UNIVERSITY OF CALIFORNIA,
IRVINE

Transferable Emulation of Expensive Computer Simulations

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Mechanical and Aerospace Engineering

by

Robert Planas

Thesis Committee:
Professor Ramin Bostanabad, Chair
Professor Athanasios Sideris
Professor Aparna Chandramowlishwaran

2021

DEDICATION

To my parents and my brother.

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vii**

ACKNOWLEDGMENTS **viii**

ABSTRACT OF THE TRANSFERABLE EMULATION OF EXPENSIVE COMPUTER SIMULATIONS **ix**

1 Introduction **1**

2 Evolutionary Gaussian Processes **4**
2.1 Technical Background 8
2.1.1 Emulation with Gaussian Processes 8
2.1.2 Evolutionary Programming 11
2.2 Evolutionary Gaussian Processes 13
2.2.1 Convergence Criterion 16
2.2.2 Historical Pareto Frontier 16
2.2.3 Global Coefficient of Variation 17
2.2.4 Discussions 18
2.2.5 Selection of the EGP Parameters 19
2.3 Results 20
2.3.1 Analytical Functions 22
2.3.2 Extrapolation for Material modeling 27
2.3.3 Symbolic Regression with EGP 29
2.4 Conclusions 30

3 Transferable Deep Learning Framework for solving PDEs in Arbitrary Domains **33**
3.1 Related Work 37
3.2 Transferable Learning of PDEs 39
3.2.1 BVP and IBVP Decomposition: Why Mosaic Works? 41
3.2.2 Genomic Flow Network (GFNet) 45
3.2.3 Mosaic Flow Predictor 47
3.3 Solving PDEs with MF predictor 50

3.3.1 Laplace Equation 50
3.3.2 Navier-Stokes Equations 61
3.3.3 Wave equation 76
3.4 Conclusions 82

4 Concluding remarks **85**

Bibliography **87**

LIST OF FIGURES

Page

2.1 (a) Extrapolation errors of various emulators, (b) GP vs. EGP 5
2.2 Simplified flowchart of our approach 7
2.3 Symbolic regression via evolutionary programming 12
2.4 Detailed flow chart of EGP 14
2.5 Example of historical Pareto frontier 17
2.6 Performance of EP, GP, and EGP (small noise scenario) 23
2.7 Performance of EP, GP, and EGP (large noise scenario) 23
2.8 Numerical stability of EGP vs. ordinary GP 25
2.9 Materials modeling with GP, EGP, and EP 26
2.10 Over confident extrapolations with EGP 31

3.1 Transferable learning of a PDE 35
3.2 Decomposition of a BVP 42
3.3 Decomposition of a time dependent PDEs 43
3.4 Primary steps of our framework for transferable PDE learning 48
3.5 Distribution of training data 51
3.6 Different architectures studied for the Laplace Equation 52
3.7 Accuracy and performance of MF predictorand PINN for the Laplace equation with two different BCs 58
3.8 Arrangement of 10 additional auxiliary genomes in the square cavity: In addition to the 9 genomes (g_{1-9}) arranged as shown in Figure 3.4b, we add 10 more auxiliary genomes (g_{10-19}) to improve MF predictor's accuracy in resolving the unseen flow. 68
3.9 Velocity magnitudes $\sqrt{u^2+v^2}$ for flow in a square lid-driven cavity with unseen BC. 68
3.10 Velocity magnitudes $\sqrt{u^2+v^2}$ for flow in a square lid-driven cavity with unseen BC 70
3.11 Velocity magnitude $\sqrt{u^2+v^2}$ and streamlines for the flow in a step-shaped lid-driven cavity and the long lid driven cavity 71
3.12 Streamlines for the predicted solutions with MF predictor, adaptive MF predictor, and ensemble MF predictor 74
3.13 Sample of the training data for the wave equation 77
3.14 FC-GRU/FC encoder architecture 80
3.15 MAE evolution with respect to the predicted time for the MF predictor . . . 81

3.16 MF predictor and the wave equation 83

LIST OF TABLES

Page

2.1 Analytical examples 21
2.2 Noise variance in analytical examples 22
2.3 Performance of GP, EGP, and EP in the engineering example 28
2.4 Regressed terms with EGP for the analytical examples 30

3.1 Accuracy of different GFNets on learning the Laplacian operator 56
3.2 XPINN vs. MF predictor with UNC 60
3.3 Data sets for the NS equations 62
3.4 GFNet's accuracy in learning the NS equations with dataset ID 1 64
3.5 66
3.6 Iterative MF predictor, adaptive MF predictor, and MF predictor ensemble applied in different cavities 69
3.7 Training accuracy of the top 5 more representative specifically trained GFNets 73

ACKNOWLEDGMENTS

First and foremost, I thank Professor Ramin Bostanabad for taking me on as his MS student. His insightful guidance and mentorship has helped me during all my master's program to challenge my limits and accomplish my goals. I really value how his experience and dedication have made me grow and develop both techinical and non-technical skills that I will treasure for the rest of my professional career.

Second, I would like to give special thanks to the Balsells fellowship program for making this unique experience possible.

Third, I would like to thank Professor Aparna Chandramowlishwaran and Dr. Henjie Wang for their guidance and support during the past year.

I would also like to thank Professor Chandramowlishwaran again and thank Professor Athanasios Sideris for being part of the committee members for this thesis.

Last but not least, I am thankful for my family, friends and lab mates. They supported me during difficult times and have cheered with me during my successes.

To all of them, thank you very much.

ABSTRACT OF THE THESIS

Transferable Emulation of Expensive Computer Simulations

By

Robert Planas

Master of Science in Mechanical and Aerospace Engineering

University of California, Irvine, 2021

Professor Ramin Bostanabad, Chair

Emulation or surrogate modeling is an indispensable part of many scientific and engineering disciplines. However, most emulators such as Gaussian processes (GPs) or deep neural networks (DNNs) are exclusively developed for interpolation/regression and have limited generalization power. In this thesis we present two approaches for improved and scalable emulation of physical systems.

In the context of finding governing dynamics for a physical system, we introduce evolutionary Gaussian processes (EGPs). EGPs are a systematic integration of GPs and evolutionary programming (EP) that boost the performance of GPs by the automatic discovery of free-form symbolic bases that regress the data. As we demonstrate via examples that include a host of analytical functions as well as an engineering problem on materials modeling, EGP can improve the performance of ordinary GPs in terms of not only extrapolation, but also interpolation/regression and numerical stability.

In the context of surrogating solvers for PDEs, we introduce a transferable framework for solving intial conditions and boundary value problems (IBVPs) via DNNs which can be trained once and used forever for various domains of unseen sizes, shapes, and boundary conditions. We present the *genomic flow network* (GFNet), a neural network that can infer the solution of an IBVP subject to arbitrary initial and boundary conditions on a

square domain called *genome*. Furthermore, we propose *mosaic flow* (MF) predictor, a novel iterative algorithm that assembles the GFNet's inferences for IBVPs on large domains with unseen sizes and shapes while preserving the spatial regularity of the solution.

Chapter 1

Introduction

Emulation or surrogate modeling has become an indispensable part of many scientific and engineering disciplines. Over the past few decades, many emulation methods have been developed. Some prominent examples include deep neural networks (DNNs) [1, 2, 3, 4, 1, 5, 6, 7, 8, 9, 10, 11, 12], Gaussian processes (GPs [13, 14, 15, 16], which are closely related to Kriging models [17, 18, 19, 20]), radial basis functions (RBF) [21], support vector regressors (SVR) [22], boosted trees [23], random forests [24], and evolutionary programming (EP) [25, 26, 27]. Among these methods, GPs and DNNs have played a pivotal role in engineering design. GPs standout as they are easy to train and interpret, can reasonably quantify prediction uncertainty [28, 29], have tractable conditional distributions, can emulate various function forms (e.g., smooth, fluctuating, rough, ...), and can handle qualitative inputs [30]. Furthermore, GPs, unlike DNNs, are also highly effective in interpolation/regression when the training data are sparse and noisy. On the other side, DNNs have extremely high latent representation power which enables them to distill highly nonlinear and hidden features and relations from a training dataset without explicit instructions. In addition, DNNs are highly scalable and versatile. For instance, while DNNs can naturally build regressors/interpolators from large and high-dimensional data, GPs rely on numerical methods such as low-rank

matrix approximation [31] to handle datasets with more than ~ 5000 samples [15, 32].

When emulating physical systems, surrogates should be generalizable and provide reliable predictions under unseen conditions, e.g., in extrapolation. However, similar to most emulators, the predictive power of both the ordinary GPs and DNNs drastically deteriorates when they are used in extrapolation. In this work, we address this limitation in two different contexts: (1) inferring the governing equations of a physical system using limited training data while building a GP, and (2) solving partial differential equations (PDEs) via DNNs that are trained *a priori* on smaller spatiotemporal domains.

Finding the exact governing equation of a system can be an extremely difficult task, if not unfeasible. However, if this equation is known or can be estimated via some data, it can be used to predict the system's state at any given configuration. We use this argument in Chapter 2 to build *evolutionary Gaussian processes* (EGPs) that can enhance the predictive power and numerical stability of ordinary GPs by estimating the governing dynamics of the system during their training. Through a wide range of analytical case studies and an engineering example we demonstrate the potential benefits of EGPs over ordinary GPs.

DNNs are increasingly used to emulate expensive numerical solvers that estimate the solution of complex PDEs. However, these DNNs are not scalable and have very limited generalization power in that they cannot predict the PDE solution in unseen domains and under unseen initial and boundary conditions. We address this limitation in Chapter 3 by introducing (1) *genomic flow network* (GFNet), a neural network that can infer the solution of a IBVP across arbitrary initial and boundary conditions on a square domain called *genome*, and (2) *mosaic flow prediction* (MF Predictor) which is a novel iterative algorithm that assembles the GFNet's inferences to estimate the PDE solution on large domains with unseen sizes and shapes while considering regularity conditions. We develop, study, and evaluate this approach in three notable PDEs: Laplace, Navier-Stokes, and wave equations and compare our results to the state-of-the-art.

The key contributions of this work are summarized in Chapter 4 and detailed in the conclusion sections of Chapters 2 and 3. Our contributions address a key limitation that most physical system surrogates face: extrapolation. We believe our results are promising and open the doors for many future research directions.

Chapter 2

Evolutionary Gaussian Processes

GPs have played a pivotal role in engineering design because they are easy to train and interpret, can reasonably quantify prediction uncertainty [28, 29], have tractable conditional distributions, can emulate various function forms (e.g., smooth, fluctuating, rough, ...), and can handle qualitative inputs [30]. GPs, unlike deep learning models or any big data based emulator, are also highly effective in interpolation/regression when the training data are sparse and noisy. However, similar to many other emulation methods, their performance deteriorates in extrapolation, see Figure 2.1.

To address this limitation, we build evolutionary Gaussian processes (EGPs) via a novel framework that systematically integrates GP modeling with EP, singular value decomposition (SVD), maximum likelihood estimation (MLE), and bootstrap sampling. Our results of comparing EGPs with ordinary GPs indicate that (*i*) EGPs generally outperform GPs in extrapolation, regression, and interpolation (see Figure 2.1(b) for a simple example), and (*ii*) EGPs are numerically more stable than GPs.

As detailed in Section 2.1.1, the predictive power of ordinary GPs significantly decreases in extrapolation. This behavior is known as reversion to the mean and is a by-product of the

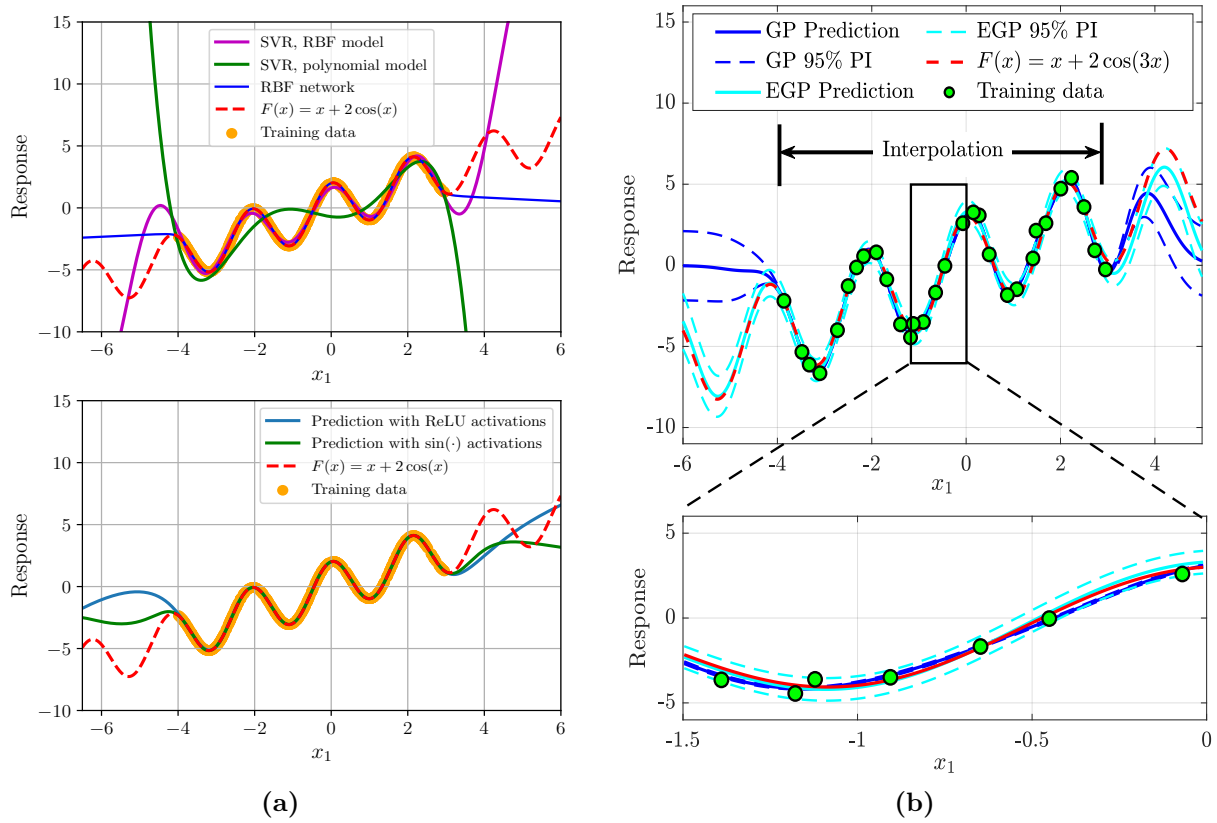


Figure 2.1 (a) Extrapolation errors of various emulators: $F(x) = x + 2\cos(3x)$ is learned over the interval $[-4, 3]$ and extrapolated over $[-6.5, 6]$. In the top panel, SVR (with two different kernels) and RBF are trained with 100 noiseless samples. In the bottom panel, 5000 noiseless samples are used to train two NNs with either ReLU or $\sin(\cdot)$ activations (both NNs have 7 hidden layers and 15 neurons per layer). **(b) GP vs. EGP:** GP and EGP both learn $F(x) = x + 2\cos(3x)$ using some noisy training data (noise variance is 0.25). EGP outperforms ordinary GP not only in extrapolation but also in interpolation. The prediction interval (PI) provided by EGP in extrapolation is also more accurate than the interval obtained via GP.

additive nature of a GP predictor which has two parts. The first part consists of a set of parametric mean functions while the second part relies on correlations between the query point and the training data. In extrapolation, these correlations become extremely weak, so the GP predictor relies primarily on the predictions provided by the parametric mean functions (hence the term reversion to the mean). In ordinary GPs, no parametric functions are used and hence their extrapolation capabilities are minimal.

Reversion to the mean in GPs has been studied previously and prior works can be divided into two primary categories. In the first approach, new covariance functions or kernels are

designed to either discover and preserve patterns [33] or decay slower as the distance between a query point and the training data increases [34, 35, 36]. Pattern preserving kernels are built by modeling the spectral density -the Fourier transform of a kernel- by a parametric model (e.g., a Gaussian mixture) whose Fourier transform can be obtained analytically. While these kernels are powerful and provide insights into the data trends, their applications are limited to low dimensional problems whose the patterns rely on trigonometric functions. GPs with slow decaying kernels break down in extrapolation if the distance between the query point and the training data is large.

In the second category, a set of parametric basis functions such as polynomials and trigonometric functions are employed in training to build the so-called universal GPs [37, 19]. Each function's significance is then determined by their coefficients which are often estimated via MLE. A large (small) coefficient indicates the importance (irrelevance) of the corresponding function in explaining the relations between the inputs and outputs in the training data. If many basis functions are used, regularization must be exercised to avoid overfitting [24]. The primary limitation of addressing reversion to the mean with this approach is that the basis functions are designed by humans who generally include simple and application-dependent functions. This limitation also applies to methods that combine sparse regression with GPs.

Our approach for training GPs with extrapolation capabilities is similar to the second category but unlike prior research we automatically discover the parametric mean functions that must be used in GP modeling. Figure 2.2 demonstrates the simplified flowchart of our framework. We start by fitting an ordinary GP model to the original training data which can be sparse and/or noisy. We then use this GP model to generate training data for EP which discovers some bases (i.e., symbolic functions) that regress the new data. Next, we examine the bases found by EP based on a statistical measure that uses SVD, MLE, and bootstrap sampling. We select the bases with the best measure, include them in the discovered bases (DBs) repository, and repeat the preceding steps while adapting some of them to consider

the most updated version of the DBs repository. Once the convergence criterion is met, we stop the iterations. The output of the framework is a GP model whose mean function is the DBs repository which improves not only the predictive power of GP (in both interpolation and extrapolation) but also its numerical stability. The rationales behind each step of our framework are detailed in Section 2.2.

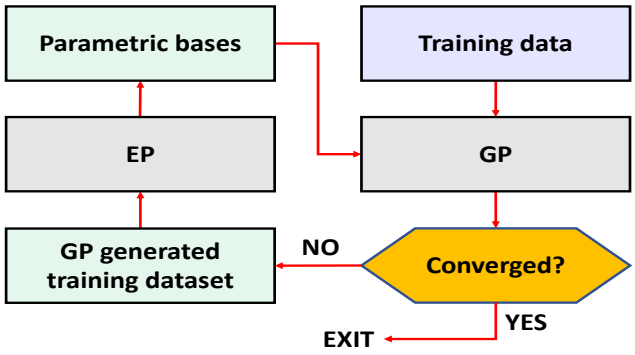


Figure 2.2 Simplified flowchart of our approach: We begin by learning an ordinary GP, i.e., one without any parametric bases. A large dataset is then generated with this GP and used in EP to find some bases. We analyze these bases and select the most fitting ones. Afterwards, we retrain the GP while using the selected bases. This iterative process is continued until the convergence criterion is met.

Our efforts towards building emulators with extrapolation capabilities have some connections with (inverse) system identification and symbolic regression. The goal of system identification is to discover the underlying governing dynamics of a system using some observed data [26, 38, 39, 40]. The objective of symbolic regression is to discover a set of symbolic functions that regress the training data well [27, 41, 42, 43, 44, 45]. With either of these two methodologies, the trained model provides extrapolation capabilities as long as the underlying physics of the data source (i) is effectively learned over the support of the training data, and (ii) does not change outside of the training domain.

Different techniques such as EP, sparse regression [41, 38, 42], and NNs [43, 44] can be used for system identification or symbolic regression but none is free from limitations. For instance, EP is very sensitive to noise and tends to output local optima that poorly extrap-

olate. Sparse regression requires a priori specification of the bases or symbolic functions. Hence, it is limited to simple and low dimensional problems. NNs with symbolic activation functions provide limited representation power because with non-trivial activation functions the gradients quickly vanish/explode or the stochastic gradient descent fails to converge to sufficiently good local optima [1, 46]. Among these methods, some of the limitations of EP can be methodically addressed by our framework and, hence, we employ EP to discover parametric bases in our approach.

The rest of the paper is Chapter 2 is organized as follows. In Section 2.1 we provide some technical background on GP modeling and EP. We elaborate on the algorithmic details of our approach in Section 2.2 and compare its performance with ordinary GPs and EP on a set of analytical functions and an engineering problem in Section 2.3. We summarize our contributions, discuss limitations, and provide future research directions in Section 2.4.

2.1 Technical Background

In this section, we review GP modeling and EP since they are the backbones of our approach. We also discuss some of their limitations that our approach aims to address.

2.1.1 Emulation with Gaussian Processes

Let us denote the output and inputs in the training data by y and $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T$, respectively, where $y \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^d$. For GP modeling, we assume that the input(s)-output relation is a single realization of the random process, $\eta(\boldsymbol{x})$, given by:

$$\eta(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})\boldsymbol{\beta} + \xi(\boldsymbol{x}) \tag{2.1}$$

where $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_h(\boldsymbol{x})]$ are some pre-determined set of bases (e.g., $\sin(x_1)$, $\log(x_1 + x_2)$,$(x_1^2 + x_2)$, $\ldots$), $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_h]^T$ are unknown coefficients, and $\xi(\boldsymbol{x})$ is a zero-mean GP. $\xi(\boldsymbol{x})$ is completely characterized by its co-variance function, $cov(\cdot, \cdot)$, defined as:

$$cov(\xi(\boldsymbol{x}), \xi(\boldsymbol{x'})) = c(\boldsymbol{x}, \boldsymbol{x'}) = \sigma^2 r(\boldsymbol{x}, \boldsymbol{x'}) \tag{2.2}$$

where σ^2 is the process variance and $r(\cdot, \cdot)$ is the correlation function. Many parametric correlation functions have been developed for GPs [14, 15, 36, 47, 48, 17, 19, 49] with the Gaussian correlation function being the most commonly used one:

$$r(\boldsymbol{x}, \boldsymbol{x'}) = \exp(-\sum_{i=1}^{d} 10^{\omega_i}(x_i - x_i')^2) \tag{2.3}$$

where $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_d]^T$,$-\infty < \omega_i < \infty$ are the roughness or scale parameters (in practice the ranges are limited to $-10 < \omega_i < 6$ to avoid numerical errors). The collection of σ^2 and $\boldsymbol{\omega}$ are called the hyper-parameters of $\xi(\boldsymbol{x})$. Following the assumption in Equation (2.1) and given the n training pairs of $(\boldsymbol{x}_{(i)}, y_{(i)})$, GP emulation requires finding a point estimate for $\boldsymbol{\beta}$, $\boldsymbol{\omega}$, and σ^2 via either MLE or cross-validation (CV). Alternatively, Bayes' rule can be employed to find the posterior distributions if some prior knowledge on these parameters is available. In this paper, the MLE approach is employed as it provides a high predictive power while minimizing the computational costs [36, 47, 50, 51, 52, 53].

The MLE estimates of $\boldsymbol{\beta}$, $\boldsymbol{\omega}$, and σ^2 maximize the likelihood of the n training data being generated by $\eta(\boldsymbol{x})$, that is:

$$\left[\widehat{\boldsymbol{\beta}}, \widehat{\sigma^2}, \widehat{\boldsymbol{\omega}}\right] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\omega}}{\arg\max} \mid 2\pi\sigma^2 \boldsymbol{R} \mid^{-\frac{1}{2}} \times \exp\left(\frac{-(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})^T (\sigma^2 \boldsymbol{R})^{-1} (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})}{2}\right)$$

which can also be written as:

$$\left[\widehat{\boldsymbol{\beta}}, \widehat{\sigma^2}, \widehat{\boldsymbol{\omega}}\right] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\omega}}{\arg\min} \frac{n}{2} \log(\sigma^2) + \frac{1}{2} \log(|\, \boldsymbol{R} \,|) + \frac{1}{2\sigma^2} (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})^T \boldsymbol{R}^{-1} (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta}), \tag{2.4}$$

where $\log(\cdot)$ is the natural logarithm, $|\cdot|$ indicates the determinant operator, $\boldsymbol{y} = \left[y_{(1)}, \ldots, y_{(n)}\right]^T$ is the $n \times 1$ vector of outputs in the training data, $\boldsymbol{R}$ is the $n \times n$ correlation matrix with $(i,j)^{th}$ element $R_{ij} = r(\boldsymbol{x}_{(i)}, \boldsymbol{x}_{(j)})$ for $i, j = 1, \ldots, n$, and $\boldsymbol{F}$ is the $n \times h$ matrix with $(k,l)^{th}$ element $F_{kl} = f_l(\boldsymbol{x}_{(k)})$ for $k = 1, \ldots, n$ and $l = 1, \ldots, h$. The point estimates of $\boldsymbol{\beta}$ and σ^2 can be represented as a function of $\boldsymbol{\omega}$ by setting the partial derivative of Equation (2.4) to zero:

$$\widehat{\boldsymbol{\beta}} = \left[\boldsymbol{F}^T \boldsymbol{R}^{-1} \boldsymbol{F}\right]^{-1} \left[\boldsymbol{F}^T \boldsymbol{R}^{-1} \boldsymbol{y}\right] \tag{2.5}$$

$$\widehat{\sigma}^2 = \frac{1}{n} (\boldsymbol{y} - \boldsymbol{F}\widehat{\boldsymbol{\beta}})^T \boldsymbol{R}^{-1} (\boldsymbol{y} - \boldsymbol{F}\widehat{\boldsymbol{\beta}}) \tag{2.6}$$

Plugging these estimates in Equation (2.4) and eliminating the constants results in:

$$\widehat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega}}{\arg\min}\, n \log(\widehat{\sigma}^2) + \log(|\, \boldsymbol{R} \,|) = \underset{\boldsymbol{\omega}}{\arg\min}\, L \tag{2.7}$$

By numerically minimizing L in Equation (2.7) one can find $\widehat{\boldsymbol{\omega}}$ and, subsequently, obtain $\widehat{\boldsymbol{\beta}}$ and $\widehat{\sigma}^2$ using Equations (2.5) and (2.6). Many heuristic global optimization methods such as genetic algorithms [54], pattern searches [55, 56], and particle swarm optimization [57] have been previously employed to solve Equation (2.7). However, gradient-based optimization techniques are commonly preferred due to their ease of implementation and superior computational efficiency [13, 47, 58]. To guarantee global optimality in this case, the optimization is done numerous times with different initial guesses. Upon completion of MLE, the following closed-form formula can be used to predict the response at any $\boldsymbol{x}^*$:

$$\mathbb{E}(y^*) = \boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}} + \boldsymbol{g}^T(\boldsymbol{x}^*) \boldsymbol{V}^{-1} (\boldsymbol{y} - \boldsymbol{F}\widehat{\boldsymbol{\beta}}), \tag{2.8}$$

where $\mathbb{E}$ denotes expectation, $\boldsymbol{f}(\boldsymbol{x}^*) = [f_1(\boldsymbol{x}^*), \ldots, f_h(\boldsymbol{x}^*)]$ are the parametric bases functions evaluated at $\boldsymbol{x}^*$, $\boldsymbol{g}(\boldsymbol{x}^*)$ is an $n \times 1$ vector where the i^{th} element is $c(\boldsymbol{x}_{(i)}, \boldsymbol{x}^*) = \widehat{\sigma}^2 r(\boldsymbol{x}_{(i)}, \boldsymbol{x}^*)$, and $\boldsymbol{V}$ is the covariance matrix where the $(i,j)^{th}$ element is $\widehat{\sigma}^2 r(\boldsymbol{x}_{(i)}, \boldsymbol{x}_{(j)})$. The posterior covariance between the responses at the two inputs $\boldsymbol{x}^*$ and $\boldsymbol{x}'$ reads:

$$cov(y^*, y') = c(\boldsymbol{x}^*, \boldsymbol{x}') - \boldsymbol{g}^T(\boldsymbol{x}^*)\boldsymbol{V}^{-1}\boldsymbol{g}(\boldsymbol{x}') + \boldsymbol{h}(\boldsymbol{x}^*)^T(\boldsymbol{F}^T\boldsymbol{V}^{-1}\boldsymbol{F})^{-1}\boldsymbol{h}(\boldsymbol{x}'), \tag{2.9}$$

where $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{f}^T(\boldsymbol{x}) - \boldsymbol{F}^T\boldsymbol{V}^{-1}\boldsymbol{g}(\boldsymbol{x})$. Equation (2.8) clearly demonstrates the reversion to the mean property of GPs in extrapolation: As the Euclidean distance between $\boldsymbol{x}^*$ and the training data increases, the elements of $\boldsymbol{g}(\boldsymbol{x}^*)$ approach zero. When $\boldsymbol{g}(\boldsymbol{x}^*) \approx 0$, the posterior mean only depends on $\boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}}$. Thus, if an incorrect or incomplete set of bases is used in training, a GP returns inaccurate predictions when extrapolating. Finally, we note that GPs can address noise and smooth the data (i.e., avoid over-fitting) via the so-called nugget or jitter parameter, δ. To this end, $\boldsymbol{R}$ is replaced with:

$$\boldsymbol{R}_\delta = \boldsymbol{R} + \delta\mathbb{I}_{n\times n}. \tag{2.10}$$

If δ is used, the estimated (stationary) noise variance in the data is $\delta\widehat{\sigma}^2$.

