgers’ and Advection problems where the input and output functions are one-dimensional. However, it has the lowest training cost for the Structural Mechanics problem where the input and output functions are one- and two-dimensional, respectively. Incorporating neural operators within our GP framework increases the training time per epoch with respect to their baseline implementations, due to the additional computations involved in the loss function (compare Equation (3.15) to MSE). We believe this additional cost is justified since the non-zero mean GPs achieve higher accuracy compared to the standalone neural operators, as shown in Table 3.1 and 3.2.

		Burgers’	Darcy	Advection	Structural Mechanics	Lid-driven Cavity
DeepONet		0.04	0.17	0.03	12.7	0.22
FNO		0.06	0.53	0.03	3.53	0.03
Ours	Zero-mean	0.12	0.24	0.12	1.74	0.20
	DeepONet-mean	0.06	0.25	0.05	13.4	0.27
	FNO-mean	0.07	1.17	0.05	3.66	0.04

Table 3.3 Training time in seconds per epoch: We report the costs for DeepONet, FNO, and our method with three different mean functions. The training time of each approach varies across the different problems based on the dimensionality of the input/output functions and their respective resolutions. For example, the zero-mean GP appears to be the slowest in problems where the input and output functions are one-dimensional, such as in the Burgers’ or Advection equations, but becomes the fastest in the Structural Mechanics problem, where the input and output functions are one- and two-dimensional, respectively. Incorporating neural operators within our GP framework increases the training time per epoch with respect to their vanilla implementations due to the additional computations involved in the loss function (see Equation (3.15) versus MSE). However, this results in higher overall accuracy, as shown in Table 3.1 and 3.2.

3.4 Concluding Remarks

We introduce a framework based on GPs for approximating single- or multi-output operators in either a purely data-driven context, or by using both data and physics. In the former case, we use MLE for parameter optimization and set the mean function of the GP to be either zero or a neural operator. In the physics-informed case, we employ GPs with neural operator mean functions whose parameters are estimated via a weighted combination of MLE and PDE residuals.

We emphasize that our framework is *general*, but we have devised several strategies to (1) ensure scalability to large data and high-dimensional feature spaces, and (2) minimize computational costs and numerical issues associated with inverting the GP’s covariance matrix. While these strategies have proven effective for the problems tested in this chapter, we recommend that practitioners explore alternative initialization settings, using our insights and guidelines as a basis for developing strong baseline models. That is, they may choose to use the framework in its generic form, i.e., Equations (3.9) and (3.13), or develop tailored mechanisms to train the model for their specific problem.

Our suggested training strategies rely on the assumption that both input and output functions are richly sampled at the same locations across samples. This allows us to (1) leverage the data structure together with the separability assumption on the kernel to formulate the GP’s covariance matrix using the Kronecker product, and (2) initialize the kernel parameters such that they require little to no tuning. In data-driven applications, these strategies enable us to build zero-mean GPs that provide high approximation accuracy in both single- and multi-output cases while having orders of magnitude fewer parameters than competing methods such as neural operators. In fact, we demonstrate that our zero-mean GPs provide competitive results even without training, i.e., we build the first zero-shot mechanism for operator learning. With NN-mean GPs, either in data-driven or physics-informed applica-

tions, our strategies eliminate the need for optimizing the kernel hyperparameters and result in a model that augments neural operators with the strengths of kernels. Using FNO and DeepONet as examples, we show that such an integrated model improves the performance of neural operators.

Dense observations may not always be available, especially when data is collected from physical experiments rather than simulations. Our GP framework is flexible enough to accommodate these situations and relax the assumptions we have made on having dense data. For instance, if a specific problem does not allow for a dense set of observations of the output function, one can choose to optimize β_y with a zero-mean GP to leverage correlations in the support of the output function. Alternatively, in the case of a neural-operator-mean GP, one can fix β_y and leverage the mean function to learn such correlations. These strategies have been successfully applied in Section 3.3.3 to problems where the output function is observed at a coarse resolution.

In all the examples in Section 3.3 we use the simple kernels in Equation (3.7b) that have at most a few hundred parameters and, hence, limited learning capacity. We studied various kernels including deep ones [70] but did not observe consistent and significant performance improvements. We believe a promising future direction is designing special kernels whose parameters can be efficiently estimated via cross-validation which is more robust to model misspecification compared to MLE. Another interesting direction is to extend the proposed GP framework in order to relax the assumption that input and output functions are sampled at the same locations across samples. We believe this could potentially be achieved by leveraging sparse GPs [144, 145, 146, 147, 148, 149] which were originally developed to increase the scalability of GPs for large datasets, or by using domain decomposition with distributed GPs [150].

Chapter 4

Learning Nonstationary Kernels

4.1 Introduction

Gaussian processes (GPs) are a class of powerful yet interpretable semi-parametric Bayesian models that define flexible prior distributions over functions [151]. Their natural ability to quantify uncertainty has made them a valuable tool for researchers and practitioners across various disciplines.

A GP, typically denoted as $GP(m(\mathbf{x}; \boldsymbol{\theta}), c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta}))$, is fully characterized by its mean function $m(\mathbf{x}; \boldsymbol{\theta})$ and covariance function or kernel $c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta})$ ¹ with parameters $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$, respectively. Many options exist for choosing the mean and covariance functions but zero-mean GPs with stationary kernels, i.e., kernels that depend only on the relative distance between data points rather than their absolute positions, have been the standard choice over the past decades [152]. While not always optimal, stationary kernels are widely used due to their general effectiveness, simplicity, and computational efficiency.

GPs with mean functions that have limited expressivity and stationary kernels can pro-

¹ $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^P$ denote two arbitrary input vectors.

vide suboptimal performance in terms of predictive mean and, more importantly, variance [153, 154]. This limitation is especially concerning for tasks where reliable uncertainty quantification (UQ) is essential, such as Bayesian optimization (BO) [137, 155]. In such cases, miscalibrated uncertainty estimates can lead to inefficient exploration and suboptimal decision-making. To address these limitations, substantial efforts have been made to enhance GP emulation capabilities to model nonstationary functions, either through flexible mean functions [138], nonstationary kernels [156, 157], or both [158]. While flexible mean functions can enhance a GP’s predictive accuracy in nonstationary problems, they fail to fully correct the predictive variance. Consequently, developing nonstationary kernels remains a more effective approach for improving both mean and uncertainty predictions.

Unlike stationary kernels which depend only on the relative distance between two inputs, i.e., $c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta}) = c(\mathbf{x} - \mathbf{x}'; \boldsymbol{\beta})$, nonstationary kernels allow the relationship to depend on the *absolute* positions of the data points, i.e., $c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta}) \neq c(\mathbf{x} - \mathbf{x}'; \boldsymbol{\beta})$. This makes nonstationary kernels much more flexible to capture fine-grained variations across the entire domain. However, effectively designing them remains an open research challenge, as it is unclear how to best provide this flexibility without introducing new issues. While one might assume that GPs are inherently safeguarded against overparameterization due to their probabilistic nature, research has shown that naively increasing kernel/mean complexity can lead to severe optimization difficulties and overfitting [159, 160, 161].

Numerous techniques have been proposed to build nonstationary kernels and we broadly categorize them into three types [154]: (1) input-dependent lengthscales [156, 157, 162], (2) input warping [163, 164, 165], and (3) mixture of GP experts [166].

The input-dependent lengthscale approach modifies the kernel function by allowing the lengthscale parameters to depend on the inputs. Notable examples include the Gibbs kernel [156], which explicitly incorporates input-dependent smoothness by modeling the length-scale function using either a GP or a neural network (NN) [157, 167, 168]. However, the

lack of structural constraints on the lengthscale function often poses overfitting and non-identifiability issues [165, 154], since many lengthscale functions can yield similar likelihood values.

The input warping approach transforms the input space via a mapping before applying the kernel. These transformations can be learned [163, 164] or predefined using a set of basis functions [156, 169]. A well-known recent example is the deep kernel [163], which employs NNs to learn an expressive input transformation before applying the kernel. The concept of input warping dates back several decades, originating from the characterization of stationary reducible and locally stationary reducible kernels [170, 171, 172] where the key idea is to learn a feature space where stationarity or local stationarity holds. A key limitation of this approach is that the learned mapping must be bijective to ensure a valid transformation [170]. For example, [164] models the mapping using the Beta cumulative distribution function, which can represent a broad class of bijective functions while having few hyperparameters.

Lastly, the mixture of experts approach partitions the input space into different regions and assigns distinct GP models to each, allowing for locally adaptive behavior [166, 173, 174]. Typically, these models employ gating functions to weight contributions from different experts and ensure smooth transitions across regions [173, 175]. However, scaling this approach to high-dimensional problems is challenging, as the number of experts must increase dramatically with the input dimensionality.

A more recent approach that lies at the intersection of the above three categories is the attentive kernel (AK) [154]. AK is designed to mitigate the training challenges of nonstationary kernels by introducing similarity attention scores to weight a predefined set of basis kernels and visibility attention scores to mask out data across sharp transitions. This approach essentially selects relevant subsets of data in prediction and has demonstrated improved mean and uncertainty estimates while reducing overfitting compared to conventional nonstationary

kernels, particularly in 2D and 3D robotics applications.

Our proposed method introduces a new category for learning nonstationary kernels. We provide a structured way to construct expressive kernels that scale well and can capture complex nonstationary patterns while being robust to overfitting. In essence, our idea is to build a nonstationary kernel via a self-adaptive composition of a set of base kernels. The self-adaptivity nature of our kernel underlies its nonstationarity feature and is achieved by integrating learnable and input-dependent weights with the base kernels such that the resulting composition not only remains interpretable, but also is guaranteed to be a valid kernel. We call our kernel Self-adaptive Explainable Kernel or SEEK.

The remainder of the chapter is structured as follows. In Section 4.2, we introduce SEEK and then present a comprehensive evaluation of its performance on multiple benchmark problems in Section 4.3. We summarize our contributions and discuss potential directions for future research in Section 4.4.

4.2 Methods

We briefly review GPs in Section 4.2.1. Then, we discuss the conditions that a valid covariance function should satisfy and the main closure properties of kernels in Sections 4.2.2 and 4.2.3, respectively. These discussions set the stage for introducing SEEK in Section 4.2.4.

4.2.1 Gaussian Processes (GPs)

A GP is a stochastic process whose samples follow a multivariate normal distribution. In the context of regression problems, we consider a training dataset $\mathcal{D} := \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, and denote the collection of inputs by $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$, where $\mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^P$, with corresponding out-

puts $\mathbf{y} = [y_1, \dots, y_N]^T$, where $y_i \in \mathbb{R}$. We assume that the samples are generated according to the model²:

$$y(\mathbf{x}) = f(\mathbf{x}) + \epsilon, \quad (4.1)$$

where $f(\mathbf{x})$ is the unknown latent function and $\epsilon \sim \mathcal{N}(0, \lambda^2)$ is the independent Gaussian noise with variance λ^2 . Under this framework, we place a GP prior on $f(\mathbf{x})$, i.e., $f(\mathbf{x}) \sim GP(m(\mathbf{x}; \boldsymbol{\theta}), c(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta}))$. Hereafter, we omit the dependencies of $m(\mathbf{x})$ on $\boldsymbol{\theta}$ and $c(\mathbf{x}, \mathbf{x}')$ on $\boldsymbol{\beta}$ to improve readability.

A key property of GPs is that they are closed under Bayesian conditioning. This implies that the posterior distribution of $f(\mathbf{x})$ conditioned on the observed data $\mathcal{D}$ remains a GP, i.e., $f(\mathbf{x})|\mathcal{D} \sim GP(\bar{m}(\mathbf{x}), \bar{c}(\mathbf{x}, \mathbf{x}'))$. Therefore, the posterior mean and variance at any unseen input $\mathbf{x}^*$ have the following closed-form expressions:

$$\bar{m}(\mathbf{x}^*) = m(\mathbf{x}^*) + c(\mathbf{x}^*, \mathbf{X})\mathbf{C}_\lambda^{-1}(\mathbf{y} - \mathbf{m}), \quad (4.2a)$$

$$\bar{c}(\mathbf{x}^*, \mathbf{x}^*) = c(\mathbf{x}^*, \mathbf{x}^*) - c(\mathbf{x}^*, \mathbf{X})\mathbf{C}_\lambda^{-1}c(\mathbf{X}, \mathbf{x}^*), \quad (4.2b)$$

where $\mathbf{C}_\lambda = \mathbf{C} + \lambda^2 \mathbf{I}$ is the $N \times N$ covariance matrix with $\mathbf{C} = c(\mathbf{X}, \mathbf{X})$, and $\mathbf{m} = m(\mathbf{X})$. For the purpose of this paper, we only consider zero-mean functions, i.e., $m(\mathbf{x}) = 0$, and focus on the kernel for modeling nonstationary functions.

Although one could use the posterior equations from Eq. 4.2 directly for predictions without estimating the kernel parameters $\boldsymbol{\beta}$ and noise variance λ^2 , it is common practice to learn them via maximum likelihood estimation (MLE), that is:

$$\left[\hat{\boldsymbol{\beta}}, \hat{\lambda}^2 \right] = \underset{\boldsymbol{\beta}, \lambda^2}{\operatorname{argmin}} \mathcal{L} = \underset{\boldsymbol{\beta}, \lambda^2}{\operatorname{argmin}} \frac{1}{2} \log |\mathbf{C}_\lambda| + \frac{1}{2} \mathbf{y}^T \mathbf{C}_\lambda^{-1} \mathbf{y}. \quad (4.3)$$

²The GP framework can be easily extended to multi-dimensional outputs but for simplicity we present the formulation for the single-output case.

Typical choices for $c(\mathbf{x}, \mathbf{x}')$ are the stationary Gaussian and Matérn kernels:

$$c(\mathbf{x}, \mathbf{x}') = \exp(-d^2), \quad (4.4a)$$

$$c(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} d \right)^\nu K_\nu \left(\sqrt{2\nu} d \right). \quad (4.4b)$$

In both cases, $d = \sqrt{(\mathbf{x} - \mathbf{x}')^T \text{diag}(10^\omega)(\mathbf{x} - \mathbf{x}')}$, where $\omega \in \mathbb{R}^P$ is the vector of lengthscale parameters. In the Matérn kernel, K_ν is the modified Bessel function of the second kind, and Γ is the gamma function. It is common practice to fix ν to half-integer values $\frac{1}{2}, \frac{3}{2}$ or $\frac{5}{2}$ to simplify the Γ and K_ν functions, leading to simpler closed-form expressions of Matérn kernel that substantially reduce computational costs.

Stationary covariance functions, such as those in Equations 4.4a and 4.4b, introduce the inductive bias that nearby input points yield correlated outputs. However, an extensive class of functions can serve as valid GP kernels provided that they satisfy certain conditions, which we review in the following sections.

4.2.2 Validity of Kernels for GPs

To serve as a valid kernel for GPs, a function must satisfy two necessary conditions:

Condition 1 (Symmetry). A function $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is symmetric if

$$c(\mathbf{x}, \mathbf{x}') = c(\mathbf{x}', \mathbf{x}), \quad \forall \mathbf{x}, \mathbf{x}'.$$

Condition 2 (Positive semi-definiteness). A function $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is positive-semidefinite (PSD) if, for any finite set of points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \subset \mathcal{X}$, the resulting covariance matrix $\mathbf{C} = [c(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n$ satisfies

$$\mathbf{a}^\top \mathbf{C} \mathbf{a} \geq 0, \quad \forall \mathbf{a} \in \mathbb{R}^n, \mathbf{a} \neq \mathbf{0}.$$

Conditions 1 and 2 ensure that $c(\mathbf{x}, \mathbf{x}')$ is symmetric and PSD, making it a valid kernel for GPs.

4.2.3 Some Closure Properties on Kernels

We now present four fundamental closure properties of kernels. These properties form the theoretical foundation of SEEK.

Let $c_1 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, $c_2 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ and $c_3 : \mathcal{Z} \times \mathcal{Z} \rightarrow \mathbb{R}$ be valid kernels, with $\psi : \mathcal{X} \rightarrow \mathcal{Z} \subset \mathbb{R}^Z$. Then, the following functions are also valid kernels:

Property 1 (Scaling).

$$c(\mathbf{x}, \mathbf{x}') = \alpha c_1(\mathbf{x}, \mathbf{x}'), \quad \forall \alpha \geq 0.$$

Property 2 (Addition).

$$c(\mathbf{x}, \mathbf{x}') = \alpha_1 c_1(\mathbf{x}, \mathbf{x}') + \alpha_2 c_2(\mathbf{x}, \mathbf{x}'), \quad \forall \alpha_1, \alpha_2 \geq 0.$$

Property 3 (Product).

$$c(\mathbf{x}, \mathbf{x}') = c_1(\mathbf{x}, \mathbf{x}') c_2(\mathbf{x}, \mathbf{x}').$$

Property 4 (Warping).

$$c(\mathbf{x}, \mathbf{x}') = c_3(\psi(\mathbf{x}), \psi(\mathbf{x}')).$$

Building upon Properties 1–3, it follows that if f is a polynomial with non-negative coefficients, then $f(c(\mathbf{x}, \mathbf{x}'))$ is also a valid kernel. Theorem 1 further generalizes kernel validity by establishing conditions under which polynomial transformations and related analytic expansions preserve kernel validity.

Theorem 1 (Kernel Validity under Analytic Transformations [176, Theorem 7.5.9]). Let $Z = [z_{ij}] \in \mathbb{R}^{n \times n}$ be positive semidefinite, where $i, j \in \{1, \dots, n\}$.

