Data-Driven Correction Fields for Turbulence Modeling: A Priori and A Posteriori Study

Affan Asif Siddiqui

Delft University of Technology

TU Delft

Data-Driven Correction Fields for Turbulence Modeling: A Priori and A Posteriori Study

by

Affan Asif Siddiqui

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 28^{th} July, 2025 at 14:00.

Student number:	5934699
Project Duration:	November, 2024 - July, 2025
Thesis committee:	Dr. R.P. Dwight, TU Delft, Supervisor
	Dr. S.J. Hulshoff, TU Delft, Committee chair
	Dr. N. Anh Khoa Doan, TU Delft, Examiner
Faculty:	Faculty of Aerospace Engineering, Delft

TU Delft

Abstract

Computational Fluid Dynamics has been widely used to model flows for engineering applications. The challenge has been to model or resolve turbulent flows accurately and there exist different approaches such as Direct Numerical Simulations (DNS), Large Eddy Simulations (LES) and Reynolds Averaged Navier Stokes (RANS). With limitations on the available computational power and time, RANS is the favorable choice to model such flows despite significant modeling errors.

With recent advancements in machine learning (ML) algorithms and availability of high-fidelity (accurate) data, the turbulence community has been actively working to improve RANS modeling by informing it from the reference data and developing corrections. This is also known as data driven turbulence modeling. There exist several algorithms that have improved RANS modeling but their computational cost and lack of physical interpretability remains an issue to gain an understanding of flow physics and the relation between different flow quantities.

The present study implements two data driven methods to provide corrections ad improve RANS modeling, focusing on Turbulent Kinetic Energy and the Reynolds Stress Tensor. However, the goal is to also obtain symbolic models and physically interpretable corrections, while ensuring computational cost remains minimum.

A priori analysis, where only the reference data is used to develop corrections, shows significant reductions in the computational cost (function calls to the CFD solver) while converging to acceptable correlations as well. The superior of the two methods is selected for *a posteriori* study where it is coupled with a CFD solver. The correction is now formulated by differentiating the CFD code. This information is given to the ML algorithm which adjusts its parameters accordingly to optimize the objective function and improve turbulence modeling. Results show that, with regularization, a generalized symbolic formula can be obtained which when tested on different geometries improves the prediction of flow quantities and turbulence modeling, as a whole.

Preface

With an interest in *Experimental Aerodynamics*, little did I know I would enjoy my thesis so much in the *AI for Fluid Mechanics - Aerodynamics* department. The thesis has helped me explore my own strengths and understand what I excel at. From being very nervous in the initial thesis discussion and not knowing a lot about data-driven turbulence modeling to exceeding my own expectations with the extent of work done, the time has been both challenging and great.

I would like to thank my supervisor, Dr. Richard Dwight firstly for his support throughout the thesis. With the weekly meetings, we set the thesis objectives step by step as we progressed and approached problems from the very basics. As the thesis went on, the duration of our meetings increased and he spent time to explain, sometimes even the most trivial issues and ensured that I was on the right track. He was able to understand my ideas and gave me the freedom to explore them. I was able to maximize my productivity under his guidance and I really appreciate his efforts. I would also like to thank members of his research group: Renzhi, Kherlen and Tyler. I was closely associated with Renzhi from day 1 and he was a great mentor. He guided me and was always quick to respond. His insights during the initial research of the thesis and frequent discussions were critical and allowed me to quickly understand concepts and proceed with my thesis at an accelerated pace.

I am also deeply thankful to my friends from my master's. Dominik, Adrian, Gleb, Jan, Sibasish, Pouyan, Arham and all the friends I made later on; the discussions, coffee breaks, the laughs and meals we shared, it had a very positive impact on me and made sure I never stayed demotivated during my thesis and allowed me to enjoy life out of academics. Friends from my bachelor's who always called to check in on me; Ahmed Tufail, Huzaifa Shafiq, Areeb, Ahmad Hassan, H. Esbhani and many more, thank you for the continuous support. I would also like to express my sincerest gratitude to Haris Shahzad for helping me from day 1 of my master's and being the best mentor I could ask for. From guiding me on courses, how to approach my master's and helping me decide my thesis, I am truly grateful for your help as I would've experienced a very different master's journey if not for you. The list goes on, but I would also like to thank my friends: Abdul Wahab, Sheryar and Mubariz Zaffar for allowing me to crash to your places when I didn't have dinner, never hesitating to accommodate me and always treated me like a younger brother when giving suggestions. I am truly grateful for your kindness as it made my life a lot easier.