2.1.2 Evolutionary Programming

Biological structures that are more successful in grappling with their environment (i.e., they are fitter) survive and reproduce at a higher rate. In other words, over time the fitness of a living individual begets its structure through natural selection, genetic crossover, and mutation. Realizing computer models as complex structures, many scholars have applied the notion of evolution to programming. This perspective on computer modeling has fathered an active field of research known as EP. The terms genetic programming [59, 60], inverse system identification [25], grammatical evolution [27] , symbolic regression [45], and structure

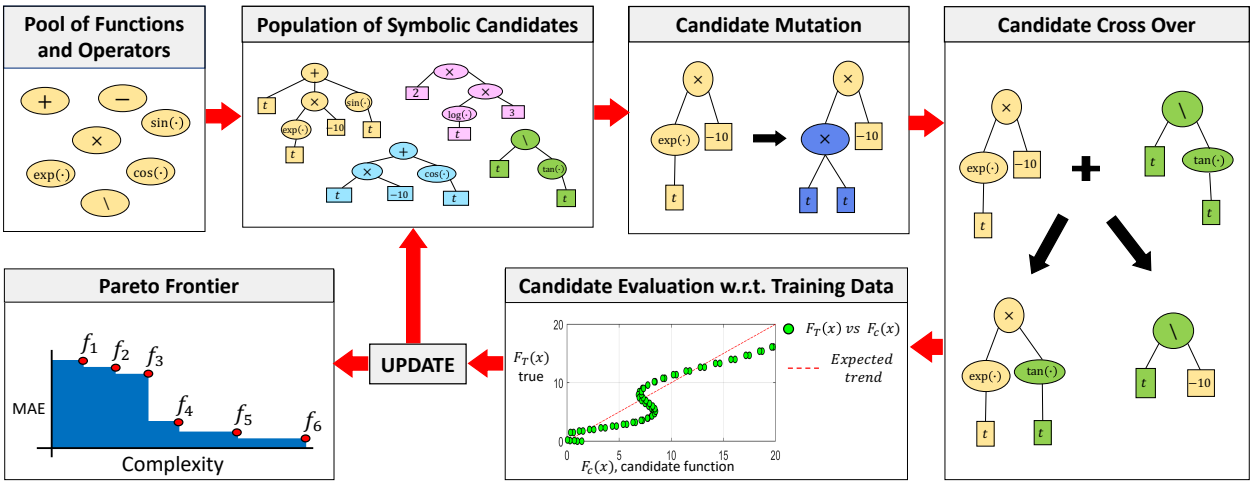


Figure 2.3 Symbolic regression via evolutionary programming: A pool of functions and operators is preinitialized. Then, random combinations of these functions are used to initiate a population of individuals. At each iteration, these individuals undergo mutation and crossover operations to produce individuals with higher fitness over the next generations. This evolutionary process is continued until convergence where a Pareto frontier of fitness vs. complexity is obtained.

learning [61] embody similar lines of research.

Since the early 1990s, EP and its many variants have been exercised to find a symbolic relation between the input(s) and output of a training dataset. As illustrated in Figure 3, the essential idea behind EP is to first build from a selected pool of functions and operators an initial population of simple solutions known as individuals or candidates. Then, these solutions are iteratively evolved by mutation and crossover operations to create more complex individuals that hopefully better relate the inputs and output based on some predefined criterion. This process is continued until convergence when the desired individuals are returned. EP usually optimizes the fitness of the individuals but multiple criteria can be optimized at the same time by means of a Pareto frontier (see Section 2.2.2).

Koza provided the first systematic approach for using EP in symbolic regression and robot planning. In his book [26], Koza models potential solutions to these problems as trees where nodes host primitive operations and functions while leaves store variables. In his approach, new solutions are found by combining these trees together and randomly changing parts of them until at least one of them can faithfully learn the data. Since Koza's seminal work, EP

and its many variants have been successfully used in many complex problems. A particularly active application has been symbolic regression where EP is used to automatically find the governing equations of simple mechanical systems [25, 45, 26, 62, 39, 42, 63, 40].

We conclude this subsection with two important notes. First, EP is fundamentally different than sparse regression [24, 38, 42, 41, 64, 65]. While in EP the input-output relation is discovered, in sparse regression a specific functional form is assumed and the data is used to estimate the parameters of that functional form. Second, applications of EP have been primarily limited to problems of low dimensionality/complexity since EP tends to overfit the data by generating extremely complicated expressions. Although regularization alleviates this issue, the proposed penalty functions have been mostly ad hoc.

2.2 Evolutionary Gaussian Processes

The algorithmic principles behind EGP are motivated by the posterior mean in Equation (2.8): To improve the predictive power of this posterior (especially in extrapolation) a set of *free-form* parametric mean functions, $\boldsymbol{f}(\boldsymbol{x})$, can be learned. Hence, we design a data-driven framework to train a GP while simultaneously searching for some $\boldsymbol{f}(\boldsymbol{x})$ that capture the trend of the data over the training domain reasonably well. To find these $\boldsymbol{f}(\boldsymbol{x})$, we use EP to search the space of parametric functions spanned by polynomials of arbitrary degree, trigonometric functions, $exp(\boldsymbol{x})$, $log(\mathbf{x})$, ..., and combinations thereof.

Figure 2.4 illustrates the mechanics of EGP which, in essence, is a systematic integration of GP and EP. Our framework is iterative (reasons to be discussed) and includes data-driven measures (inspired by the MLE and bootstrap sampling) that vet the bases found by EP.

Given a noisy and/or sparse dataset, we start the first iteration by fitting an ordinary GP, and computing and storing the iteration's convergence parameter (α^1). The computation

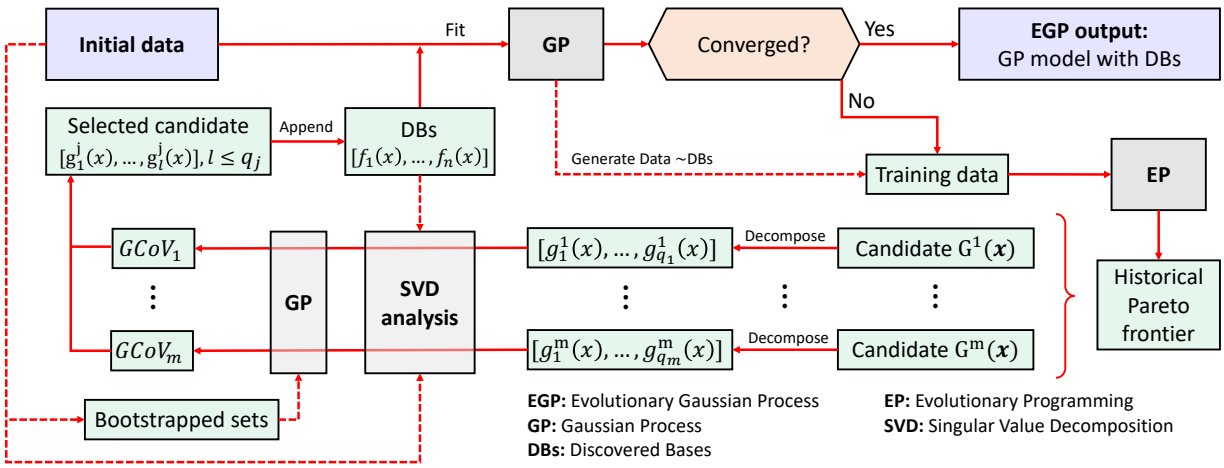


Figure 2.4 Detailed flow chart of EGP. We use the initial training data to fit a GP whose mean function is composed of the parametric functions stored in the discovered bases (DBs) repository. We then generate training data for EP via this GP such that the effect of DBs is excluded. We decompose the candidate solutions discovered by EP into their primitive bases, remove linear dependencies, and analyze their robustness via several trainings on bootstrapped sets. We select the best candidate and append its linearly independent bases to the DBs. We retrain the GP with the updated DBs and check for convergence.

procedures and motivations for the convergence parameter α^i are detailed in Section 2.2.1). Then, we use this GP to generate a new training dataset for EP. This new data have $\lfloor \mu \times n \rfloor$ samples ($\mu \in \mathbb{R}$ and $\mu \geq 1$) and are generated via Sobol sequence [66, 67] over the same domain as the original data. They are better suited for EP since GP reduces noise and can generate as many data points as EP needs.

We then use EP for multi-objective symbolic regression to find a Pareto frontier of candidate solutions ($G^i(\boldsymbol{x})$) that optimizes for both fitness (mean absolute error, MAE) and complexity (for more details see Section 2.2.2). Next, we decompose each $G^i(\boldsymbol{x})$ into its components or *bases* $[g_1^i(\boldsymbol{x}), \dots, g_q^i(\boldsymbol{x})]$ which are defined as its individual additive terms (excluding the coefficients). For instance, the bases in the candidate $G^i(\boldsymbol{x}) = x_1 + x_2(x_3 + 2)$ are $[g_1^i(\boldsymbol{x}), g_2^i(\boldsymbol{x}), g_3^i(\boldsymbol{x})] = [x_1, x_2, x_2 x_3]$). As explained below, this decomposition strategy is adopted to allow for not only removing redundant bases (that EP overfits to the data) but also refining the coefficient of the informative bases that capture the trend of the data.

To remove redundant bases in $G^i(\boldsymbol{x})$, we use SVD to eliminate the linearly dependent ones since they adversely affect the numerical stability of the ensuing steps that involve GP (if a set of linearly dependent bases are used in GP modeling, the $\boldsymbol{F}^T\boldsymbol{R}^{-1}\boldsymbol{F}$ term in Equation (2.5) will not be invertible). Our SVD analysis is performed iteratively on the $\boldsymbol{F}$ matrix, which we build using the original training data and the bases. If dependencies are detected (i.e., the minimum singular value is below a threshold T), the most complex component is removed. This process is repeated until the set of bases is free from linear dependencies. Once the linearly independent bases of $G^i(\boldsymbol{x})$ are determined, we use them to calculate the global coefficient of variation (GCoV, detailed in Section 2.2.3) of $G^i(\boldsymbol{x})$. The candidate with the smallest GCoV is then selected and its linearly independent bases are temporarily included in the repository of discovered bases (DBs). Finally, we use the stored basis in the DBs and the original training data to train a GP and calculate α^2. α^2 is compared to α^1 and the result used as the convergence criterion (detailed in Section 2.2.1). If $\alpha^2 < \gamma\alpha^1$, with $\gamma \in (0, 1)$, the bases are appended to the DBs repository and the algorithm progresses to the next iterations and continues until this convergence criterion is met. If $\alpha^2 >= \gamma\alpha^1$, the iteration is repeated and if the convergence parameter does not improve in this second trial, the algorithm is terminated.

Iterations two, three, ... differ from the first one in two major aspects. First, the following equation is used to generate training data for EP:

$$y^*(\boldsymbol{x}^*) = y^*_{GP}(\boldsymbol{x}^*) - \boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}}.$$

where $\boldsymbol{f}$ and $\widehat{\boldsymbol{\beta}}$ are, respectively, all the bases from the DBs repository and its corresponding coefficients estimated via MLE. This strategy greatly helps EGP in that the evolutionary process no longer needs to discover what is learned in the previous iterations. The second major difference is that all the bases in the DBs repository are also included in the SVD analyses to prevent the addition of bases to the repository that are linearly dependent with

the available ones. It is noted that to prevent forgetting what the algorithm has already learned, the SVD analyses of iteration i never eliminate the bases discovered in the previous iterations.

2.2.1 Convergence Criterion

The convergence parameter α^i, for a particular iteration i, ensures that the updates on the DBs after the iteration increase the predictive power of EGP (especially in extrapolation). to do that, using only the data available on the training set α^i is defined as follows. Let $D \in \mathbb{R}^n$ be the training domain. Let $H \subset D$ be an hypercube inside the training domain such that $V(H)$, volume of H, is the 80% of the $V(D)$. Then, α^i is the mean-squared error of a GP trained over H with the bases from the DBs repository (if any) when evaluated on $D \backslash H$. We refer to H as the interpolation set and to $D \backslash H$ the extrapolation set and they are, by definition, mutually exclusive parts of D.

The value of α^i across two consecutive iterations governs convergence. In particular, EGP converges in iteration $i > 1$ if $\alpha^{i+1} \geq \gamma \alpha^i$ with $\gamma \in (0, 1)$, i.e., if the updates of the DBs in iteration $i + 1$ incorrectly model the data or do not sufficiently increase the accuracy.

2.2.2 Historical Pareto Frontier

EP optimizes for both complexity and fitness, quantified via MAE, which are generally opposing objectives. That is, complex candidate solutions or individuals are unfavorable but they tend to have better fitness values (i.e. lower MAE). As a result, EP builds a Pareto frontier of individuals that indicate the fittest candidate for any complexity, see Figure 2.5. The complexity of a candidate solution is calculated by summing the complexity numbers assigned to its entailed operations: Simple operations $(+, -, \times)$ have complexity

ID	Equation
1	x
2	$0.17+x$
3	$0.14+1.07x$
4	$x+0.06x^2$
5	$x+\cos(3.00x)$
6	$1.03x+0.94\cos(3.00x)-0.02$
7	$1.05x+0.92\cos(2.99x)-0.06x\cos(2.98+x)$
8	$1.05x+0.92\cos(2.98x)-0.04-0.07x\cos(2.54+x)$
9	$1.07x+0.95\cos(2.93x)-0.11-0.16x\cos(2.03+1.26x)$
10	$(1.06x+0.96\cos(2.91x)-0.12-0.18x\cos(1.93+1.27x)$ $-0.01x^2\cos(1.93+1.27x)$

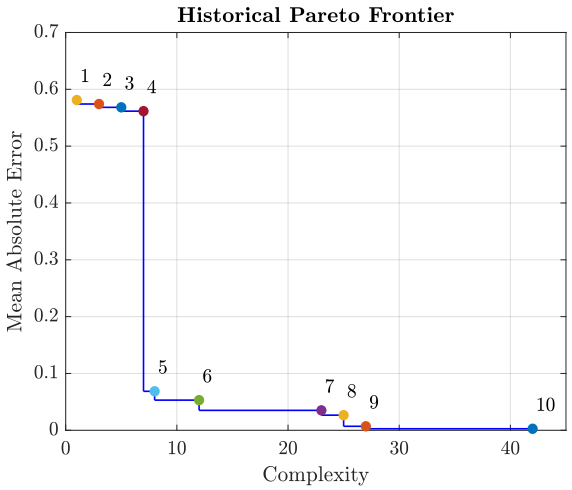


Figure 2.5 Example of historical Pareto frontier: EGP emulates the function $F(x) = x + 2\cos(3x)$ using some noisy training data. The individual candidates comprising this historical Pareto frontier are obtained by EP in the first iteration of EGP.

1, division has complexity 2, and functions (e.g., trigonometric, exponential, logarithm, ...) have complexity 3.

The Pareto frontier is usually built with the individuals of the last generation. However, our studies on EP indicate that these individuals do not necessarily provide the optimum compromise between fitness and complexity. This is because EP either achieves local optimality or overfits the noise that GP has failed to filter out. To address this issue, we track all the individuals generated during the entire evolution process (regardless of the population or generation number) and build the historical Pareto frontier.

2.2.3 Global Coefficient of Variation

The candidates obtained via EP generally overfit the data even though we use the historical Pareto frontier, filter out noise with GP, and learn them iteratively. Not all of the bases in one candidate correctly model the underlying physics of the problem, so we desing a metric to filter those candidates that contain unlikely bases: the global coefficient of variannce, GCoV. The GCoV is inspired by MLE and bootstrap sampling and measures the robustness

of the coefficients of a candidate's bases to variations in the training data. The candidate whose linearly independent bases are more robust has a smaller GCoV and is selected over other candidates as it is understood that it better represents the underlying behavior of the data.

To obtain GCoV for the candidate solutions, we first bootstrap the initial data to generate b datasets of size $n' < n$ where n is the size of the initial data. Then, for $G^i(\boldsymbol{x})$ with l linearly independent bases, we use the following equation to calculate the CoVs for the coefficients of its bases:

$$CoV_j = \frac{\text{std}(\beta_j^1, \ldots, \beta_j^b)}{\text{mean}(\beta_j^1, \ldots, \beta_j^b)}, j = 1, \ldots, l.$$

Where β_j^k is the coefficient of basis j when the set k is used in MLE. The MLE process involves fitting a GP using the l linearly independent bases (in addition to the bases in the DBs repository) as parametric mean functions. Once these CoVs are calculated, we determine the GCoV of the corresponding candidate via penalized weighted averaging, i.e.,:

$$GCoV = \frac{\sum_{j=1}^{l} \left(CoV_j\right)^p}{l^q},$$

where p and l^q penalize (i.e., increase) the GCoV of a candidate if it has unstable coefficients or too many bases, respectively.

2.2.4 Discussions

The underlying assumption behind EGP is that there are some free-form parametric bases that can model the data source reasonably well. As a result, if we can *discover* these bases via some training data and use them in GP modeling, we can build GPs that outperform ordinary GPs in extrapolation, interpolation, and regression. If this assumption is incorrect (or EGP fails to find bases that explain the data well), the output of our algorithm would be an ordinary GP.

As we show in Section 2.3.1, another potential advantage of EGP over other GP modeling approaches is increased numerical stability. We can quantify this advantage by analyzing the smallest eigenvalue of $\boldsymbol{R}_{opt} = \boldsymbol{R}_\delta(\boldsymbol{\omega} = \widehat{\boldsymbol{\omega}})$. In the extreme case where $\boldsymbol{f}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}$ completely explains the data, $\xi(\boldsymbol{x})$ in Equation (2.1) models either the noise (i.e., the data is noisy and $\boldsymbol{f}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}$ regresses the data) or a random process that is zero everywhere (i.e., the data does not have noise and $\boldsymbol{f}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}$ interpolates the data). In either of these scenarios, $\widehat{\omega}_i \ll 0$ (if the nugget is used correctly in the optimization [47]) and hence $\boldsymbol{R}_\delta = \boldsymbol{R} + \delta\mathbb{I}_{n\times n} \approx \boldsymbol{1}_{n\times n} + \delta\mathbb{I}_{n\times n}$ which is well-conditioned as the smallest eigenvalue is $1 + \delta$.

In most problems, the extreme case considered above is not observed and hence $\boldsymbol{R} \neq \boldsymbol{1}_{n\times n}$. Since $\boldsymbol{R}$ is positive-definite, the smallest eigenvalue of $\boldsymbol{R}_{opt} = \boldsymbol{R}_\delta(\boldsymbol{\omega} = \widehat{\boldsymbol{\omega}}) = \boldsymbol{R}(\boldsymbol{\omega} = \widehat{\boldsymbol{\omega}}) + \delta\mathbb{I}_{n\times n}$ is close to δ and thus $\boldsymbol{R}_{opt}$ is again well-conditioned. It is also noted that our SVD analyses ensure that the term $\boldsymbol{F}^T\boldsymbol{R}^{-1}\boldsymbol{F}$ is of full rank and hence invertible.

Lastly, we point out that while the goal of EGP is not symbolic regression, it can be used for that purpose. We demonstrate this capability of EGP in Section 2.3.3.

2.2.5 Selection of the EGP Parameters

The performance of EGP depends on μ, γ, n', T, b, p, and q. In this section, we specify the parameter choice used for all the examples in Section 2.3 and discuss the rationales behind them so any user can adjust them to specific problems.

We use $\mu = 1.5$ to generate a denser dataset for EP which helps it in finding informative bases. We discourage the use of $\mu \gg 1$ as computational costs would increase, and the symbolic regression could overfit. $\gamma = 0.9$ is used to account for a minimum amount of improvement in terms of extrapolation after each iteration. With a γ too close to one, highly complex bases will be accepted in the DBs repository. With $\gamma < 0.9$, it will be difficult to regress the

least relevant terms correctly. Taking this into account, we recommend selecting γ based on the data configuration. In a low noise and dense dataset, where overfitting is not an issue as the noise will be properly filtered by the GP, we advise using a γ closer to 1. Otherwise, the user should be conservative on the selection of γ.

The choice for n' and T is related. We use $n' = \frac{1}{3} \times n$ while bootstrapping to allow for variations to be developed while having a decent number of samples. With very small bootstrapped sets, high variations will be observed across the sets but robustness (numerical stability) decreases. So, we select $T = 10^{-4}$ to avoid any numerical issues while training the GPs. Significantly denser (sparser) datasets than those used in the examples would require a lower (higher) value of n' while T can remain the same. In highly complex data, where linear dependencies between the bases are unlikely, a smaller value for T can be selected.

We use $b = 10$ to assess the sensitivity of the individual CoVs robustly. A higher value of b will be computationally expensive, but we encourage the user to increase it if the time and the computational power are not a constraint. We select $p = 3$ to penalize candidates with unstable bases whose CoV is high. When using a larger value of p, no improvements are usually observed. Finally, with $q = 0.7$ we penalize large, overfitting candidates. On the examples studied with $q \geq 1$, the linear independent bases of the overfitting candidates were usually accepted while with $q < 0.5$ single base candidates, which were not part of the true expression, were accepted. We recommend the inexperienced user to use the suggested defaults.

2.3 Results

In this section, we compare the performance of EGP in interpolation and extrapolation against ordinary GP and EP. In Section 2.3.1, we compare them using eight analytical

ID	Analytical Function	Min($\boldsymbol{x}$) for $\boldsymbol{x} \in D$	Max($\boldsymbol{x}$) for $\boldsymbol{x} \in D$	Mean Range (y)
1	$y(x) = 943.61(x_1 - 2.8)^2 + 409.40(x_2 - 2.0941)^2$ $+4.15(1 - \cos(2x_3)) + 3.28\left((\frac{3.46}{x_4})^3 - (\frac{3.46}{x_4})^2\right)$	$[2.12, 0.85, 0, 2]$	$[4.32, 3.35, 3.14, 7]$	2602
2	$y(x) = x_7^3 - x_5 + 0.2x_7x_3x_1 + 4x_6 - x_2x_3 + x_1 - x_2$ $+15\sin(2x_1) - 10\cos(x_3) - 5$	$[-4.5, -6, 1, -3, -3, -2.5, -3.5]$	$[3.5, 2, 6, 4, 2, 3.5, 4.5]$	188
3	$y(x) = 2\sin(x_1) - 3\cos(x_2) + x_1 + x_2^2$	$[-3, -5]$	$[2, 2]$	36
4	$y(x) = x_1^2 + \frac{x_1}{2x_2 - 1} + \sqrt{(2)}x_3x_4 + 2.3\mathrm{e}^{-x_5} + \sqrt{(x_2)}$	$[-3, 0.54, -2, -3, -4.5]$	$[3.5, 4.54, 3, 1, 0.5]$	219.97
5	$y(x) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + 4x_2^4 - 4x_2^2$	$[-2, -1]$	$[2, 1]$	5.58
6	$y(x) = \log(\lvert x_3 \rvert)x_1 + 3.2\tan(x_1) + \sqrt{\lvert x_2 + 3 \rvert}$	$[-1.2, -2, 0.7]$	$[1.2, +4, 5]$	19.6
7	$y(x) = \frac{2\pi x_1(x_2 - x_3)}{\left(\log\left(\frac{x_4}{x_5}\right)\left(1 + \frac{x_1}{x_6} + \frac{2x_1x_7}{\log\left(\frac{x_4}{x_5}\right)x_5^2x_8^2}\right)\right)}$	$[63070, 990, 700, 100, 0.05,$ $63.1, 1120, 9855]$	$[115600, 1110, 820, 50000,$ $0.15, 116, 1680, 12045]$	223
8	$y(x) = \Big(\exp(-(x_2 - 1)^2) + \exp(-0.8(x_2 + 1)^2)$ $-0.05\sin(8(x_2 + 0.1))\Big)\Big(\exp(-(x_1 - 1)^2)$ $+\exp(-0.8(x_1 + 1)^2) - 0.05\sin(8(x_1 + 0.1))\Big)$	$[-3, -3]$	$[3, 3]$	1.11

Table 2.1 Analytical examples: The functions posses different dimensionality, complexity, and ranges. The average range of each function is calculated using multiple training sets that are generated over the input space.

examples that include different dimensions and noise levels. We also use these examples to demonstrate the superior numerical stability of EGP over ordinary GP. In Section 2.3.2, we apply these three emulation techniques to an engineering problem where very limited prior knowledge on the functional form of the response is available. Finally, in Section 2.3.3 we briefly discuss the results of using EGP as a symbolic regression tool.

In all examples, the following parameters are selected for EGP: $\mu = 1.5$, $\gamma = 0.9$, $T = 10^{-4}$, $n' = \frac{1}{3} \times n$, $b = 10$, $p = 3$, and $q = 0.7$, whose rationales have been introduced in Section 2.2.5.

We use the algorithm described in [47] for GP modeling which employs an adaptive mechanism to estimate the nugget variance. For symbolic regression, we use Eureqa [45] which is a highly sophisticated implementation of EP. In all of our studies, the convergence criterion of Eureqa is based on the stagnation of MAE across the candidates. We note that many aspects of EP (e.g., population size, crossover rate, and mutation rate) are internally determined in Eureqa and cannot be manually tuned.

	ID 1	ID 2	ID 3	ID 4	ID 5	ID 6	ID 7	ID 8
Small Noise Variance	0.75^2	1^2	0.8^2	1^2	0.04^2	0.1^2	1^2	0.01^2
Large Noise Variance	2.5^2	3^2	3^2	3.5^2	0.4^2	1^2	3^2	0.05^2

Table 2.2 Noise variance in analytical examples: Normal noise is added to the training data. For each example, two noise levels are considered (one small and one large). The range of the function (see last column of Table 2.1) is used to determine the magnitude of the noise variances.

2.3.1 Analytical Functions

The analytical functions we study are shown in Table 2.1. Many of them are relatively high dimensional functions and include terms that possess asymptotic behavior, have a high-frequency, or contain nested functions. Each function is emulated under two different scenarios where the training data are corrupted by a Gaussian noise with either a small or large noise variance, see Table 2.2. The noise variances are selected based on the function range (see the last column in Table 2.1) where a small (large) noise variance corrupts the response by up to 1% (10%).