1. The Hadamard powers $Z^{(k)} = [z_{ij}^k]$ are positive semidefinite for all $k = 1, 2, \dots$; they are positive definite if Z is positive definite.
2. Let $f(z) = a_0 + a_1 z + a_2 z^2 + \dots$ be an analytic function with nonnegative coefficients and radius of convergence $R > 0$. Then $[f(z_{ij})]$ is positive semidefinite if $|z_{ij}| < R$ for all $i, j \in \{1, \dots, n\}$; it is positive definite if, in addition, Z is positive definite and $a_k > 0$ for some $k \in \{1, 2, \dots\}$.
3. The Hadamard exponential matrix $[e^{z_{ij}}]$ is positive semidefinite; it is positive definite if and only if no two rows of Z are identical.

From this theorem, it follows that any analytic function $f(\cdot)$ which can be expressed (or approximated) as a power series (e.g., a Taylor series expansion) with all non-negative coefficients, preserves Condition 2 (PSD) within its radius of convergence. Moreover, since such a function maintains symmetry (Condition 1), it then preserves the validity of the kernel.

This result can be leveraged to show that the Gaussian kernel is indeed a valid kernel, or more generally, that the exponential transformation $\exp(c(\mathbf{x}, \mathbf{x}'))$ preserves kernel validity. This reasoning naturally extends to other transformations, including $\sinh(c(\mathbf{x}, \mathbf{x}'))$ and $\cosh(c(\mathbf{x}, \mathbf{x}'))$.

Regarding Property 4, we note that it underlies input warping techniques such as deep kernels [163] where a learned feature mapping $\psi(\mathbf{x})$ is used to transform the input space before applying a base kernel.

These closure properties provide a principled and structured framework for systematically constructing new kernels from existing ones while preserving the fundamental requirements of symmetry and positive semi-definiteness, as outlined in Conditions 1 and 2. In Section 4.2.4, we leverage Properties 1–4 and Theorem 1 to introduce SEEK.

4.2.4 SEEK: Our Proposed Kernel

SEEK is inspired by the architecture of a single artificial neuron in NNs [177]. In a typical neuron, each feature is multiplied by a weight, a bias term is added, and the result is passed through an activation function to capture complex patterns. Following this intuition, we design SEEK by applying an appropriate nonlinear activation function to a weighted sum of base kernels and an added bias term, ensuring validity through kernel closure properties (Properties 1–4) and Theorem 1. In SEEK, the weights and the bias are learnable functions of the inputs (e.g., parameterized by NNs), making the overall kernel nonstationary. Definition 1 lays out the mathematical form of SEEK which is formulated in Equation (4.5).

Definition 1 (SEEK Kernel). Let $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^P$ be two input vectors, and consider vector-valued functions $\mathbf{w}_m(\mathbf{x}; \boldsymbol{\beta}_{\mathbf{w}_m}) : \mathbb{R}^P \rightarrow \mathbb{R}^{W_m}$ and $\mathbf{b}(\mathbf{x}; \boldsymbol{\beta}_b) : \mathbb{R}^P \rightarrow \mathbb{R}^B$, parameterized via $\boldsymbol{\beta}_{\mathbf{w}_m}$ and $\boldsymbol{\beta}_b$, respectively³. Given a set of M base kernels $\{c_m(\mathbf{x}, \mathbf{x}'; \boldsymbol{\beta}_{c_m})\}_{m=1}^M$ and an appropriate activation function $\phi : \mathbb{R} \rightarrow \mathbb{R}$, we define SEEK as:

$$c(\mathbf{x}, \mathbf{x}') = \phi(z(\mathbf{x}, \mathbf{x}')), \quad (4.5)$$

with the pre-activation $z(\mathbf{x}, \mathbf{x}')$ defined as:

$$z(\mathbf{x}, \mathbf{x}') = \sum_{m=1}^M \mathbf{w}_m^\top(\mathbf{x}) \mathbf{w}_m(\mathbf{x}') c_m(\mathbf{x}, \mathbf{x}') + \mathbf{b}^\top(\mathbf{x}) \mathbf{b}(\mathbf{x}'). \quad (4.6)$$

where we have dropped the dependence of the functions in Equations 4.5 and 4.6 on their parameters to improve readability.

Figure 4.1 illustrates the overall workflow of SEEK, where the base kernels $c_m(\mathbf{x}, \mathbf{x}')$ are weighted, combined, and then passed through an activation function to produce the final

³Unless otherwise specified, all vectors are treated as column vectors.

kernel output. This formulation enables SEEK to flexibly capture complex, nonlinear relationships by modulating the contributions of multiple base kernels adaptively across the entire input space. We now discuss three important features of SEEK.

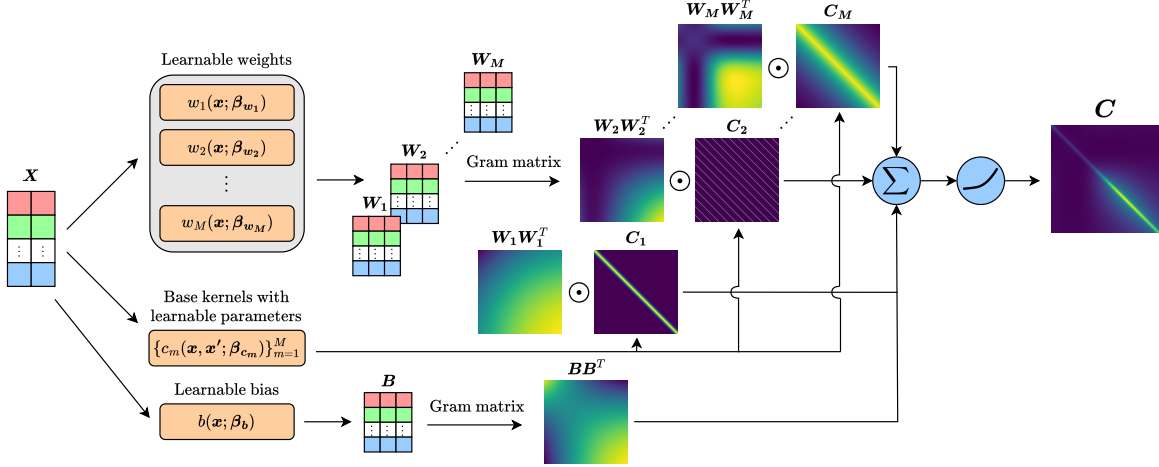


Figure 4.1 Schematic illustration of our kernel: SEEK has a set of weighted base kernels with learnable hyperparameters where the weights are learnable functions. These weighted kernels, along with a learnable bias term, are summed and then passed through an appropriate activation function to produce the final nonstationary covariance function.

First, the weight functions $w_m(\mathbf{x})$ and the bias function $\mathbf{b}(\mathbf{x})$ can be modeled using different function approximators, each offering varying levels of expressiveness and complexity. Potential choices include polynomials, NNs, or GPs.

The second feature is the flexibility in the choice of base kernels $\{c_m(\mathbf{x}, \mathbf{x}')\}_{m=1}^M$, which can include any valid stationary (e.g., Gaussian kernel) and nonstationary kernels (e.g., Gibbs kernel). This flexibility allows SEEK to capture a wide range of behaviors, from globally stationary phenomena to cases where smoothness and variability change across the input domain. We highlight that SEEK can use M kernels of the same type (e.g., M Gaussians), with each component learning specific patterns in different parts of the domain.

The last key feature is that each term in Equation 4.6 leverages kernel closure Properties 1–4 to ensure its validity. For instance, the terms $\mathbf{w}_m^\top(\mathbf{x})\mathbf{w}_m(\mathbf{x}')$ and $\mathbf{b}^\top(\mathbf{x})\mathbf{b}(\mathbf{x}')$ are valid kernels, as they result from applying Property 4 to the linear kernel. The product

$\mathbf{w}_m^\top(\mathbf{x})\mathbf{w}_m(\mathbf{x}')c_m(\mathbf{x}, \mathbf{x}')$ also ensures validity, according to Property 3. More generally, by applying Properties 1–3, it can be shown that the pre-activation $z(\mathbf{x}, \mathbf{x}')$ itself is a valid kernel.

Choosing an appropriate activation function $\phi(\cdot)$ is essential to guarantee that the final result is a valid kernel. As discussed in Section 4.2.3, potential candidates for $\phi(\cdot)$ include, but are not limited to, $\exp(\cdot)$, $\sinh(\cdot)$ and $\cosh(\cdot)$. The third feature discussed above is in fact essential for ensuring the well-posedness of *any* nonstationary kernel. Just as PointNet [178], an NN architecture for processing point clouds, ensures permutation invariance at each of its building blocks, nonstationary kernels must guarantee kernel validity at every stage of their construction.

This combination of features enables SEEK to provide a flexible, customizable, and interpretable framework for kernel learning while ensuring kernel validity *by construction*. In Section 4.2.5, we demonstrate the explainability of the proposed kernel, and in Section 4.3.2, we conduct sensitivity analyses to provide more insights on its key design choices.

4.2.5 Explainability of SEEK: Illustrative Example

We demonstrate the behavior of the proposed method on the **Analytic I** problem introduced in Section 4.3.1. We use a zero-mean GP with SEEK, trained via MLE on the same 50 data points as in [160]. As base kernels, we employ Gaussian, periodic and Matérn kernels, i.e., $M = 3$, and use two independent neural networks—each with two hidden layers and four neurons per layer—to learn the weights $\{\mathbf{w}_m(x; \boldsymbol{\beta}_{\mathbf{w}_m})\}_{m=1}^M$ and bias $\mathbf{b}(x; \boldsymbol{\beta}_{\mathbf{b}})$.

The main results are illustrated in Figure 4.2. Subplot (a) presents the predicted mean and 95% confidence interval of the model. Subplots (b,c) show the learned weighted base covariances for the Gaussian and periodic kernels⁴, given by $\mathbf{w}_m^\top(x)\mathbf{w}_m(x')c_m(x, x')$ for $m = 1, 2$.

⁴For brevity, we omit the Matérn kernel and bias term in the subplots from Figure 4.2.

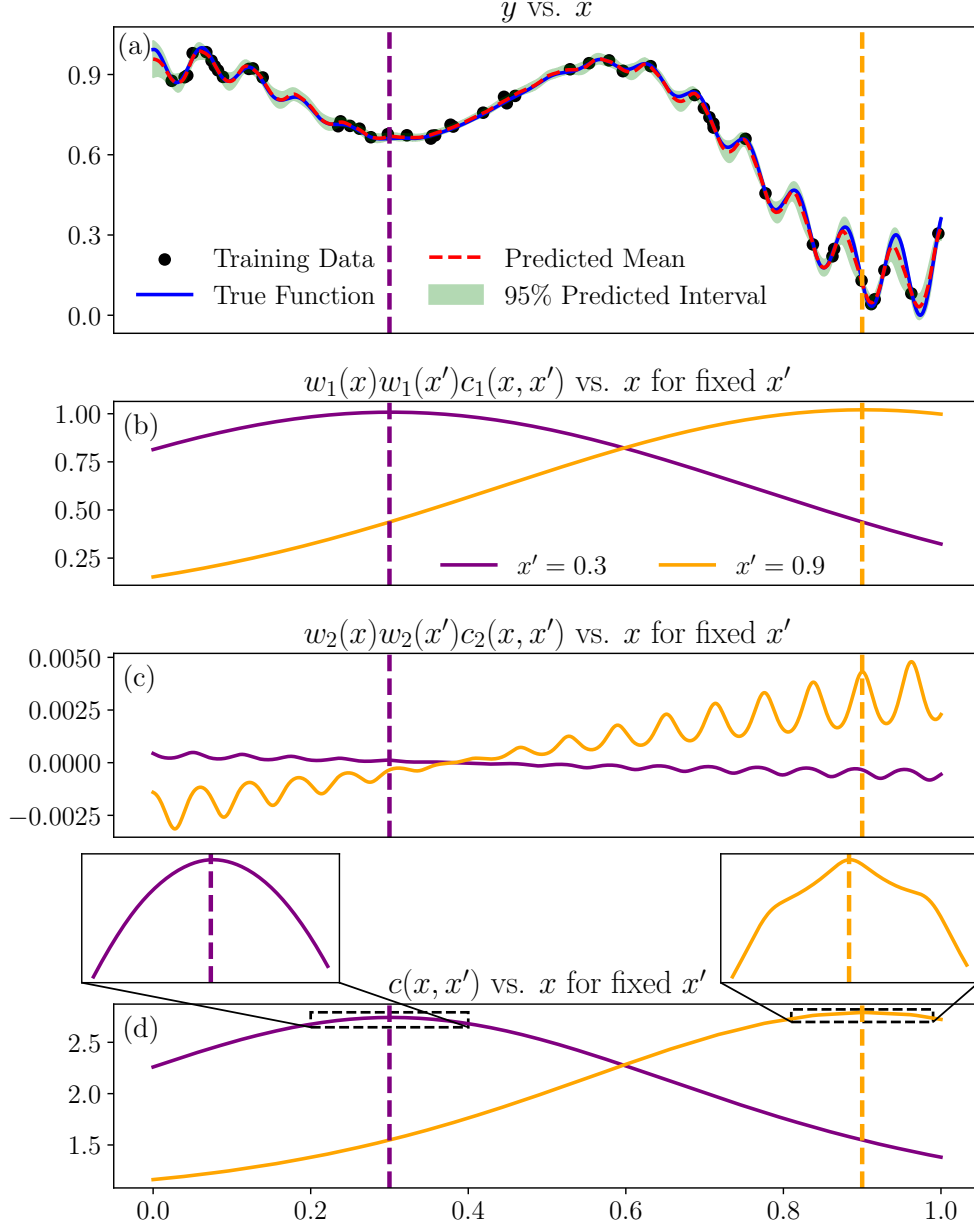


Figure 4.2 Prediction of a GP using SEEK on the Analytic I problem, along with two of the learned weighted base kernels and the resulting covariance function evaluated at two reference points.

Subplot (d) illustrates the resulting covariance function $c(x, x')$ from SEEK. The functions in subplots (b-d) are evaluated at two reference points exhibiting different behaviors: smooth at $x' = 0.3$ (purple) and high-frequency variations at $x' = 0.9$ (orange).

From Figure 4.2(a), we observe that our GP (1) produces highly accurate predictions, achiev-

ing an RMSE of 0.013 which is notably smaller than 0.038 reported in [160], and (2) provides well-calibrated prediction intervals, obtaining an NNOIS of 0.189 (see Eq. 4.10b). Figures 4.2(b) and 4.2(c) illustrate the learned weighted Gaussian and periodic kernels, respectively. The weighted Gaussian kernel exhibits a similar trend at both reference points where it has a maximum that is smoothly decreased. Unlike the Gaussian kernel, the weighted periodic kernel varies adaptively. Specifically, for $x' = 0.3$, the kernel exhibits little to no variation across the entire domain, aligning with the smooth nature of the underlying function. In contrast, at $x' = 0.9$, the kernel adjusts to local variations, i.e., it exhibits an oscillating pattern only in regions where the underlying function does as well. Finally, Figure 4.2(d) presents the resulting covariance function obtained by Eq. 4.5. At a large scale, the covariance functions for both reference points appear similar. However, a closer look (see insets) reveals clear distinct behaviors: $c(x, x' = 0.9)$ exhibits a more oscillatory structure, driven by the learned weighted periodic kernel, whereas $c(x, x' = 0.3)$ presents a smoother behavior.

These results highlight the flexibility and interpretability of SEEK for kernel learning in GPs. By dynamically adjusting the contribution of different base kernels, the model effectively captures both smooth and high-frequency behaviors in different regions of the input space. This adaptive behavior is crucial for modeling complex, nonstationary functions where stationary kernels inherently fall short.

4.3 Results and Discussion

We begin this section by introducing the datasets and evaluation metrics. Then, we conduct sensitivity analyses to evaluate the dependence of our method on various design choices. Finally, we present a series of comparative studies where we assess the performance of SEEK against other nonstationary kernels. The numerical experiments in this section are implemented using the open-source Python package GP+ [179].

All simulations are repeated 16 times independently (except engineering problems in Section 4.3.4 and Bayesian optimization (BO) runs in Section 4.3.5, which are repeated 8 times) to ensure the results are representative across random initializations. Also, all the GPs are trained via the L-BFGS optimizer with a learning rate of 0.01. For further implementation and optimization details, please see Section C.1 in the Appendix.

4.3.1 Benchmark Problems and Metrics

We consider seven benchmark problems with varying levels of nonlinearity and dimensionality. These include analytic functions adopted from the literature [160, 180] and real-world datasets derived from engineering applications [154, 181, 182, 183].

Analytic I: we consider the following 1D function [160]:

$$f(x) = \frac{1}{3.94} (\sin(5x) + \cos(10x)) + 1.435(x - 0.4)^2 \cos(100x) + 0.659 \quad (4.7)$$

where $x \in [0, 1]$. We use 55 points randomly drawn via Sobol sequence for our training dataset, and corrupt these samples with noise $\epsilon \sim \mathcal{N}(0, \lambda^2 = 10^{-4})$.

Analytic II: we design the following 1D function:

$$f(x) = \begin{cases} \text{env}(x) + 0.1 \sin(8\pi x), & 0 \leq x < 2, \\ 0.5 e^{x-2} \sin(2\pi x), & 2 \leq x < 4, \\ \sin\left(2\pi\left[2 + (x-4)^2\right] \frac{x-4}{2}\right), & 4 \leq x < 6, \\ \begin{cases} 16t, & 0 \leq t < 0.25, \\ 8 - 16t, & 0.25 \leq t < 0.5, \end{cases} & 6 \leq x < 8, \\ \sin(2\pi x) + 0.5 \sin(8\pi x), & 8 \leq x \leq 10, \end{cases} \quad (4.8)$$

where

$$t = (x - 6) \bmod 0.5,$$

with $x \in [0, 10]$ and

$$\text{env}(x) = \begin{cases} 4x, & 0 \leq x < 0.5, \\ 2, & 0.5 \leq x \leq 1.5, \\ 4(2 - x), & 1.5 < x \leq 2. \end{cases}$$

We draw 140 samples via Sobol sequence, and corrupt them with noise $\epsilon \sim \mathcal{N}(0, \lambda^2 = 10^{-4})$.