Finally, there's not enough strong words that I know of to convey this but my boundless gratitude goes to my parents and my sister. All this would not be possible without their support and their happiness in seeing me succeed. My sister would often school me whenever I stressed about the simple things and would be on call for hours comforting me and making sure I was alright. To my parents, who often sacrificed their sleep just to talk to me for a few minutes when I was working on my thesis day-night, it made me realize if at all I am at this stage and have done great in my thesis and my master's, it is not because I am a hard worker, or smart, but the credit goes to their efforts, unwavering support and faith in me. Thank you mom, dad and Abeer for making me stand where I am now!

Affan Asif Siddiqui
Delft, July 2025

Contents

List of symbols

Symbol	Description	Units
Greek Symbols		
α	Model coefficient in turbulence models	-
β	Context 1: k-ω SST model coefficient	-
	Context 2: Correction field	-
γ	Specific scalar invariant	-
δ_{ij}	Kronecker delta	-
ϵ	Dissipation rate	m^2/s^3
λ	Regularization parameter	-
μ	Dynamic viscosity	kg/(m·s)
ν	Kinematic viscosity	m^2/s
ν_t	Turbulent (eddy) viscosity	m^2/s
ρ	Density	kg/m^3
σ_b	Coefficient for b_{ij}^{Δ} correction	-
τ_{ij}	Reynolds stress tensor	m^2/s^2
ρ	Context 1: Density	kg/m^3
	Context 2: Pearson correlation coefficient	-
ϕ	Context 1: KANs activation function	-
	Context 2: Flux field	varies
ψ	Adjoint vector	-
ω	Specific dissipation rate	1/s
Latin Symbols		
b_{ij}	Anisotropic part of Reynolds stress tensor	-
b_{ij}^{Δ}	Correction term for b_{ij}	-
d	Wall distance	m
e	Specific internal energy	m^2/s^2
$f_1(x), f_2(x)$	Benchmark symbolic functions	varies
H	Height of periodic hill geometry	m
k	Turbulent kinetic energy	m^2/s^2
L	Characteristic length	m
L_x, L_y, L_z	Domain lengths	m
p	Pressure	Pa
$\hat{P}_k$	Modified turbulent production term	m^2/s^3
q_Q, q_T	Non-dimensional input features	-
q_{Re}	Local Reynolds number scaling	-
q_γ	Specific scalar invariant	-
$q_{\tau_k^B}$	Boussinesq stress anisotropy invariant	-
$q_{prod}, q_{diff}, q_{destr}$	RITA-based invariants	-
Re	Reynolds number	-
R^2	Coefficient of determination	-
S_{ij}	Strain-rate tensor	1/s
t	Time	s
T	Temperature	K
U_i, U_j	Velocity components	m/s
$\vec{U}$	Velocity vector	m/s
$\mathcal{U}_o$	Bulk (reference) velocity	m/s

V	Velocity magnitude	m/s
W_{ij}	Rotation-rate tensor	1/s
y^+	Wall-normal viscous coordinate	-
Other Notations		
∇	Gradient operator	1/m
$\nabla\cdot$	Divergence operator	1/m
∂	Partial derivative	varies
$\exp$	Exponential function	-
$\log$	Natural logarithm	-
$\tanh$	Hyperbolic tangent activation	-
$\mathcal{L}$	Loss function (MSE/RMSE)	varies
$\mathcal{R}$	Residual vector	varies
$\mathcal{I}$	Objective function	varies
RMSE	Root Mean Square Error	varies
$\vec{f}$	External body force vector	N/m^3