Sobol sequence [66, 67] is used to generate the training and validation data. The size of the training dataset is $50 \times Dim$ where Dim indicates the dimensionality of the underlying function. The validation dataset for each case is corrupted by noise as well and comprises two mutually exclusive parts designed to assess the interpolation and extrapolation capabilities of GP, EP, and EGP. The interpolation error of each emulator is calculated over the training domain. However, the extrapolation error is measured over a domain that excludes the training region and its bounds are obtained by expanding $min(\boldsymbol{x})$ and $max(\boldsymbol{x})$ in Table 2.1 by 40%. For instance, if the training range is $[0, 1]$, the interpolation and extrapolation errors are calculated over, respectively, $[0, 1]$ and $[-0.2, 1.2]$ ranges. Errors are quantified via normalized root mean squared error (NRMSE) using $500 \times Dim$ samples. To account for sample-to-sample variations, the training-validation process is repeated five times. Lastly, in example 3, trigonometric functions are intentionally excluded from the primitive function set of EP.

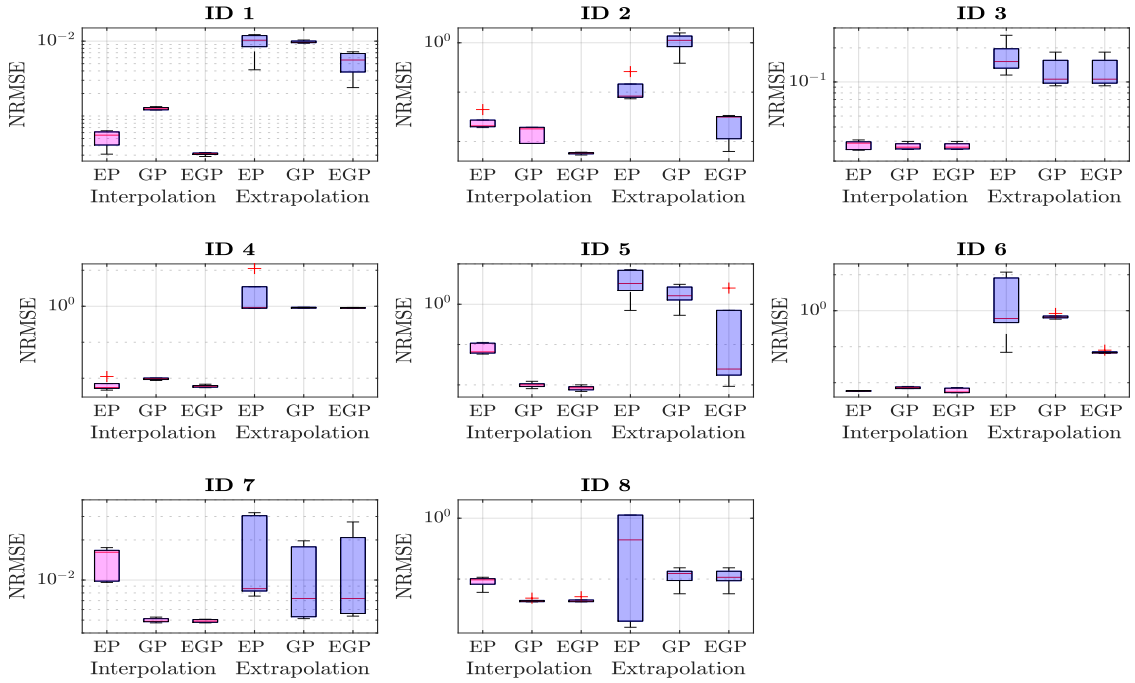


Figure 2.6 Performance of EP, GP, and EGP (small noise scenario) : Each subplot corresponds to one example and includes interpolation and extrapolation errors in terms of NRMSE on log scale.

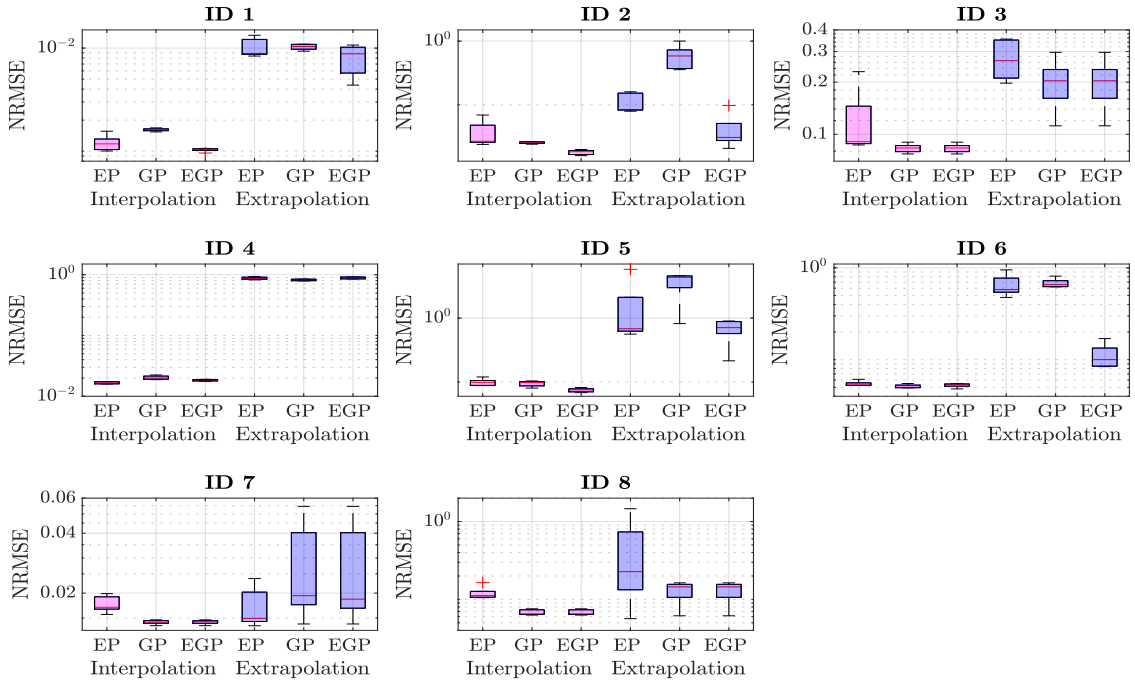


Figure 2.7 Performance of EP, GP, and EGP (large noise scenario): Each subplot corresponds to one example and includes interpolation and extrapolation errors in terms of NRMSE on log scale.

Results

Figure 2.6 summarizes the results for the cases where the noise variance is small. Interpolation-wise (see box plots in light pink), EGP outperforms EP across all our cases except for example 6 where EP results in slightly less variations across the repetitions. When comparing EGP to ordinary GP, two trends are observed. In examples where some bases are discovered by our algorithm (IDs 1,2,4,5,6), EGP achieves smaller interpolation errors than GP. In examples where no bases are discovered (IDs 3,7,8), the performance of EGP and GP is identical.

The primary advantage of EGP is observed in extrapolation (see box plots in light purple in Figure 2.6). In examples 1, 2, 5, and 6 where some adequate bases are discovered, the extrapolation capability of EGP is more than EP and ordinary GP. This higher accuracy is because EGP (i) does not suffer from reversion to the mean as much as ordinary GP does, and (ii) has multiple mechanisms (unlike a standalone EP software such as Eureqa) that exclude bases which incorrectly capture the trend of the data. In example 4, although some adequate bases are discovered, the extrapolation performance is not improved. This is because the base $\frac{x_1}{2x_2-1}$, which has a large asymptotic behavior on the training domain, is not properly learned. In examples 3, 7, and 8 where no bases are included in EGP, the performance is still better than EP (by avoiding overfitting) but similar to ordinary GP. We note that in one of the repetitions of example 7 (which is highly complex) an incorrect basis is accepted and hence the performance variability of EGP is more than GP (compare the height of the corresponding box plots). 2.2.4

Figure 2.7 summarizes the results for the cases where the noise variance is large. The observations are largely consistent with those reported in Figure 2.6 with the exception of example 7. In this example, EP discovers some bases that capture part of the trend of the data in extrapolation and hence achieves smaller errors than EGP which excludes these bases as they insufficiently decrease the convergence parameter, α. Although tuning α in

our algorithm can increase the performance of EGP in this example, we avoid tailoring EGP to specific problems.

With either large or small noise variance, two interesting observations are made. Firstly, in example 3 where trigonometric functions are intentionally excluded from the evolutionary search process. This example is designed to not only resemble applications (such as the one in Section 2.3.2) where the underlying data source does not have a closed-form functional form, but also applications where the underlying physics is not effectively learned (e.g., due to the failure of the evolutionary search process). In this case, unlike EP, EGP avoids fitting high-degree polynomials to the data and hence achieves much better extrapolation accuracy. The second observation is on the numerical stability of EGP and how it compares to ordinary GP. Figure 2.8 quantifies the variations of smallest eigenvalue of $\boldsymbol{R}_{opt}$ in all of our studies and indicates that whenever some bases are discovered by EGP (examples 1, 2, 4, 5, and 6), this measure and thus the numerical stability are higher.

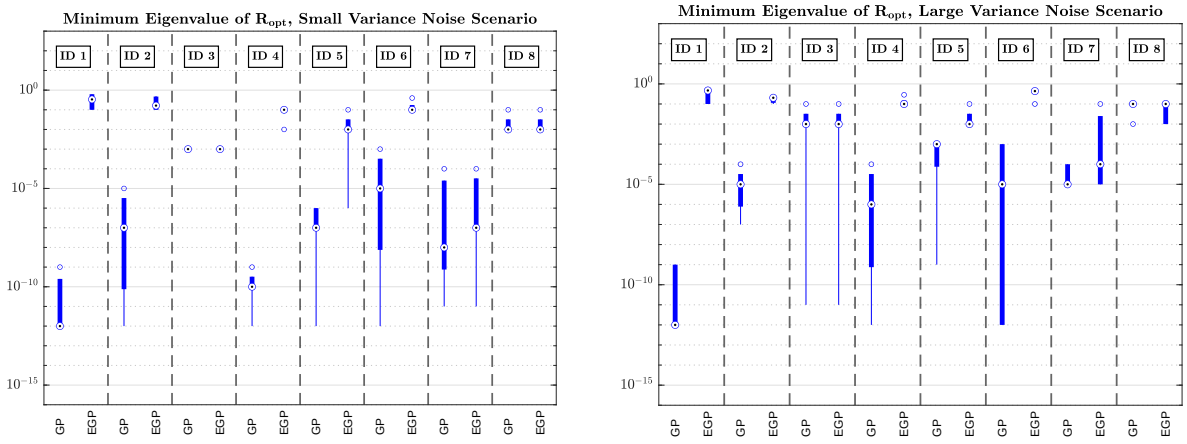


Figure 2.8 Numerical stability of EGP vs. ordinary GP: The left (right) panel indicates the distribution of the smallest eigenvalue of $\boldsymbol{R}_{opt}$ across the eight examples when training data are corrupted by a Gaussian noise with small (large) variance.

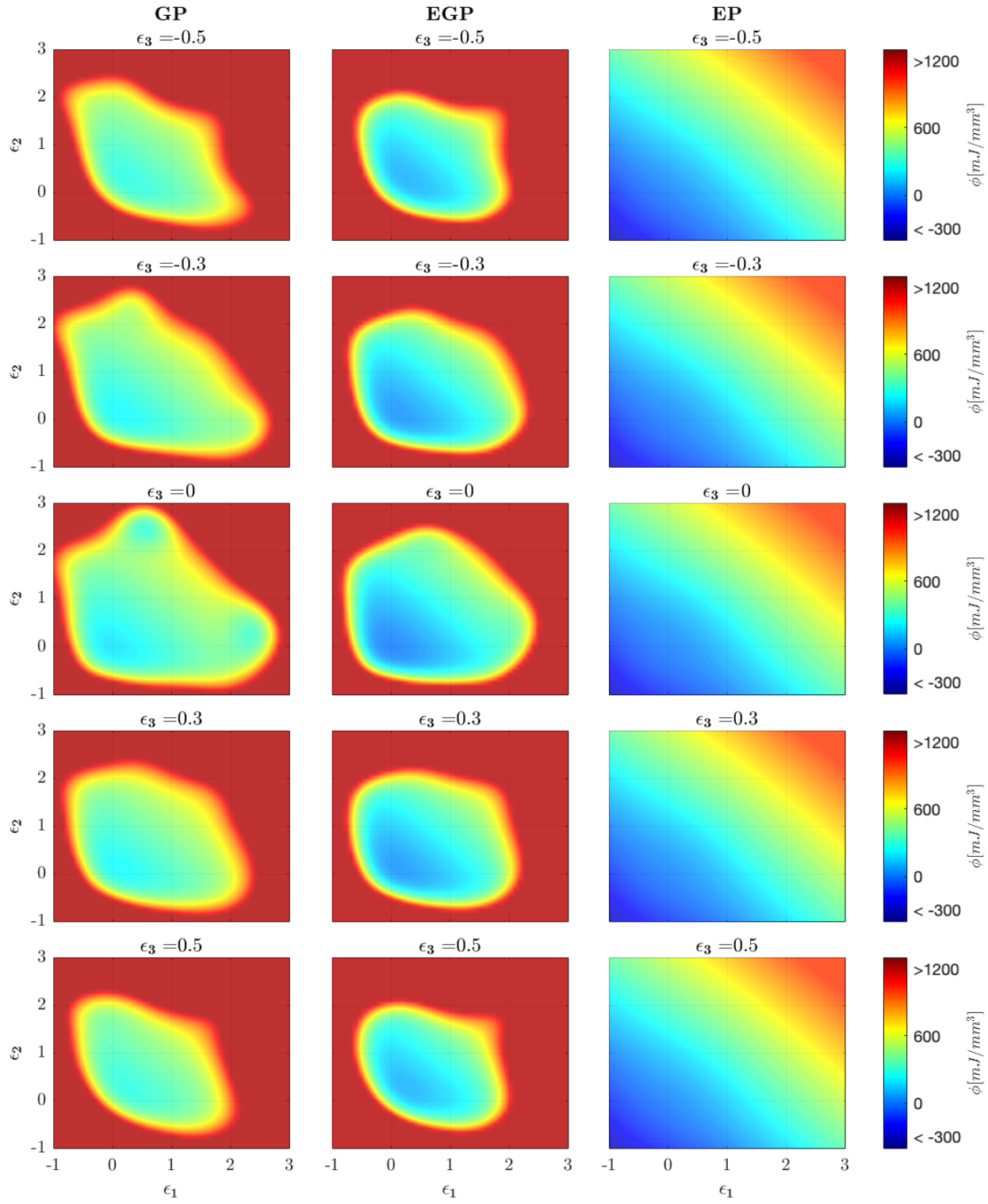


Figure 2.9 Materials modeling with GP, EGP, and EP: ϵ_1, ϵ_2, and ϵ_3 are Green strains and ϕ is the stored potential in the composite microstructure. The prediction domain is larger than the training domain which is $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -0.085 \leq \epsilon_1, \epsilon_2 \leq 1.275; -0.34 \leq \epsilon_3 \leq 0.34\}$.

2.3.2 Extrapolation for Material modeling

In most real-world applications, the symbolic relation between the independent and dependent variables in a dataset is unknown. To study how our algorithm performs in these scenarios, we use EGP to learn the constitutive law of a 2D heterogeneous composite. The matrix material of the composite is a soft compressible elastomer modeled by the Arruda Boyce [68] hyperelastic constitutive model. The composite reinforcements are randomly distributed elliptical particles that are modeled as compressible Neo-Hookean elastomers (see more details in [69, 70]). The constitutive law of our composite microstructure is unknown and, thus, numerical methods such as the finite element method (FEM) are required to find its response (i.e., the stored potential) to any applied strains. The high computational costs of FEM hinder multiscale simulations where the fine-scale behavior is governed by this microstructure. One solution for decreasing the costs is to replace FEM with a surrogate as follows. A training dataset is first built by computing (via FEM) the stored potential in the microstructure under various strain states. Then, an emulator is fitted to this dataset to serve as the constitutive law of the microstructure.

Our dataset has 1000 samples and its inputs and output are, respectively, Green strains $(\epsilon_1, \epsilon_2, \epsilon_3)$,and the stored deformation potential (ϕ) $[mJ/mm^3]$. The data has a small amount of noise which is primarily due to numerical instabilities, excessive mesh distortion, and round off errors experienced in FEM (see [70] for more technical details). It is important to note that in this problem, obtaining samples that entail large deformations (i.e., large strains) takes much longer than samples that undergo small deformations. Hence, we are particularly interested in building an emulator that can predict the stored potential at large deformations using training data on small deformations.

We divide our dataset into three mutually exclusive and collectively exhaustive sets for training, interpolation testing, and extrapolation testing. The original domain sampled

	Training Error	Interpolation Error	Extrapolation Error
EP	27.08	48.66	169.18
GP	1.69e-05	0.185	92.73
EGP	1.28e-05	0.139	56.08

Table 2.3 Performance of GP, EGP, and EP in the engineering example: MSE results of EP, GP and EGP on the different sets. The response range in the original training domain is 383.74.

with FEM is $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -0.1 \leq \epsilon_1, \epsilon_2 \leq 1.5; -0.4 \leq \epsilon_3 \leq 0.4\}$. For training and interpolation test sets, we select 85% of the original domain, i.e., $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -0.085 \leq \epsilon_1, \epsilon_2 \leq 1.275; -0.34 \leq \epsilon_3 \leq 0.34\}$. This domain contains 616 data points. 75% of these data points (i.e., 462 samples) are randomly selected and used for training while the remaining 25% (i.e., 154 samples) are employed to obtain the interpolation accuracy. Data points that lay outside of this new domain (i.e., $1000 - 616 = 384$ samples) comprise the extrapolation set.

We fit three emulators to the training set via ordinary GP, EP and EGP and predict the stored potential in the other two sets. As there is no prior knowledge on the underlying expression, we select the following primitive functions for symbolic regression in both EP and EGP: simple operations $(+, -, \times)$, division, and exponential function.

To compare the three emulators we use (i) MSE associated with interpolation and extrapolation test sets, and (ii) the physical constraints that the emulators should satisfy: since they predict a stored potential, the emulators should be convex and also satisfy $\phi(0, 0, 0) = 0$ [71].

Table 2.3 summarizes the MSE results and indicates that EGP outperforms ordinary GP and EP in all cases. As for the physical constraints, the predicted stored potential at zero-deformation are $\phi(0, 0, 0)_{EP} = 6.71$, $\phi(0, 0, 0)_{GP} = 0.0027$, and $\phi(0, 0, 0)_{EGP} = 0.0021$ which show that EGP is more accurate. The constraint on convexity is evaluated by visually comparing the response surface of each emulator over the region $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -1 \leq$

$\epsilon_1, \epsilon_2 \leq 3; -0.5 \leq \epsilon_3 \leq 0.5\}$ which is much larger than the original domain. The response surfaces are plotted in Figure 2.9 and demonstrate that EGP neither fluctuates (unlike EP) nor regresses to the mean (unlike ordinary GP).

2.3.3 Symbolic Regression with EGP

While the goal of EGP is not symbolic regression, it can be used for that purpose. We illustrate this by comparing the discovered symbolic regressions from EGP (final components of the DBs repository) with the bases that Eureqa finds in examples 1, 3, and 7.

In example 1, the EP emulator (from Eureqa) includes a wide variety of incorrect bases (x_3, x_3^4, x_4) while EGP does not. EGP generally includes the term $\frac{1}{x_4}$ which, while not part of the underlying expression, better captures the trend than x_4 (term usually included by EP). Similar results are observed in examples 2, 4, 5, and 6.

In example 3 the trigonometric functions are intentionally excluded from the primitive pool of functions when the evolutionary search process is conducted. As a result, EP tries to interpolate these trigonometric functions with high degree polynomial terms, which adversely affect extrapolation. EGP, however, detects that these terms are not part of the true underlying analytical function and discards them.

In example 7, the underlying expression is highly complex and the evolutionary search incorrectly finds simple terms. Most of these simple terms are excluded in the results of EGP as they cannot extrapolate. However, EP generally includes many incorrect bases in the final results. Similar results are observed in example 8.

Terms / Method	x_1^2	x_1	x_2^2	x_2	$\cos(2x_3)$	$\frac{1}{x_4^2}$	$\frac{1}{x_4^3}$	**False Terms (Percentage of Appearance %)**
EGP Small Noise	100%	100%	100%	100%	100%	40%	0%	x_4 (20%), $\frac{1}{x_4}$ (60%)
EGP Large Noise	100%	100%	100%	100%	80%	40%	0%	x_4 (20%), x_4^2 (20%)
EP Small Noise	100%	100%	100%	100%	40%	40%	0%	x_3(40%), x_3^2 (40%) x_4 (80%)
EP Large Noise	100%	100%	100%	100%	40%	40 %	0%	x_3(40%), x_3^2 (40%) x_4 (80%)

Terms / Method	$\sin(x_1)$	$\cos(x_2)$	x_1	x_2^2	**False Terms (Percentage of Appearance %)**
EGP Small Noise	0%	0%	0%	0%	-
EGP Large Noise	0%	0%	0%	0%	-
EP Small Noise	0%	0%	100%	100%	$x_2^6(20\%)$, $x_2^4(60\%)$, $x_1^3(100\%)$, $\frac{1}{x_2}(20\%)$, $x_2^5(40\%)$, $x_2^3(40\%)$, $x_2(60\%)$, $x_1^2(20\%)$, $x_2x_1^2(20\%)$, $x_2x_1(40\%)$
EP High Noise	0%	0%	100%	100%	$x_2^3(40\%)$, $x_2(40\%)$, $x_1^3(40\%)$, $x_2^2(20\%)$ $\frac{1}{x_2}(20\%)$, $\frac{x_2}{x_1}(20\%)$, $x_2^5(40\%)$, $x_1^2x_2(20\%)$, $x_1^2(20\%)$, $x_2^4(20\%)$, $x_1x_2^2(20\%)$, $\frac{1}{1.901x_1-0.389}(20\%)$, $x_1x_2(20\%)$, $x_2x_1^4(20\%)$

Terms / Method	a	b	**False Terms (Percentage of Appearance %)**
EGP Small Noise	0%	0%	$x_2x_5^2(20\%)$, $x_3x_5^2(20\%)$, $x_7x_5^2(20\%)$, $x_8x_5^2(20\%)$
EGP Large Noise	0%	0%	-
EP Small Noise	0%	0%	$x_2x_5^2(40\%)$, $x_8x_2^2x_5^2(20\%)$, $x_8x_3^2x_5^2(20\%)$, $x_7x_2^2x_5^2(40\%)$, $x_2x_5x_8(40\%)$, $x_2^2x_5^2(40\%)$, $x_3^2x_7^2(40\%)$, $x_8(40\%)$ $x_5x_7x_3^2(40\%)$, $x_4(20\%)$, $\sin(126x_4)(20\%)$, $x_2x_3x_8x_5^2(20\%)$, $x_1x_6(20\%)$
EP Large Noise	0%	0%	$\frac{x_8x_2^2x_5^2}{x_7x_3^3}(20\%)$, $\frac{1}{x_7x_3^3}(20\%)$, $x_2x_5x_8(20\%)$, $x_3^2x_7^2(20\%)$, $x_8(20\%)$, $x_5x_7x_3^2(20\%)$, $x_5(20\%)$, $x_8x_2^2x_5^2(20\%)$, $x_3x_5(20\%)$, $x_2x_5^2(40\%)$, $x_3x_7x_8x_5^2(20\%)$, $x_8x_5^2(40\%)$, $x_7x_5^2(20\%)$, $x_3x_5^2(40\%)$, $\frac{x_2^2x_5^2}{x_7}(20\%)$, $x_2^2x_5^2(20\%)$

Table 2.4 Regressed terms with EGP for the analytical examples. From top to bottom, the tables show the symbolic regression results for examples 1, 3, and 7. For each true term the tables include the percentage of appearance of it across the 5 different fits. The last column shows the regressed terms that are not part of the true underlying function and its percentage of appearance.

$$a = x_1x_3\left(\log\left(\frac{x_4}{x_5}\right)\left(1+\frac{x_1}{x_6}+\frac{2x_1x_7}{\log\left(\frac{x_4}{x_5}\right)}x_5^2x_8^2\right)\right)^{-1}, b = x_1x_3\left(\log\left(\frac{x_4}{x_5}\right)\left(1+\frac{x_1}{x_6}+\frac{2x_1x_7}{\log\left(\frac{x_4}{x_5}\right)}x_5^2x_8^2\right)\right)^{-1}$$

2.4 Conclusions

In Chapter 2, we introduced EGP which leverages EP to build a GP model with automatically discovered symbolic bases. We tested the performance of EGP against ordinary GP and EP using some analytical functions as well as an engineering problem on learning the constitutive law of a heterogeneous microstructure. The analytical examples showed

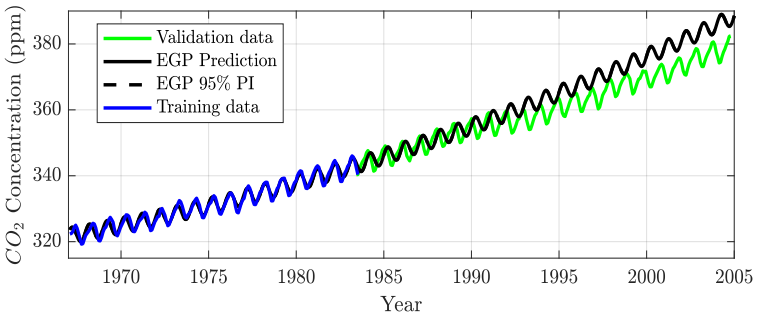


Figure 2.10 Over confident extrapolations with EGP: The monthly mean CO_2 concentration of Mauna Loa, Hawai, is learned and extrapolated. The uncertainties in extrapolation are smaller that 10^{-4}.

that, when our algorithm discovers bases that can explain the data reasonably well, EGP achieves lower extrapolation errors, is more numerically robust, and performs better in interpolation/regression. In the engineering example on materials modeling, EGP produced an emulator that better satisfies the physical constraints.

While we believe the primary advantages of EGP are realized in engineering problems where the underlying relation between the inputs and outputs are unknown, our studies on the analytical examples indicated the following three limitations. Firstly, the computational cost of training an EGP is higher than that of an ordinary GP. The additional cost is unfavorable if our algorithm results in an ordinary GP which happens when the underlying assumptions do not hold or when EP does not discover any informative bases. In this case, the output of EGP is an ordinary GP, but several expensive computations are performed that add no performance improvement compared to an ordinary GP. The second limitation is on interfacing with Eureqa which we use for symbolic regression. Eureqa is not open-source and hence each iteration of our algorithm involves (i) manual copy-pasting of the GP-generated data set into Eureqa, and (ii) writing sub-routines that automatically convert the results of Eureqa into Python-readable objects. While EGP generally converges in a few iterations, the manual data transfer is tedious and prone to error. We have tried open-source symbolic regression packages such as DEAP [60] which can be easily integrated with our codes. However, the symbolic regression power of Eureqa far exceeds other available

packages and thus we used Eureqa in our approach. The third limitation arises when the discovered bases regress the data very well which results in over confident extrapolation, see Figure 2.10.

Future research directions include extension to mixed-variable problems that include qualitative and quantitative inputs, improving interpolation/regression capabilities of GPs without the need for improving their extrapolation power, implementing an in-house symbolic regression package that can effectively interface with the rest of our algorithm in Python, and extensions to handle very large [72, 73] or high dimensional [74] datasets.

Chapter 3

Transferable Deep Learning Framework for solving PDEs in Arbitrary Domains

Partial differential equations (PDEs) are ubiquitously used in sciences and engineering to model physical phenomena. Three notable PDEs that have far-reaching applications are the Navier-Stokes (NS) equations which model the motion of viscous fluids [75] and Laplace equation which extensively appears in fluid dynamics, electrostatics, and steady-state heat transfer [76], and the wave equation present in many fields of the acoustics, electromagnetics and fluid dynamics. Solving such PDEs on large domains with arbitrary initial and boundary conditions (ICs and BCs) relies on numerical methods such as finite element (FE) [77] and finite difference (FD) [78]. While these methods are extremely powerful, they are computationally expensive. Because of these high costs, researchers face the challenge of drawing conclusions using a limited number of time-consuming studies. For instance, NASA's CFD vision 2030 [79] estimates that achieving a 24-hour turnaround time on a wall-modeled large eddy simulation of a full-wing at Reynolds $Re = 10^7$ requires 180 PFlops/s which is only

achievable on the most powerful supercomputer today. Such high costs make it infeasible to conduct compute-intensive studies such as uncertainty quantification (UQ) or airfoil shape optimization. To address this challenge, particularly in large-scale and inverse problems, simulations are augmented with inexpensive surrogates.