Volcano: we consider a dataset containing the terrain elevation of Mount Saint Helens as a function of planar spatial location [154]. This terrain exhibits nonstationary behavior due to the presence of prominent environmental features, which contribute to spatially varying smoothness and complexity of the terrain. We use a training dataset consisting of 400 data points drawn via Sobol sequence, and corrupt them with noise $\epsilon \sim \mathcal{N}(0, \lambda^2 = 1)$.

Hartmann: we consider the 6D Hartmann function [184, 180]:

$$f(\mathbf{x}) = - \sum_{i=1}^4 \alpha_i \exp \left(- \sum_{j=1}^6 A_{ij} (x_j - P_{ij})^2 \right), \quad (4.9)$$

where $\mathbf{x} \in [0, 1]^6$ and the constants α_i , A_{ij} , and P_{ij} are typically set as follows:

$$\boldsymbol{\alpha} = [1.0, 1.2, 3.0, 3.2],$$

$$A = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix},$$

$$P = 10^{-4} \times \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}.$$

This function has multiple local minima and is inherently nonstationary where small input perturbations can induce large variations in the output. We use 800 samples for training drawn via Sobol sequence, and corrupt them with noise $\epsilon \sim \mathcal{N}(0, \lambda^2 = 10^{-4})$.

ROM: we consider our in-house dataset derived from a reduced-order model for multiscale fracture simulations of cast aluminum alloys [181]. Each sample contains 6 real-valued features (four microstructural descriptors and two material properties) and a single output corresponding to the toughness of the microstructure. In each repetition, we train the models with 200 samples randomly drawn from the full dataset and test their performance on the remaining 50 samples.

Elevators: this dataset [182] captures the behavior of an aircraft elevator control system.

It has 18 continuous inputs representing the aircraft’s current flight state and recent elevator commands, and a single response showing the desired elevator deflection for the next control step. We discard eight of the input features as they do not affect the response and for each repetition we randomly draw 1,750 training samples (20% of the full training set) and evaluate the performance of the models on 7,847 test samples.

Pumadyn: this dataset [183] was derived from a realistic high-fidelity simulation of the dynamics of a Unimation PUMA 560 robot arm. We use the *puma32H* variant with moderate output noise, where each data point contains 32 continuous input features including angular positions, velocities, torques and additional dynamic parameters of the robot arms, and a single continuous target corresponding to the angular acceleration of the arm’s sixth link. For each repetition, we randomly draw 2,699 training samples (60% of the full training set) and use 3,692 test samples to assess the performance of the models.

We note that we downsampled the training data due to scalability constraints. We aim to address these limitations in future work, as discussed in Section 4.4.

For all our experiments, we scale both inputs and output using the mean and standard deviation computed from the training set (to avoid data leakage). The same transformation is then applied to the test set for consistency. We evaluate the performance of all models across all examples using a noiseless test set consisting of N_{test} samples. Our evaluation metrics are the normalized root mean squared error (NRMSE) and the normalized negatively oriented interval score (NNOIS):

$$\text{NRMSE} = \frac{1}{s} \sqrt{\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (\bar{m}(\mathbf{x}_i) - f(\mathbf{x}_i))^2}, \quad (4.10a)$$

$$\begin{aligned}
\text{NNOIS} = & \frac{1}{s} \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (\bar{u}(\mathbf{x}_i) - \bar{l}(\mathbf{x}_i)) \\
& + \frac{2}{\alpha} (\bar{l}(\mathbf{x}_i) - f(\mathbf{x}_i)) \mathbb{1}\{f(\mathbf{x}_i) < \bar{l}(\mathbf{x}_i)\} \\
& + \frac{2}{\alpha} (f(\mathbf{x}_i) - \bar{u}(\mathbf{x}_i)) \mathbb{1}\{f(\mathbf{x}_i) > \bar{u}(\mathbf{x}_i)\},
\end{aligned} \tag{4.10b}$$

where s is the output standard deviation of the N_{test} test samples, and $\bar{l}(\mathbf{x}_i)$ and $\bar{u}(\mathbf{x}_i)$ denote the predicted lower and upper bounds of the prediction interval for the i -th test sample, respectively. We employ 95% prediction intervals, i.e., $\alpha = 0.05$, and thus its endpoints can be computed via $\bar{l}(\mathbf{x}_i) = \bar{m}(\mathbf{x}_i) - 1.96\bar{c}(\mathbf{x}_i, \mathbf{x}_i)$ and $\bar{u}(\mathbf{x}_i) = \bar{m}(\mathbf{x}_i) + 1.96\bar{c}(\mathbf{x}_i, \mathbf{x}_i)$. $\mathbb{1}\{\cdot\}$ is an indicator function which is 1 if its condition holds and zero otherwise. For both metrics in Eq. 4.10 lower values are better, with NRMSE reflecting the accuracy of the mean predictions and NNOIS accounting for the quality of the prediction intervals.

4.3.2 Sensitivity Studies

We conduct four sensitivity studies to assess the impact of key design choices on the performance of SEEK:

- **Sensitivity Study 1 (Convergence behavior):** we examine the convergence behavior as a function of data size N . For SEEK, we use a single Gaussian kernel as the base kernel and compare it to the Gaussian kernel in Equation 4.4a.
- **Sensitivity Study 2 (Kernel structure):** we analyze the influence of the number and types of base kernels used within SEEK. In the plots, we use the format “letter-number” to indicate the type and total number of base kernels. Specifically, “G” refers to the Gaussian kernel, “PE” to the power exponential kernel, and “H” to a hybrid kernel (a combination of Gaussian, periodic, and Matérn kernels).
- **Sensitivity Study 3 (Kernel activation function):** we evaluate the impact of ϕ

by testing exponential, hyperbolic sine, hyperbolic cosine, and identity (denoted as “iden”) activation functions.

- **Sensitivity Study 4 (Network architecture):** we investigate the impact of the number of neurons and the choice of activation function in the NNs⁵ employed for learning $\mathbf{w}$ and $\mathbf{b}$. We test three different activation functions: softplus, tanh, and identity (i.e., no activation).

We evaluate the performance of the models in terms of NRMSE and NNOIS. The results of these studies are summarized in Figure 4.3, where each row corresponds to a different study. The analyses are assessed on the **Analytic II** function and the **Volcano** dataset. We note that the solid lines and shaded bands in panels (a-d) and (m-p) represent the mean and 95% confidence intervals across repetitions, respectively. The red lines in panels (e-h) and (i-l) indicate the first, second and third quartiles across repetitions, while the blue line indicates the mean.

From Sensitivity Study 1 in Figure 4.3(a-d), we observe that SEEK achieves lower NRMSE and NNOIS compared to the Gaussian kernel in both benchmark problems for all sample sizes, except for $N = 100$ in the **Volcano** dataset. This discrepancy is attributed to the fact that 100 samples are insufficient to capture meaningful patterns about the underlying function, leading to both models clearly underfitting the data. In addition, as the number of samples increases, the performance gap between SEEK and the Gaussian kernel decreases. This behavior is expected due to the interpolation capabilities inherent to GPs. More specifically, from Equations 4.2a and 4.2b, and assuming noiseless samples, we observe that, when making predictions on the training dataset $\mathbf{X}$:

$$\bar{m}(\mathbf{X}) = m(\mathbf{X}) + \cancel{c(\mathbf{X}, \mathbf{X})} \mathbf{C}^{-1} \overset{\mathbf{I}}{\mathbf{y}} - \mathbf{m} = \mathbf{y},$$

⁵Only two hidden layers are used.

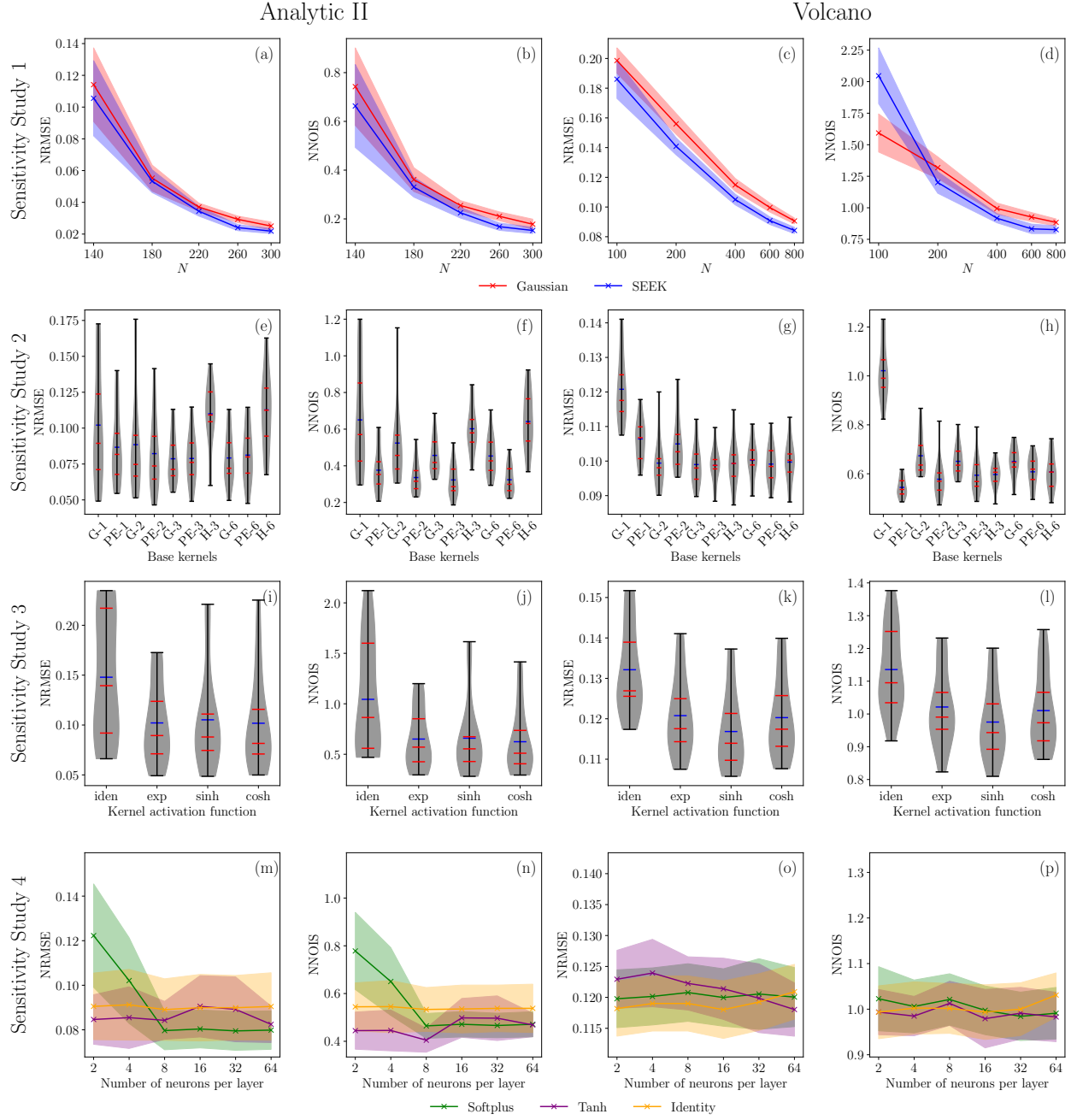


Figure 4.3 Sensitivity studies: We conduct four sensitivity studies on two benchmark problems to evaluate the impact of key design choices on the performance of SEEK. The results are based on 16 repetitions.

$$\bar{c}(\mathbf{X}, \mathbf{X}) = c(\mathbf{X}, \mathbf{X}) - \cancel{c(\mathbf{X}, \mathbf{X})} \mathbf{C}^{-1} \mathbf{I} c(\mathbf{X}, \mathbf{X}) = \mathbf{0}.$$

These results highlight the importance of nonstationary kernels in low-to-mid data regimes,

where they significantly influence the model’s predictions. However, in high data regimes, the inherent interpolation capabilities of GPs dictate the predictions, making the choice of kernel less relevant.

This observation poses an interesting dilemma: while nonstationary kernels are more beneficial in low-to-mid data regimes, they typically rely on a larger number of parameters. This increased flexibility makes them more prone to overfitting compared to stationary kernels, especially when samples are scarce. This trade-off implies that one must constrain the kernel’s complexity (i.e., number of parameters) according to both the input dimensionality and the available dataset size to prevent overfitting.

From Sensitivity Study 2 in Figure 4.3(e-h), we observe that SEEK benefits significantly from an increased number of base kernels in both problems. For example, the model performs noticeably better when using 6 Gaussian kernels (G-6) compared to just 1 (G-1). However, the model does not show substantial improvement from combining different types of kernels when a high number of base kernels is used, as seen in the comparison between models using only Gaussian or power exponential kernels versus the hybrid kernel.

To visualize the above behavior, we present the predictions of a GP with SEEK using 1 and 6 Gaussian kernels as the base kernels, see Figure 4.4. While the model’s performance with a single Gaussian kernel in Figure 4.4(a) is already strong, the use of six Gaussian kernels in Figure 4.4(b) provides enhanced flexibility, allowing the model to learn more complex patterns. This results in a noticeable reduction in overall error and more confident predictions.

From Sensitivity Study 3 in Figure 4.3(i-l), we observe that the model performs better in terms of both NRMSE and NNOIS if $\phi(\cdot)$ has nonlinearity. This is consistent with the discussion in Section 4.2.4, given that the activation function enhances the flexibility of the kernel by introducing additional interactions between the weighted base kernels. Further-

more, we do not observe a significant difference in performance as $\phi(\cdot)$ switches between the exponential, hyperbolic sine, or hyperbolic cosine activation functions.

From Sensitivity Study 4 in Figures 4.3(m-p), we observe that the model performs well even $\mathbf{w}$ and $\mathbf{b}$ are based on the identity activation function, suggesting that a simple linear mapping (instead of an NN) could provide decent accuracy for learning these functions (note that the model does benefit from nonlinear activation functions such as tanh or softplus but this benefit is not substantial in these examples). Once again, the model exhibits strong robustness against overparameterization since its performance remains consistent as the number of neurons is increased.

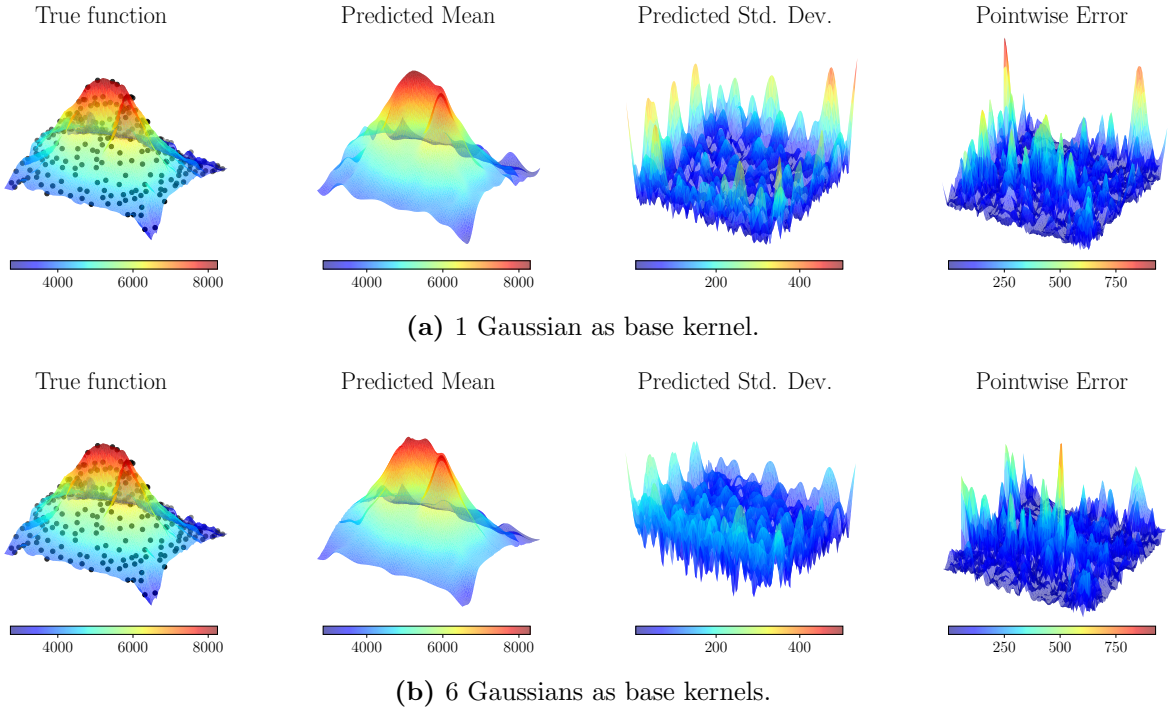


Figure 4.4 Our proposed kernel SEEK with (a) 1 and (b) 6 Gaussian kernels as base kernels on the Volcano dataset: Increasing the number of base kernels reduces error and improves confidence, as reflected in the colorbars (same ranges are used in (a) and (b) to facilitate direct comparison). Black dots represent the training data points.

4.3.3 Comparative Studies

We compare SEEK with one Gaussian kernel as the base kernel against four kernels: Gaussian (Eq. 4.4a), Gibbs [156], deep kernel (DK) [163], and attentive kernel (AK) [154].

To ensure a fair comparison, we use two hidden layers for the NNs used in the nonstationary kernels, and ensure that the number of learnable parameters remains comparable across different approaches. All hidden layers employ the softplus activation function, while all output layers use a linear (identity) activation. More specifically, in SEEK we use a single Gaussian as the base kernel, and use two NNs with two hidden layers each having $2P$ neurons to model $\mathbf{w}(\mathbf{x})$ and $\mathbf{b}(\mathbf{x})$, where P is the input dimensionality. The output layer of $\mathbf{w}$ contains a single neuron, while $\mathbf{b}$ has two output neurons. The NN in the deep kernel consists of two hidden layers with $4P$ neurons each, followed by a linear output layer of dimension P . Similarly, the NN parameterizing the lengthscales in the Gibbs kernel comprises two hidden layers of size $4P$ and a linear output layer with P neurons. Lastly, our implementation of the attentive kernel employs 10 Gaussian kernels with fixed, equally spaced lengthscale parameters. The two NNs (denoted as the z - and w -networks in [154]) used to compute similarity and visibility attention scores have been unified, following the authors' recommendations, and designed so that the number of learnable parameters is on par with the other nonstationary kernels.