List of Abbreviations

Abbreviation	Definition
CFD Methods	
CFD	Computational Fluid Dynamics
DNS	Direct Numerical Simulation
LES	Large Eddy Simulation
RANS	Reynolds-Averaged Navier-Stokes
RSM	Reynolds Stress Model
SA	Spalart-Allmaras (turbulence model)
Turbulence & Physics	
RST	Reynolds Stress Tensor
TKE	Turbulent Kinetic Energy
NS	Navier-Stokes
PDE	Partial Differential Equation
Re	Reynolds Number
CD	Cross-Diffusion
Machine Learning	
ML	Machine Learning
PI-ML	Physics-Informed Machine Learning
ANN	Artificial Neural Network
TBNN	Tensor Basis Neural Network
KAN(s)	Kolmogorov-Arnold Network(s)
CFD-ML	CFD with Machine Learning Integration
Genetic Programming	
GP	Genetic Programming
CGP	Conventional Genetic Programming
LGP	Linear Genetic Programming
TGP	Tree-based Genetic Programming
GEP	Gene Expression Programming
HOF	Hall of Fame (evolutionary computing)
Optimization Methods	
TRF	Trust Region Reflective (optimizer)
LM	Levenberg-Marquardt (optimizer)
L-BFGS	Limited-memory Broyden-Fletcher-Goldfarb-Shanno
Adam	Adaptive Moment Estimation
Error Metrics	
RMSE	Root Mean Square Error
MSE	Mean Squared Error
Case Studies	
SD	Square Duct
PH	Periodic Hill
Software & Tools	
DAFoam	Design Adjoint with OpenFOAM

DEAP	Distributed Evolutionary Algorithms in Python
General Terms	
UQ	Uncertainty Quantification
QoI(s)	Quantity(ies) of Interest
AD	Automatic Differentiation
IC	Initial Conditions
BC	Boundary Conditions
PDF	Probability Density Function
CPU	Central Processing Unit
RITA	Relative Importance Term Analysis

1

Introduction

Ranging from the strong atmospheric winds, external flows over an airfoil to internal flows in ducts, engines and HVAC systems, the flow regime transitions to a turbulent flow state inevitably. It is a regime characterized by chaotic and irregular motion, resulting in separation regions and vortices. Given their presence, it is essential to model them accurately as it further has consequences on the aerospace, automotive and other engineering industries.

Computational Fluid Dynamics, based on the Navier-Stokes equations, is used to model turbulence in almost all engineering applications and simulate the flow to get an understanding of it. However, these equations are very complex and thus, there exists different approaches to model turbulence. While accuracy is preferred, the computational cost of such methods make them unlikely to be used for engineering applications. Direct Numerical Simulation, for example, is a high-fidelity approach where all the scales of the turbulent flow are resolved and the equations are solved numerically [51]. However, their costs can go as high as $\mathcal{O}(Re^3)$, making it impossible to be performed on non high-performance systems [12]. Large Eddy Simulation is another approach which introduces where the large scales are resolved while the small scales are modeled [85]. Although much cheaper at the expense of accuracy, it still remains unfavorable for everyday simulations.

The most common approach is Reynold Averaged Navier Stokes (RANS). It decomposes the flow into a mean and a fluctuating component and the resulting Navier-Stokes equations now have an additional stress term (Reynolds Stress Tensor), defined by the averaged product of fluctuating velocity components [3]. The RST is responsible to represent turbulent anisotropy and capture phenomena like separation and reattachment and wall shear distribution. However, most of the RANS models are based on the Eddy viscosity hypothesis which assumes a linear relation between the anisotropy and the mean rate of strain [61]. This does not hold for different flow regimes. Although computationally cheapest, it introduces a lot of errors due to modeling of the RST, aside from other sources such as averaging and coefficient calibration. The modeling of the RST is a well known problem and is referred commonly to as the closure problem. There also exist other RANS approaches such as Reynolds Stress Models to model turbulence. However, the improvement in results is not consistent and they are also computationally expensive.

High-fidelity data has generally been used to gather insights on RANS closure modeling and has been increasingly made available especially for canonical flow cases such as Periodic Hills. Furthermore, with recent developments in efficient algorithms and possible to store large data, the turbulence community is working to leverage these high-fidelity data sets and improve RANS modeling [19]. This is known as Data Driven Turbulence Modeling where the goal is to calibrate RANS models and inform them to obtain more correct results.

There have been several machine learning methods used to accomplish this such as Neural Networks, Random Forest and Evolutionary Algorithms among others. However, methods like Neural Networks, though used extensively, have a black box nature that results in a lack of interpretability of the modeling

process [5]. Evolutionary Algorithms can produce a symbolic formula but are computationally intensive to train as they require evaluating the objective function (individual quality) multiple times [93].