Although a wide range of surrogates such as Gaussian processes (GPs) [80, 81, 31] and trees [82, 83] are available, most recent applications of surrogate modeling employ deep neural networks (DNNs) [3, 4, 1, 5, 6, 7, 8, 9, 10, 11, 12]. The increasing use of DNNs in emulation is largely because they have extremely high latent representation power which enables them to distill highly nonlinear and hidden features and relations from a training dataset without explicit instructions. In addition, DNNs are highly scalable and versatile. For instance, while DNNs can naturally build regressors/interpolators from large and high-dimensional data, GPs rely on numerical methods such as low-rank matrix approximation [31] to handle datasets with more than ~ 5000 samples [15, 32]. Or, unlike trees, DNNs can easily handle both classification and regression/interpolation problems with high accuracy [1].

A major driver in using DNNs for surrogating PDE-governed systems that it allows to systematically and easily embed a system's governing equations in a network's training stage to build physics informed neural networks (PINNs) [84]. This inductive bias not only reduces the reliance on training data obtained from traditional solvers, but also increases the adherence of the DNN to the physics of the problem. The equation can be embeded using FD, but, more interestingly, they can be embedded using automatic differentiation (AD) [85] which dispenses with discretization errors (since AD is exact [86]). To demonstrate such a physics-informed training process, consider the $2D$ boundary value problem (BVP):

$$\begin{aligned} \nabla^2 u(\boldsymbol{x}) &= 0, \qquad \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{3.1}$$

where $\boldsymbol{x} = [x, y]$, ∇^2 is the Laplacian operator $(\partial^2/\partial x^2 + \partial^2/\partial y^2)$, Ω denotes the domain,

and $g(\boldsymbol{x})$ is the boundary value function. When training a DNN that surrogates $u(\boldsymbol{x})$, the loss function can be designed as:

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= L_1(\boldsymbol{\theta}) + \alpha L_2(\boldsymbol{\theta}) + \beta L_3(\boldsymbol{\theta}), \\
L_1(\boldsymbol{\theta}) &= \frac{1}{N_1} \sum_{i=1}^{N_1} \left(u(\boldsymbol{x}_i) - \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\theta}) \right)^2, \\
L_2(\boldsymbol{\theta}) &= \frac{1}{N_2} \sum_{j=1}^{N_2} \left(\nabla^2 \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\theta}) \right)^2, \\
L_3(\boldsymbol{\theta}) &= |\boldsymbol{\theta}|^2
\end{aligned}
\tag{3.2}
$$

where $\boldsymbol{\theta}$ are the parameters of the network (weights and biases), $\mathcal{N}$ is the DNN surrogate, $L_1(\boldsymbol{\theta})$ is the network's error in predicting the available data on the solution (these data can be on $\partial\Omega$ or in Ω), $L_2(\boldsymbol{\theta})$ is the residual error which enforces $\mathcal{N}$ to satisfy the Laplace equation on some randomly selected points (aka collocation points) in Ω, $L_3(\boldsymbol{\theta})$ denotes Tikhonov regularization that prevents overfitting, and α and β are constants that control the contributions of, respectively, $L_2(\boldsymbol{\theta})$ and $L_3(\boldsymbol{\theta})$ to $L(\boldsymbol{\theta})$. $L_2(\boldsymbol{\theta})$ in Equation 3.2 involves partial derivatives which can be analytically calculated via AD or approximated via FD.

Training a DNN that surrogates the solution of PDEs can be a time-consuming and chal-

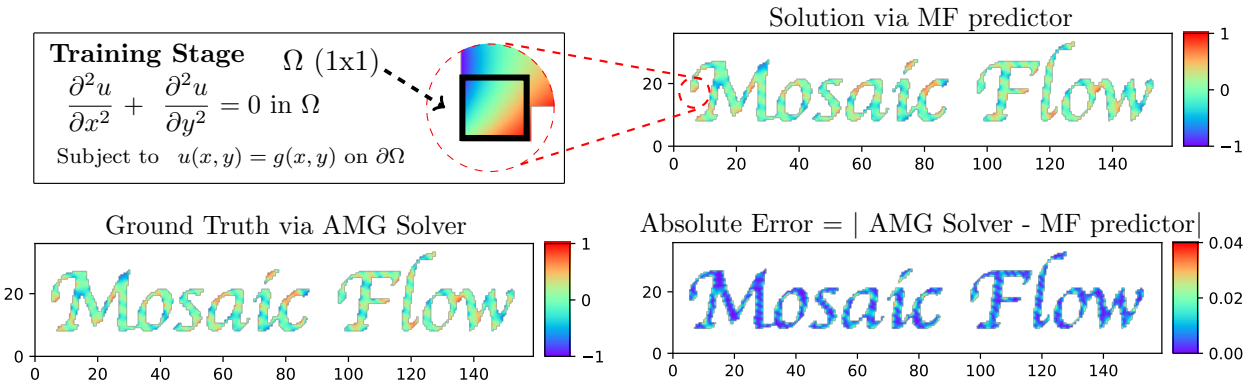


Figure 3.1 Transferable learning of a PDE: We train our network in a small domain of size 1×1 and use it to solve the Laplace equation in a domain that is $1200\times$ larger. Prediction time is about 20 minutes on a single NVIDIA V100 GPU.

lenging task primarily because (1) obtaining training data via high-fidelity simulations is expensive, (2) designing an efficient network architecture is an iterative process, (3) optimizing the network's parameters (such as weights and biases) is demanding, (4) calculating the residual loss is computationally costly because each time a differential operator is applied to the network, it almost doubles the number of computations associated with backpropagation [1, 87], and (5) choosing the optimizer's parameters (such as batch size and learning rate) as well as balancing the contributions of different loss components to the overall loss (e.g., α and β in Equation 3.2) are iterative processes. These challenges are exacerbated when training DNNs that need many parameters to accurately surrogate the solution of complex PDEs (such as the NS equations) in large domains.

We argue that the benefits of DNNs that surrogate PDE-governed systems outweigh the high costs and challenges associated with their training if the DNNs are *transferable*, in other words, if they can extrapolate. Unfortunately, existing methods [88, 89, 90, 91, 92] fail in this regard, i.e., there is currently no mechanism for estimating the solution of PDEs with a *pre-trained* DNN. For example, a PINN that surrogates $u(\boldsymbol{x})$ in Equation 3.1 would be largely useless if Ω and $g(\boldsymbol{x})$ change.

To bridge this gap, we develop a novel framework that builds a transferable DNN surrogate that solves time dependent PDEs in unseen domains with arbitrary BCs and ICs (see Figure 3.1). We present our idea in the context of steady flow prediction (i.e., surrogating the solution of the NS equations for steady flow) but note that the framework is readily applicable to both time and non-time dependent PDEs types such as the wave equation and Laplace equations.

For on-the-fly prediction of the flow in an unseen large domain with unseen ICs and/or BCs, we first decompose the domain into small subdomains called *genomes*. Then, we predict the flow in each genome with a pre-trained DNN, called *genomic flow network* (GFNet), such that the iterative assembly of these genome-wise predictions surrogates the flow in the large

domain. We denote this assembled flow as a *mosaic*. In essence, in the following section we make the following contributions:

- We introduce GFNet which can solve any Initial Conditions and Boundary Conditions Boundary Problem (IBVP) or any BVP with arbitrary conditions on a square domain called *genome*. We design GFNet's architecture based on the characteristics of the PDE and show that such a design significantly improves accuracy. For instance, GFNet outperforms the state-of-the-art in surrogating the solution of the Laplace equation by 2× (Section 3.2.2).

- We develop *mosaic flow* (MF) predictor, a novel algorithm that iteratively assembles GFNet's inferences to predict the solution of the PDE system for domains with unseen sizes, shapes, and BCs. We show that MF predictor can scale up GFNet's inferences by ∼1200×, 12× and 9× in the case of, respectively, Laplace, NS equations and the wave equation (Section 3.2.3).

- With GFNet and MF predictor, we eliminate the need to re-train a DNN for unseen domains or BCs. This transferability delivers 3-5 orders of magnitude speedup compared to the start-of-the-art PINN for solving the Laplace and NS equations while achieving comparable or better accuracy (Section 3.3).

- Our GFNets are not tailored to specific applications and hence can be used by others. This reusability saves energy and benefits researchers with limited access to GPUs.

3.1 Related Work

Existing approaches for building DNNs that solve PDEs employ a wide range of training mechanisms and network architectures. These choices largely depend on the application and

the characteristics of the PDEs. The architectures are typically built with fully connected (FC), convolutional, or recurrent [93, 94, 95] layers. In many works a combination of these layers, with or without residual connections [96], are employed as well. For instance, networks similar to Resnet [96] and UNet [97], whose building blocks are convolutional neural networks (CNNs), are generally the backbone of super-resolution frameworks that aim to reconstruct a high-resolution solution from a coarse solution [89, 98]. Such networks have also been trained in the Fourier space to learn a PDE operator using the modal space [91, 99]. CNNs, which may also include FC layers, are extensively used as surrogates for predicting steady laminar flows around 2D and 3D objects [100], estimating 2D flows around specific airfoils [101, 102], flow visualization [103], and many other applications [104, 105, 106, 107].

Solving PDEs via CNN-based networks is data-efficient (due to parameter sharing and learning spatially invariant kernels [3, 108, 109, 110]), but it requires developing a mechanism for interpolating the solution (and its gradients) on non-grid points [111]. One strategy to avoid such interpolation errors is to only employ FC layers. This strategy has been used for inverse estimation of PDE coefficients [84], inverse estimation of flow fields given observations on a passive scalar [87], predicting the Reynolds stress in Reynolds-averaged NS simulations [112, 113], and solving Poisson equation and eigenvalue problems [114]. Networks based on FC layers have also been reformulated for variational learning (to improve predictions on the boundaries) [115, 57], surrogating fractional PDEs (where AD cannot calculate residual errors) [116], emulating long time-dependent PDEs (where long temporal features render the training very difficult) [92], and learning operators that map functions to functions [117, 118, 119].

Regardless of the architecture, existing works lack *transferability*, i.e., they build models that are largely useless if the domain, the BCs or the ICs associated with the PDE system are modified. This important issue has been rarely addressed. DNNs that learn a PDE operator [120, 119], can in theory handle varying BCs and ICs but they are not domain-agnostic and

their applications thus far have been limited to simple PDEs over small, fixed domains. In [121], CNNs are used to solve the $2D$ Poisson equation on rectangular domains with different aspect ratios. In [122], the DNN is trained to solve the NS equations over backward-facing steps with varying heights. In [123], the DL model infers the lift and drag of various elliptic objects with different aspect ratios. In all these works, the geometry type (e.g., rectangle, ellipse) is fixed and only varies within a very limited range with one parameter. A few studies [124, 107, 101, 103, 102] have proposed more general techniques to embed geometry information into the training stage of a DNN such that it can predict the quantities of interests in unseen domains. However, the overall domain size is still fixed, grid data are required, and the variability of the objects is limited to specific applications.

Since the training costs of a PDE-solving DNN are generally high, parallel and distributed computations have been explored in their training [88]. However, the overall training costs are still high given the iterative and combinatorial process of tuning the architecture, optimization parameters, and loss components. To justify these high training costs, a DNN is expected to be transferable across multiple applications.

3.2 Transferable Learning of PDEs

Our goal is to surrogate the solution to the IBVP of the following homogeneous PDE in an *arbitrary* domain Ω given any BCs $g(\boldsymbol{x}, t)$ and ICs $\boldsymbol{f}(x)$:

$$\begin{aligned} &H(u(\boldsymbol{x}, t)) = 0, \quad \boldsymbol{x} \in \Omega,\ t > 0, \\ &u(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \partial\Omega,\ t > 0 \\ &\boldsymbol{F}(u(\boldsymbol{x}, t = 0)) = \boldsymbol{f}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \end{aligned} \tag{3.3}$$

where $H(\cdot)$ denotes a partial differential operator composed of spatial and time derivatives. Note that this problem formulation encloses non time dependent PDEs, yielding to the well known BVP problem:

$$\begin{aligned} H(u(\boldsymbol{x})) &= 0, \quad \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}, t) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{3.4}$$

where $H(\cdot)$, here, denotes a partial differential operator composed only of spatial derivatives. For notational convenience, we drop the dependence of u to $\boldsymbol{x}$ and t and we highlight that while Equation 3.3 is scalar, our approach is directly applicable to PDE systems such as the NS equations, see Section 3.3.2.

Building a *single* DNN that solves the IBVP of Equation 3.3 in *any* domain and under *any* BCs and ICs is an extremely difficult (if not infeasible) learning task because considering arbitrarily large variations in domains, BCs and ICs in training drastically amplifies the challenges described in the introduction of the chapter. In our framework, we solve this difficult task by decomposing it into two simpler (but highly coupled) tasks that correspond to (1) training a GFNet that solves 3.3 and 3.4 in a genome (i.e., a small square) given any $\boldsymbol{f}$ and/or g, and (2) building MF predictorthat surrogates the solution in any given large domain by stitching the genome-wise predictions made by the pre-trained GFNet. The advantage of our approach is that predicting the solution of the PDE in any new, unseen domain only relies on task 2, i.e., there is no need to train or re-train a DNN anymore.

Our task decomposition strategy is motivated by curriculum learning [125] which was originally developed for animal training [126] where difficult tasks are divided into simpler tasks which are easier to learn and collectively make up the original task. In Section 3.2.1 we demonstrate that this approach is applicable to solving BVPs, and extend this results to time dependent PDEs and IBVP. Then, we elaborate on how to build a GFNet and an MF predictorin Sections 3.2.2 and 3.2.3, respectively.

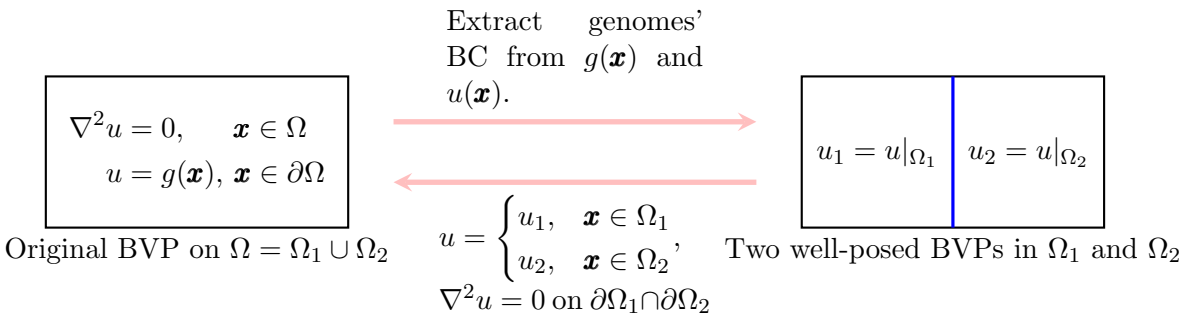
3.2.1 BVP and IBVP Decomposition: Why Mosaic Works?

The IBVP in Equation 3.3 is well-posed if the solution, u, is unique. For such a configuration, u on any arbitrary sub-domain inside Ω, inheriting the corresponding ICs and BCs, also forms a well-posed IBVP.

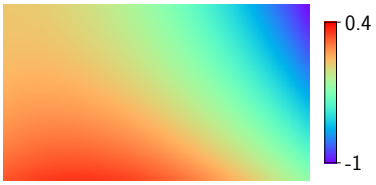
To illustrate this, let us drop the time dependence and consider the simpler BVP. As shown in Figure 3.2a for the Laplace equation, the domain Ω is split into two genomes, namely, Ω_1 and Ω_2. The boundaries of these genomes are denoted by $\partial\Omega_1$ and $\partial\Omega_2$ and they verify that $\partial\Omega_1 \in \Omega_2$ and $\partial\Omega_2 \in \Omega_1$, so the domains meet in the middle of Ω (marked blue in the figure). Once the Laplace equation is solved in Ω, u will be available on $\partial\Omega_1$ and $\partial\Omega_2$. We can now solve two BVPs: one in Ω_1 and the other in Ω_2 to get the solutions u_1 and u_2, respectively. The above well-posedness relation indicates $u_1 = u|_{\Omega_1}$ and $u_2 = u|_{\Omega_2}$.

Conversely, if we can find u_1 and u_2 while ensuring that the PDE is satisfied on the shared region, i.e., $\nabla^2 u = 0$ on $\Omega_1 \cap \Omega_2$, then we will have u since $u = u_1 \cup u_2$. Indeed, to find u_1 and u_2 we need to solve two BVPs (one in Ω_1 and the other in Ω_2) both of which require u on $\partial\Omega_1$ and $\partial\Omega_2$. Since u is unknown, we propose an iterative strategy to find it on the shared region: we first randomly assign values to u on $\partial\Omega_1$ and $\partial\Omega_2$ and solve two BVPs in Ω_1 and Ω_2 to find u_1 and u_2, respectively. Then, we use u_1 and u_2 to update our initial guess on the opposite borders and repeat this process until a convergence criterion is met (e.g., until values on $\partial\Omega_1$ and $\partial\Omega_2$ negligibly change across consecutive iterations). Our framework is similar to this approach except that in all iterations we use a pre-trained GFNet to (1) quickly find the solution in each genome (i.e., u_1 and u_2), and (2) update the solution on the inner borders.

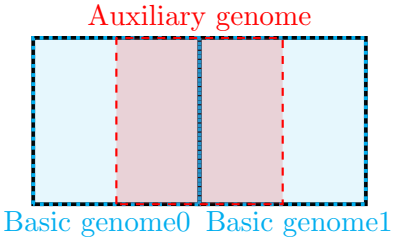
The rationale behind using GFNet for updating the shared values is that it greatly accelerates convergence by increasing the pace of information propagation from $\partial\Omega$ to Ω. We illustrate this property by solving the BVP in Equation 3.1 in a domain of size 2×1 (i.e., $\Omega =$



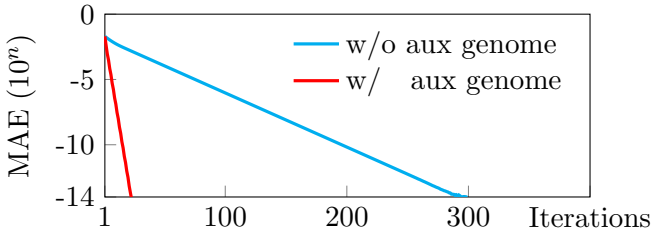
(a) Interchangeability between a BVP and its genomes' BVPs



(b) Direct solution via AMG



(c) Domain decomposition



(d) Convergence history in log-scale

Figure 3.2 Decomposition of a BVP: The Laplace equation is solved in $\Omega = [0, 2] \times [0, 1]$ with three methods: a direct and two iterative approaches. The iterative approach that leverages an auxiliary genome converges to the direct solution 12× faster.

$[0, 2] \times [0, 1]$) subject to the boundary condition $g(\boldsymbol{x}) = x^2 - y^2 + xy$. We first use algebraic multi-grid (AMG) [127] to directly find u in Ω, see Figure 3.2b. Then, we use AMG within our iterative approach in two strategies and show that, while both strategies converge to the true solution obtained by the direct approach, one is significantly faster.

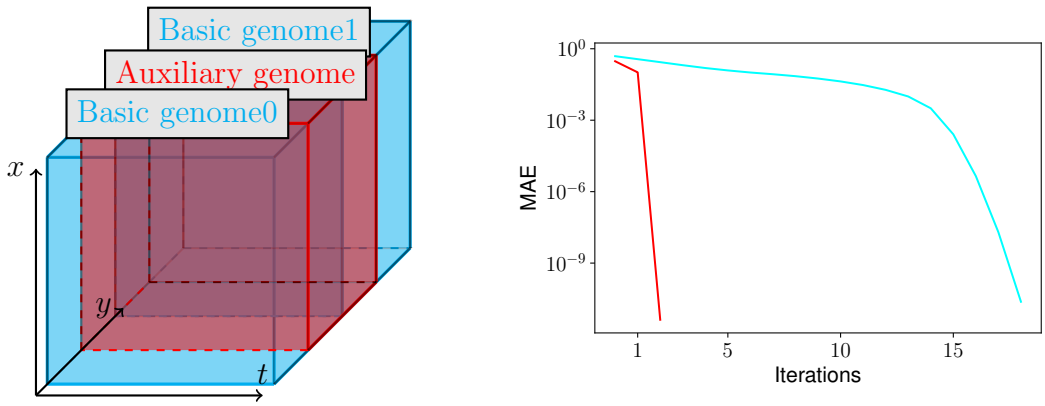
Since Ω is of size 2×1, we decompose the domain into two *basic* genomes of size $1 + \delta \times 1$ that share an infinitesimal non empty region defined by $\delta > 0$, see Figure 3.2c. In the first iterative strategy, we initialize the inner borders with zero values for u, solve for u_1 and u_2 using AMG, and exchange data across the borders. This iterative process converges in 300 iterations when the mean absolute error (MAE) compared to the direct solution drops below

1e-14.

The second iterative strategy differs from the former in the update step: after finding u_1 and u_2, we update u on the shared border by solving the Laplace equation in an *auxiliary* genome whose BCs on its two vertical borders are inside Ω_1 and Ω_2, see Figure 3.2c. Note that we do not need $\delta > 0$ in these scenarios. This approach introduces a stronger and more effective coupling across the basic genomes which, in turn, accelerates the rate of information propagation from $\partial\Omega$ to the shared border. The higher rate of information propagation increases the convergence rate by almost 12 times compared to the first iterative strategy, see Figure 3.2d.

Our framework is akin to the second iterative strategy, except that we always use a single GFNet to predict the solution in either the basic genomesor the auxiliary genomes.

We demonstrated that this strategy can successfully by applied to solve BVP problems, but the same approach can used to solve IBVP. In this case, the updates need to be on the shared border of the basic genomes for any time t. To showcase that the same principles apply, let



(a) Domain decomposition **(b)** Convergence history in log scale

Figure 3.3 Decomposition of a time dependent PDEs: The Wave equation is solved in $\Omega = [0, 2] \times [0, 1]$ and $t \in [0, 2]$ using basic and auxiliary genomes solved with finite differences.

us consider a particular IBVP for the wave equation

$$\begin{aligned} u(\boldsymbol{x},t)_{tt} - c \cdot \nabla^2 u(\boldsymbol{x},t) = 0, \qquad & \boldsymbol{x} \in \Omega, t > 0, \\ u(\boldsymbol{x},t) = g(\boldsymbol{x},t) \quad & \boldsymbol{x} \in \partial\Omega, t > 0 \\ u_t(\boldsymbol{x},t) = f_1(\boldsymbol{x}), \quad & \boldsymbol{x} \in \Omega, t = 0 \\ u(\boldsymbol{x},t=0) = f_2(\boldsymbol{x}) \quad & \boldsymbol{x} \in \Omega, t = 0 \end{aligned} \tag{3.5}$$

with $c = 1$, $f_1(\boldsymbol{x}) = 0$, $f_1(\boldsymbol{x}) = 0$, $g(\boldsymbol{x},t) = x^2 + y^2 + x \cdot y \cdot \sin(5t)$, and $f_2(\boldsymbol{x}) = x^2 + y^2$. We use the same two strategies to solve it in the domain $\Omega = [0,2] \times [0,1]$ for time in $T = \{t | t \in [0,2]\}$. First, we decompose the larger domain into subdomains, $\Omega_1 \times T = [0, 1+\delta] \times [0,1] \times [0,2]$ and $\Omega_2 \times T = [1-\delta, 2] \times [0,1] \times [0,2]$ (see Figure 3.3a). Then, we solve each Ω_i for $t \in [0,2]$ and iteratively update the values of $g(\boldsymbol{x},t)$ for all $t \in T$ at $\partial\Omega_1$ and $\partial\Omega_2$, like in the Laplace example. As shown in Figure 3.3b, and as we are solving two well posed problems, this will converge to the actual solution when for any t the values of u on $\partial\Omega_1$ and $\partial\Omega_2$ negligibly change across consecutive iterations. Like its analogous BVP, if this strategy is used with auxiliary genomes, it further increases the convergence speed, see Figure 3.3a for the domain decomposition and 3.3b for the speed convergence comparison.

3.2.2 Genomic Flow Network (GFNet)

The iterative nature of MF predictor requires a GFNet to be able to estimate the PDE solution inside a genome for a wide range of BCs and ICs. Thus, the inputs must present the following configuration:

$$\underbrace{g(\boldsymbol{x}_1^{bc}, t_1),\ g(\boldsymbol{x}_2^{bc}, t_1),\ \cdots,\ g(\boldsymbol{x}_{N_{bc}}^{bc}, t_1), \ldots, g(\boldsymbol{x}_1^{bc}, t_n),\ g(\boldsymbol{x}_2^{bc}, t_n),\ \cdots,\ g(\boldsymbol{x}_{N_{bc}}^{bc}, t_n)}_{\boldsymbol{g} \text{ of size } N_{var} \times N_{bc} \times N_{time}},$$

$$\underbrace{u(x_1^{int}, t=0),\ \ldots,\ , u(x_N^{int}{}_{IC})}_{u \text{ of size } N_{out} \times N_{IC}},\ \underbrace{f_1(x_1^{int_1}),\ \ldots\ , f_1(x_N^{int_1}{}_{IC_1}, t=0)}_{f_1 \text{ of size } N_{out} \times N_{IC_1}},\ \ldots,$$

$$\underbrace{f_k(x_1^{int_k}),\ \ldots\ , f_k(x_N^{int_k}{}_{IC_k})}_{f_k \text{ of size } N_{out} \times N_{IC_k}},\ \underbrace{\boldsymbol{x}}_{N_{dim}},\ t$$

where N_{bc}, N_{var}, N_{time}, N_{dim}, N_{IC}, N_{IC_i} and N_{out}, denote the number of points on the genome boundary for a fixed time, the number of points on the boundary for a fixed spatial coordinate, the number of variables, the spatial dimensions, the number of the interior points for the initial conditions and for its ith time derivative, and the number of outputs variables. $\boldsymbol{x}$ and t indicate the coordinates and time where GFNet predicts the solution.

The above inputs include discretized versions of $g(\boldsymbol{x}, t)$, $u(\boldsymbol{x}, t = 0)$, and $f_k(\boldsymbol{x})$ both in the spatial and time dimensions. For the discretization of $g(\boldsymbol{x}, t)$, note that $t_1, \ldots . t_n = T$ discretizes the interval $(0, T]$ and the predicted time t must verify that $t \leq T$.

For the BVP problem on non time dependent PDEs, the inputs can be simplified as follows, if the time dependence is dropped,

$$\underbrace{g(\boldsymbol{x}_1^{bc}),\ g(\boldsymbol{x}_2^{bc}),\ \cdots,\ g(\boldsymbol{x}_{N_{bc}}^{bc})}_{\boldsymbol{g} \text{ of size } N_{var} \times N_{bc}},\ \underbrace{\boldsymbol{x}}_{N_{dim}}$$

In both problems, the necessity of discretizing the borders comes with the possibility of discretizing the whole domain, using the sampled points of the input to build a gird. Hence, we can solve the PDE and obtain a discrete output on all the grid points inside the domain, getting rid of the inputs $\boldsymbol{x}, t$. This is discussed in Section 3.3 as it simplifies the training and favors the model performance.