The results of the comparative studies are summarized in Figure 4.5. For the **Analytic I** problem, SEEK achieves the best performance in terms of NRMSE and NNOIS across all kernels. This is a remarkable result, especially considering that, as shown in Figures 4.3(e-h), SEEK achieves even better performance if provided with a richer set of base kernels. However, as mentioned earlier, we intentionally restricted the model to a single base kernel to maintain a comparable number of learnable parameters. Other nonstationary kernels, such as the Gibbs and deep kernels, also show noticeable improvements over the Gaussian kernel.

However, they do not demonstrate the same level of robustness across different repetitions compared to SEEK, as reflected by their wider variations in the violin plots.

In the **Analytic II** and **Volcano** problems, a different trend is observed: although SEEK remains the top-performing kernel, the stationary Gaussian kernel performs comparably or even better than the other nonstationary kernels. We attribute this to the higher density of training data in these two problems, which reduces the benefits of modeling nonstationarity by increasing kernel’s flexibility. This aligns with the expected behavior of GPs, as discussed in Section 4.3.2: in higher data regimes, the natural interpolation properties of GPs, combined with the smooth prior induced by the Gaussian kernel, can yield sufficiently strong predictive performance.

In the **Hartmann** problem, we observe that SEEK outperforms the Gaussian, deep and attentive kernels in terms of NRMSE and NNOIS. However, the Gibbs kernel occasionally achieves lower metric values (but also higher ones), as indicated by its wider violin plots.

In our studies, we noticed that more complex nonstationary kernels (with higher number of parameters) generally require more data to outperform the Gaussian kernel in high-dimensional, sparse settings where effectively learning input-dependent variations becomes especially challenging. We attribute this to the fact that, while these nonstationary kernels introduce additional flexibility to capture such local variations, they also incur a higher risk of overfitting when data are limited. In contrast, the Gaussian kernel imposes a globally smooth prior, acting as an effective regularizer. This observation aligns with prior works [168, 185, 186], which combine flexible nonstationary kernels with sparse approximation techniques to allow the GP to leverage large datasets in high dimensions.

Finally, we would like to comment on the attentive kernel [154]. As shown in Figure 4.5, we found that it does not perform well on any of the benchmark problems. While we have carefully validated our implementation against the authors’ provided code in [154] to ensure

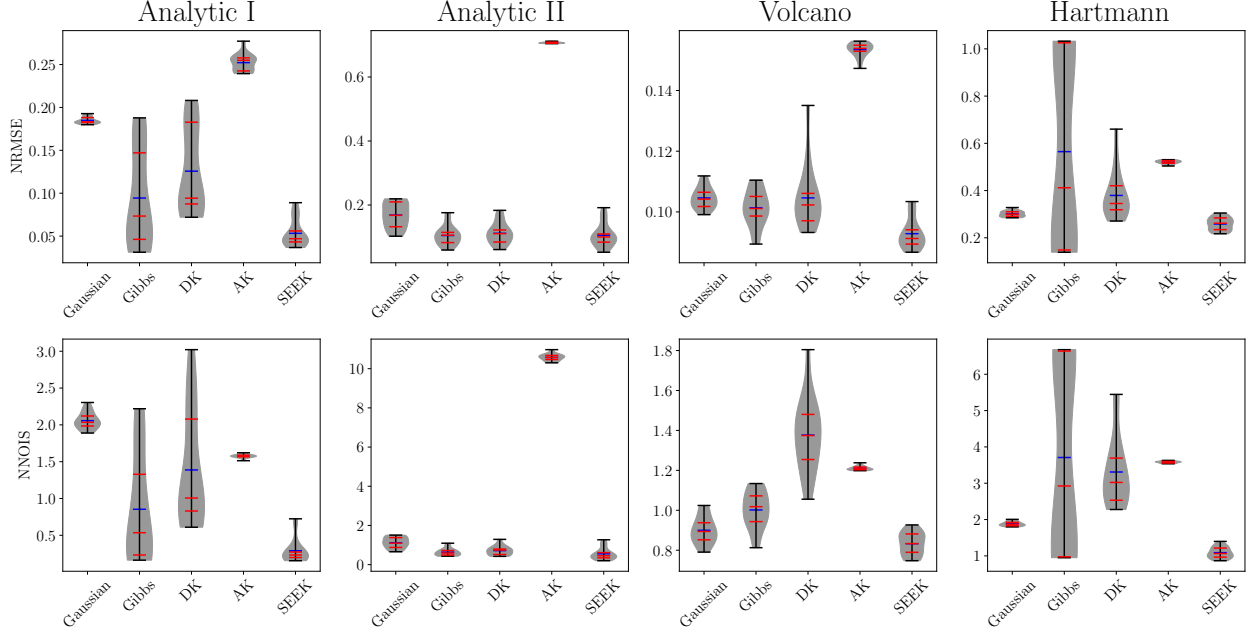


Figure 4.5 Comparative studies: We compare SEEK against other nonstationary kernels on four benchmark problems with varying degrees of nonlinearity and dimensionality.

consistency, the observed performance in our studies may be caused due to variations in experimental settings or hyperparameter choices. In addition, we suspect that a key factor contributing to its inefficiency is that the lengthscale range recommended by the authors may not generalize effectively across problems with varying dimensionality and complexity.

4.3.4 Engineering Problems

In this section, we evaluate SEEK on three challenging, high-dimensional engineering regression problems (described in Section 4.3.1): our in-house ROM dataset, and the popular **Elevators** and **Pumadyn** benchmarks. For these three problems, we compare SEEK against the standard Gaussian kernel. The **Elevators** and **Pumadyn** datasets have been widely tested in the literature, which allows us to also compare SEEK’s performance against regression models other than GPs [187, 188, 189, 190, 191]. These studies aim to show that SEEK provides more accurate mean predictions and improved uncertainty quantification in realistic

engineering datasets.

Similar to Section 4.3.3, we parameterize the weight function $\mathbf{w}(\mathbf{x})$ and bias function $\mathbf{b}(\mathbf{x})$ with two separate feed-forward NNs. Each network comprises two hidden layers, with (4, 2) neurons for **ROM**, (8, 4) for **Elevators**, and (16, 8) for **Pumadyn**. We chose these layer widths to roughly scale with input dimensionality while keeping the total parameter count reasonable and manageable.

Figure 4.6 summarizes the results where we observe that in the **ROM** dataset SEEK achieves roughly a two-times improvement in terms of NRMSE and NNOIS across repetitions. In the **Elevators** dataset, SEEK also consistently outperforms the Gaussian kernel, achieving a marginally lower NRMSE while reducing the NNOIS by over three times. However, we observe a different trend in the **Pumadyn** dataset: the single-Gaussian kernel yields an slightly lower NRMSE, but SEEK still delivers an approximately three-fold improvement in terms of NNOIS. These results highlight, once again, SEEK’s capability in providing more reliable

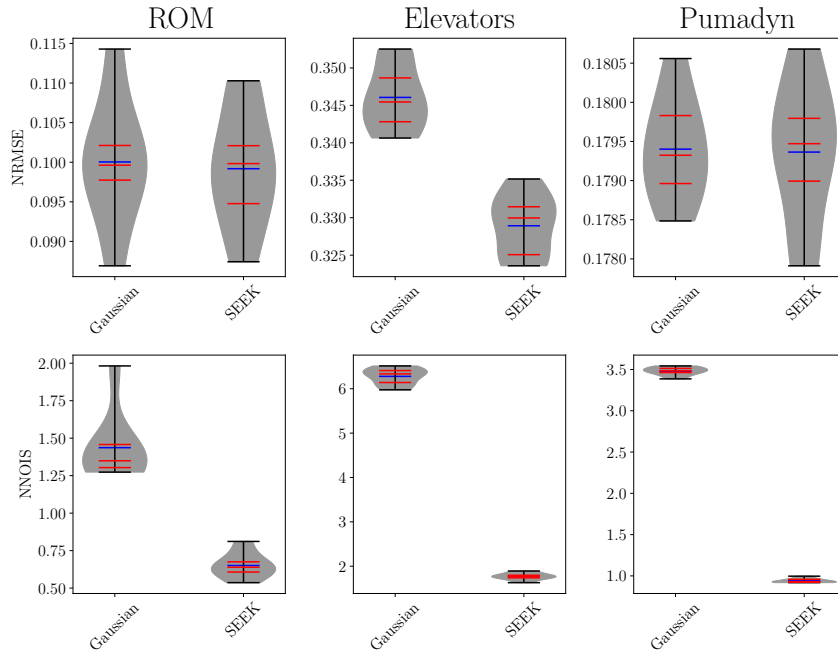


Figure 4.6 Engineering problems: We compare the performance of GP using SEEK versus a standard Gaussian kernel on three high-dimensional engineering problems.

and informative uncertainty estimates, which is particularly valuable in engineering design applications where trustworthy UQ is critical.

Finally, we highlight that across all our experiments we have found that GP models incorporating SEEK tend to converge more consistently and in fewer iterations than those using conventional kernels. In particular, on the **Pumadyn** problem, we observed that the single-Gaussian kernel failed to converge under the L-BFGS optimizer and only produced the results shown in Figure 4.6 when trained with AdamW at an unusually high learning rate of 1.0 to escape local minima. In contrast, GPs with SEEK converged smoothly under the standard optimizer settings employed throughout our studies. This observation holds across a variety of datasets and kernel settings, suggesting that reasonably increased kernel flexibility can not only boost predictive performance but also improve optimization stability and efficiency.

4.3.5 Bayesian Optimization

To further demonstrate the applicability of SEEK in design tasks, we evaluate its performance in the context of BO where the uncertainty estimates of an emulator are used to balance

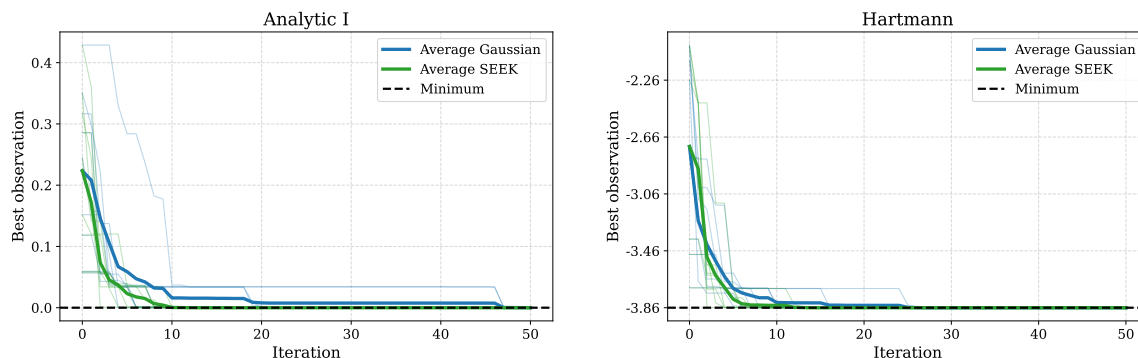


Figure 4.7 Bayesian optimization: We compare the performance of GPs using SEEK or the standard Gaussian kernel on two of the benchmarks.

exploration and exploitation while optimizing expensive black-box functions.

We consider two benchmark problems in this section, namely the **Analytic I** and the three-dimensional version of **Hartmann**, previously introduced in Section 4.3.1. For each problem, we compare SEEK with the Gaussian kernel under identical BO settings. We employ the Log Expected Improvement (LogEI) acquisition function and perform the optimization for each kernel across 8 independent seeds. Each run is initialized with 6 samples and continued for 50 BO iterations (in each iteration, the optimum kernel hyperparameters are obtained by via 32 restarts).

In the case of SEEK, we parameterize the weight function $\mathbf{w}(\mathbf{x})$ and the bias function $\mathbf{b}(\mathbf{x})$ with two separate feed-forward NNs. Each network comprises two hidden layers, with $(1, 1)$ neurons for **Analytic I**, and $(3, 3)$ for **Hartmann**, building kernels with 14 and 76 learnable parameters, respectively. In comparison, the GPs with a Gaussian kernel have 3 and 5 learnable parameters for the two benchmark problems.

Figure 4.7 shows the best objective values obtained across iterations. We observe that SEEK achieves superior convergence, typically reaching near-optimal values within 10–20 iterations, while the Gaussian kernel requires substantially more evaluations. Moreover, SEEK exhibits reduced variability across seeds, demonstrating the robustness of the proposed method.

We note that training costs of SEEK are slightly higher per iteration compared to the Gaussian kernel but this additional cost is justified since SEEK converges faster and hence reduces the overall sampling costs which is a much more important factor than emulator training costs in BO.

4.4 Concluding Remarks

We introduce SEEK, a flexible and interpretable framework for kernel learning in GPs. We believe this method offers a novel and systematic approach for designing nonstationary kernels while guaranteeing kernel validity. Our sensitivity analyses highlight that GPs that use SEEK are robust against overparameterization. In addition, our extensive comparative studies demonstrate that SEEK outperforms other stationary/nonstationary kernels in both prediction accuracy and uncertainty quantification.

A natural direction for extending SEEK is to integrate it with sparse approximation techniques to enhance its scalability to big data applications. In addition, as we discussed in Section 4.2, SEEK is inspired by artificial neurons in NNs. This suggests a promising direction: exploring its performance when stacked layer by layer, similar to how neurons are combined in NN architectures. Another potential extension is to make the activation function itself learnable, which could further boost the model’s performance. Finally, applying SEEK to real-world problems that require reliable uncertainty quantification also presents an exciting avenue for future work.

Chapter 5

Conclusions

This dissertation has developed three novel hybrid frameworks that leverage the complementary strengths of deep learning architectures and Gaussian processes to tackle complex problems in computational science and engineering.

A central challenge in physics-informed machine learning lies in the complex optimization landscape induced by multiterm loss functions derived from PDE systems. While several strategies have been proposed to dynamically balance the contributions of PDE residuals, BCs and ICs, the optimal weighting of these terms remains an open research question. In Chapter 2, we introduced a hybrid framework for solving problems governed by nonlinear PDEs that directly addresses this challenge. This framework exploits the closure properties of GPs to enforce BCs and IC, while leveraging deep NNs in the GP mean function to handle nonlinear constraints. This design eliminates the need for explicit BC/IC loss terms, leading to a simpler and more robust training process. Although in this chapter we enforce only the BCs and ICs through the GP kernel, the approach naturally extends to incorporate additional linear constraints in the kernel. Extensive experiments across a variety of PDE systems demonstrate that the proposed framework consistently outperforms competing methods,

achieves fast convergence in inverse problems, and exhibits strong robustness to optimizer choice and parameter initialization. Its practical utility has further been demonstrated in downstream engineering applications such as topology optimization [76] and compliance minimization [192]. A current limitation of the approach is its inability to enforce constraints in the absence of labeled data, such as periodic BCs. Therefore, devising specialized periodic kernels represents a promising direction to address this limitation, particularly for multiscale simulations where PDEs with periodic BCs frequently arise in fine-scale analyses.

In Chapter 3, we extended the applicability of the previous framework to parametric PDEs by introducing a general GP-based approach for operator learning. The proposed framework effectively approximates mappings between function spaces by reformulating the operator learning problem as an equivalent regression problem. It can optionally leverage neural operators as the GP mean function and incorporate prior physical knowledge in the training process. Through a wide array of numerical benchmarks, we demonstrate the competitiveness of the framework and highlight two key advantages: (1) it enhances the performance of a base neural operator when used as the mean function of the GP, and (2) it enables the construction of zero-shot data-driven models that can make accurate predictions without any prior training. The framework naturally extends to multi-output operators, which frequently arise in PDEs such as the Navier–Stokes equations, and benefits from substantial computational speed-ups through Kronecker product matrix representations of the underlying kernel matrices. However, the current framework relies on the assumption that both input and output functions are discretized consistently at the same locations across all samples. This limits its applicability to problems with fixed topologies, such as structured grids or uniform point clouds. Real-world scenarios can diverge from this setup, as observation points may vary across samples based on the geometry. For instance, predicting flow fields around objects with different shapes requires adapting observation locations—often represented as point clouds—to the unique geometry of each object. We are actively pursuing a promising direction to address this limitation by developing a domain decomposition method with dis-

tributed GPs [150]. This approach splits the computational domain into smaller, manageable subdomains for parallel problem-solving within each subdomain and subsequently combines the results to form a global solution.

A standard choice for GPs—including in the frameworks presented in Chapter 2 and Chapter 3—is to employ stationary kernels such as the Gaussian or Matérn kernel. While simple and widely adopted, these kernels can produce suboptimal predictions and miscalibrated uncertainty estimates in nonstationary settings with limited data. To address this challenge, in Chapter 4 we introduced SEEK, a novel class of learnable kernels designed to model nonstationary functions within the GP paradigm. The proposed kernel draws inspiration from artificial neurons and is constructed from first principles to guarantee kernel validity through symmetry and positive semi-definiteness. SEEK achieves nonstationary modeling by learning input-dependent compositions of base kernels where the weights are parameterized by NNs. Our experiments on both analytic and real-world engineering datasets demonstrate that SEEK consistently outperforms competing kernels in terms of predictive accuracy, uncertainty quantification, and robustness to overparameterization. Looking ahead, several directions should be considered. One is the integration of SEEK with sparse approximation techniques to enhance its scalability to big data applications. Another is the exploration of hierarchical SEEK architectures, i.e., stacking SEEK kernels layer by layer in analogy to deep neural networks, which could further enhance the expressive power of the model for capturing increasingly complex nonstationary phenomena.