The loss function of GFNet is adopted from Equation 3.2 where the size of the training data (e.g., N_1 and N_2) and coefficients (e.g., α and β) are chosen based on the PDE and the GFNet architecture. For instance, our studies indicate that while residual loss improves the accuracy of some GFNets when learning the Laplace equation, it always reduces the accuracy in the case of the NS equations, see Section 3.3 for the details and our reasoning. When residual errors improve the accuracy of GFNet, we introduce a mechanism to adaptively (instead of randomly) choose the location of collocation points in the genome.

We will discuss and study different architectures in order to refine the GFNet because the subsequent steps are greatly conditioned by its accuracy.

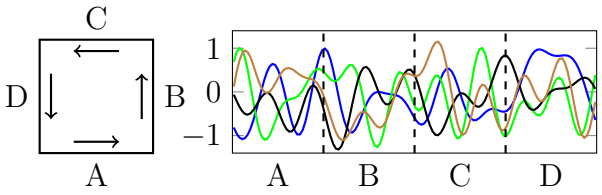
Since a GFNet takes $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k$ and $\boldsymbol{g}$ as an inputs, a wide range of well-posed (and ideally negligibly correlated) ICs and BCs must be used in training. For linear PDEs such as the Laplace equation and the wave equation, the PDE has a unique solution as long as the boundary function $g(\boldsymbol{x})$ and the initial conditions $f_i(\boldsymbol{x})$ are smooth, i.e., its derivatives in $H(\cdot)$ exist. Hence, for the Laplace equation we use GPs to generate a smooth function in $1D$ and then wrap that function around the perimeter of the genome, see Figure 3.4a. For the wave equation we use GPs to generate a $3D$ smooth function where we sample the ICs and BCs for any time t. GPs, widely discussed in Chapter 2, are stochastic processes that place a prior distribution on functions. By judiciously selecting the hyperparameters of their kernel, we can generate a wide range of smooth functions that are minimally correlated. For highly nonlinear PDEs such as the NS equations, the BCs must satisfy mass, momentum, and energy conservation laws which makes it highly challenging to create well-posed BCs

via a GP. Hence, we first solve the PDEs with realistic BCs on a large domain and then extract solutions for small genomes embedded in the domain. This strategy is demonstrated in Figure 3.4a where the flow in a lid-driven cavity is swept by a genome while recording the flow on the boundary as well as inside the genome (the cavity's domain spans $[0, 1] \times [0, 1]$ while the genome has a size of 0.5×0.5). As argued in Section 3.2.1, this procedure generates valid data since the flow on each genome's boundary constitutes the BCs for the BVP on that genome.

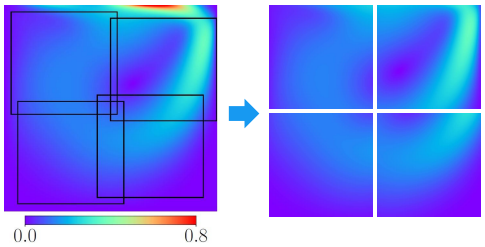
Note that the genomes in training and prediction have the same shape because GFNet does not learn the effect of the genome's shape. We can address this limitation by including the boundary's geometrical information as input but this is beyond the scope of this work.

3.2.3 Mosaic Flow Predictor

As detailed in Section 3.2.1 we can (1) convert a IBVP in a large domain to multiple IBVPs in genomes that cover that domain and then solve these IBVPs iteratively, and (2) employ auxiliary genomes to significantly accelerate the convergence. Based on the above two attributes, MF predictor decomposes an unseen domain into two sets of genomes, see Figure 3.4b. The first set consists of basic genomes that cover the entire domain and can overlap with each other. The BCs on the borders of basic genomes are either known (if the borders lie on the large domain's boundary) or must be inferred via MF predictor. The second set consists of auxiliary genomes that overlap with the basic genomes to accelerate the propagation of boundary information into the domain, i.e., to speed up the iterative process of estimating the unknown BCs of the basic genomes. We distribute these two sets of genomes based on the following general guidelines. First, we arrange the basic genomes to cover the entire domain with as little overlap as possible ($g1$ through $g4$ in Figure 3.4b). Then, we place auxiliary genomes on the vertical and horizontal shared borders of the basic

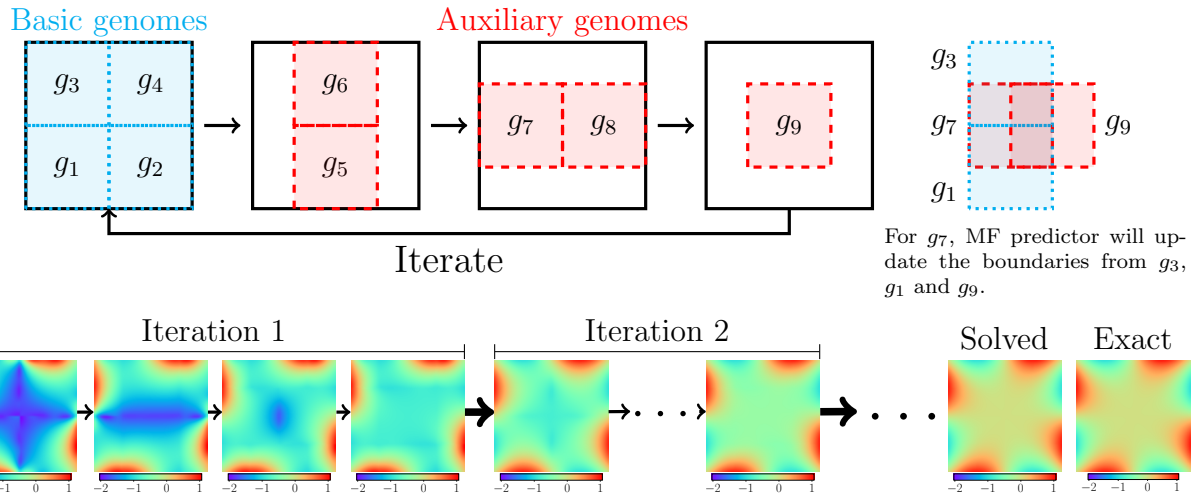


Method 1: sample genome's BCs with GP.



Method 2: extract genomes from flow solutions.

(a) Step 1: Data generation



(b) Step 3: Iterative nature of MF predictor

Figure 3.4 Primary steps of our framework for transferable PDE learning: Steps 1 and 2 are done only once and their cost is fully amortized in step 3 which can be applied to unseen domains. (*a*) Training data are generated by either solving a BVP whose BCs are sampled from a GP, or by sweeping the solution of the BVP in a large domain with a genome (the plot in method 2 corresponds to the velocity magnitude in a lid-driven cavity simulation). (*b*) Mosaic flow (MF) predictor estimates the solution in an unseen domain by systematically coupling the predictions of a GFNet on basic and auxiliary genomes. Note that basic genomescan overlap with each other and auxiliary genomescan be placed anywhere inside the domain (we have avoided these scenarios to improve clarity).

genomes ($g5$ through $g9$ in Figure 3.4b). Finally, we place an additional auxiliary genome on every corner/region where four basic genomes connect/overlap ($g9$ in Figure 3.4b). We have developed this spatial genome distribution to balance the pace of information transfer (which directly affects convergence speed) and implementation efforts. As we demonstrate in Section 3.3.2, for highly nonlinear PDEs such as the NS equations, more auxiliary genomes can facilitate information propagation when the GFNet fails to do so. In such cases, following the above procedure, we place additional overlapping auxiliary genomes on the shared borders of other auxiliary genomes.

To solve an arbitrary IBVP, MF predictor finds the solution in the genomes set by set using a pre-trained GFNet and iterates until convergence. The basic genomes' borders are initialized with some fixed ICs either with the domain's BCs if they lie on the domain's boundary or with 0 if they lie between basic genomes. The predictions in basic genomes provide the BCs for auxiliary genomes and then the predictions in auxiliary genomes update the unknown BCs of the basic genomes. In other words, one iteration of MF predictor is completed when GFNet is applied to all the genomes exactly once. For non-time dependent PDEs the iterations stop when the change in the inferred BCs of basic genomes is smaller than a user-defined tolerance ϵ. For time dependent PDEs, MF predictor only solves for a given time window T, dependent on the training and the inputs. Hence, once the solution for the first time window T converges, the MF predictor uses it as ICs for the next time window, inferring for any time $t > T$.

Figure 3.4b illustrates the spatial distribution of basic and auxiliary genomes and the iterative nature of MF predictor for solving the Laplace equation in $\Omega = [0, 2] \times [0, 2]$ with genomes of size 1×1. In this figure, there are 4 basic genomes that, without any overlap, cover Ω. The BCs are only known on two sides of the basic genomes and hence the MF predictor aims to predict the unknown BCs on the other two sides. There are also 5 auxiliary genomes that accelerate the pace of inferring the unknown BCs of the basic genomes. In this example, MF

predictor converges to the exact solution in 18 iterations.

3.3 Solving PDEs with MF predictor

We showcase both of our approaches, GFNet and MF predictor, by solving the linear Laplace equation, the non-linear incompressible NS equations and the wave equation in Sections 3.3.1, 3.3.2, and 3.3.3, respectively. In each section, we present the details on data generation, the different GFNet architectures considered and its accuracy, and the accuracy and performance (used to refer to computational costs) when the designed GFNets are used in MF predictor. For the Laplace and NS equation, we compare our approach against the state-of-the-art PINNs. Finally, we introduce three extra procedures to update the values on the shared borders.

We use Tensorflow [128] and ADAM [129] optimizer. We implement GFNet with TensorFlow 2 (2.4.0) whereas the PINN related sources [84, 130] used for comparison are built with TensorFlow 1 (1.8.0).

3.3.1 Laplace Equation

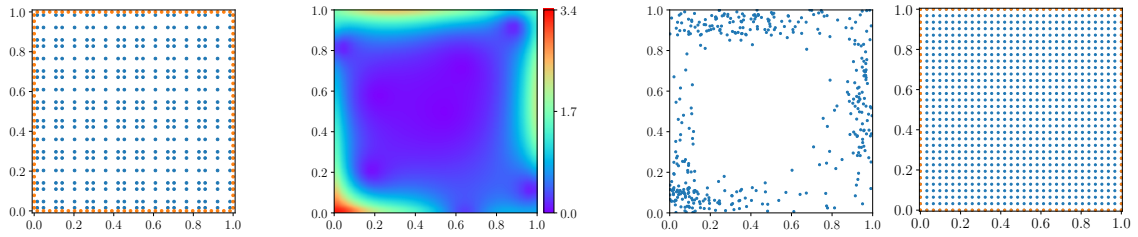
Laplace equation is a BVP that ubiquitously appears in fluid dynamics and heat transfer [76], see Equation 3.1. In this section, we learn the $2D$ Laplacian operator via a GFNet in a unit square domain, i.e., the genome covers $[0, 1] \times [0, 1]$. Then, we use MF predictorto solve the Laplace equation in domains that are up to 1200 times larger and we compare the accuracy and performance of GFNet and MF predictoragainst PINN [84] and XPINN [88]. Finally, we introduce an optimization variant of MF predictor

Training Data.

As described in Section 3.2.2, we use GPs to generate BCs. In particular, we use Sobol sequence [67] to sample the hyperparameters of the Gaussian kernel of a $1D$ GP. Then, we draw a sample function (i.e., a $1D$ curve) from each GP and wrap it around the genome, see Figure 3.6a. Finally, for each BC, we numerically solve the Laplace equation via PyAMG [127] which relies on FD to provide the solution on two different settings. First, a uniform grid that consists of 32×32 cells and 128 boundary points (dataset 1), see Figure 3.5a, and second, a uniform grid of 31×31 cells sampled at the corner of the cells (dataset 2), see Figure 3.5d. In models trained with dataset 2, we do not include the coordinates as inputs and we enforce the Laplacian via FD. In models trained with dataset 1, we include the coordinates as inputs and AD is used to enforce the PDE.

Architecture and Training of GFNet.

For the GFNet, we consider a total of 4 different architectures that we group in two categories depending on the inputs of the network. For the first category (category I), trained with dataset 1, the GFNet's inputs include $\boldsymbol{x}$ and the discretized boundary value



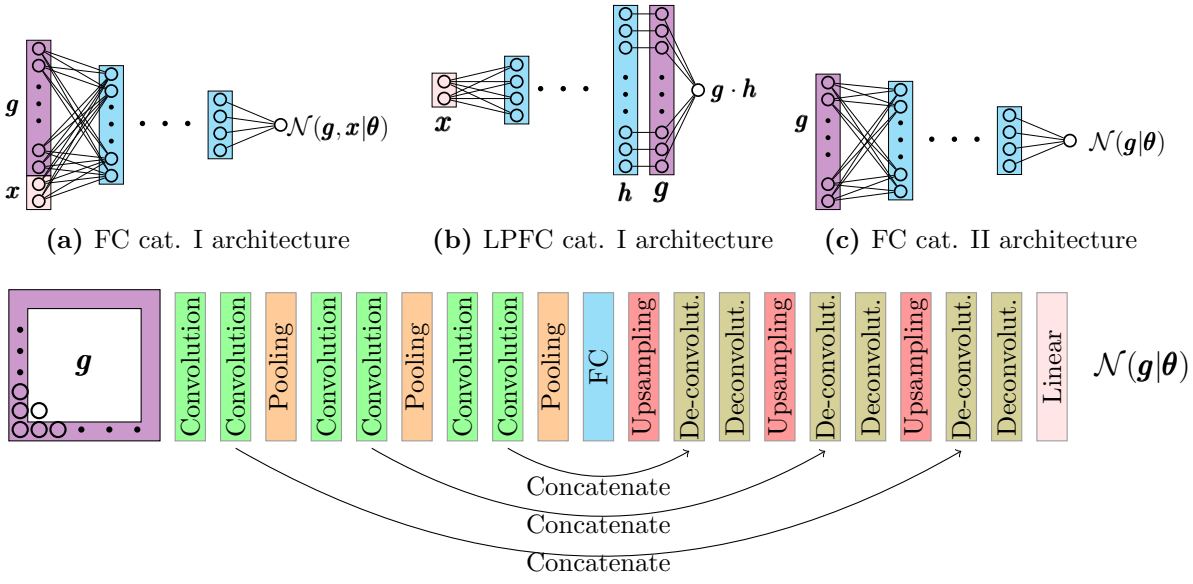
(a) Boundary and data points, first dataset

(b) GFNet's gradient magnitude, $|\nabla \mathcal{N}|$

(c) Collocation points

(d) Boundary and data points, second dataset

Figure 3.5 Distribution of training data: (a-b-c) Data distribution for dataset 1. There are 128 boundary points, 400 data points, and 400 collocation points. The locations of boundary and data points are fixed during training while those of the collocation points can adaptively change based on $|\nabla \mathcal{N}|$. (d) Data distribution for dataset 2. There are 124 boundary points, 900 data points, and 900 collocation points.



(a) FC cat. I architecture **(b)** LPFC cat. I architecture **(c)** FC cat. II architecture

(d) UNC cat. II architecture

Figure 3.6 Different architectures studied for the Laplace Equation: Architectures (a) and (b) fall into category I, specifying the coordinates for the output. Architectures (c) and (d) fall into category II, providing the output for the whole domain. Architecture (d) is a fixed architecture based on the U-Net convolutions architecture. Every convolution doubles the number of filters and every de-convolution reduces them by half. The pooling layer uses average pooling of size 2 and the upsampling uses bilinear sampling.

function $\boldsymbol{g}$ which in this case has $N_{bc} = 128$ points. Since Laplace equation has one scalar variable (i.e., $N_{var} = 1$), GFNet has a total of $N_{bc} \times N_{var} + N_{dim} = 130$ inputs. The output of GFNet is a single value $\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta})$ that approximates $u(\boldsymbol{x})$. This architectures have the advantage that they can solve for any point inside the domain and that AD can be used to enforce the PDE.

In category I, we study the following architectures:

- *Fully Connected (FC)* architecture where $\boldsymbol{g}$ and $\boldsymbol{x}$ are concatenated into one input vector and passed through all the FC hidden layers, see Figure 3.6a. We describe the structure of the network as $k^i \otimes l^j$, which denotes i FC layers of size k followed by j FC layers of size l.

- *Linearity-preserving fully connected(LPFC)* architecture, a novel architecture that pre-

serves the linearity of the Laplace equation. In LPFC, we pass the coordinates of the input point $\boldsymbol{x}$ through multiple fully connected hidden layers and set the last hidden layer's outputs, $\boldsymbol{h}$, to have the same size as $\boldsymbol{g}$. The output of GFNet is a linear combination of $\boldsymbol{g}$ with $\boldsymbol{h}$, i.e., $\boldsymbol{g} \cdot \boldsymbol{h}$. With this configuration, $\boldsymbol{h}$ leverages the spatial correlations in Ω and $\boldsymbol{g} \cdot \boldsymbol{h}$ enforces it to conform to the linearity of the PDE. See Figure 3.6a.

We choose tanh as the activation function for both FC and LPFC architectures. The loss function of GFNet is defined in Equation 3.2 and uses $\beta = 0$ and α =1e-3. When minimizing this loss, three data types are used in each epoch: 128 boundary points, 400 data points (to reduce training costs, we do not use all the 32×32 points where PyAMG provides the solution), and 400 collocation points.

While the location of boundary and data points are determined by the dataset 1 (and hence fixed), we adaptively choose the location of collocation points during training based on the spatial gradients that GFNet estimates inside the genome. These gradients are estimated via AD and their magnitude, $|\nabla \mathcal{N}|$, controls the spatial distribution of the collocation points by increasing the point density in regions where $|\nabla \mathcal{N}|$ is large. Figure 3.5b demonstrates the spatial distribution of these three data types in one randomly selected epoch during training. In this figure, boundary and data points are on a grid (since PyAGM generates them based on FD) while collocation points concentrate near the genome's border where the solution tends to change sharply.

For the second category of models (category II), trained with dataset 2, the GFNets' inputs include only the discretized boundary value function $\boldsymbol{g}$ which in this case has $N_{bc} = 124$ points. The outputs of GFNet are the approximations of $u(\frac{i}{31}, \frac{j}{31})$ for all $i, j \in \mathbb{Z}$ and $0 \leq i, j \leq 31$. The main advantages of category II architectures can be found on its training simplicity, as we can easily include multiple solutions in a single batch. The training of category I architectures is limited because to include a single solution we need to include

all the 400 used data points. For example, for a batch size of 10 solutions, category I architectures needs 4000 different training samples while categroy II only requires 10 samples. This improved training provides better accuracy and increased performance in prediction. As AD can not be used in this coordinate-less architectures, the PDE is enforced by FD approximations of the spatial derivatives. In category II, we study the following architectures:

- *Fully Connected (FC)* architecture where $\boldsymbol{g}$ is passed through all the FC hidden layers, see Figure 3.6c.
- *U-Net convolutional (UNC)* architecture inspired in the U-Net architecture [131]. In this architecture, the inputs $\boldsymbol{g}$ are considered as an image, padding with zeros all the values inside of the domain. The inputs are passed through a series of convolutions and max pooling layers to extract features. This features are then passed through several FC hidden layers that preserve dimensionality. Finally, the domain is reconstructed by a series of de-convolutions and upsampling layers. See Figure 3.6d.

We choose *swish* as the activation function for both FC category II and UNC architectures. The loss function of GFNet is defined in Equation 3.2 and uses $\beta = 1e-15$ and α =1e-6 or α =0 . When minimizing this loss, we use all the data points in the grid. This corresponds to 124 boundary points, 900 data points and 900 collocation points evaluated using FD.

Accuracy for Unseen BCs.

Both categories of architectures are trained with 8000 Laplace solutions of the corresponding dataset. For each, we test several combinations of hyper-parameters and fine tune the individual models. The best architectures found are the following. For the FC category I, the best architecture consists of 14 hidden layers of sizes $32^2 \otimes 64^2 \otimes 96^5 \otimes 128^5$ which make a cone as shown in Figure 3.6a. The inverse the order of these hidden layers is used to get

the best architecture for LPFC. For the FC category II, the best architecture consists of 5 hidden layers of sizes $100^5 \otimes 1024$, which greatly reduces the number of parameters needed to learn the operator. Regarding the UNC architectures, the best model follows the structure shown in 3.6d. The initial number of filters for the convolutions is 16, which is doubled after every two convolutions, and the kernel size used is 3. The FC structure of the UNC consists of 4 hidden layers of sizes $100^3 \otimes 64$. The FC output is later de-convoluted to its original shape using transpose convolutions of kernel size 3 that reduces the number of filters by half every two de-convolutinos.

The resulting GFNets are tested on 900 unseen Laplace solutions generated by PyAMG where the MAE, MAR or the MSR (mean squared error for the residual) are evaluated at all the 32×32 grid points that compose the grid of dataset 2.

From Table 3.1 we observe some interesting conclusions. First, the more samples used in training, the more accuracy the model presents. However, this behavior is not linear and increasing the number of samples further than 8000 produces from few to non gain. Second, category II architectures present a more favorable training. Not only we can use a bigger batch size with the same computing resources but this models can be trained using residual loss only, which eliminates the need for data generation. We were unsuccessful when trying to train category I architectures with only residual loss. However, only when data points are used in training we reach the best accuracy. Third, and regarding category I architectures only, we observe that LPFC learns the Laplacian operator more accurately than FC category I by one order of magnitude. This superior accuracy is because FC applies a non-linear activation function (tanh) to $\boldsymbol{g}$ which conflicts with the linearity of the Laplace equation. On the contrary, LPFC applies the non-linear activation function only to $\boldsymbol{x}$ and retains the linearity of the Laplacian operator. Fourth, the overall best model by half an order of magnitude, is the UNC architecture. Therefore, we select UNC architecture trained without collocation points as the GFNet for the final model in MF predictor.

Network	Precision	Training			Test on unseen BCs		
		Arch.	Samples	Points	MAE	MAR	MSR
FC Cat. I	Single	A	2000	100\400	2.64e-3	1.72e-1	-
FC Cat. I	Single	B	4000	400\400	1.30e-3	7.12e-2	-
FC Cat. I	Single	B	8000	400\400	8.79e-3	4.17e-2	-
FC Cat. I	Double	B	8000	400\400	8.12e-4	3.84e-2	-
LPFC	Single	A	2000	100\400	5.42e-4	2.33e-2	-
LPFC	Single	B	4000	400\400	4.33e-4	2.05e-2	-
LPFC	Single	B	8000	400\400	4.10e-4	1.30e-2	-
LPFC	Double	B	8000	400\400	4.07e-4	1.51e-2	-
LPFC	Single	B	18000	400\400	4.10e-4	1.46e-2	-
LPFC	Double	B	18000	400\400	4.10e-4	1.46e-2	-
FC Cat. II	Single	C	8000	900\900	3.48e-4	-	1.94e-6
FC Cat. II	Single	D	8000	900\0	3.06e-4	-	2.43e-1
FC Cat. II	Single	E	8000	900\0	4.28e-4	-	8.19e-1
FC Cat. II	Single	F	8000	0\900	4.49e-2	-	1.99e+1
UNC	Single	G	8000	900\900	1.29e-3	-	1.80e-2
UNC	Single	H	8000	0\900	2.06e-3	-	3.32e+0
UNC	Single	G	8000	900\0	8.28e-5	-	6.27e-2

Table 3.1 Accuracy of different GFNets on learning the Laplacian operator: For Category I architectures, LPFC architecture learns the Laplacian operator more accurately than FC. For the Category II, overall more accurate than Category I, UNC outperforms FC. Different models configurations were tested, but only the best results are reported. The training column includes the number of samples used, data/collocation points, and the *alpha* value used in the loss. The test column reports the MAE and the mean absolute residual (MAR) or the mean squared residual (MSR) which are calculated by evaluating GFNet's predictions against PyAMG [127]. $A = 32^2 \otimes 64^2 \otimes 96^2 \otimes 128^2$, $B = 32^2 \otimes 64^2 \otimes 96^5 \otimes 128^5$, $C = 100^5 \otimes 1024$, $D = 200^5 \otimes 1024$, $E = 150^4 \otimes 1024$, $F = 128^3 \otimes 1024$, G = U-Net with 100^3, H = U-Net with 100^7

For all the architectures, further increasing the number of hidden layers or units did not noticeably affect accuracy.

We also test single and double-precision computations but no significant difference in accuracy is observed for GFNet (see Table 3.1). Since the former is 17% faster when used with MF predictor(see Figure 3.7), we choose the model trained with single-precision.

Accuracy on Unseen Domains Subject to Unseen BCs.

We first evaluate MF predictor on square domains of area $A > 1$ with two boundary functions $g_1(s) = \sin(2\pi s/\sqrt{A})$ and $g_2(s) = \sin(2\pi s)$ where s parameterizes the boundary. We gradually increase A and execute MF predictor using the best GFNets for each category. The accuracy of MF predictor is measured by the MAE between the converged solution and the ground truth generated with PyAMG. Figure 3.7 summarizes the results.

At $A = 1$, the error is solely due to GFNet (i.e., inferring the solution for an unseen BC). For both $g_1(s)$ and $g_2(s)$, the MAE spikes at $A = 2 \times 2$ as it starts to include additional error accumulated during the iterative process in MF predictor. However, as A increases, the influence of BC on the inner region of the domain reduces, and therefore MF predictor can more accurately predict the solution. As a result, MAE decays and plateaus once $A > 4 \times 4$.

The converged MAEs for $g_1(s)$ and $g_2(s)$ are different. For $g_1(s)$, the boundary function oscillates less across the domain boundary as A increases which simplifies the genome-wise BVP compared to $A = 1$. Therefore, at $A = 8 \times 8$, even with the additional accumulated error due to MF predictor, the MAE is less than that at $A = 1$. However, for $g_2(s)$ the oscillation frequency does not depend on A and hence the MAE at $A = 8 \times 8$.

We also demonstrate the transferability and scalability of MF predictorby solving the Laplace equation in a domain that resembles "Mosaic Flow" and is 1222.5× larger than the training domain (i.e., a genome), see Figure 3.1. The prescribed BC is $g(\boldsymbol{x}) = \sin(2\pi(x/6 + y/5))$ and 2020 genomes (basic and auxiliary) are used. MF predictor with LPFC converges in 9821 seconds with an MAE of 2.61e-3 and MF predictor with UNC converges in 1370 seconds with an MAE of 2.12e-3, evidencing the increased performance of category II architectures.

With these results, we evidence yet another the benefit of using category II architectures over category I ones as for larger domains we see several orders of magnitude of improvement in

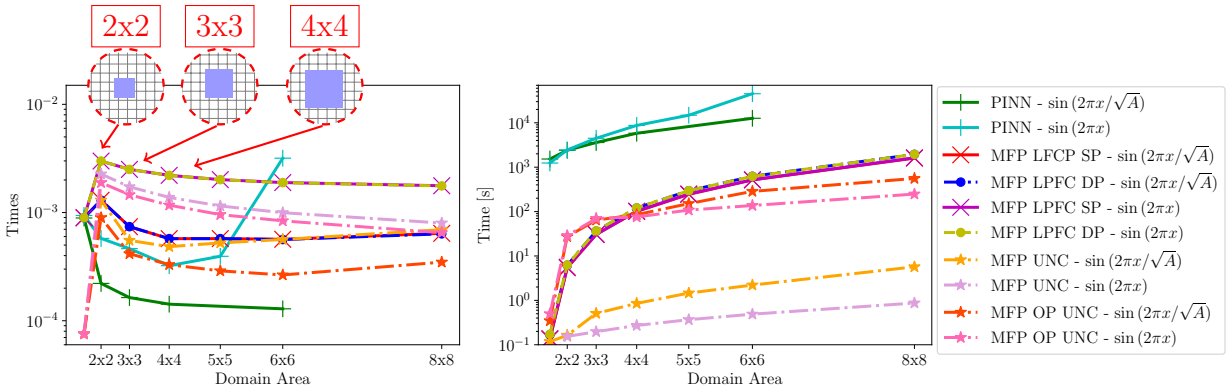


Figure 3.7 Accuracy and performance of MF predictorand PINN for the Laplace equation with two different BCs: For the boundary condition $g_1(s) = \sin(2\pi s/\sqrt{A})$, PINN achieves better accuracy than MF predictor. However, for the more complex BC $g_2(s) = \sin(2\pi s)$, the accuracy of PINN drops, especially in larger domains. Note that while PINN is re-trained for every new BC, MF predictor has never seen these BCs. In terms of computational costs (sum of training and inference times), PINN is several orders of magnitude slower than MF predictor (we were unable to train PINN on the 8×8 domain due to very high costs). In addition, MF predictor with single (SP) and double-precision (DP) yields near identical results. The best results are obtained by MF predictor with optimzation, see Section 3.3.1

the performance. This is because, for updating the shared borders, category I architectures need to call the model up to 128 different times for each genome while category II models require a single call per genome. Although both categories' performance can benefit from the use of batched inputs, category II models are still more desirable.

Comparison with PINN and XPINN.

We compare the accuracy and computational costs of GFNet and MF predictor against a state-of-the-art PINN which has the feed-forward architecture of $60 \otimes 60 \otimes 60 \otimes 60$. To maximize the accuracy of this PINN, we train it with a two-stage optimization process [132] that starts with 40000 iterations using ADAM [129] and is followed by a second-order optimization method based on L-BFGS [133]. We use $20000 \times A$ collocation points and $32 \times 4 \times \sqrt{A}$ boundary points to consider the effect of A. Contrary to PINN, L-BFGS is not used in fine-tuning GFNet.

The evaluation results are illustrated in Figure 3.7. PINN and GFNet achieve a similar accuracy for a single genome, unless the UNC architecture is used. For the simpler BC, $g_1(s) = \sin(2\pi x/\sqrt{A})$, PINN achieves smaller MAEs compared to MF predictor for $A > 1$. This higher accuracy is because (1) the parameters of PINN are fine-tuned with L-BFGS while GFNet is trained via Adam (GFNet cannot be trained via L-BFGS since it has too many parameters), and (2) PINN is specifically trained on this BC (and cannot be used for any other BC) while MF predictor assembles the predictions from a GFNet that has never seen this BC. For the more complex BC, $g_2(s) = \sin(2\pi x)$, PINN fails to scale up to large A; indicating that its architecture is domain-specific and must be optimized anew. Unlike PINN, MF predictor scales up quite robustly to large domains with unseen BCs without the need to re-train or fine-tune its GFNet. Remarkably, MF predictor is between 3-5 orders-of-magnitude faster than PINN depending on the domain size when category II architectures are used which underscores its potential for scalable inference, especially on large and complex domains (for instance, we were unable to train a PINN on an 8×8 domain due to the extremely high training costs).

Finally, we test MF predictor against XPINN [88] which is an extension of PINN that aims to address PINN's scalability issues. XPINN divides a domain into some sub-domains and trains a PINN on each one while ensuring continuity across the sub-domains. In our comparison, we choose a square domain of size 2×2 subject to the BC: $g_1(s) = \sin(2\pi x/\sqrt{A})$ and divide it into 4 square sub-domains of equal sizes. The 4 PINNs used in XPINN have an architecture of $20 \otimes 20 \otimes 20 \otimes 20$ and are trained using Adam for 5000 epochs with 16000 collocation points and 128 boundary points. Table 3.2 enumerates the accuracy of XPINN and MF predictor and shows that our approach is two orders of magnitudes more accurate.

Method	MAE
MF predictor	4.84e-04
XPINN	2.12e-2

Table 3.2 XPINN vs. MF predictor with UNC: Both methods surrogate the Laplace operator in a square domain of size 4×4.

MF predictor with optimization

In the previous results, MF predictor updates the unknown border values iteratively, merging the predictions of the basic and the auxiliary genomes. We refer to it as iterative iterative MF predictor. Although the approach converges for all the studied domains, there is no strong guarantee that the iterative MF predictor will converge in for any scenario, specially if the GFNet used presents a low accuracy. To bypass this problem, we can substitute the iterative updates of the unknown border values for an optimization problem with the following loss:

$$Loss = \sum_{\substack{\forall i \in \{1,...N\} \\ \forall j \in \{1,...M\}}} (\mathcal{N}(\boldsymbol{g^b}_i|\boldsymbol{\theta}) \cap \boldsymbol{g^a}_j - \boldsymbol{x} \cap \boldsymbol{g^a}_j)^2 + \sum_{\substack{\forall i \in \{1,...N\} \\ \forall j \in \{1,...M\}}} (\mathcal{N}(\boldsymbol{g^a}_j|\boldsymbol{\theta}) \cap \boldsymbol{g^b}_i - \boldsymbol{x} \cap \boldsymbol{g^b}_i)^2$$

where $\boldsymbol{x}$ is the set of shared borders, $\boldsymbol{g^b}$ represents the set of border values for the N basic genomes, and $\boldsymbol{g^a}$ represents the set of border values for the M auxiliary genomes.

As shown in Figure 3.7 the accuracy for the MF predictor with optimization is greater than the iterative approach, but, as it requires more computational cost, the performance sharply decreases.

3.3.2 Navier-Stokes Equations

The $2D$ incompressible NS equations are:

$$\begin{aligned} H_1 &: \partial_x u + \partial_y v = 0, \\ H_2 &: u\partial_x u + v\partial_y u + \partial_x p - \nu\nabla^2 u/\rho = 0, \\ H_3 &: u\partial_x v + v\partial_y v + \partial_y p - \nu\nabla^2 v/\rho = 0, \end{aligned} \tag{3.6}$$

where $u(\boldsymbol{x}), v(\boldsymbol{x})$, and $p(\boldsymbol{x})$ denote velocity components and pressure, respectively. We solve Equation 3.6 in a lid-driven cavity problem where the top lid moves with an arbitrary horizontal velocity, i.e., $u = g(x), v = 0$, and the remaining boundaries are treated as static viscous walls where $u = v = 0$. We fix $\rho = 1.0$ and $\nu = 0.002$ and aim to predict $u(\boldsymbol{x}), v(\boldsymbol{x})$, and $p(\boldsymbol{x})$ in unseen domains that have unseen BCs on the top lid and are larger than a genome. Our imposed lid velocity profiles can be highly nonlinear functions and collectively generate Reynolds numbers in the $(0, 500)$ range. The corresponding flow fields are also more complex than cases where the top lid moves with an uniform velocity.

Training Data.

To generate the training data we solve the NS equations with OpenFOAM [134] for two types of flows: cavity flow and channel flow. The velocity profile for the top moving lid and the inlet are sampled via GPs. For each solution field, we obtain the solution on a uniform grid with cells of size of $h = 1/64$. We sweep this grid with a genome of size 0.5×0.5 and record the values on the boundary and inside the genome, see Figure 3.4a. To generate a wide variety of data we sample the flow for different square cavities of size 2×2, 1×2, 1.5×1, 2×1, 3×1, for a step-shaped cavity of size 2×2, and for a channel of developed flow of width 1.

ID	Flow Type	Shape	Size	Sampling Scheme	Samples
1	Cavity	square	2×1 and 1×2	I	2634
2	Cavity	square	2×1 and 1×2	II	6000
3	Cavity	square	2×2	III	900
4	Cavity	step	2×2	III	900
5	Cavity	square	1.5×1	III	900
6	Cavity	square	3×1	III	1800
7	Channel	-	-	I	877

Table 3.3 Data sets for the NS equations: By solving different NS problems we can obtain data from several sources. The flow fields can be sampled using different schemes to prevent correlation on the training samples, understanding that, with more correlation, more samples are available.

Including any genome inside those domains in the training data can lead to highly correlated training samples which would adversely affect GFNet's training. Hence, to get rid of possible correlations we sample the solution field in three ways. By randomly sampling the solutions and discarding a sample if the Euclidean distance of its BC to another sample's BC, i.e., $|\boldsymbol{g}_i - \boldsymbol{g}_j|$, is small ($I$), by sampling every solution with basic genomes of size 0.5×0.5 that covers the domain (II), and by space filling sampling of the domain (III). With the different flows available we generate a total of 7 data sets to train the GFNet's.

Architecture and Training of GFNet.

Following the same procedure as for the Laplace operator, we also design two categories of architectures (I and II). For category I architectures, the GFNet's inputs include the velocities (u, v), extracted at the 128 boundary points located at the grid vertices located on the boundary, and $\boldsymbol{x}$, i.e., a total of $N_{var} \times N_{bc} + N_{dim} = 128 \times 2 + 2 = 258$ inputs. Following [132], we exclude the BC on p from the inputs to simplify the learning task. Category I architecture's outputs $\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta}) = (\hat{u}, \hat{v}, \hat{p})$ are $3D$ and approximate the solution (u, v, p) at $\boldsymbol{x}$. For category II architectures, the GFNet's inputs include (u, v, p), extracted at the same grid vertices, so a total of $N_{var} \times N_{bc} = 128 \times 3 = 384$ inputs. Due to the increased training advantages of category II architectures, we do not need to simplify the learning task

and can include p as an input, which actually provides more accurate models. The outputs $\mathcal{N}(\boldsymbol{g}|\boldsymbol{\theta}) = (\hat{u}, \hat{v}, \hat{p})$ are $3D$ and approximate the solution (u, v, p) at all the grid vertices of the domain.

The loss function of GFNet is adopted from Equation 3.2 as follows:

$$\begin{aligned} L(\boldsymbol{\theta}) = \frac{1}{N_0} \sum^{N_0} \left(\gamma_1(\hat{u} - u)^2 + \gamma_2(\hat{v} - v)^2 + \gamma_3(\hat{p} - p)^2\right) \\ + \frac{1}{N_1} \sum^{N_1} \left(\alpha_1 H_1^2 + \alpha_2 H_2^2 + \alpha_3 H_3^2\right) + \beta|\boldsymbol{\theta}|^2. \end{aligned} \tag{3.7}$$

where the first summation minimizes the error on predictions, the second summation penalizes the residuals, and the last term represents the Tikhonov regularization. $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \gamma_3]$, $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \alpha_3]$, and β balance the contributions to the overall loss. In case of category I architectures H_1, H_2, and H_3 are obtained using AD on the inputs, while in category II architectures they are approximated via FD.

In category I, due to the non-linear nature of NS equations, we only study the FC architecture which uses tanh activation function in all the layers but the last, which uses *linear*. In category II, we study both the FC architecture and three modified versions of the UNC. UNC with global convolution and global de-convolutions (UNC-GC-GD), which recreates the exact same configuration UNC but the inputs and outputs have 3 channels corresponding to (u, v, p). UNC with global convolution and particular de-convolutions (UNC-GC-PD), where after the FC layers, each channel has its own de-convolution process. And UNC with particular convolutions and particular de-convolutions (UNC-PC-PD), where each input is treated independently both in the convolution and the de-convolution processes, and the FC layers are used to relate the three different parts. For all the category II architectures we use *swish* activation functions in all the layers but the last, which uses *linear* activation functions.

The depth and size of our GFNet are considerably larger than the networks in related works

[84, 130, 132, 135] because we (1) use BCs (which are high-dimensional) as inputs, and (2) find that a deep network with Tikhonov regularization generalizes better than shallow networks that exclude regularization. To find the optimal values of $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$, and β, we set $\gamma_1 = \gamma_2$ and $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$ since u and v are equally important and the three equations are highly coupled, i.e., no reason to penalize one equation's residual more than the others. We alter β between 1e-25 and 1e-9 and for each value of β, test a few combinations of γ_1, γ_3 and α.

For data-only category I GFNets, i.e., $\alpha = 0$, reducing pressure's contribution to gradient descent improves the overall accuracy. We use $\gamma_3 = 0.5$ since further decreasing it to $\gamma_3 = 0.2$ negligibly improves the predictions for velocities while increasing the error on pressure by 27%. Following these principles, $\gamma_3 = 0.5$ was adopted by all the category II GFNet architectures.

While using collocation points in category I architectures reduces the accuracy by more than 50%, category II architectures do not suffer from this pathology. Like in the Laplace example, category II GFNets can be trained using collocation points only. The main reason

Network	Arch.	Points	γ_1, γ_3		α	β	Validation MAE			Validation PDE loss		
FC Cat. I	A	961/-	1.0	1.0	0	1e-11	1.02e-3	1.01e-3	6.49e-4	-	-	-
FC Cat. I	A	961/-	1.0	0.5	0	1e-10	6.81e-4	6.43e-4	5.51e-4	-	-	-
FC Cat. I	A	961/-	1.0	0.2	0	1e-10	6.42e-4	6.30e-4	7.05e-4	-	-	-
FC Cat. I	A	576/1200	1.0	0.5	1e-3	0	1.48e-3	1.24e-3	8.79e-4	8.30e-4	2.58e-3	4.51e-4
FC Cat. I	A	576/1200	1.0	0.5	1e-2	1e-10	1.88e-3	1.56e-3	9.84e-4	3.20e-4	4.57e-4	1.94e-4
FC Cat. II	B	961/-	1.0	0.5	0	1e-15	5.05e-4	4.80e-4	3.09e-4	-	-	-
FC Cat. II	B	961/961	1.0	0.5	1e-4	1e-15	5.13e-04	5.13e-04	3.20e-04	1.11e-4	9.32e-5	8.48e-5
FC Cat. II	B	-/961	1.0	0.5	1	1e-15	7.05e-3	6.79e-3	6.82e-3	4.94e-4	8.49e-4	6.39e-4
UNC-GC-GC	C	961/-	1.0	0.5	0	1e-25	1.06e-3	9.94e-4	5.05e-4	-	-	-
UNC-GC-PC	D	961/-	1.0	0.5	0	1e-25	1.18e-3	1.04e-4	4.47e-4	-	-	-
UNC-PC-PC	D	961/-	1.0	0.5	0	1e-25	6.72e-04	8.76e-04	2.81e-04	-	-	-

Table 3.4 GFNet's accuracy in learning the NS equations with dataset ID 1: The first column indicates the network used and the second the structure of the corresponding FC layer in that architecture. The first and second values in column three indicate the number of data and collocation points, respectively. γ_1, γ_3, α, and β are the coefficients in Equation 3.8. The three validation error and loss values correspond to $u(\boldsymbol{x}), v(\boldsymbol{x})$, and $p(\boldsymbol{x})$ which denote the velocity components and pressure, respectively. $A = 256^3 \otimes 128^3 \otimes 96^3 \otimes 64^3 \otimes 32^2 \otimes 3$, $B = 350^5 \otimes 3267$, $C = 100^3$, $D = 200^3$

why category I architectures cannot be successfully trained with PDE loss is the following. In the training, AD is used to compute the derivatives analytically and regularizes the differential form of the NS equations while OpenFOAM solves the integral form of the NS equation. Note that the velocity and pressure from OpenFOAM are accurate but their derivatives differ from the ones obtained by analytical differentiation and dissatisfy the PDE residual in Equation 3.8. For instance, OpenFOAM reports an average residual of 9e-10 for H_1 using the velocities shown in Figure 3.10b. However, the same velocities result in an average residual of 3e-4 if FD is used to calculate the derivatives in Equation 3.6. As a result, the data loss and PDE residual contradict each other during gradient descent and decrease the overall accuracy of GFNet. In category II architectures, as FD is used to approximate the derivatives, this differences are reduced and the undesirable effect of it to the PDE loss mitigates, specially when models are trained with a low α.

In Table 3.4, we find the best architectures trained using dataset ID 1 only. As for the lid-driven cavity problem, we are interested in inferring the velocity accurately across unseen BCs and unseen domains, we choose as GFNet the model with the lowest MAE in velocity which is obtained for the category II FC architecture of size $350^5 \otimes 3267$. The subsequent results will be provided using this architecture.

Increasing the accuracy of GFNet.

In the previous section, we selected category II FC as the best architecture that has been studied. However, the MAEs reported at 3.4 were achieved by only considering dataset ID 1. We can further increase the accuracy of GFNet by repeating the training with other combinations of datasets.

The results for these trainings, displayed in Table 3.5, demonstrate that the use of more boundary conditions and flow solutions benefits the accuracy of the GFNet. By comparing

Model ID	Datasets							Test on unseen BCs		
	1	2	3	4	5	6	7	MAE u	MAE v	MAE p
ID 1	Y	N	N	N	N	N	N	5.05e-4	4.80e-4	3.09e-4
ID 2	Y	N	Y	Y	Y	Y	N	2.49e-4	2.36e-4	1.49e-4
ID 3	Y	N	N	N	N	N	Y	5.20e-4	4.03e-4	3.39e-4
ID 4	Y	N	Y	Y	N	N	Y	4.90e-4	4.11e-4	3.26e-4
ID 5	Y	N	Y	Y	Y	Y	Y	5.28e-4	4.14e-4	3.31e-4
ID 6	N	Y	N	N	N	N	N	2.12e-4	2.10e-4	1.41e-4
ID 7	N	Y	Y	Y	N	N	N	2.15e-4	2.11e-4	1.37e-4
ID 8	N	Y	Y	Y	Y	Y	N	1.83ee-4	1.74e-4	1.16e-4
ID 9	N	Y	Y	Y	Y	Y	Y	3.99e-4	3.28e-4	2.55e-4

Table 3.5 Improving the accuracy of the GFNet: Category II FC architectures of size $350^5 \otimes 2883$ trained using different datasets. Y (yes) indicates that a dataset is used while N (no) indicate that it is not. For each sample in the data set, 951 data points are used in training but no collocation points are present. Each resulting training set is randomly divided three minor sets of proportions 8:1:1. The first and second sets are used as training and validation data while the third set is used after the training to test the GFNet. The MAEs are reported using the data on the test set.

the accuracy of model ID 1 against the one for model ID 6, we observe that variety of flows is more important than the variety of BCs when learning of the NS operator. In dataset ID 1, the data is selected to avoid correlations between BC which penalizes the low magnitude flows. However they are as important as the others in terms of learning the NS operator. For the same reason, the more datasets used in the training the more accurate the model becomes.

We note that the pressure and velocity scales of the channel data are different from the ones in the cavity flows. Thus, when models are trained with dataset 8, the test accuracy is reduced. Sampling more data from this kind of flow fields would not only reduce the negative effects of including it but would also increase the accuracy of the model, as we see when we add other cavity flows. After this extended training, the best GFNet for solving NS equation is considered to be model ID 8.

Accuracy for Unseen Domains subject to Unseen BCs and Comparison with PINN.

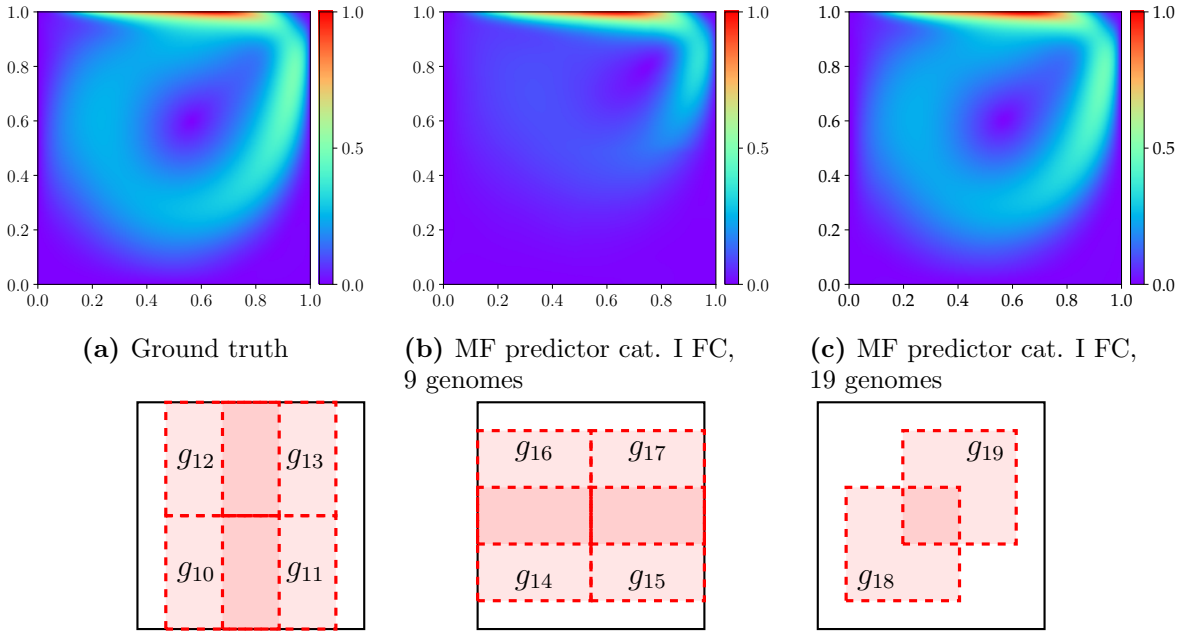
We evaluate the accuracy of MF predictor in three different cavities where the unseen BC is again generated via a GP and determines the velocity profile of the top lid:

- A square cavity of size $[0,1] \times [0,1]$ which is four times larger than the genome used in training the GFNet.
- An step cavity domain of size $\big([0,2] \times [0,2]\big) \setminus \big([0,1] \times [1,2]\big)$ which is twelve times larger than the genome used in training the GFNet.
- A long cavity of size $[0,3] \times [0,1]$ which is twelve times larger than the genome used in training the GFNet.

To cover the domain, we use MF predictor with basic and auxiliary genomes arranged as shown in Figure 3.4b, the same distribution as the one used throughout Section 3.3.1. However, the highly non-linear nature of the NS equations complicates the propagation of the boundary information. During our preliminary studies, we observed that only using basic and auxiliary genomes does not always provide sufficient accuracy if the GFNet itself is not accurate enough to propagate the boundary conditions. Figure 3.8b exemplifies this pathology when the best category I FC architecture is used in solving the square cavity unseen domain. Using only 9 genomes does not provide sufficient accuracy and the MAE for velocity components u and v are 7.15e-2 and 6.82e-2, respectively. To address this issue, we place another layer of overlapping auxiliary genomes that facilitates the information propagation, see Figure 3.8d. As shown in Figure 3.8c, using 19 genomes significantly improves the accuracy and reduces the MAE of velocity components u and v to 5.68e-3 and 4.96e-3, respectively. However, more accurate GFNets, model ID 8 for instance, successfully propagate the boundary conditions and do not benefit for the use these extra overlapping genomes.

When only basic and auxiliary genomes are used, the MAE for velocity components u and v are 2.28e-4 and 2.29e-4, respectively. If the extra genomes are used, the accuracies are 2.45e-4 and 1.99e-4. In Table 3.6, we show how in some problems using overlapping genomes negatively affects the accuracy of the models as we are introducing more GFNet errors in the predictions and the BCs are already successfully propagated. In terms of performance, the solving time present the same order of magnitude because although the performance of single iterations decreases using overlapping genomes, less iterations are needed to converge.

To compare accuracy and performance with the state-of-the-art models, we construct the PINN designed in [130] to solve the lid-driven cavity problem with $u = 1, v = 0$ on the top lid (we were able to reproduce the accurate results reported in [130]). We re-train the model for 40000 epochs with Adam using the architecture $50 \otimes 50 \otimes 50 \otimes 50 \otimes 50$, 4×129



(a) Ground truth **(b)** MF predictor cat. I FC, 9 genomes **(c)** MF predictor cat. I FC, 19 genomes

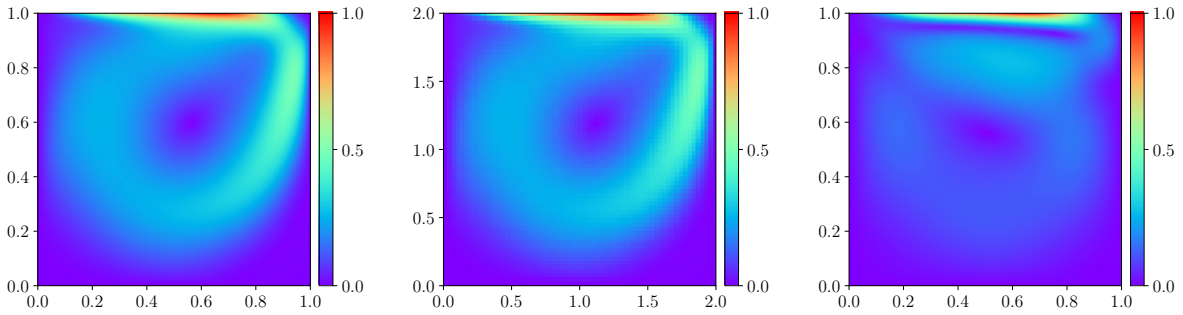
(d) Arrangement of 10 additional auxiliary genomes in the square cavity

Figure 3.9 Velocity magnitudes $\sqrt{u^2+v^2}$ for flow in a square lid-driven cavity with unseen BC: (a) The ground truth is simulated by OpenFOAM [134]. (b) MF predictor, with category I FC architecture, fails to accurately predict the velocities with 9 genomes. (c) MF predictor, with category I FC architecture, accurately predicts the unseen conditions with 19 genomes and achieves an MAEs of 5.68e-3 and 4.96e-3 for velocities (u, v), respectively. (d) In addition to the 9 genomes (g_{1-9}) arranged as shown in Figure 3.4b, we add 10 more auxiliary genomes (g_{10-19}) to improve MF predictor's accuracy in resolving the unseen flow.

boundary points, and 10000 collocation points. However, as shown in Figure 3.10c, PINN is unable to resolve the flow field and the MAE for velocities (u, v) are as large as 1.52e-1 and 1.39e-1. This indicates that the architecture of PINN only applies to the specific BC used in [130], i.e., changing the BC requires not only re-training the PINN but also re-designing its architecture. Regarding computational costs, MF predictor with 9 genomes converges to

Approach	GFNet	Cavity	Overlapping Genomes	MAEs		Time [s]
				MAE u	MAE v	
Iterative	A	square	No	2.28e-4	2.29e-4	4.8
			Yes	2.45e-4	2.03e-4	7.5
		step	No	3.71e-3	3.91e-3	20
			Yes	6.83e-3	6.65e-3	18
		long	No	1.64e-3	1.29e-3	15
			Yes	2.15e-3	2.28e-3	17
Adaptive	B	square	No	2.38e-3	1.61e-3	14
			Yes	1.639e-3	1.25e-3	37
		step	No	2.24e-3	2.63e-3	62
			Yes	3.07e-3	3.77e-3	51
		long	No	2.37e-3	1.64e-3	23
			Yes	3.06e-3	2.59e-3	53
Ensemble	C	square	No	1.94e-4	2.09e-4	42
			Yes	1.86e-4	1.67e-4	73
		step	No	2.79e-3	3.09e-3	182
			Yes	4.96e-3	5.12e-3	170
		long	No	1.40e-3	6.4e-4	63
			Yes	2.06e-3	1.63e-3	200

Table 3.6 Iterative MF predictor, adaptive MF predictor, and MF predictor ensemble applied in different cavities: Three different updating schemes for MF predictor are evaluated in three different domains: square cavity, step cavity, and long cavity. Y (yes) in the overlapping genomes column indicates that MF predictor uses an extra layer of auxiliary genomes to favor the boundary condition propagation while N (no) indicates that only the regular basic and auxiliary genomes are being used. As convergence criteria, we use the stagnation of the solved flows in terms of MSE. The time to solve each domain is displayed in the last column. A represents a single GFNet of size $350^5 \otimes 3267$. B represents a combination of 5 GFNets of sizes $350^5 \otimes 3267$. In the selected five GFNets we find one original GFNet, two I-GFNets and two B-GFNets. C represents a combination of 11 GFNets of sizes $350^5 \otimes 3267$. Multiple GFNets, I-GFNets and B-GFNets are used.