Bibliography

- [1] Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew M Stuart. Solving and learning nonlinear pdes with gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.
- [2] Maarten V de Hoop, Daniel Zhengyu Huang, Elizabeth Qian, and Andrew M Stuart. The cost-accuracy trade-off in operator learning with neural networks. *arXiv preprint arXiv:2203.13181*, 2022.
- [3] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.
- [4] Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496:112549, 2024.
- [5] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc Natl Acad Sci U S A*, 113(15):3932–7, 2016.
- [6] Hayden Schaeffer, Russel Caflisch, Cory D Hauck, and Stanley Osher. Sparse dynamics for partial differential equations. *Proceedings of the National Academy of Sciences*, 110(17):6634–6639, 2013.
- [7] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, and M. A. Bessa. Deep learning predicts path-dependent plasticity. *Proc Natl Acad Sci U S A*, 116(52):26414–26420, 2019.
- [8] S. Rahimi-Aghdam, V. T. Chau, H. Lee, H. Nguyen, W. Li, S. Karra, E. Rougier, H. Viswanathan, G. Srinivasan, and Z. P. Bazant. Branching of hydraulic cracks enabling permeability of gas or oil shale with closed natural fractures. *Proc Natl Acad Sci U S A*, 116(5):1532–1537, 2019.
- [9] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proc Natl Acad Sci U S A*, 116(31):15344–15349, 2019.

- [10] S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proc Natl Acad Sci U S A*, 115(39):9684–9689, 2018.
- [11] Marc Santolini and Albert-László Barabási. Predicting perturbation patterns from the topology of biological networks. *Proceedings of the National Academy of Sciences*, 115(27):E6375–E6383, 2018.
- [12] Didier Lucor, Atul Agrawal, and Anne Sergent. Simple computational strategies for more effective physics-informed neural networks modeling of turbulent natural convection. *Journal of Computational Physics*, 456:111022, 2022.
- [13] Qian Fang, Xuankang Mou, and Shibei Li. A physics-informed neural network based on mixed data sampling for solving modified diffusion equations. *Scientific Reports*, 13(1):2491, 2023.
- [14] Ameya D Jagtap, Zhiping Mao, Nikolaus Adams, and George Em Karniadakis. Physics-informed neural networks for inverse problems in supersonic flows. *Journal of Computational Physics*, 466:111402, 2022.
- [15] GP Purja Pun, R Batra, R Ramprasad, and Y Mishin. Physically informed artificial neural networks for atomistic modeling of materials. *Nature communications*, 10(1):2339, 2019.
- [16] Mohammad Lotfollahi, Sergei Rybakov, Karin Hrovatin, Soroosh Hediye-Zadeh, Carlos Talavera-López, Alexander V Misharin, and Fabian J Theis. Biologically informed deep learning to query gene programs in single-cell atlases. *Nature Cell Biology*, 25(2):337–350, 2023.
- [17] Raphaël Pestourie, Youssef Mroueh, Chris Rackauckas, Payel Das, and Steven G. Johnson. Physics-enhanced deep surrogates for partial differential equations. *Nature Machine Intelligence*, 5(12):1458–1465, 2023.
- [18] Daniel J Kozuch, Frank H Stillinger, and Pablo G Debenedetti. Combined molecular dynamics and neural network method for predicting protein antifreeze activity. *Proceedings of the National Academy of Sciences*, 115(52):13252–13257, 2018.
- [19] Lachlan Coin, Alex Bateman, and Richard Durbin. Enhanced protein domain discovery by using language modeling techniques from speech recognition. *Proceedings of the National Academy of Sciences*, 100(8):4516–4520, 2003.
- [20] Stefano Curtarolo, Gus LW Hart, Marco Buongiorno Nardelli, Natalio Mingo, Stefano Sanvito, and Ohad Levy. The high-throughput highway to computational materials design. *Nature materials*, 12(3):191–201, 2013.
- [21] Keith T Butler, Daniel W Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555, 2018.
- [22] Gus LW Hart, Tim Mueller, Cormac Toher, and Stefano Curtarolo. Machine learning for alloys. *Nature Reviews Materials*, 6(8):730–755, 2021.

- [23] Zhe Shi, Evgenii Tsymbalov, Ming Dao, Subra Suresh, Alexander Shapeev, and Ju Li. Deep elastic strain engineering of bandgap through machine learning. *Proceedings of the National Academy of Sciences*, 116(10):4117–4122, 2019.
- [24] W. K. Lee, S. Yu, C. J. Engel, T. Reese, D. Rhee, W. Chen, and T. W. Odom. Concurrent design of quasi-random photonic nanostructures. *Proc Natl Acad Sci U S A*, 114(33):8734–8739, 2017.
- [25] Wing Kam Liu, Miguel A Bessa, Francisco Chinesta, Shaofan Li, and Nathaniel Trask. Special issue of computational mechanics on machine learning theories, modeling, and applications to computational materials science, additive manufacturing, mechanics of materials, design and optimization. *Computational Mechanics*, 72(1):1–2, 2023.
- [26] Timnit Gebru, Jonathan Krause, Yilun Wang, Duyun Chen, Jia Deng, Erez Lieberman Aiden, and Li Fei-Fei. Using deep learning and google street view to estimate the demographic makeup of neighborhoods across the united states. *Proceedings of the National Academy of Sciences*, 114(50):13108–13113, 2017.
- [27] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.
- [28] L. Lu, M. Dao, P. Kumar, U. Ramamurty, G. E. Karniadakis, and S. Suresh. Extraction of mechanical properties of materials through deep learning from instrumented indentation. *Proc Natl Acad Sci U S A*, 117(13):7052–7062, 2020.
- [29] Hengjie Wang, Robert Planas, Aparna Chandramowlishwaran, and Ramin Bostanabad. Mosaic flows: A transferable deep learning framework for solving pdes on unseen domains. *Computer Methods in Applied Mechanics and Engineering*, 389:114424, 2022.
- [30] Ziad Aldirany, Régis Cottureau, Marc Laforest, and Serge Prudhomme. Multi-level neural networks for accurate solutions of boundary-value problems. *Computer Methods in Applied Mechanics and Engineering*, 419:116666, 2024.
- [31] Jakob GR von Saldern, Johann Moritz Reumschüssel, Thomas L Kaiser, Moritz Sieber, and Kilian Oberleithner. Mean flow data assimilation based on physics-informed neural networks. *Physics of Fluids*, 34(11), 2022.
- [32] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
- [33] Badis Djeridane and John Lygeros. Neural approximation of pde solutions: An application to reachability computations. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 3034–3039. IEEE, 2006.
- [34] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

- [35] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
- [36] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
- [37] Levi McClenny and Ulisses Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism. *arXiv preprint arXiv:2009.04544*, 2020.
- [38] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.
- [39] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.
- [40] Jie Bu and Anuj Karpatne. Quadratic residual networks: A new class of neural networks for solving forward and inverse problems in physics involving pdes. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 675–683. SIAM, 2021.
- [41] Han Gao, Luning Sun, and Jian-Xun Wang. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.
- [42] Ameya D Jagtap and George E Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In *AAAI spring symposium: MLPS*, volume 10, 2021.
- [43] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:1–43, 2018.
- [44] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018.
- [45] Remco van der Meer, Cornelis W Oosterlee, and Anastasia Borovykh. Optimally weighted loss functions for solving pdes with neural networks. *Journal of Computational and Applied Mathematics*, 405:113887, 2022.
- [46] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

- [47] Pola Lydia Lagari, Lefteri H Tsoukalas, Salar Safarkhani, and Isaac E Lagaris. Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions. *International Journal on Artificial Intelligence Tools*, 29(05):2050009, 2020.
- [48] Suchuan Dong and Naxian Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435:110242, 2021.
- [49] Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8):1221–1233, 2009.
- [50] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.
- [51] Petr Karnakov, Sergey Litvinov, and Petros Koumoutsakos. Solving inverse problems in physics by optimizing a discrete loss: Fast and accurate learning without neural networks. *PNAS Nexus*, 2024.
- [52] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.
- [53] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- [54] Sancho Salcedo-Sanz, José Luis Rojo-Álvarez, Manel Martínez-Ramón, and Gustavo Camps-Valls. Support vector machines in engineering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):234–267, 2014.
- [55] Houman Owhadi, Clint Scovel, and Florian Schäfer. Statistical numerical approximation. *Notices of the AMS*, 2019.
- [56] Jiahao Zhang, Shiqi Zhang, and Guang Lin. Pagp: A physics-assisted gaussian process framework with active learning for forward and inverse problems of partial differential equations. *arXiv preprint arXiv:2204.02583*, 2022.
- [57] Tomoharu Iwata and Zoubin Ghahramani. Improving output uncertainty estimation and generalization in deep learning via neural network gaussian processes. *arXiv preprint arXiv:1707.05922*, 2017.
- [58] Rui Meng and Xianjin Yang. Sparse gaussian processes for solving nonlinear pdes. *Journal of Computational Physics*, 490:112340, 2023.
- [59] Pau Batlle, Matthieu Darcy, Bamdad Hosseini, and Houman Owhadi. Kernel methods are competitive for operator learning. *Journal of Computational Physics*, 496, 2024.

- [60] Kang Wang, Lei Zhang, and Shaoqiang Tang. Discovery of pdes driven by data with sharp gradient or discontinuity. *Computers & Mathematics with Applications*, 140:33–43, 2023.
- [61] Carl Edward Rasmussen. *Gaussian processes for machine learning*. 2006.
- [62] Amin Yousefpour, Zahra Zanjani Foumani, Mehdi Shishehbor, Carlos Mora, and Ramin Bostanabad. Gp+: A python library for kernel-based learning via gaussian processes. *arXiv preprint arXiv:2312.07694*, 2023.
- [63] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.
- [64] R. Bostanabad, T. Kearney, S. Y. Tao, D. W. Apley, and W. Chen. Leveraging the nugget parameter for efficient gaussian process modeling. *International Journal for Numerical Methods in Engineering*, 114(5):501–516, 2018.
- [65] R. Bostanabad, Y. C. Chan, L. W. Wang, P. Zhu, and W. Chen. Globally approximate gaussian processes for big data with application to data-driven metamaterials design. *Journal of Mechanical Design*, 141(11), 2019.
- [66] N. Oune and R. Bostanabad. Latent map gaussian processes for mixed variable meta-modeling. *Computer Methods in Applied Mechanics and Engineering*, 387:114128, 2021.
- [67] Matthew Plumlee and Daniel W. Apley. Lifted brownian kriging models. *Technometrics*, 59(2):165–177, 2017.
- [68] Liang Ding, Simon Mak, and CF Wu. Bdrygp: a new gaussian process model for incorporating boundary information. *arXiv preprint arXiv:1908.08868*, 2019.
- [69] Liwei Wang, Suraj Yerramilli, Akshay Iyer, Daniel Apley, Ping Zhu, and Wei Chen. Scalable gaussian processes for data-driven design using big data with categorical factors. *Journal of Mechanical Design*, 144(2), 2021.
- [70] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.
- [71] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [72] Taku Ohwada. Cole-hopf transformation as numerical tool for the burgers equation. *Appl. Comput. Math*, 8(1):107–113, 2009.
- [73] COMSOL Multiphysics. Introduction to comsol multiphysics®. *COMSOL Multiphysics, Burlington, MA, accessed Feb, 9(2018):32*, 1998.

- [74] Jasbir Singh Arora. *Introduction to optimum design*. Elsevier, 2004.
- [75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [76] Amin Yousefpour, Shirin Hosseinmardi, Carlos Mora, and Ramin Bostanabad. Simultaneous and meshfree topology optimization with physics-informed gaussian processes. *arXiv preprint arXiv:2408.03490*, 2024.
- [77] Roger G Ghanem and Pol D Spanos. *Stochastic finite elements: a spectral approach*. Courier Corporation, 2003.
- [78] James O Ramsay and Bernard W Silverman. Fitting differential equations to functional data: Principal differential analysis. *Functional data analysis*, pages 327–348, 2005.
- [79] James O Ramsay and Bernard W Silverman. *Applied functional data analysis: methods and case studies*. Springer, 2002.
- [80] David J Lucia, Philip S Beran, and Walter A Silva. Reduced-order modeling: new approaches for computational physics. *Progress in aerospace sciences*, 40(1-2):51–117, 2004.
- [81] Vikramjit Mitra, Yucel Ozbek, Hosung Nam, Xinhui Zhou, and Carol Y Espy-Wilson. From acoustics to vocal tract time functions. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4497–4500. IEEE, 2009.
- [82] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, and Juan J Alonso. Su2: An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2016.
- [83] Nikola Kovachki, Burigede Liu, Xingsheng Sun, Hao Zhou, Kaushik Bhattacharya, Michael Ortiz, and Andrew Stuart. Multiscale modeling of materials: Computing, data science, uncertainty and goal-oriented optimization. *Mechanics of Materials*, 165:104156, 2022.
- [84] Gabriele Boncoraglio and Charbel Farhat. Active manifold and model-order reduction to accelerate multidisciplinary analysis and optimization. *AIAA Journal*, 59(11):4739–4753, 2021.
- [85] Bogdan Raonic, Roberto Molinaro, Tobias Rohner, Siddhartha Mishra, and Emmanuel de Bezenac. Convolutional neural operators. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.
- [86] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

- [87] Nicholas H Nelsen and Andrew M Stuart. Operator learning using random features: A tool for scientific computing. *SIAM Review*, 66(3):535–571, 2024.
- [88] Dongbin Xiu. *Numerical methods for stochastic computations: a spectral method approach*. Princeton university press, 2010.
- [89] Dongbin Xiu and George Em Karniadakis. The wiener–askey polynomial chaos for stochastic differential equations. *SIAM journal on scientific computing*, 24(2):619–644, 2002.
- [90] Olivier Le Maître and Omar M Knio. *Spectral methods for uncertainty quantification: with applications to computational fluid dynamics*. Springer Science & Business Media, 2010.
- [91] Dongbin Xiu and Jie Shen. Efficient stochastic galerkin methods for random diffusion equations. *Journal of Computational Physics*, 228(2):266–281, 2009.
- [92] 2003.
- [93] P. D. Spanos and Roger Ghanem. Stochastic finite element expansion for random media. *Journal of Engineering Mechanics*, 115(5):1035–1053, 1989.
- [94] Houman Owhadi and Clint Scovel. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design*, volume 35. Cambridge University Press, 2019.
- [95] Houman Owhadi. Bayesian numerical homogenization. *Multiscale Modeling & Simulation*, 13(3):812–828, 2015.
- [96] Florian Schäfer and Houman Owhadi. Sparse recovery of elliptic solvers from matrix-vector products. *SIAM Journal on Scientific Computing*, 46(2):A998–A1025, 2024.
- [97] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhat-tacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.
- [98] Nikola B Kovachki, Samuel Lanthaler, and Andrew M Stuart. Operator learning: Algorithms and analysis. *arXiv preprint arXiv:2402.15715*, 2024.
- [99] Nicolas Boullé and Alex Townsend. A mathematical guide to operator learning. *arXiv preprint arXiv:2312.14688*, 2023.
- [100] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhat-tacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.
- [101] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans Neural Netw*, 6(4):911–7, 1995.

- [102] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.
- [103] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deepnets. *Science advances*, 7(40):eabi8605, 2021.
- [104] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *arXiv preprint arXiv:2006.09535*, 2020.
- [105] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.
- [106] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- [107] Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.
- [108] 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. *ACM/JMS Journal of Data Science*, 2021.
- [109] 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. *Advances in Neural Information Processing Systems*, 36, 2024.
- [110] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.
- [111] Wolfgang Hackbusch. *Multi-grid methods and applications*, volume 4. Springer Science & Business Media, 2013.
- [112] Juncal He and Jinchao Xu. Mgnet: A unified framework of multigrid and convolutional neural network. *Science china mathematics*, 62:1331–1354, 2019.
- [113] Juncal He, Xinliang Liu, and Jinchao Xu. Mgno: Efficient parameterization of linear operators via multigrid. *arXiv preprint arXiv:2310.19809*, 2023.

- [114] James O Ramsay and CJ1125714 Dalzell. Some tools for functional data analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 53(3):539–561, 1991.
- [115] Charles A Micchelli and Massimiliano Pontil. On learning vector-valued functions. *Neural computation*, 17(1):177–204, 2005.
- [116] Leo Breiman and Jerome H Friedman. Predicting multivariate responses in multiple linear regression. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 59(1):3–54, 1997.
- [117] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, 2004.
- [118] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine learning*, 73:243–272, 2008.
- [119] Mauricio A Alvarez, Lorenzo Rosasco, Neil D Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.
- [120] Houman Owhadi. Do ideas have shape? idea registration as the continuous limit of artificial neural networks. *Physica D: Nonlinear Phenomena*, 444:133592, 2023.
- [121] Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew M Stuart. Solving and learning nonlinear pdes with gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.
- [122] Houman Owhadi. Computational graph completion. *Research in the Mathematical Sciences*, 9(2):27, 2022.
- [123] Matthew Lowery, John Turnage, Zachary Morrow, John D Jakeman, Akil Narayan, Shandian Zhe, and Varun Shankar. Kernel neural operators (knos) for scalable, memory-efficient, geometrically-flexible operator learning. *arXiv preprint arXiv:2407.00809*, 2024.
- [124] Kurt Cutajar, Mark Pullin, Andreas Damianou, Neil Lawrence, and Javier González. Deep gaussian processes for multi-fidelity modeling. *arXiv preprint arXiv:1903.07320*, 2019.
- [125] José Betancourt, François Bachoc, Thierry Klein, Déborah Idier, Rodrigo Pedreros, and Jérémy Rohmer. Gaussian process metamodeling of functional-input code for coastal flood hazard assessment. *Reliability Engineering & System Safety*, 198:106870, 2020.
- [126] Samuel Stanton, Wesley Maddox, Ian Delbridge, and Andrew Gordon Wilson. Kernel interpolation for scalable online gaussian processes, 2021.