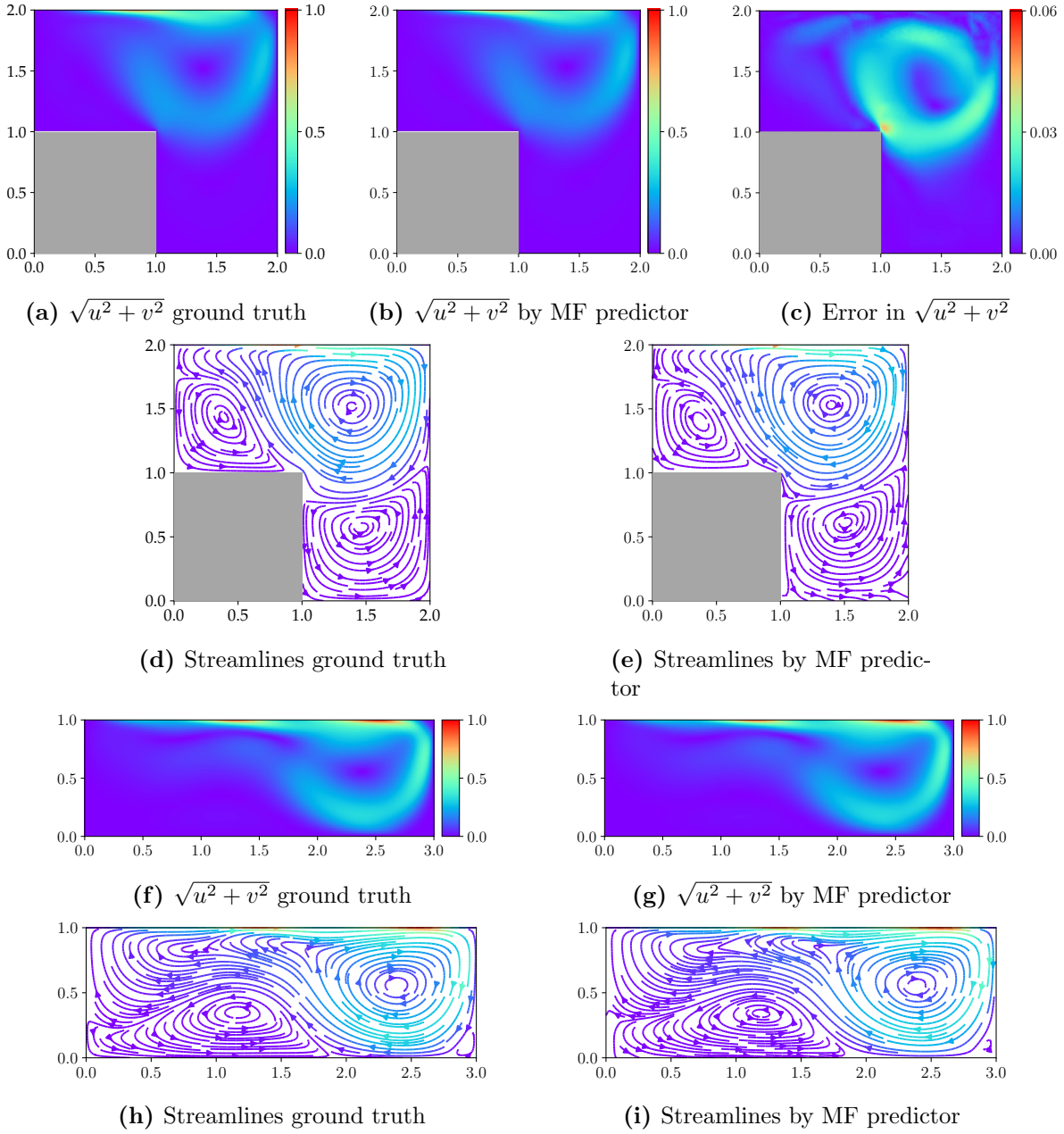


(a) Ground truth **(b)** MF predictor best model **(c)** PINN

Figure 3.10 Velocity magnitudes $\sqrt{u^2+v^2}$ for flow in a square lid-driven cavity with unseen BC: (a) The ground truth is simulated by OpenFOAM [134]. (b) MF predictor, model ID 8, accurately predict the velocities with 9 genomes. (c) PINN fails to resolve the flow and achieves MAEs 1.52e-1 and 1.39e-1 for velocities (u, v).

an accurate solution in 4.8 seconds while the PINN's training takes 8846 seconds.

We further evaluate MF predictor by predicting the flow in a step-shaped lid-driven cavity that has an unseen velocity profile set to the top lid, see Figure 3.11. This unseen cavity is 12 times larger than our genomes and is decomposed into 12 basic and 21 auxiliary genomes (or in 12 basic and 42 auxiliary genomes, if overlapping genomes are used). The accuracy and performance results can be found at Table 3.6. Figure 3.11 compares the results from MF predictor and the ground truth simulated by OpenFOAM [134] using the same grid resolution as in data generation. MF predictor converges in 20 seconds which is almost 4 times more than the cost of solving the square cavity with MF predictor in 3.10c. This increase in inference time relates with the 3:1 area ratio between the two domains. The MAE for velocities (u, v) are 3.71e-3 and 3.91e-3, respectively, which are larger than the MAE for the square cavity by 1 order of magnitude. As shown in Figure 3.11c and 3.11d, the increased error primarily comes from complex flow patterns. During training, GFnet only captures the flow on a local scale and does never see the intricate interactions that the flow over the step corner creates. Nonetheless, MF predictor still captures all three vortex structures in the cavity as shown by the streamlines in Figure 3.12c. We also observe some nonphysical flows in the near-wall regions where the streamlines should be parallel to the wall. This error is



(a) $\sqrt{u^2+v^2}$ ground truth **(b)** $\sqrt{u^2+v^2}$ by MF predictor **(c)** Error in $\sqrt{u^2+v^2}$

(d) Streamlines ground truth **(e)** Streamlines by MF predictor

(f) $\sqrt{u^2+v^2}$ ground truth **(g)** $\sqrt{u^2+v^2}$ by MF predictor

(h) Streamlines ground truth **(i)** Streamlines by MF predictor

Figure 3.11 Velocity magnitude $\sqrt{u^2+v^2}$ and streamlines for the flow in a step-shaped lid-driven cavity and the long lid driven cavity:. (*a*,b,c,d,e) For the step-shaped cavity, MF predictor, model ID 8, achieves MAEs of 3.71e-3 and 3.91e-3 for velocity components u and v, respectively, compared to the ground truth simulated by OpenFOAM [134]. The streamline plots (colored by velocity magnitude) reveal that the step induces a complex flow pattern that consists of three inter-related vortices and MF predictor successfully captures all these vortex structures while the GFNet has never seen such scenarios during training. (*f*,g,h) For the long cavity, MF predictor, model ID 8, achieves MAEs of 1.64e-3 and 1.29e-3 for velocity components u and v, respectively. The streamline plots (colored by velocity magnitude) reveal that MF predictor successfully captures the two inter-related vortices that the long cavity shape produces.

caused by the inaccuracies of GFNet in predicting the flow on the boundaries.

Lastly, we evaluate the flow generated in a long cavity as shown in Figure 3.11. Again, MF predictor can successfully predict the intricate flow behavior that arises with this cavity, extrapolating form a local scale to a global one. The MAEs for the u and v velocities are 1.64e-3 and 1.29e-3, respectively and the solving time is 15 seconds. Three times more than the time needed to solve predict the smaller square cavity. Again, this evidences that the approach scales almost linearly with the domain.

Adaptive MF predictor

By inspection on the streamlines of the three predicted flows, we observe that MF predictor fails to produce physical feasible solutions near the boundaries of the domain. See in Figure 3.12d how at the bottom of the step cavity the predicted streamlines end into the walls, which creates an unfeasible flow. To address this issue, we would like to use specifically trained GFNets in those regions, adaptively solving the flow and improving the accuracy of MF predictor, in physical terms, near the borders. To do so, we design the *adaptive MF predictor*, a novel upgrade for MF predictor that automatically selects, out of N specifically trained GFNets, the one that can better solve the flow at each different location on the domain. The selection is automated using a classifier DNN, trained to learn which of the N GFNets provides the smallest MAE given the current u, v, p in the genome.

We obtain the specifically trained GFNets in two steps. First, we expand the number of available training datasets with two new sampling schemes: (IV) by randomly sampling genomes laying in the borders only and (V) by randomly sampling genomes laying in the interior of the domain only. We extract these genomes from the current solved square cavities of size 2×1, 1×2 and 2×2 1.5×1 and 3×1, from the solved step cavities, and from the solved channel flows. In total, this generates 14 new datasets, 7 with interior genomes,

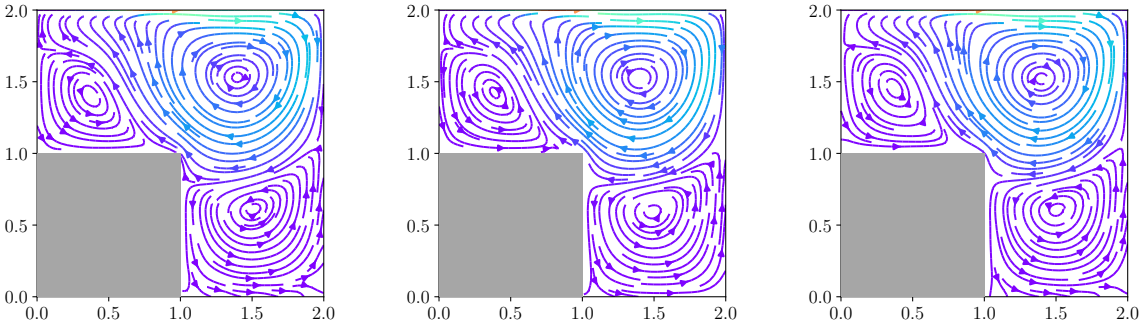
GFNet type	Datasets							Test on unseen BCs		
	1	2	3	4	5	6	7	MAE u	MAE v	MAE p
GFNet	N	Y	Y	Y	Y	Y	N	1.83ee-4	1.74e-4	1.16e-4
I-GFNet	Y	Y	Y	Y	Y	Y	N	1.45ee-4	1.39e-4	8.80e-5
I-GFNet	Y	Y	Y	Y	Y	Y	Y	5.27e-4	3.51e-4	3.48e-4
B-GFNet	Y	Y	Y	Y	Y	Y	N	1.68e-4	1.65e-4	1.10e-4
B-GFNet	Y	Y	Y	Y	Y	Y	Y	4.96e-4	3.43e-4	3.37e-4

Table 3.7 Training accuracy of the top 5 more representative specifically trained GFNets: Category II FC architecture of size $350^5 \otimes 2883$ trained using different datasets. I-GFNet indicates that only interior genomes are used in training while B-GFNet indicates that only border genomes are used. We represent with Y (yes) that a dataset is used while N (no) indicates that it is not. For each sample in the data set, 951 data points are used in training but no collocation points are present. Each resulting training set is randomly divided three minor sets of proportions 8:1:1. The first and second sets are used as training and validation data while the third set is used after the training to test the GFNet. The MAEs are reported using the data on the test set.

and 7 with border genomes. Second, we retrain the best GFNet architecture, category II FC with size $350^5 \times 3267$, using different combinations of these new datasets and old ones. This leads to three groups of specifically trained GFNets ready to be used in prediction: border GFNets (B-GFNet), trained with border genomes only; interior GFNet (I-GFNet), trained with interior genomes only; the original GFNets, trained with the original datasets.

Once we have the specifically trained GFNets, we can obtain the classifier trained to adaptively select one of these GFNets when solving each genome. As the number M of trained models can be large, we ease the learning task of the classifier by only selecting the top N most representative models. To find out how representative a model is, we record how many times a model k presents the best MAE out of all the M trained networks when evaluated in the all of the available data samples (original and expanded datasets). The more times the model has the best MAE, the more representative it is. Note that this selection is based on the relative performance of the M models, and other selection procedures will be evaluated in the future.

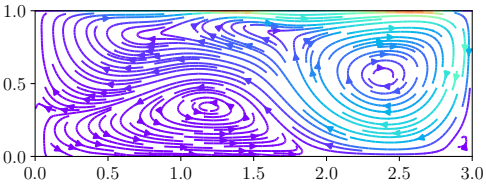
To test the *adaptive MF predictor* we reproduce the steps above with $N = 5$ and with

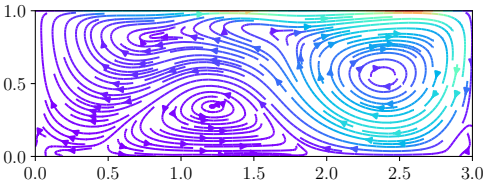


(a) Streamlines by iterative MF predictor

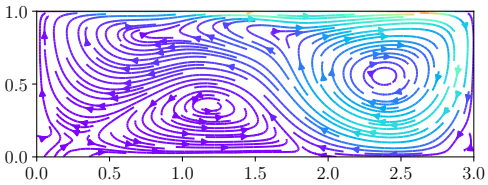
(b) Streamlines by adaptive MF predictor

(c) Streamlines by ensemble MF predictor





(d) Streamlines by iterative MF predictor **(e)** Streamlines by adaptive MF predictor



(f) Streamlines by ensemble MF predictor

Figure 3.12 Streamlines for the predicted solutions with MF predictor, adaptive MF predictor, and ensemble MF predictor: (*a*,b,c) For the step cavity, adaptive MF predictor mitigates the unfeasible flow pathology, observed at the bottom of the domain, while the usual approach and the ensemble MF predictor do not. (*d*,e,f) For the long cavity, adaptive MF predictor, although it produces a more feasible flow than its counterparts fails to properly capture the left vortex. Note that although the MAE of the ensemble approach is the lowest, see Table 3.6 it also fails to properly capture this flow.

$M = 21$. We display the specifically trained GFNets selected and their test accuracy during training in Table 3.7. As a classifier, we use a FC architecture of size $100^5 \otimes 5$ with *swish* activation functions. The model is trained with itercalated dropout layers with a probability of drop out equal to 0.2 to avoid overfitting. We terminate the training of the classifier when the validation accuracy no longer decreases and the validation loss starts to increase. The trained classifier presents a validation accuracy of 0.9004 when selecting the most representative model in the test data.

We evaluate adaptive MF predictor accuracy and performance in the three testing domains, see the results at Table 3.6. For the square cavity, the benefits of using adaptive MF predictor over the iterative approach are minimal and only occur if the extra layer of overlapping genomes is used. Note that the single cavity is the wider domain out of all the testing ones so it will benefit the most from a better propagation of the boundary conditions. The single cavity solution was already physically feasible when using the usual MF predictor.

For the step cavity we see a clear improvement on both the accuracy and the physical meaning of the streamlines, specially at the bottom right cavity, see Figure **??**. However, when adaptive MF predictor is applied to the long cavity, it fails to properly learn the left bottom vortex. By analyzing the predictions at the converged solution, we identify that the error is caused by a poor selection of the GFNet done by the classifier. A perfect adaptive MF predictor model should use only B-GFNets for the basic genomes as they are all laying in the boundaries. However, in the used trained version of adaptive MF predictor, the genomes placed at the bottom of the domain are inferred using I-GFNets. In future works, we plan to enforce the classifier to select feasible GFNets only at each particular location in the domain (i.e. do not use a I-GFNet to predict for a genome laying at the border).

Ensemble MF predictor

Motivated by the amount of trained models and to further extend the accuracy of the predictions, we design yet another updating approach for MF predictor, *ensemble MF predictor*. The principle behind this approach is to reduce the error of the GFNet predictions by ensembling the predictions of N GFNets via the point-wise median. To do so, first we select the top N most representative GFNets out of all the M available, and then, we use the combined predictions to perform the updates on the iterative MF predictor.

We test the *ensemble MF predictor* with $N = 12$ and with $M = 21$. The 21 GFNets are

a combination of 7 original GFNets, 7 I-GFNets, and 7 B-GFNets. Table 3.6 shows the accuracy and performances of this approach.

By using ensemble MF predictor, we notice an approximate improvement of 20% of accuracy, on average, with respect to the iterative MF predictor. However, this improvement is not reflected in the streamlines of the solved flows, see Figure 3.12. We argue that this approach can be refined by restricting the models used at each genome depending if it lays in the border or in the inner domain and we will explore in these directions in future works. However, this approach has two major disadvantages. First, we need perform N predictions for each genome and, second, we need compute the point-wise median of these predictions. An operation that can become prohibiting for larger domains. For the tested domains we already see this effect, making it up to 10 times slower than the simpler approach.

We note that we also tested other ensembling procedures but the median was the one that produced best results in terms of accuracy.

3.3.3 Wave equation

In order to show that MF predictor can be used in time dependent PDEs and solve the IBVP of Equation 3.3, we consider the 2D homogeneous wave equation with with $c = 1$, see Equation 3.5. We aim to solve it for $t > 0$ given arbitrary BCs $g(\boldsymbol{x}, t)$ for $\boldsymbol{x} \in \partial\Omega$ and $t > 0$, and given arbitrary ICs $f_1(\boldsymbol{x}) = u(\boldsymbol{x}, t = 0)$ and $f_2(\boldsymbol{x}) = u_t(\boldsymbol{x}, t = 0)$ for $\boldsymbol{x} \in \Omega$.

The results showed in this section are still in a preliminary stage of research.

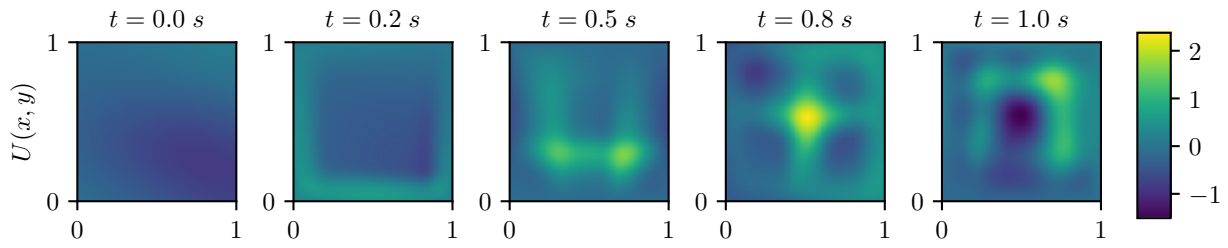


Figure 3.13 Sample of the training data for the wave equation: Wave equation solved for $t \in [0, 1]$s. The intial conditions are generated using GP and the u_t is assumed to be zero. At $t > 0.5$ we observe how the initial wave propagates to the center of the domain and interferes with the waves created by the other borders, generating interferences of both constructive and destructive nature. To be able to solve for any time t, the model must be trained to handle this situations.

Training Data.

As described in Section 3.2.2, we use GPs to generate the ICs and the BCs. In particular, we use Sobol sequence [67] to sample the hyperparameters of the Gaussian kernel of a $3D$ GP, where we interpret the dimensions as x, y and t, respectively. Using this GP, we draw the initial conditions at the vertices of a uniform grid composed of 31×31 cells that covers the domain $[0, 1] \times [0, 1]$ at $t = 0$, and we draw the boundary conditions at the borders of this gird for every $\Delta t = 0.005$s until $t = 2$s. For each set of ICs and BCs, we numerically solve the homogeneous wave equation, with $c = 1$, using FD under the assumption that $u_t(\boldsymbol{x}, t = 0) = 0$. This provides us with the solution for any time t such that $t = k \times \Delta t$ with $k \in \mathbb{Z}^+$ and $\Delta t = 0.005$ s. We repeat this process to generate 5000 different solution under different BCs and ICs.

Given that the domain size is $[0, 1] \times [0, 1]$ and that the wave velocity $c = 1$, we make sure that the data introduces wave interference at the center of the domain, see Figure 3.13, by solving for all $t < 2$. This is a necessary condition to be able to extrapolate for any time $t > 0$ given any BC and ICs.

Note that all of 5000 different solutions assume that $u_t(\boldsymbol{x}, t = 0) = 0$. For this reason,

from these 5000 solutions we randomly sample, with an space filling design [136], at total of 100000 different time windows at different times. each time window is composed of 11 solutions evenly distributed in time. For every sample, we also include a FD approximate version of u_t at the initial time.

We generate two datasets by choosing two different time windows, 0.055 and 0.22 seconds. The first one covers short term predictions where the distance between solutions is $\Delta t = 0.005$ seconds (short term predictions dataset, STPD). The second covers long term predictions where the distance between solutions is $4 \times \Delta t = 0.02$ s (long term predictions dataset, LTPD).

Architecture and Training of GFNet.

Solving the IBVP for wave equation, unlike the Laplace and NS equations, involves making future predictions given present and past information. A group of architectures specially desirable in predicting for future times are recurrent neural networks, RNN [137], in particular long-short term memory cells, LSTM [138] and gated recurrent units, GRU [139].

We design the architectures such that we can take advantage of these properties. Thus, the units of the FC architectures for the wave equation are GRU cells. Thus, if we buld a FC of size L hidden layers, the units for the first $L-1$ are GRU cells with *swish* activation functions and *sigmoid* as the recurrent activation functions. The Lth layer is fully connected linear layer that maps to the domain. We name this structure FC-GRU.

The GFNet's inputs include the boundary conditions and the initial conditions, i.e., a total of $N_{time} \times N_{var} \times N_{bc} + N_{var} \times N_{IC} + N_{var} \times N_{IC_1} = 10 \times 1 \times 128 + 1 \times 1024 + 1 \times 1024$ inputs, where N_{time} represents the number of solution in a time window. Given the available datasets, $N_{time} = 10$, so in total the FC-GRU handles $r = 10$ recursions in the network. For each recursion, the FC-GRU takes one input, $\boldsymbol{g}(t = r * \Delta t)$, and one hidden input, $\boldsymbol{h}_{r-1}$, encoding

information about the past states and predicts one output, $\boldsymbol{h}_r = \mathcal{N}(\boldsymbol{g}(t = r \times \Delta t), \boldsymbol{h}_{r-1}|\boldsymbol{\theta})$. However, while $\boldsymbol{h}_{r-1}$ is available for all $r > 1$, it is unknown for $r = 1$. In most GRU models, it is usually assumed to be $\boldsymbol{h}_0 = \boldsymbol{0}$, but this would imply that the ICs for the model are always the same, which is not the case when solving the wave equation with arbitrary ICs. To bypass this issue, we encode the initial conditions using a FC model and use the output of this encoder as $\boldsymbol{h}_0$, i.e. $\boldsymbol{h}_0 = \mathcal{N}(f_1, f_2|\boldsymbol{\theta})$.

As a result, the GFNet architecture, for which we show a detailed representation in Figure 3.14, is the composition of two different architectures: a FC-GRU and a FC encoder. We denote it by first indicating the FC-GRU structure followed by the FC encoder structure (i.e. $100^3 \otimes 1024/100^2$, where the GRU structure is 3 hidden layers of size 100 followed by a fully connected layer with linear activation of size 1024 and the FC encoder architecture is 2 hidden layers of size 100).

Due to the complexity and the size of the training data, only category II architectures are considered. Note that both the FC encoder and the FC-GRU are trained at the same time and the loss function used in training is adopted from Equation 3.2 as follows:

$$\begin{aligned} L(\boldsymbol{\theta}) = & \frac{1}{N_0} \sum^{N_0} \left(\hat{u} - u\right)^2\big) \\ & + \frac{1}{N_1} \sum^{N_1} \left(\alpha H^2\right) + \beta |\boldsymbol{\theta}|^2. \end{aligned} \tag{3.8}$$

Based on previous results, $\beta = 1e - 15$ and $\alpha = 0$ are selected when training the models.

During the early stages of our studies, we noticed that using $f_1(\boldsymbol{x}) = u(\boldsymbol{x}, t = 0)$ and $f_2(\boldsymbol{x}) = u_t(\boldsymbol{x}, t = 0)$ leads to models with poor accuracy. Between the values of $u(\boldsymbol{x}, t = 0)$ and $u_t(\boldsymbol{x}, t = 0)$ there is a big range difference which negatively affects the learning. By substituting the value of $u_t(\boldsymbol{x}, t = 0)$ by $u(\boldsymbol{x}, t = -\Delta t)$, which we approximate by $u(\boldsymbol{x}, t = -\Delta t) = u(\boldsymbol{x}, t = 0) - \Delta t \cdot u_t(\boldsymbol{x}, t = 0)$, we observe an average of an order of magnitude of improvement in the accuracy of the models tested.

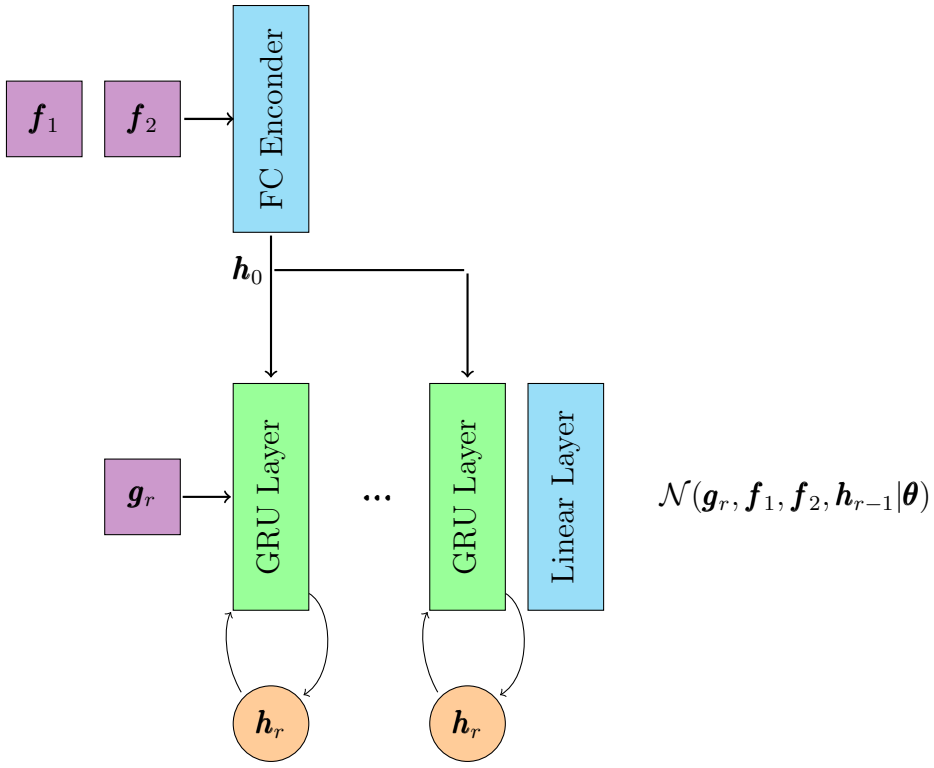


Figure 3.14 FC-GRU/FC encoder architecture: Architecture used in the GFNet to solve time dependent PDEs. The architecture is composed by two different architectures. An FC architecture which encodes the ICs and inputs them as the hidden state h_0 in the GRU (FC encoder) and a FC-GRU, which is a fully connected architecture where its units for the first $L-1$ layers are all GRU cells with swish activation functions and with sigmoid recurrent activation functions.

Using these initial conditions, we manage to design the best models. For the STPD, the best achieved accuracy in terms of MAE is 2.61e-4 given by an architecture of size $200^4 \otimes 1024/150^3$. For the LTPD, the best accuracy is 5.71e-3 given by an architecture of size $150^5 \otimes 1024/150^3$. We adopt these two architectures as the short term predictions GFNet (ST-GFNet) and as the long term predictions GFNet (LT-GFNet). A deeper study in the architecture is still necessary to conclude the optimally of these configurations and these hyper-parameters.

Accuracy for Unseen Domains subject to Unseen BCs.

We evaluate the accuracy of the MF predictor in three domains of sizes $[0,1] \times [0,1]$, $[0,2] \times [0,2]$ and $[0,3] \times [0,3]$, by solving the wave equation for $t \in (0,2]$ at $t = k \times \Delta t$ where

$k \in \mathbb{Z}^+$ and $\Delta t = 0.005$ seconds.

As our goal is to make successful long term predictions, we aim to use MF predictor with the LT-GFNet. However, this model only provides solutions at $t = k \times 4\Delta t$. To predict for any $t = k \times \Delta t$, we first need to use the MF predictor with ST-GFNet and extract extra ICs for all $t = l \times \Delta t$ with $l = \{0, 1, 2, 3\}$. With now four ICs available, we can use MF predictor with LT-GFNet to solve for all $t \in (l, 10 \times 4 \times \Delta t + l)$ for each l and then couple the predictions. However, as the LT-GFNet only solves for a time window of $10 \times 4 \times \Delta t$, we need to iterate until we solve for all $t \leq 2$ s (see Section 3.2.3 for a deeper development on how the updates are made).

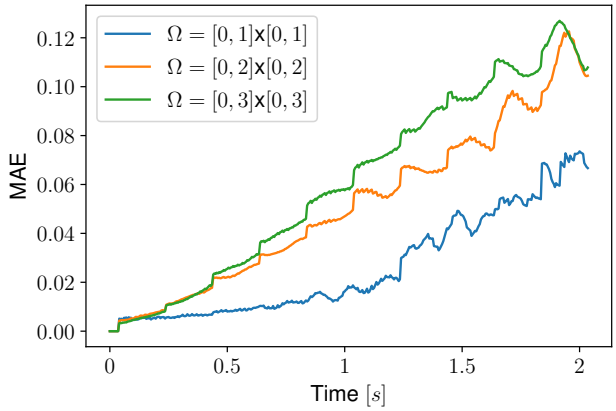


Figure 3.15 MAE evolution with respect to the predicted time for the MF predictor: We use MF predictor to approximate the wave solution for the domain of sizes $[0, 1] \times [0, 1]$, $[0, 2] \times [0, 2]$, and $[0, 3] \times [0, 3]$ and compare the predicted solution with the solution provided by finite differences. We observe that there are two main sources of error, one caused by the errors of the GFNet and another one caused by the errors generated when solving for the mosaic.

Figure 3.15 shows the evolution of the MAE against the predicted time when the MF predictor solution is compared to the FD solution. By inspecting these scenarios, we argue that MF predictor faces mainly two sources of error. First, the accumulated error when predicting for future steps, which can be clearly observed when MF predictor is used to solve for the $[0, 1] \times [0, 1]$ domain as all the BCs are known. We expected this error to have a continuous increase across time, but the results show that it increases in a step fashion every time the LT-GFNet is used on a new set of 10 time steps. We attribute this to a poor encoding of the

h_0 inputs. Second, the error caused by the iterative MF predictor approach, observed when MF predictor is used to solve for the $[0, 2] \times [0, 2]$ and $[0, 3] \times [0, 3]$. In these scenarios the error drastically propagates and accumulates in time reaching an MAE of 0.12 at times close to $t = 2$s. Figure 3.16 shows a detailed the evolution of the predicted wave for the domain $[0, 2] \times [0, 2]$.

Performance wise, MF predictor needs 4, 228 and 489 seconds to solve for the $[0, 1] \times [0, 1]$, $[0, 2] \times [0, 2]$, and $[0, 3] \times [0, 3]$, respectively. This apparent decrease in performance when compared to the NS and Laplace equations is due to the fact that we need to solve several IBVP with MF predictor to solve for times bigger than the window covered by LT-GFNet. In fact, a single solution for MF predictor converges within approximately 22s when used in the domain $[0, 2] \times [0, 2]$ which is comparable to the time needed to solve the NS equations for the square cavity.

3.4 Conclusions

This chapter has demonstrated that a well-designed and well-trained deep neural network that takes BCs as inputs (GFNet), coupled with a novel iterative algorithm for assembling its predictions (MF predictor) can result in a *transferable* deep learning framework for operator learning. To the best of our knowledge, such a framework is the first of its kind. The main advantage of this framework is that the model, GFNet, needs to be trained only once and can then be used forever without re-training to solve any IBVP or BVP in larger and complex domains with unseen sizes, shapes, and BCs.

Our framework demonstrates the capability to infer the solution of Laplace, Navier-Stokes and wave equations on domains that are 1200×, 12×, and 9× times larger that the training domains, respectively. Such scalable predictions are achieved without any re-training.

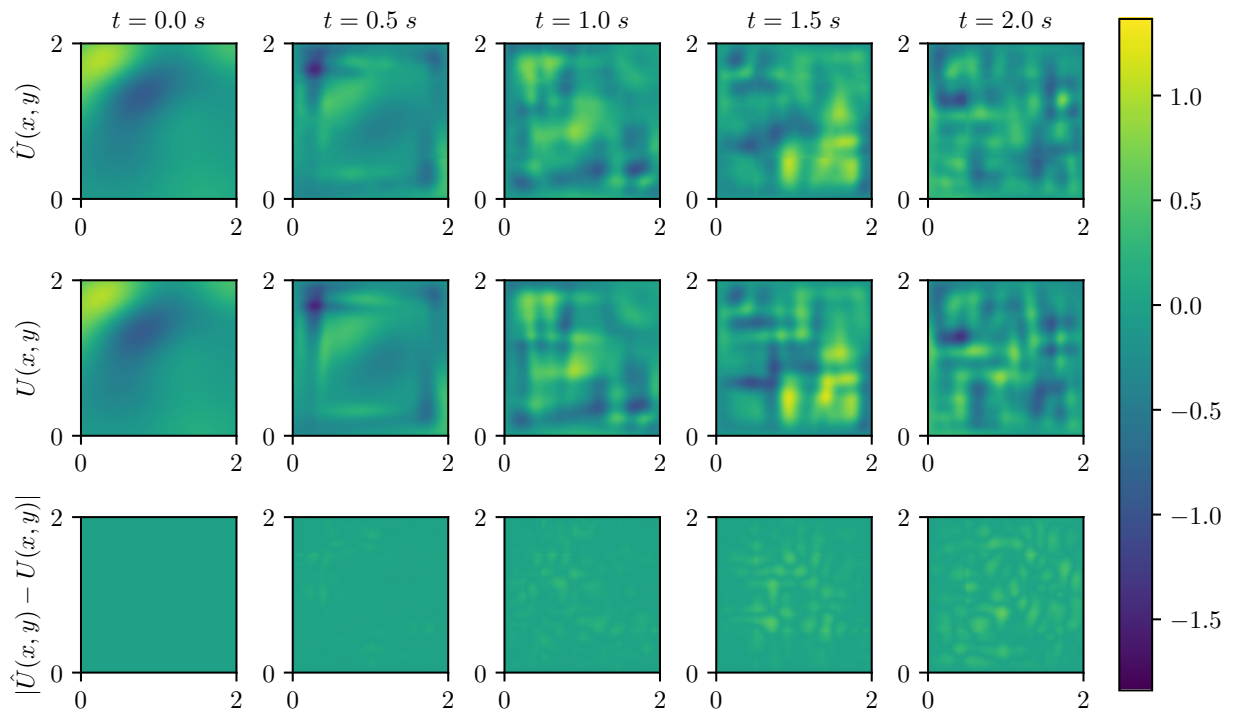


Figure 3.16 MF predictor and the wave equation: MF predictor solution for the wave equation. We use MF predictor to approximate the wave solution for a domain of size $[0, 2] \times [0, 2]$ for all $t \leq 0$. The predictions are compared with the exact solution solved using finite differences. At time $t = 0.5$s the MAE is 2.2e-2, at time $t = 1$s is 4.8e-2, at time $t = 1.5$s is 7.8e-2 and at time $t = 2$s it is 1.0e-1.

When compared with the state-of-the-art PINN, only for the Laplace and the NS PDEs, we demonstrate a remarkable 3-5 orders of magnitude speedups while achieving comparable or better accuracy across a range of BCs and domains unseen during training. We anticipate this research will open new directions for physics-informed surrogate modeling in computational sciences and engineering.

More work remains to be done for GFNet and MF predictor to be widely used for a larger class of problems in engineering. Some points to consider for future research lines that were not successfully solved in this chapter follow.

- Although we have introduced some innovative updating schemes to solve the complex problem of NS (adaptive MF predictor and ensemble MF predictor), they still require

fine tuning and research to significantly surpass the original MF predictor.

- The results on the wave equation can be considered preliminary and they generated some obvious questions still to be answered: Which is the best GFNet architecture for the wave equation? Why does the error present an step-like behavior and how can we solve it? How can we determine when the MF predictor is not reliable anymore?

- Our studies employed square genomes which cannot accurately represent curved objects such as airfoils. Category II GFNets and its better training performance open a lot of possibilities in this line of research.

- We only considered steady flows and boundary value problems whose right-hand-side function can be either zero or a periodic function whose period in each direction equals the size of the genome. The latter choice implies that the BVP solution on one genome is invariant to the genome's position in a large domain which is not a valid assumption for PDEs with space-dependent forcing terms.

- For category I GFNets, the training is a compromise between accuracy and limited time and GPU resources we can access. However, they provide the solution at any point $\boldsymbol{x}$ in the domain. Is there a way to combine both architectures, having the advantages of both?

,

Addressing all these questions is an important future exploration that can lead to some novel and groundbreaking results.

Chapter 4

Concluding remarks

This thesis presents different surrogate models for physical systems, based on GPs and DNNs, that are designed to provide enhanced inferences during extrapolation.

In Chapter 2, we focused on finding the governing dynamics for a physical system. We introduced evolutionary Gaussian processes (EGPs), a systematic integration of GPs and evolutionary programming (EP) that boost the performance of GPs by the automatic discovery of free-form symbolic bases that regress the data. We demonstrated this improved performance via examples that include a host of analytical functions as well as an engineering problem on materials modeling. The main contributions are threefold:

- We designed EGP as the systematic integration of GPs and EP. We showed that EGP can improve the performance of ordinary GPs in terms of not only extrapolation, but also interpolation/regression and numerical stability.
- By solving a real world engineering problem, we showed that EGP is a reliable surrogate for physical modelling that preserves the underlying physics where other emulators fail.
- We demonstrate, using a thought analysis on analytical functions, that EGP can pro-

vide accurate symbolic regressions that can give an extra insight on the governing dynamics of the problem.

In Chapter 3, we focused on surrogating the solution of a wide raneg of PDEs. We introduced a transferable framework for solving initial and boundary value problems (IBVPs) via deep neural networks which can be trained once and used forever for various domains of unseen sizes, shapes, and conditions. We tested our framework on three different well-known PDEs and showed that it outperforms the state-of-the-art approaches. There are three main contributions made in this chapter:

- We introduced *genomic flow network* (GFNet), a neural network that can infer the solution of an IBVP across arbitrary initial and boundary conditions on a square domain. For BVPs, we showed that GFNet excels in both performance and accuracy when compared with state-of-the-art approaches.
- We proposed *mosaic flow* (MF) predictor, a novel iterative algorithm that assembles the GFNet's inferences for IBVPs on large domains with unseen sizes and shapes while considering regularity of the solution.
- We improved the presented MF predictor using different updating schemes that include the MF predictor with optimization, the adaptive MF predictor, and the ensemble MF predictor.

We believe our contributions have promising future directions which are discussed in 2.4 and 3.4.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[2] Ramin Bostanabad. Reconstruction of 3d microstructures from 2d images via transfer learning. *Computer-Aided Design*, 128:102906, 2020.

[3] Ramin Bostanabad. Reconstruction of 3d microstructures from 2d images via transfer learning. *Computer-Aided Design*, 128:102906, 2020.

[4] Yu-Chin Chan, Faez Ahmed, Liwei Wang, and Wei Chen. Metaset: Exploring shape and property spaces for data-driven metamaterials design. *Journal of Mechanical Design*, 143(3), 2021.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–44, 2015.

[6] M Mozaffar, R Bostanabad, W Chen, K Ehmann, Jian Cao, and MA Bessa. Deep learning predicts path-dependent plasticity. *Proc Natl Acad Sci U S A*, 116(52):26414–26420, 2019.

[7] Stephan Rasp, Michael S. Pritchard, and Pierre Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018.

[8] Sourav Saha, Zhengtao Gan, Lin Cheng, Jiaying Gao, Orion L. Kafka, Xiaoyu Xie, Hengyang Li, Mahsa Tajdari, H. Alicia Kim, and Wing Kam Liu. Hierarchical deep learning neural network (hidenn): An artificial intelligence (ai) framework for computational science and engineering. *Computer Methods in Applied Mechanics and Engineering*, 373:113452, 2021.

[9] Youngjoon Suh, Ramin Bostanabad, and Yoonjin Won. Deep learning predicts boiling heat transfer. *Scientific Reports*, 11(1):5622, 2021.

[10] Liwei Wang, Yu-Chin Chan, Faez Ahmed, Zhao Liu, Ping Zhu, and Wei Chen. Deep generative modeling for mechanistic-based learning and design of metamaterial systems. *Computer Methods in Applied Mechanics and Engineering*, 372:113377, 2020.

[11] Huaiqian You, Yue Yu, Nathaniel Trask, Mamikon Gulian, and Marta D'Elia. Data-driven learning of nonlocal physics from high-fidelity synthetic data. *Computer Methods in Applied Mechanics and Engineering*, 374:113553, 2021.

[12] François Chollet. *Deep learning with python*. Manning Publications Co., 2017.

[13] I. Hassaninia, R. Bostanabad, W. Chen, and H. Mohseni. Characterization of the optical properties of turbid media by supervised learning of scattering patterns. *Sci Rep*, 7(1):15259, 2017.

[14] R. Bostanabad, B. Liang, J. Y. Gao, W. K. Liu, J. Cao, D. Zeng, X. M. Su, H. Y. Xu, Y. Li, and W. Chen. Uncertainty quantification in multiscale simulation of woven fiber composites. *Computer Methods in Applied Mechanics and Engineering*, 338:506–532, 2018.

[15] 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.

[16] Robert Planas, Nicholas Oune, and Ramin Bostanabad. Extrapolation With Gaussian Random Processes and Evolutionary Programming. In *Volume 11A: 46th Design Automation Conference (DAC)*, page V11AT11A004, Virtual, Online, August 2020. American Society of Mechanical Engineers.

[17] N. Cressie. The origins of kriging. *Mathematical Geology*, 22(3):239–252, 1990.

[18] J.D. Martin and T.W. Simpson. Use of kriging models to approximate deterministic computer models. *AIAA Journal*, 43(4):853–863, 2005.

[19] Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[20] Ramin Bostanabad, Biao Liang, Anton van Beek, Jiaying Gao, Wing Kam Liu, Jian Cao, Danielle Zeng, Xuming Su, Hongyi Xu, Yang Li, et al. Multiscale simulation of fiber composites with spatially varying uncertainties. In *Uncertainty Quantification in Multiscale Materials Modeling*, pages 355–384. Elsevier, 2020.

[21] Carl Edward Rasmussen. *Gaussian processes for machine learning*. The MIT Press, 2006.

[22] Mariette Awad and Rahul Khanna. Support Vector Regression. In *Efficient Learning Machines*, pages 67–80. Apress, Berkeley, CA, 2015.

[23] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[24] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[25] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proc Natl Acad Sci U S A*, 104(24):9943–8, 2007.

[26] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–5, 2009.

[27] Ioannis G Tsoulos and Isaac E Lagaris. Solving differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 7(1):33–54, 2006.

[28] Zihan Wang and Hongyi Xu. Quantitative Representation of Aleatoric Uncertainties in Network-Like Topological Structural Systems. *Journal of Mechanical Design*, 143(031713), January 2021.

[29] Sangjune Bae, Chanyoung Park, and Nam H. Kim. Estimating Effect of Additional Sample on Uncertainty Reduction in Reliability Analysis Using Gaussian Process. *Journal of Mechanical Design*, 142(111706), June 2020.

[30] Nicholas Oune and Ramin Bostanabad. Latent map gaussian processes for mixed variable metamodeling, 2021.

[31] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[32] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *arXiv preprint arXiv:1809.11165*, 2018.

[33] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation, 2013.

[34] S. Ba and V. R. Joseph. Composite gaussian process models for emulating expensive functions. *Annals of Applied Statistics*, 6(4):1838–1860, 2012.

[35] N. Zhang and D. W. Apley. Fractional brownian fields for response surface metamodeling. *Journal of Quality Technology*, 46(4):285–301, 2014.

[36] M. Plumlee and D. W. Apley. Lifted brownian kriging models. *Technometrics*, 59(2):165–177, 2017.

[37] R. Paulo. Default priors for gaussian processes. *Annals of Statistics*, 33(2):556–582, 2005.

[38] 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.

[39] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.

[40] Agustín Somacal, Leonardo Boechi, Matthieu Jonckheere, Vincent Lefieux, Dominique Picard, and Ezequiel Smucler. Uncovering differential equations from data with hidden variables, 2020.

[41] H. Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proc Math Phys Eng Sci*, 473(2197):20160446, 2017.

[42] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Sci Adv*, 3(4):e1602614, 2017.

[43] Georg Martius and Christoph H. Lampert. Extrapolation and learning equations, 2016.

[44] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–12, 2020.

[45] Michael Schmidt and Hod Lipson. *Symbolic regression of implicit equations*, pages 73–85. Springer, 2010.

[46] F. Chollet. *Deep learning with python*. Manning Publications Co., 2017.

[47] 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.

[48] W. Z. Zhang, R. Bostanabad, B. Liang, X. M. Su, D. Zeng, M. A. Bessa, Y. C. Wang, W. Chen, and J. Cao. A numerical bayesian-calibrated characterization method for multiscale prepreg preforming simulations with tension-shear coupling. *Composites Science and Technology*, 170:15–24, 2019.

[49] Hongyi Xu. Constructing Oscillating Function-Based Covariance Matrix to Allow Negative Correlations in Gaussian Random Field Models for Uncertainty Quantification. *Journal of Mechanical Design*, 142(7), 03 2020. 074501.

[50] R. B. Gramacy and D. W. Apley. Local gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics*, 24(2):561–578, 2015.

[51] B. MacDonald, P. Ranjan, and H. Chipman. GPfit: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs. *Journal of Statistical Software*, 64(12):1–23, 2015.

[52] P. Ranjan, R. Haynes, and R. Karsten. A computationally stable approach to gaussian process interpolation of deterministic computer simulation data. *Technometrics*, 53(4):366–378, 2011.

[53] J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31(1):41–47, 1989.

[54] D.J.J. Toal, N.W. Bressloff, and A.J. Keane. Kriging hyperparameter tuning strategies. *AIAA Journal*, 46(5):1240–1252, 2008.

[55] Charles Audet and John E Dennis Jr. Analysis of generalized pattern searches. *SIAM Journal on optimization*, 13(3):889–903, 2002.

[56] L. Zhao, K. K. Choi, and I. Lee. Metamodeling method using dynamic kriging for design optimization. *Aiaa Journal*, 49(9):2034–2046, 2011.

[57] D.J. Toal and et al. The development of a hybridized particle swarm for kriging hyperparameter tuning. engineering optimization. *Engineering optimization*, 43(6):675–699, 2011.

[58] Siyu Tao, Kohei Shintani, Ramin Bostanabad, Yu-Chin Chan, Guang Yang, Herb Meingast, and Wei Chen. Enhanced gaussian process metamodeling and collaborative optimization for vehicle suspension design optimization. In *ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 2B. American Society of Mechanical Engineers, 2017.

[59] J. R. Koza. Genetic programming as a means for programming computers by natural-selection. *Statistics and Computing*, 4(2):87–112, 1994.

[60] F. A. Fortin, F. M. De Rainville, M. A. Gardner, M. Parizeau, and C. Gagne. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(Jul):2171–2175, 2012.

[61] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[62] Dario Izzo, Francesco Biscani, and Alessio Mereta. Differentiable genetic programming. In *European Conference on Genetic Programming*, pages 35–51. Springer, 2017.

[63] Maxime Bassenne and Adrián Lozano-Durán. Computational model discovery with reinforcement learning. *arXiv preprint arXiv:2001.00008*, 2019.

[64] Jie Xiong, San-Qiang Shi, and Tong-Yi Zhang. A machine-learning approach to predicting and understanding the properties of amorphous metallic alloys. *Materials & Design*, 187:108378, 2020.

[65] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[66] Il'ya Meerovich Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967.

[67] Ilya M Sobol. On quasi-monte carlo integrations. *Mathematics and Computers in Simulation*, 47(2-5):103–112, 1998.

[68] Ellen M. Arruda and Mary C. Boyce. A three-dimensional constitutive model for the large stretch behavior of rubber elastic materials. *Journal of the Mechanics and Physics of Solids*, 41(2):389 – 412, 1993.

[69] R. Bostanabad, W. Chen, and D. W. Apley. Characterization and reconstruction of 3d stochastic microstructures via supervised learning. *J Microsc*, 264(3):282–297, 2016.

[70] M. A. Bessa, R. Bostanabad, Z. Liu, A. Hu, D. W. Apley, C. Brinson, W. Chen, and W. K. Liu. A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. *Computer Methods in Applied Mechanics and Engineering*, 320:633–667, 2017.

[71] Ted Belytschko, Wing Kam Liu, Brian Moran, and Khalil Elkhodary. *Nonlinear finite elements for continua and structures*. John Wiley & sons, 2013.

[72] Jerome H. Friedman and Werner Stuetzle. Projection Pursuit Regression. *Journal of the American Statistical Association*, 76(376):817–823, December 1981.

[73] Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1–67, March 1991.

[74] Songqing Shan and G. Gary Wang. Metamodeling for High Dimensional Simulation-Based Design Problems. *Journal of Mechanical Design*, 132(5):051009, May 2010.

[75] Robert W Fox, Alan T McDonald, and John W Mitchell. *Fox and McDonald's introduction to fluid mechanics*. John Wiley & Sons, 2020.

[76] Frank P Incropera, Adrienne S Lavine, Theodore L Bergman, and David P DeWitt. *Fundamentals of heat and mass transfer*. Wiley, 2007.

[77] Ted Belytschko, Wing Kam Liu, Brian Moran, and Khalil Elkhodary. *Nonlinear finite elements for continua and structures*. John wiley & sons, 2013.

[78] Erwin Kreyszig. *Advanced engineering mathematics*. John Wiley & Sons, 2010.

[79] Jeffrey Slotnick, Abdollah Khodadoust, Juan Alonso, David Darmofal, William Gropp, Elizabeth Lurie, and Dimitri Mavriplis. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. Technical Report March, 2014.

[80] Marc C Kennedy and Anthony O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[81] Robert Planas, Nicholas Oune, and Ramin Bostanabad. Evolutionary gaussian processes. *Journal of the Mechanical Design*, 2021.

[82] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[83] Terry Therneau, Beth Atkinson, Brian Ripley, and Maintainer Brian Ripley. rpart: Recursive partitioning and regression trees, 2014.

[84] 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.

[85] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[86] Andreas Griewank. On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107, 1989.

[87] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[88] Ameya D. Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[89] Chiyu Max Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Anima Anandkumar, and Others. MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework. 2020.

[90] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *arXiv preprint arXiv:.05385*, 2020.

[91] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. 2020.

[92] Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. Ppinn: Parareal physics-informed neural network for time-dependent pdes. *Computer Methods in Applied Mechanics Engineering*, 370:113250, 2020.

[93] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Trans Neural Netw Learn Syst*, 28(10):2222–2232, 2017.

[94] Jürgen Hochreiter, Seppassd Schmidhuber. Long short-term memory. *Neural Comput*, 9(8):1735–80, 1997.

[95] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

[96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[97] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

[98] Chiyu Jiang, Karthik Kashinath, and Philip Marcus. Enforcing physical constraints in cnns through differentiable pde layer. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

[99] Kailiang Wu and Dongbin Xiu. Data-driven deep learning of partial differential equations in modal space. *Journal of Computational Physics*, 408:109307, 2020.

[100] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490.

[101] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

[102] Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowlishwaran. CFDNet: a deep learning-based accelerator for fluid simulations. In *Proceedings of the International Conference on Supercomputing*, 2020.

[103] Nils Wandel, Michael Weinmann, and Reinhard Klein. Fast Fluid Simulations in 3D with Physics-Informed Deep Learning. *arXiv*, pages 1–10, 2020.

[104] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[105] Shaoxing Mo, Yinhao Zhu, Nicholas Zabaras, Xiaoqing Shi, and Jichun Wu. Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media. *Water Resources Research*, 55(1):703–728, 2019.

[106] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[107] Wenqian Dong, Jie Liu, Zhen Xie, and Dong Li. Adaptive neural network-based approximation to accelerate eulerian fluid simulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–22, 2019.

[108] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[109] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[110] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[111] Nathaniel Trask, Ravi G. Patel, Ben J. Gross, and Paul J. Atzberger. Gmls-nets: A framework for learning from unstructured data. *arXiv preprint arXiv:1909.05371*, 2019.

[112] N. Geneva and N. Zabaras. Quantifying model form uncertainty in reynolds-averaged turbulence models with bayesian deep neural networks. *Journal of Computational Physics*, 383:125–147, 2019.

[113] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[114] E Weinan and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[115] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:.00873*, 2019.

[116] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[117] Beichuan Deng, Yeonjong Shin, Lu Lu, Zhongqiang Zhang, and George Em Karniadakis. Convergence rate of deeponets for learning operators arising from advection-diffusion equations. *arXiv preprint arXiv:2102.10621*, 2021.

[118] Samuel Lanthaler, Siddhartha Mishra, and George Em Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. *arXiv preprint arXiv:2102.09618*, 2021.

[119] Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. pages 1–22, 2019.

[120] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[121] Ali Girayhan Özbay, Sylvain Laizet, Panagiotis Tzirakis, Georgios Rizos, and Björn Schuller. Poisson CNN: Convolutional Neural Networks for the Solution of the Poisson Equation with Varying Meshes and Dirichlet Boundary Conditions. pages 1–36, 2019.

[122] Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. Accelerating RANS turbulence modeling using potential flow and machine learning. pages 1–21, 2019.

[123] Shantanu Shahane, Purushotam Kumar, and Surya Pratap Vanka. Convolutional Neural Network for Flow over Single and Tandem Elliptic Cylinders of Arbitrary Aspect Ratio and Angle of Attack. *arXiv*, 2020.

[124] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. *34th International Conference on Machine Learning, ICML 2017*, 7:5258–5267, 2017.

[125] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

[126] Kai A Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394, 2009.

[127] L N Olson and J B Schroder. PyAMG: Algebraic Multigrid Solvers in Python v4.0, 2018.

[128] Eugene Brevdo Martín Abadi, Ashish Agarwal, Paul Barham, Andy Davis Zhifeng Chen, Craig Citro, Greg S. Corrado, Ian Goodfellow Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Yangqing Jia Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Mike Schuster Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Jonathon Shlens Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Paul Tucker Benoit Steiner, Ilya Sutskever, Kunal Talwar, Fernanda Viégas Vincent Vanhoucke, Vijay Vasudevan, Martin Wicke Oriol Vinyals, Pete Warden, Martin Wattenberg, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

[129] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[130] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and Mitigating Gradient Pathologies in Physics-Informed Neural Networks. *arXiv*, pages 1–28, 2020.

[131] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[132] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[133] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[134] Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, and Others. OpenFOAM: A C++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik Croatia, 2007.

[135] Shengze; Cai, Zhicheng; Wang, Sifan; Wang, and Perdikaris; Paris. Physics-Informed Neural Networks (PINNs) for Heat Transfer Problems. *Journal of heat transfer*, 2021.

[136] R L Iman, J M Davenport, and D K Zeigler. Latin hypercube sampling (program user's guide). [lhc, in fortran].

[137] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[138] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[139] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.