- [127] Jonathan Tammer Eweis-Labolle, Nicholas Oune, and Ramin Bostanabad. Data fusion with latent map gaussian processes. *Journal of Mechanical Design*, 144(9), 2022.
- [128] Zahra Zanjani Foumani, Mehdi Shishehbor, Amin Yousefpour, and Ramin Bostanabad. Multi-fidelity cost-aware bayesian optimization. *Computer Methods in Applied Mechanics and Engineering*, 407:115937, 2023.
- [129] Xianjin Yang and Houman Owhadi. A mini-batch method for solving nonlinear pdes with gaussian processes. *arXiv preprint arXiv:2306.00307*, 2023.
- [130] Yifan Chen, Houman Owhadi, and Florian Schäfer. Sparse cholesky factorization for solving nonlinear pdes via gaussian processes. *Mathematics of Computation*, 2024.
- [131] Carlos Mora, Amin Yousefpour, Shirin Hosseinmardi, and Ramin Bostanabad. Neural networks with kernel-weighted corrective residuals for solving partial differential equations. *arXiv preprint arXiv:2401.03492*, 2024.
- [132] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.
- [133] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
- [134] Edwin V Bonilla, Kian Chai, and Christopher Williams. Multi-task gaussian process prediction. *Advances in neural information processing systems*, 20, 2007.
- [135] Stefano Conti and Anthony O’Hagan. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640–651, 2010. 531at Times Cited:195 Cited References Count:30.
- [136] Mehdi Shishehbor, Shirin Hosseinmardi, and Ramin Bostanabad. Parametric encoding with attention and convolution mitigate spectral bias of neural partial differential equation solvers. *Structural and Multidisciplinary Optimization*, 67(7):128, 2024.
- [137] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.
- [138] Tomoharu Iwata and Zoubin Ghahramani. Improving output uncertainty estimation and generalization in deep learning via neural network gaussian processes. *arXiv preprint arXiv:1707.05922*, 2017.
- [139] R. Planas, N. Oune, and R. Bostanabad. Evolutionary gaussian processes. *Journal of Mechanical Design*, 143(11):111703, 2021.
- [140] Hengjie Wang, Robert Planas, Aparna Chandramowliswaran, and Ramin Bostanabad. Mosaic flows: A transferable deep learning framework for solving pdes on unseen domains. *Computer Methods in Applied Mechanics and Engineering*, 389:114424, 2022.

- [141] H. Jasak and Z. Tukovic. Automatic mesh motion for the unstructured finite volume method. *Transactions of FAMENA*, 30(2):1 – 18, 2007. Cited by: 2.
- [142] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deepnets. *Science advances*, 7(40):eabi8605, 2021.
- [143] 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. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024.
- [144] Christopher KI Williams, Carl Edward Rasmussen, A Scwaighofer, and Volker Tresp. Observations on the nyström method for gaussian process prediction. 2002.
- [145] Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*.
- [146] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264.
- [147] 2009.
- [148] Edward Snelson and Zoubin Ghahramani. Variable noise and dimensionality reduction for sparse gaussian processes. *arXiv preprint arXiv:1206.6873*, 2012.
- [149] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.
- [150] Marc Deisenroth and Jun Wei Ng. Distributed gaussian processes. In *International conference on machine learning*, pages 1481–1490. PMLR, 2015.
- [151] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.
- [152] Karl Ezra Pilario, Mahmood Shafiee, Yi Cao, Liyun Lao, and Shuang-Hua Yang. A review of kernel methods for feature extraction in nonlinear process monitoring. *Processes*, 8(1):24, 2019.
- [153] Marcus M Noack and Kristofer G Reyes. Mathematical nuances of gaussian process-driven autonomous experimentation. *MRS Bulletin*, 48(2):153–163, 2023.
- [154] Weizhe Chen, Roni Khardon, and Lantao Liu. Adaptive robotic information gathering via non-stationary gaussian processes. *The International Journal of Robotics Research*, 43(4):405–436, 2024.
- [155] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

- [156] Mark N Gibbs. *Bayesian Gaussian processes for regression and classification*. PhD thesis, Citeseer, 1998.
- [157] Christopher Paciorek and Mark Schervish. Nonstationary covariance functions for gaussian process regression. *Advances in neural information processing systems*, 16, 2003.
- [158] Shan Ba and V Roshan Joseph. Composite gaussian process models for emulating expensive functions. *The Annals of Applied Statistics*, pages 1838–1860, 2012.
- [159] Sebastian W Ober, Carl E Rasmussen, and Mark van der Wilk. The promises and pitfalls of deep kernel learning. In *Uncertainty in Artificial Intelligence*, pages 1206–1216. PMLR, 2021.
- [160] Marcus M Noack, Hengrui Luo, and Mark D Risser. A unifying perspective on non-stationary kernels for deeper gaussian processes. *APL Machine Learning*, 2(1), 2024.
- [161] Joost Van Amersfoort, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal. On feature collapse and deep kernel learning for single forward pass uncertainty. *arXiv preprint arXiv:2102.11409*, 2021.
- [162] Markus Heinonen, Henrik Mannerström, Juho Rousu, Samuel Kaski, and Harri Lähdesmäki. Non-stationary gaussian process regression with hamiltonian monte carlo. In *Artificial Intelligence and Statistics*, pages 732–740. PMLR, 2016.
- [163] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.
- [164] Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for bayesian optimization of non-stationary functions. In *International conference on machine learning*, pages 1674–1682. PMLR, 2014.
- [165] Anthony Tompkins, Rafael Oliveira, and Fabio T Ramos. Sparse spectrum warped input measures for nonstationary kernel learning. *Advances in Neural Information Processing Systems*, 33:16153–16164, 2020.
- [166] Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193, 2012.
- [167] Christian Plagemann, Kristian Kersting, and Wolfram Burgard. Nonstationary gaussian process regression using point estimates of local smoothness. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 204–219. Springer, 2008.
- [168] Sami Remes, Markus Heinonen, and Samuel Kaski. Neural non-stationary spectral kernel. *arXiv preprint arXiv:1811.10978*, 2018.

- [169] Ying Xiong, Wei Chen, Daniel Apley, and Xuru Ding. A non-stationary covariance-based kriging method for metamodelling in engineering design. *International Journal for Numerical Methods in Engineering*, 71(6):733–756, 2007.
- [170] Marc G Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.
- [171] Marc G Genton and Olivier Perrin. On a time deformation reducing nonstationary stochastic processes to local stationarity. *Journal of Applied Probability*, 41(1):236–249, 2004.
- [172] Paul D Sampson and Peter Guttorp. Nonparametric estimation of nonstationary spatial covariance structure. *Journal of the American Statistical Association*, 87(417):108–119, 1992.
- [173] Carl Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems*, 14, 2001.
- [174] Martin Trapp, Robert Peharz, Franz Pernkopf, and Carl Edward Rasmussen. Deep structured mixtures of gaussian processes. In *International conference on artificial intelligence and statistics*, pages 2251–2261. PMLR, 2020.
- [175] Edward Meeds and Simon Osindero. An alternative infinite mixture of gaussian process experts. *Advances in neural information processing systems*, 18, 2005.
- [176] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.
- [177] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.
- [178] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.
- [179] Amin Yousefpour, Zahra Zanjani Foumani, Mehdi Shishehbor, Carlos Mora, and Ramin Bostanabad. Gp+: a python library for kernel-based learning via gaussian processes. *Advances in Engineering Software*, 195:103686, 2024.
- [180] Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and multidisciplinary optimization*, 48:607–626, 2013.
- [181] Shiguang Deng, Carlos Mora, Diran Apelian, and Ramin Bostanabad. Data-driven calibration of multifidelity multiscale fracture models via latent map gaussian process. *Journal of Mechanical Design*, 145(1):011705, 2023.
- [182] Rui Camacho. Elevators dataset. University of Porto Regression Repository. Experiments of Rui Camacho (rcamacho@garfield.fe.up.pt).

- [183] DELVE repository. Pumadyn32h dataset. DELVE repository, University of Toronto. Publicly available at DELVE repository.
- [184] Laurence Charles Ward Dixon. The global optimization problem: an introduction. *Towards Global Optimiation 2*, pages 1–15, 1978.
- [185] Mark D Risser and Daniel Turek. Bayesian inference for high-dimensional non-stationary gaussian processes. *Journal of Statistical Computation and Simulation*, 90(16):2902–2928, 2020.
- [186] Marcus M Noack, Harinarayan Krishnan, Mark D Risser, and Kristofer G Reyes. Exact gaussian processes for massive datasets via non-stationary sparsity-discovering kernels. *Scientific reports*, 13(1):3155, 2023.
- [187] Thiago JM Moura, George DC Cavalcanti, and Luiz S Oliveira. Mine: A framework for dynamic regressor selection. *Information Sciences*, 543:157–179, 2021.
- [188] Tianxun Zhou and Keng-Hwee Chiam. Synthetic data generation method for data-free knowledge distillation in regression neural networks. *Expert Systems with Applications*, 227:120327, 2023.
- [189] Fabio Sigrist. Ktboost: Combined kernel and tree boosting. *Neural Processing Letters*, 53(2):1147–1160, 2021.
- [190] Daniel Waxman and Petar M Djurić. Dynamic online ensembles of basis expansions. *arXiv preprint arXiv:2405.01365*, 2024.
- [191] Nathan Lay and Adrian Barbu. The artificial regression market. *arXiv preprint arXiv:1204.4154*, 2012.
- [192] Xiangyu Sun, Amin Yousefpour, Shirin Hosseinmardi, and Ramin Bostanabad. Compliance minimization via physics-informed gaussian processes. *arXiv preprint arXiv:2507.09968*, 2025.
- [193] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [194] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
- [195] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.
- [196] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [197] Finbarr O’sullivan, Brian S Yandell, and William J Raynor Jr. Automatic smoothing of regression functions in generalized linear models. *Journal of the American Statistical Association*, 81(393):96–103, 1986.

- [198] George Kimeldorf and Grace Wahba. Some results on tchebycheffian spline functions. *Journal of mathematical analysis and applications*, 33(1):82–95, 1971.
- [199] Richard Szeliski. Regularization uses fractal priors. In *Proceedings of the sixth National conference on Artificial intelligence-Volume 2*, pages 749–754, 1987.

Appendix A

Solving Forward and Inverse Problems Involving PDEs

A.1 Properties of a Gaussian Process Surrogate

We use an analytic one-dimensional ($1D$) function to demonstrate some of the most important characteristics of GP surrogates. Specifically, we leverage a set of examples to argue that GPs: (1) have interpretable parameters, (2) can regress or interpolate highly nonlinear functions, (3) suffer from reversion to the mean phenomena in data scarce regions, (4) can have ill-conditioned covariance matrices if their mean function interpolates the data, and (5) with manually chosen hyperparameters can faithfully surrogate a function if sufficient training samples are available. These properties underpin our decision for manually selecting the kernel parameters in module one of our framework. They also demonstrate the effects of a GP’s mean function on its prediction power and numerical stability.

As demonstrated in Figure A.1 our experiments involve sampling from a sinusoidal function where we study the effects of frequency, noise, data distribution, function differentiability,

adopted prior mean function, and hyperparameter optimization on the behavior of GPs. For all of these studies we endow the GP with the following parametric kernel:

$$c(x, x'; \sigma^2, \phi, \delta) = \sigma^2 \exp\{-\phi(x - x')^2\} + \mathbb{1}\{x == x'\}\delta, \quad (\text{A.1-1})$$

where $\boldsymbol{\lambda} = [\sigma^2, \phi, \delta]^T$ are the kernel parameters. In this equation, σ^2 is the process variance which, looking at Equation (2.1) in the main text, does not affect the posterior mean and hence we simply set it to 1 in our framework (this feature of our framework is in sharp contrast to other methods such as [1] whose performance is quite sensitive to the selected kernel parameters). The rest of the parameters in Equation (A.1-1) are defined as follows. $\phi = 10^\omega$ where ω is the length-scale or roughness parameter that controls the correlation strength along the x -axis, $\mathbb{1}\{\cdot\}$ returns 1/0 if the enclosed statement is true/false, and δ is the so-called nugget or jitter parameter that is added to the kernel for modeling noise and/or improving the numerical stability of the covariance matrix. We quantify the numerical stability of the covariance matrix via its condition number or κ .

Given some training data, $\boldsymbol{\lambda}$ can be quickly estimated via maximum likelihood estimation (MLE). We denote parameter estimates obtained via this process by appending the subscript MLE to them, i.e., $\hat{\boldsymbol{\lambda}}_{MLE}$. Alternatively, we can manually assign specific values to $\boldsymbol{\lambda}$.

We first study the effect of noise by training two GPs where both GPs aim to emulate the same underlying function but one has access to noise-free responses while the other is trained on noisy data, see Figure A.1 (a) and Figure A.1 (b), respectively. We observe in Figure A.1 (a) that the estimated value for $\hat{\delta}_{MLE}$ is very small since the data is noise-free (the small value is added to reduce κ) while in Figure A.1 (b) the estimated nugget parameter is much larger and close to the noise variance (2.48e−3 vs. 2.50e−3). Additionally, comparing Figure A.1 (a) and Figure A.1 (c) we observe a direct relation between the frequency of the underlying function and the estimated kernel parameters. In particular, the magnitude

of $\hat{\omega}_{MLE}$ increases as $u(x)$ becomes rougher since the correlation between two points on it quickly dies out as the distance between those points increases (for this reason, ω is also sometimes called the roughness parameter). Further increasing the frequency of $u(x)$ to the extent that it resembles a noise signal directly increases $\hat{\omega}_{MLE}$. These points indicate that the kernel parameters of a GP are interpretable.

We next study the reversion to the mean behavior and numerical instabilities of GPs in Figure A.1 (d) and Figure A.1 (e). In both of these scenarios the training data is only available close to the boundaries. However, we set the prior mean of the GP in Figure A.1 (d) and Figure A.1 (e) to zero and $m(x; \theta) = \theta \times \sin(2\pi x)$, respectively. The reversion to the mean behavior is clearly observed in Figure A.1 (d) where the expected value of the posterior distribution is almost zero in the $(-0.5, 0.5)$ range where the correlations with the training data die out. The reversion to the mean behavior is also seen in Figure A.1 (e) but this time it is not undesirable since the functional form of the chosen parametric mean function is similar to $u(x)$ (note that a large neural network can also reproduce the training data but such a network cannot match $u(x)$ in interior regions where there are no labeled data). This similarity forces the kernel to regress residuals that are mostly zero (i.e., the kernel must regress a constant value in the entire domain). Since any two points on a constant function have maximum correlation, regressing such residuals requires $\phi \rightarrow 0$ which, in turn, renders the covariance matrix ill-conditioned to the extent that $\kappa \rightarrow +\infty$. Based on these observations, in our framework we do not estimate the kernel parameters jointly with the weights and biases of the deep neural network (NN).

Lastly, in Figure A.1 (e) we demonstrate that GPs can interpolate non-differentiable functions as long as they are provided with sufficient training data. The power and efficiency of GPs in learning from data is quite robust to the hyperparameters. As shown in Figure A.1 (g) through Figure A.1 (i) GPs with manually selected ω can accurately surrogate $u(x)$ regardless of its frequency (the nugget value in these three cases is chosen such that κ does

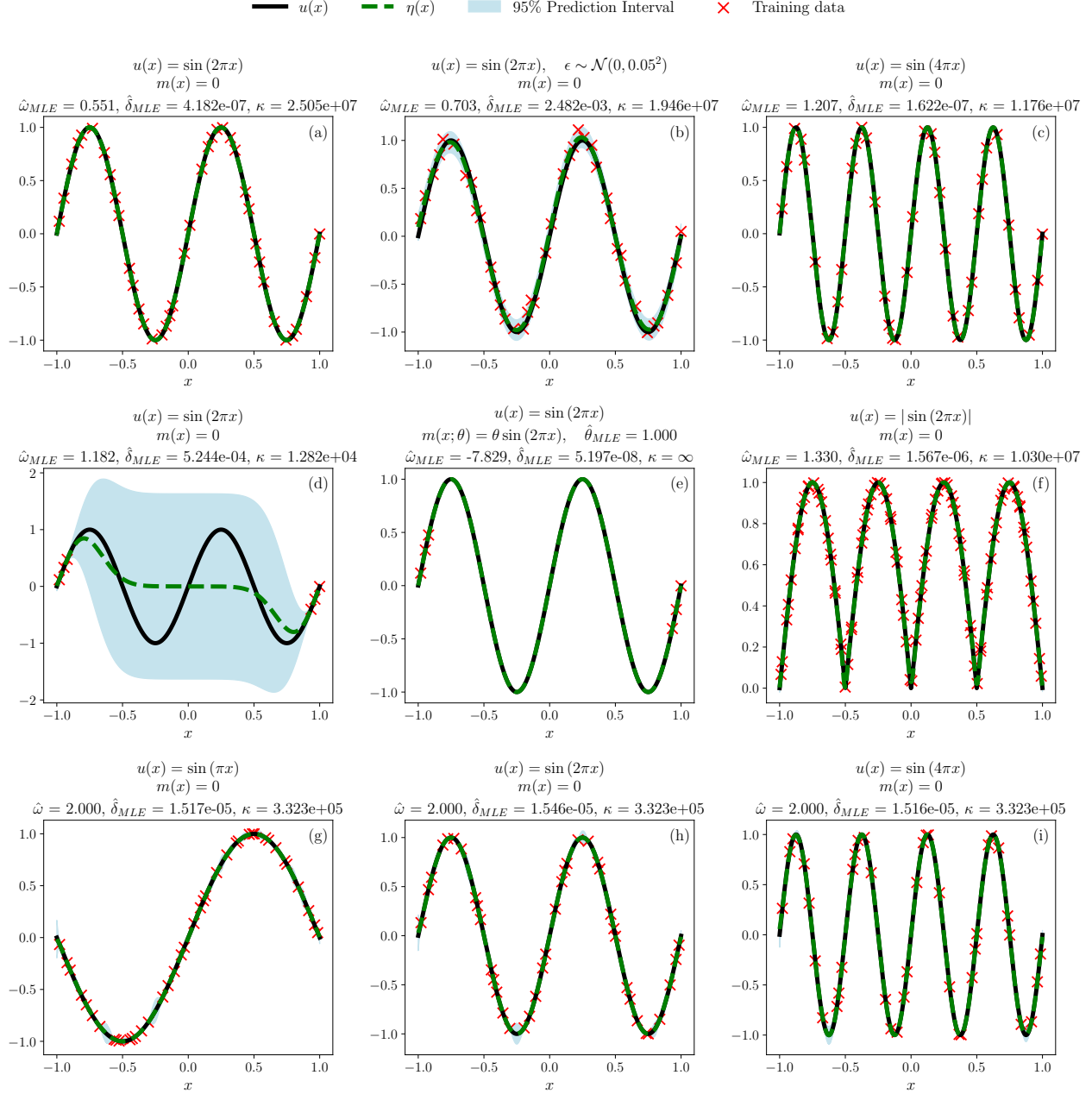


Figure A.1 Properties of Gaussian processes: We demonstrate that GPs have interpretable hyperparameters and can regress a wide range of functions. The dependency of regression quality on the hyperparameters rapidly decreases as the size of the training data increases.

not exceed a predetermined value). This attractive behavior forms the basis of our choice to manually fix ϕ in the first module of our framework. It is highlighted that the manual parameter selection results in sub-optimal prediction intervals but this issue does not affect our framework since we do not leverage these intervals.

A.2 Methods Description

Below, we briefly introduce the four PIML models that we have used in our comparative studies. To be able to directly compare the implementation of the four PIML models, we use Burgers' equation in the following descriptions. The PDE system is:

$$u_t + uu_x - \nu u_{xx} = 0, \quad \forall x \in [-1, 1], t \in (0, 1] \quad (\text{A.2-2a})$$

$$u(-1, t) = u(1, t) = 0, \quad \forall t \in [0, 1] \quad (\text{A.2-2b})$$

$$u(x, 0) = -\sin(\pi x), \quad \forall x \in [-1, 1] \quad (\text{A.2-2c})$$

where $\mathbf{x} = [x, t]$ are the independent variables, u is the PDE solution, and ν is a constant that denotes the kinematic viscosity. Also, we denote the output of the NN models via $m(\mathbf{x}; \boldsymbol{\theta})$ throughout this section. Note that we also employ $m(\mathbf{x}; \boldsymbol{\theta})$ for denoting the NN in the mean function of NN-CoRes.

A.2.1 Physics-Informed Neural Networks (PINNs)

As schematically shown in Figure A.2, the essential idea of PINNs is to parameterize the relation between u and $\mathbf{x}$ with a deep NN [35], i.e., $u(\mathbf{x}) = m(\mathbf{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ are the network's weights and biases. The parameters of m are optimized by iteratively minimizing a loss function, denoted by $\mathcal{L}(\boldsymbol{\theta})$, that encourages the network to satisfy the PDE system in Equation (A.2-2c). To calculate $\mathcal{L}(\boldsymbol{\theta})$, we first obtain the network's output at n_{BC} points on the $x = -1$ and $x = 1$ boundaries, n_{IC} points on the $t = 0$ boundary which marks the initial condition, and n_{PDE} collocation points (CPs) inside the domain, see Figure A.2b. For the $n_{BC} + n_{IC}$ points on the boundaries, we can directly compare the network's outputs to the specified boundary and initial conditions in Equations (A.2-2b) and (A.2-2c). For each of the n_{PDE} CPs, we evaluate the partial derivatives of the output and calculate the residual

in Equation (A.2-2a). Once these three terms are calculated, we obtain $\mathcal{L}(\boldsymbol{\theta})$ by summing them up as follows:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \mathcal{L}_{PDE}(\boldsymbol{\theta}) + \mathcal{L}_{BC}(\boldsymbol{\theta}) + \mathcal{L}_{IC}(\boldsymbol{\theta}) \\ &= \frac{1}{n_{PDE}} \sum_{i=1}^{n_{PDE}} (m_t(\mathbf{x}_i; \boldsymbol{\theta}) + m(\mathbf{x}_i; \boldsymbol{\theta})m_x(\mathbf{x}_i; \boldsymbol{\theta}) - \nu m_{xx}(\mathbf{x}_i; \boldsymbol{\theta}))^2 + \\ &\quad \frac{1}{n_{BC}} \sum_{i=1}^{n_{BC}} (m(\mathbf{x}_i; \boldsymbol{\theta}) - 0)^2 + \frac{1}{n_{IC}} \sum_{i=1}^{n_{IC}} (m(\mathbf{x}_i; \boldsymbol{\theta}) + \sin(\pi x_i))^2\end{aligned}\tag{A.2-3}$$

The loss function in Equation (A.2-3) is typically minimized via either the Adam [193] or L-BFGS [194] methods which are both gradient-based optimization algorithms. With either Adam or L-BFGS, the parameters of the network are first initialized and then iteratively updated to minimize $\mathcal{L}(\boldsymbol{\theta})$. These updates rely on partial derivatives of $\mathcal{L}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ which can be efficiently obtained via automatic differentiation [43].

While Adam and L-BFGS are both gradient-based optimization techniques, they have some major differences [195]. Adam is a first-order method while L-BFGS is not since it is a quasi-Newton optimization algorithm. Compared to Adam, L-BFGS is more memory-intensive and has a higher per-epoch computational cost since it uses an approximation of the Hessian matrix during the optimization. Moreover, Adam scales to large datasets better than L-BFGS which does not accommodate mini-batch training. However, L-BFGS typically provides lower loss values and requires fewer epochs for convergence compared to Adam.

A.2.2 PINNs with Dynamic Loss Weights

One of the challenges associated with minimizing the loss function in Equation (A.2-3) is that the three terms on the right-hand side disproportionately contribute to $\mathcal{L}(\boldsymbol{\theta})$. To mitigate this issue, a popular approach is to scale each loss component independently before summing

them up, that is:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{PDE}(\boldsymbol{\theta}) + w_{BC}\mathcal{L}_{BC}(\boldsymbol{\theta}) + w_{IC}\mathcal{L}_{IC}(\boldsymbol{\theta}). \quad (\text{A.2-4})$$

Since the scale of the three loss terms can change dramatically during the optimization process, these weights must be dynamic, i.e., their magnitude must be adjusted during the training. In our experiments, we follow the process described in [39] for dynamic loss balancing and highlight that this approach is only applicable to cases where Adam is used.

A.2.3 PINNs with Hard Constraints

An alternative approach to dynamic weight balancing is to eliminate $\mathcal{L}_{BC}(\boldsymbol{\theta})$ and $\mathcal{L}_{IC}(\boldsymbol{\theta})$ from Equation (A.2-3) by requiring the model's output to satisfy the boundary and initial conditions by construction [50]. To this end, we now denote the output of the network by

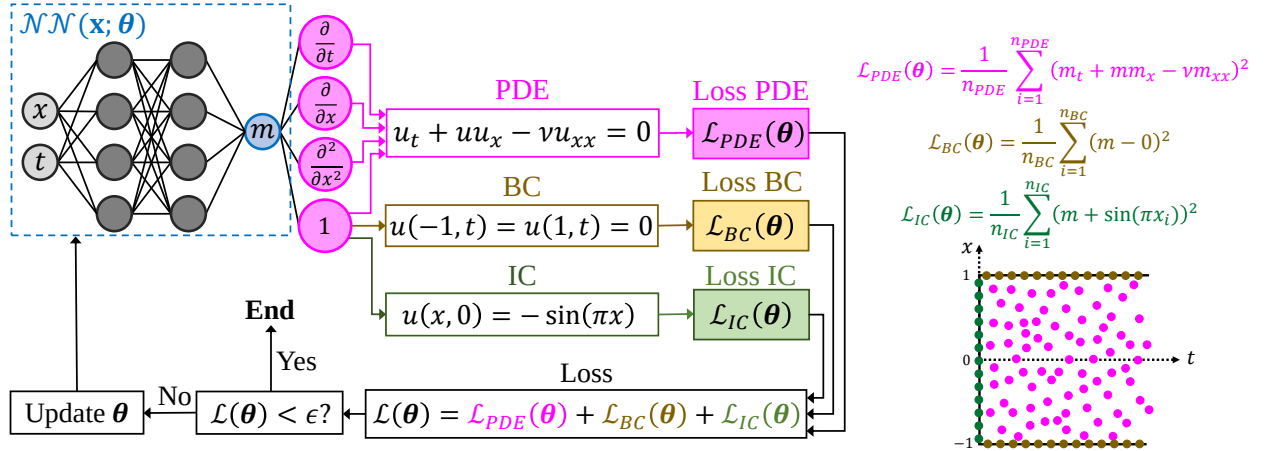


Figure A.2 Physics-informed neural network (PINN): The model parameters, $\boldsymbol{\theta}$, are optimized by minimizing the three-component loss function that encourages the network to satisfy the PDE inside the domain while reproducing the initial and boundary conditions. These loss components are obtained by querying the network on a set of points that are distributed inside the domain or on its boundaries.

$\tilde{m}(\mathbf{x}; \boldsymbol{\theta})$ and then formulate the final output of the model as:

$$m(\mathbf{x}; \boldsymbol{\theta}) = a(\mathbf{x})\tilde{m}(\mathbf{x}; \boldsymbol{\theta}) + b(\mathbf{x}), \quad (\text{A.2-5})$$

where $a(\mathbf{x})$ and $b(\mathbf{x})$ are analytic functions that ensure $m(\mathbf{x}; \boldsymbol{\theta})$ satisfies Equations (A.2-2b) and (A.2-2c) regardless of what $\tilde{m}(\mathbf{x}; \boldsymbol{\theta})$ produces at $\mathbf{x}$. A common strategy is to choose $a(\mathbf{x})$ to be the signed distance function that vanishes on the boundaries and produces finite values inside the domain. The construction of $b(\mathbf{x})$ is application-specific since one has to formulate a function that satisfies the applied boundary and initial conditions while generating finite values inside the domain. For the PDE system in Equation (A.2-2c), one option is $b(\mathbf{x}) = \frac{-2 \sin \pi x}{1+e^{-t}}$.

A.2.4 Optimal Recovery

This recent approach leverages zero-mean GPs for solving nonlinear PDEs [1]. Specifically, let us denote the kernel of a zero-mean GP via $c(\cdot, \cdot)$. We associate $c(\cdot, \cdot)$ with the reproducing kernel Hilbert space (RKHS) $\mathcal{U}$ where the RKHS norm is defined as $\|u\|$. Following these definitions, we can approximate $u(\mathbf{x})$ by finding the minimizer of the following optimal recovery problem:

$$\begin{aligned} & \underset{u \in \mathcal{U}}{\text{minimize}} \quad \|u\| \\ & \text{subject to} \\ & \quad u_t(\mathbf{x}_i) + u(\mathbf{x}_i)u_x(\mathbf{x}_i) - \nu u_{xx}(\mathbf{x}_i) = 0, \quad \forall i = 1, \dots, n_{PDE} \\ & \quad u(\mathbf{x}_i) = 0, \quad \forall i = 1, \dots, n_{BC}, \\ & \quad u(\mathbf{x}_i) = -\sin(\pi x_i), \quad \forall i = 1, \dots, n_{IC}, \end{aligned} \quad (\text{A.2-6})$$

where n_{PDE} , n_{BC} , n_{IC} are the number of nodes inside the domain, on the $x = -1$ and $x = 1$ lines where the boundary conditions are specified, and on the $t = 0$ line where the initial condition is specified, respectively. We denote the collection of these $n_{PDE} + n_{BC} + n_{IC}$ points via $\mathbf{X}$.

The optimization problem in Equation (A.2-6) is infinite-dimensional and hence [1] leverage the representer theorem to convert it into a finite-dimensional one by defining the slack variable $\mathbf{z} = [z^{(1)}, z^{(2)}, z^{(3)}, z^{(4)}]$:

$$\begin{aligned}
& \underset{\mathbf{z} \in \mathbb{R}^N}{\text{minimize}} \quad \mathbf{z}^T \mathbf{\Theta}^{-1} \mathbf{z} \\
& \text{subject to} \\
& \quad z_i^{(2)} + z_i^{(1)} z_i^{(3)} - \nu z_i^{(4)} = 0, \quad \forall i = 1, \dots, n_{PDE} \\
& \quad z_i^{(1)} = 0, \quad \forall i = 1, \dots, n_{BC}, \\
& \quad z_i^{(1)} = -\sin(\pi x_i), \quad \forall i = 1, \dots, n_{IC},
\end{aligned} \tag{A.2-7}$$

where $N = 4(n_{PDE} + n_{BC} + n_{IC}) + 3n_{PDE}$ and $\mathbf{\Theta}$ is the covariance matrix (see Section 3.4.1 of [1] for details on $\mathbf{\Theta}$). Equation (A.2-7) can be reduced to an unconstrained optimization problem by eliminating the equality constraints following the process described in Subsection 3.3.1 of [1]. Once $\mathbf{z}$ is estimated, the PDE solution can be estimated at the arbitrary point $\mathbf{x}$ in the domain via GP regression.

We note that the process of defining the slack variables and obtaining the equivalent finite-dimensional optimization problem needs to be repeated for different PDE systems (e.g., in a PDE system one may have to define some of the slack variables as the Laplacian of the solution rather than the solution itself). Also, per the recommendations in [1], $c(\cdot, \cdot)$ is set to an anisotropic kernel and its parameters are chosen manually (i.e., they do not need to be jointly estimated with $\mathbf{z}$) but, unlike our approach, this choice must be done carefully since it affects the results. In our comparative studies, we use the values reported in [1] for the

kernel parameters.

A.3 Neural Networks with Kernel-Weighted Corrective Residuals Reproduce the Data

Proof. We prove that the error of our model in reproducing the boundary data converges to zero as we increase the number of sampled boundary data. For the sake of completeness, we begin by a definition and invoking two theorems and then proceed with our proof.

Definition A.1 (Reproducing kernel Hilbert space). *Let $\mathcal{H}$ be a Hilbert space of real functions u defined on an index set $\mathcal{X}$. Then, $\mathcal{H}$ is called a Reproducing kernel Hilbert space (RKHS) with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ if the function $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ with the following properties exists:*

- *For any $\mathbf{x}$, $c(\mathbf{x}, \mathbf{x}')$ as a function of $\mathbf{x}'$ is in $\mathcal{H}$,*
- *c has the reproducing property, that is $\langle u(\mathbf{x}'), c(\mathbf{x}', \mathbf{x}) \rangle_{\mathcal{H}} = u(\mathbf{x})$.*

Note that the norm of u is $\|u\|_{\mathcal{H}} = \sqrt{\langle u, u \rangle_{\mathcal{H}}}$ and that $\langle c(\mathbf{x}, \cdot), c(\mathbf{x}', \cdot) \rangle_{\mathcal{H}} = c(\mathbf{x}, \mathbf{x}')$ since both $c(\mathbf{x}', \cdot)$ and $c(\mathbf{x}, \cdot)$ are in $\mathcal{H}$.

Theorem A.1 (Mercer's Theorem). *The eigenfunctions of the real positive semidefinite kernel $c(\mathbf{x}, \mathbf{x}')$ whose eigenfunction expansion with respect to measure π is $c(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^N \alpha_i \psi_i(\mathbf{x}) \psi_i(\mathbf{x}')$, are orthonormal. That is:*

$$\int \psi_i(\mathbf{x}) \psi_j(\mathbf{x}) d\pi = \delta_{ij} \tag{A.3-8}$$

where δ_{ij} denotes the Kronecker delta function. Following this theorem, we note that for a Hilbert space defined by the linear combinations of the eigenfunctions, that is $u(\mathbf{x}) =$

$\sum_{i=1}^N u_i \psi_i(\mathbf{x})$ with $\sum_{i=1}^N u_i / \alpha_i < \infty$, we have $\|u\|_{\mathcal{H}}^2 = \langle u, u \rangle_{\mathcal{H}} = \sum_{i=1}^N u_i / \alpha_i$.

Theorem A.2 (Representer Theorem). *Each minimizer $u(\mathbf{x}) \in \mathcal{H}$ of the following functional can be represented as $u(\mathbf{x}) = \sum_{i=1}^n \alpha_i c(\mathbf{x}, \mathbf{x}_i)$:*

$$F[u(\mathbf{x})] = \frac{\beta}{2} \|u(\mathbf{x})\|_{\mathcal{H}}^2 + P(\mathbf{h}, \mathbf{u}). \quad (\text{A.3-9})$$

where $\mathbf{h} = [h_1, \dots, h_n]^T$ is the observation vector, $u(\mathbf{x})$ denotes the function that we aim to fit to $\mathbf{h}$, $\mathbf{u} = [u(\mathbf{x}_1), \dots, u(\mathbf{x}_n)]^T = [u_1, \dots, u_n]^T$ are the evaluations of $u(\mathbf{x})$ at configurations where $\mathbf{h}$ are observed, β is a scaling constant that balances the contributions of the two terms on the right hand side (RHS) to $F[u(\mathbf{x})]$, and $P(\cdot, \cdot)$ is a function that evaluates the quality of $u(\mathbf{x})$ in reproducing $\mathbf{h}$. For proof of this theorem, see [196, 197, 198].

In our case, to prove that our model can reproduce the boundary data we first assume that the initial and boundary conditions are sufficiently smooth functions and that the neural network (i.e., the mean function of the GP) produces finite values on the boundaries. These assumptions simplify the proof by allowing us to work with the difference of these two terms.

We now consider a specific form of Equation (A.3-9):

$$F[u(\mathbf{x})] = \frac{1}{2} \|u(\mathbf{x})\|_{\mathcal{H}}^2 + \frac{\lambda^2}{2} \sum_{i=1}^n (h_i - u(\mathbf{x}_i))^2, \quad (\text{A.3-10})$$

where $u(\mathbf{x})$ is the zero-mean GP predictor and $\|u(\mathbf{x})\|_{\mathcal{H}}$ is the RKHS norm with kernel $c(\cdot, \cdot)$. The second term on the right hand side corresponds to the negative log-likelihood of a Gaussian noise model with precision λ^2 and hence the minimizer of Equation (A.3-10) is the posterior mean of the GP [199]. Hence, we now need to show that as $n \rightarrow \infty$ the minimizer of Equation (A.3-10), which is our GP, can reproduce the data $\mathbf{h}$. We denote the ground truth function that we aim to discover and the variance around it by, respectively, $h(\mathbf{x})$ and $\tau^2(\mathbf{x}) = \int (h - h(\mathbf{x}))^2 d\pi(h|\mathbf{x})$ where $\pi(\mathbf{x}, h)$ is the probability measure that generates the

data $(\mathbf{x}_i, h_i)$.

We rewrite the second term on the right hand side of Equation (A.3-10) as:

$$\begin{aligned} \mathbb{E} \left[\sum_{i=1}^n (h_i - u(\mathbf{x}_i))^2 \right] &= n \int (h - u(\mathbf{x}))^2 d\pi(\mathbf{x}, h) = \\ n \int (h - h(\mathbf{x}) + h(\mathbf{x}) - u(\mathbf{x}))^2 d\pi(\mathbf{x}, h) &= \\ n \int \tau^2(\mathbf{x}) d\pi(\mathbf{x}) + 0 + n \int (h(\mathbf{x}) - u(\mathbf{x}))^2 d\pi(\mathbf{x}). \end{aligned} \quad (\text{A.3-11})$$

where the zero on the last line is due to the definition of $h(\mathbf{x})$, i.e., $h(\mathbf{x}) = \mathbb{E}[h|\mathbf{x}]$. Since $\tau^2(\mathbf{x})$ is independent of $u(\mathbf{x})$, we can use Equation (A.3-11) to rewrite Equation (A.3-10) as:

$$F_\pi[u(\mathbf{x})] = \frac{1}{2} \|u(\mathbf{x})\|_{\mathcal{H}}^2 + \frac{n\lambda^2}{2} \int (h(\mathbf{x}) - u(\mathbf{x}))^2 d\pi(\mathbf{x}). \quad (\text{A.3-12})$$

We now invoke Mercer's theorem to write $u(\mathbf{x}) = \sum_{i=1}^{\infty} u_i \psi_i(\mathbf{x})$ and $h(\mathbf{x}) = \sum_{i=1}^{\infty} h_i \psi_i(\mathbf{x})$ where ψ_i are the eigenfunctions of the nondegenerate kernel of the GP. Since $\{\psi_i\}$ form an orthonormal basis, we can write:

$$F_\pi[u(\mathbf{x})] = \frac{1}{2} \sum_{i=1}^{\infty} \frac{u_i^2}{\alpha_i} + \frac{n\lambda^2}{2} \sum_{i=1}^{\infty} (h_i - u_i)^2. \quad (\text{A.3-13})$$

We take the derivative of Equation (A.3-13) with respect to u_i and set it to zero to obtain:

$$u_i = \frac{\alpha_i h_i}{\alpha_i + 1/n\lambda^2}. \quad (\text{A.3-14})$$

Since $1/n\lambda^2 \rightarrow 0$ as $n \rightarrow \infty$, in the limit $u_i \rightarrow h_i$, i.e., our zero-mean GP predictor corrects for the error that $m(\mathbf{x}, \boldsymbol{\theta})$ has on reproducing the initial and boundary conditions. Note that the convergence in Equation (A.3-14) does not depend on $\tau^2(\mathbf{x})$ and hence holds for the case where the observation vector $\mathbf{h}$ is noisy. $\square$

A.4 Additional Experiments

In the following subsections, we summarize the findings from additional experiments in forward problems. First, we provide further details on the results obtained for the LDC problem. Next, we present the results obtained using NN-CoRes for the 2D Helmholtz equation. Finally, we compare and discuss the computational cost of our method with that of PINN.

A.4.1 Lid-Driven Cavity Problem

The solution of the LDC problem consists of three dependent variables which are the pressure $p(\mathbf{x})$ and the two velocity components in the x and y directions, $u(\mathbf{x})$ and $v(\mathbf{x})$, respectively. In the main text we report the mean of the Euclidean norm of the error on the three outputs (see Table 2.1 in the main text). In Table A.1 we provide the errors for the individual outputs of this benchmark problem and observe the same trend where NN-CoRes consistently outperforms other methods. We also notice that all the models predict pressure with less accuracy compared to the velocity components. This trend is due to the facts that not only the scale of $p(x, y)$ is smaller than the velocity components, but also $p(x, y)$ is known at a single point on the boundaries whereas $u(x, y)$ and $v(x, y)$ are known everywhere on the boundaries.

Capacity Problem		NN-CoRes		GP _{OR}		PINN		PINN _{DL}		PINN _{HC}	
		4 × 10	4 × 20	1,000	2,000	4 × 10	4 × 20	4 × 10	4 × 20	4 × 10	4 × 20
LDC (A = 3)	u	1.92e−1	8.56e−2	—	—	2.66e−1	1.23e−1	2.99e−1	1.26e−1	3.99e−1	4.97e−1
	v	1.74e−1	8.25e−2	—	—	2.78e−1	1.28e−1	3.07e−1	1.23e−1	3.06e−1	3.95e−1
	p	1.91e−1	9.19e−2	—	—	2.72e−1	1.33e−1	2.98e−1	1.25e−1	5.92e−1	6.96e−1
LDC (A = 5)	u	2.49e−1	2.22e−1	—	—	6.01e−1	5.67e−1	5.97e−1	5.18e−1	1.02e0	7.57e−1
	v	2.51e−1	2.20e−1	—	—	6.32e−1	5.92e−1	6.29e−1	5.43e−1	7.04e−1	5.64e−1
	p	4.33e−1	3.94e−1	—	—	9.17e−1	8.72e−1	9.23e−1	8.09e−1	1.36e0	1.41e0

Table A.1 Summary of comparative studies for the LDC problem: We report $L_{2,e}$ of different methods as a function of model capacity and A . The symbol $\otimes$ indicates the network architecture (e.g., $4 \otimes 10$ is an NN which has four 10– neuron hidden layers). Unlike NN-based methods, GP_{OR}’s accuracy relies on the number of interior nodes which we set to 1,000 or 2,000. GP_{OR} is not applied to LDC as it relies on manual derivation of the equivalent variational problem which, unlike the first three PDEs, is not done by the developers [1].

A.4.2 2D Helmholtz Equation

In Figure A.3 we solve a canonical PDE system known as Helmholtz [39] which is defined as:

$$\begin{aligned}
u_{xx}(x, y) + u_{yy}(x, y) + u(x, y) &= q(x, y), \quad \forall x, y \in (-1, 1)^2 \\
u(x, -1) = u(x, 1) &= 0, \quad \forall x \in [-1, 1] \\
u(-1, y) = u(1, y) &= 0, \quad \forall y \in [-1, 1]
\end{aligned} \tag{A.4-15}$$

In Equation (A.4-15), $q(x, y)$ is constructed such that the analytic solution is $u(x, y) = \sin(a_1\pi x) \sin(a_2\pi y)$ where a_1 and a_2 are two constants that control the frequency along the x and y directions, respectively. The Helmholtz equation is a well-studied benchmark problem since PINNs fail to accurately solve it. To address this shortcoming, recent works have introduced quite complex architectures which typically leverage adaptive loss functions. We test our framework on this benchmark problem by setting $a_1 = 1$ and $a_2 = 4$ while using the same architecture and training procedure that are used in our comparative studies. As shown in Figure A.3 our predictions accurately capture both the high- and low-frequency features of the solution. We note that the solution in Figure A.3 is 5 times more accurate than the one reported in [39] which employs a considerably larger architecture ($4 \otimes 50$) and leverages the adaptive loss function described in Equation (A.2-4).

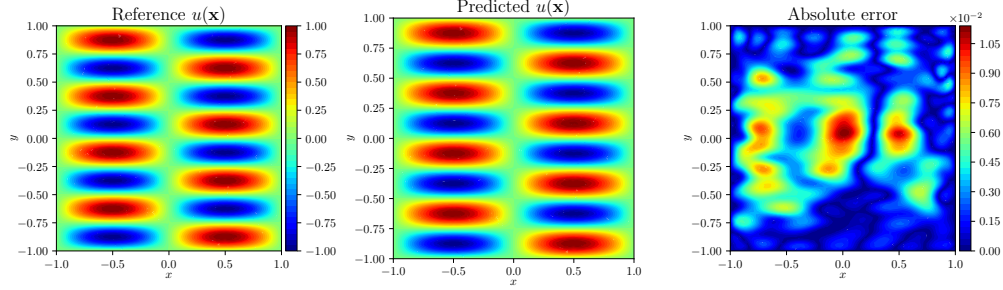


Figure A.3 Reference solutions, predicted solution and error map on the Helmholtz equation: Our approach with a $4 \otimes 20$ architecture provides much lower errors compared to other methods and automatically adapts to high- and low-frequency solutions.

A.4.3 Computational Cost

To provide a benchmark for the computational cost of NN-CoRes, we report in Table A.2 its training time per epoch compared to PINN (using the same architecture and optimizer settings). As it can be observed, the time per epoch of NN-CoRes is slightly higher compared to PINN across different problems. We attribute this behavior to the fact that our method needs to evaluate the kernels in addition to the NN-based mean function for making predictions. While this slightly increases the inference costs, it offers the advantage of automatically satisfying the BCs/IC and facilitating the training of the NN by only requiring one term in the loss function (see Figure 2.1). This leads to faster convergence and higher accuracies across all problems, as demonstrated in Figure 2.9 and 2.10.

Model \ Problem	Problem			
	Burgers' ($\nu = \frac{0.01}{\pi}$)	Elliptic ($\alpha = 30$)	Eikonal ($\epsilon = 0.01$)	LDC ($A = 5$)
NN-CoRes ($4 \otimes 20$)	0.91	1.18	1.11	1.94
PINN ($4 \otimes 20$)	0.70	0.84	0.83	1.24

Table A.2 Training time in seconds per epoch for NN-CoRes and PINN across different problems with L-BFGS: NN-CoRes takes slightly more time per epoch across all problems compared to the baseline PINN. However, the curves shown in Figure 2.9 and 2.10 show that NN-CoRes requires fewer epochs to converge than PINN while consistently achieving higher accuracies. The symbol $\otimes$ indicates the network architecture (e.g., $4 \otimes 20$ is an NN which has four 20-neuron hidden layers).

Appendix B

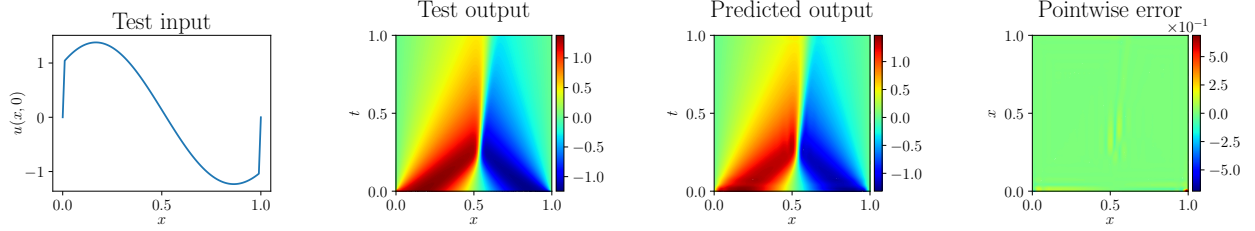
Operator Learning in Parametric PDEs

B.1 1D Burgers' Equation with Large Gradients

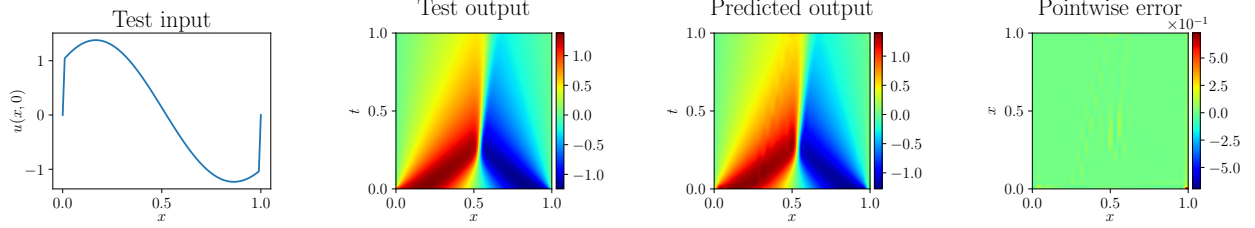
To evaluate the performance of the proposed method when the solution maps exhibit large gradients, we consider the 1D Burgers' problem with zero Dirichlet BCs and $\zeta = 0.01$:

$$\begin{aligned} v_t + vv_x - \zeta v_{xx} &= 0, \quad \forall x \in (0, 1), \quad t \in (0, 1] \\ v &= 0, \quad \forall x = \{0, 1\}, \quad t \in [0, 1] \\ v &= u, \quad \forall x \in [0, 1], \quad t = 0. \end{aligned} \tag{B.1-1}$$

We use $N = 800$ samples and set $p = 100$ and $q = 34 \times 34$. We employ both zero- and DeepONet-mean GPs in our framework which achieve $L2$ relative test errors of 2.84% and 3.03%, respectively. Figure B.1 shows the results obtained for a random test sample. As illustrated, both the zero- and DeepONet-mean GP are able to accurately learn the sharp gradients in the solution map.



(a) Our GP framework with zero-mean is able to accurately capture the large gradients in the solution.



(b) Our GP framework with DeepONet-mean is able to accurately capture the large gradients in the solution.

Figure B.1 Example test input, output, predictions and pointwise error: We provide the results obtained with our zero-mean and DeepONet-mean GP for one test example for the 1D Burgers' equation with zero Dirichlet BCs and $\zeta = 0.01$. The L_2 test relative error across 200 test samples for each model is 2.84% and 3.03% respectively.

B.2 Fixing versus Optimizing β_y

In Section 3.2.3, we propose a training strategy that involves fixing β_y to a large value, e.g., 10^3 , based on the assumption that output functions are typically richly sampled for operator learning purposes. This approach enables the GP to effectively capture correlations in Ω_v without any training, while ensuring the numerical stability of $\mathbf{C}_y$ and eliminating the need to invert it in every optimization iteration. However, dense observations may not always be available, especially when data is collected from physical experiments rather than simulations. Fortunately, our GP framework is flexible enough to accommodate these situations by optimizing β_y instead of fixing it.

To demonstrate the effect of fixing β_y versus optimizing it, we conduct an experiment on the Burgers' equation with Dirichlet BCs, i.e., Equation (3.5). We report the test relative L_2 error of our zero-mean GP as a function of the output function resolution q under two

scenarios: (1) fixing β_y at 10^3 and following the proposed strategy, and (2) optimizing β_y alongside the other parameters. For this study, we set $N = 800$ and $p = 100$. The results of these studies are summarized in Figure B.2. We observe that fixing β_y achieves comparable or even better performance as the output function resolution increases. In contrast, when the output function is sampled at coarser resolutions, optimizing β_y yields better results since it allows the GP to more accurately capture correlations within Ω_v . Therefore, practitioners always have the option of optimizing β_y if they suspect the observation density is low. Additionally, given the low computational cost of our approach, one can experiment with both strategies and choose the one that performs best.

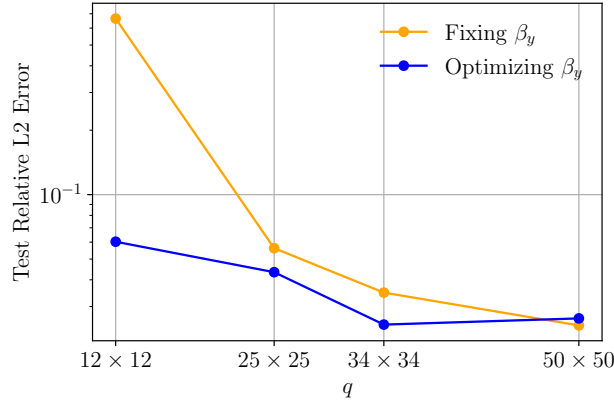


Figure B.2 Optimizing versus fixing β_y in training with our zero-mean GP for the Burgers' equation with Dirichlet BCs: Fixing β_y achieves comparable or slightly better performance as the output function is observed at finer resolutions. Optimizing β_y is the better choice when the output function is not observed at a fine enough resolution.

Appendix C

Learning Nonstationary Kernels

C.1 Implementation And Optimization Details

All experiments in Sections 4.3.2 and 4.3.3 have been implemented using the Python package GP+ [179]. For the optimization of the models, we used the L-BFGS optimizer from PyTorch with a fixed learning rate of 0.01. To reduce the risk of converging to suboptimal solutions, we used the strategy of rerunning the optimization of models with different initializations of the learnable parameters. This is a common and well-known strategy, especially for training models like GPs. Although it is not necessary to use this many reruns, to give each model enough opportunity to show its best performance and reduce the effect of the initial values of learnable parameters on the performance of the models, we used 80 reruns.

In practice, we don't need to reinitialize these many times to get decent results. Yet, it should be mentioned that it makes more sense to increase the number of initializations as the problem's dimensionality increases, since the optimization space typically grows with it.

Each model was trained for a maximum of 2,000 epochs and we applied early stopping if the

computed loss failed to improve for 20 consecutive epochs. All experiments were conducted on a machine equipped with an Intel Core i9-14900KF CPU and 64 GB of RAM.