The modeling improvement can be accomplished in two ways: *a priori* and the *a posteriori* framework. *A priori* setting involves using high fidelity data as a reference to improve prediction of quantities such as RST and Turbulent Kinetic Energy based on fixed data. The idea is to develop equations which when inserted into the turbulence model should give improved results. However, the performance improvement may not be as good when implemented. It is explained that these discrepancies could exist due to time-averaged high fidelity data, which tend to produce good models for anisotropy stress tensor, but not so for other quantities[93]. To correct this, the *a posteriori* framework is used which involves running CFD simulations and optimizing algorithm parameters simultaneously which is accomplished by differentiating the CFD code. Thus, the corrections for instance, are produced by taking input from the CFD solution where the velocity field is updated at each solve, rather than relying on fixed data. The high fidelity data is now only used to evaluate the quality of the converged RANS results. In principal, the number of function calls in the *a priori* setting equals the number of CFD runs in an *a posteriori* setting. The *a posteriori* framework is also much more difficult as many models suffer from stability issues and the CFD solution is more susceptible to divergence [65].

With this, the objective of this thesis is to improve turbulent flow modeling in a RANS model by incorporating it with the necessary corrections. However, the focus of the thesis will not be on flow modeling only but also to develop computationally cheaper algorithms. There already exist many methods that can produce excellent correlations in the *a priori* setting. However, the challenge is to use them in the *a posteriori* setting where the number of function calls are the important factor. In addition, it is desirable to obtain a correction that can be physically interpretable and composed of flow quantities. In the *a posteriori* setting, the goal is to obtain a generalizable model that when applied to different geometries can produce improved results by improving turbulence modeling and not just serving as an ad-hoc correction.

Thus, this thesis will study algorithms first in the *a priori* setting and then select the optimum one and move to the *a posteriori* testing. Since the algorithms will be developed and are relatively new, they will be first tested on benchmark regression problems before moving to CFD datasets.

The thesis is outlined as follows: Chapter 2 gives a detailed background on CFD approaches and the governing equations. It highlights the work that has been done to to improve RANS modeling and concludes with the research questions for this study. Chapter 3 explains the methodology adopted in the development of the algorithms and the testing process by first implementing on benchmark datasets, and then the CFD datasets. It explains in detail how the algorithms work and the how the *a posteriori* framework is programmed. Chapter 4 presents the results of the algorithms on the benchmark datasets followed by conclusions which serve as strong indicators about the effectiveness of these algorithms before moving to Chapter 5 where these algorithms are then applied to CFD cases. The 2 CFD cases under consideration are Periodic Hill and Square Duct which serve as canonical flow cases where errors related to Turbulent Kinetic Energy and RST are dominant respectively. The optimum algorithm is selected and Chapter 6 explains the results when applied in the *a posteriori* framework. Finally, reflecting on the results, conclusions are drawn in Chapter 7, followed by future recommendations mentioned in Chapter 8.

2

Literature Review

This chapter begins with an overview of the governing equations in a Computational Fluid Dynamics (CFD) simulation, followed by approaches to model turbulence. Focusing on RANS, the limitations of these models in accurately capturing turbulent phenomena is discussed, providing the basis to study the role of data-driven turbulence modeling in improving their predictions of flow variables like turbulent production over regions of adverse pressure gradients. Since the study focuses on the implementation of Genetic Programming (GP), and Kolmogorov-Arnold Networks (KANs), recent advancements in these techniques are reviewed. The chapter concludes by emphasis on combining GP with gradient methods to enhance the modeling accuracy and convergence of such data-driven models, and the capabilities of KANs, followed by the research questions.

2.1. Navier-Stokes Equations

A CFD solution is governed by the conservation of mass, momentum, and energy. Mathematically, they are expressed in the form of continuity (Equation 2.1), momentum (Equation 2.2), and energy (Equation 2.3). These are referred to as the Navier-Stokes (NS) equations, and are the statements of fundamental physical principles, which influence the motion of fluid.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0, \tag{2.1}$$

$$\frac{\partial (\rho \vec{V})}{\partial t} + \vec{V} \cdot \nabla (\rho \vec{V}) = -\nabla p + \rho \vec{f} + \mu \nabla^2 \vec{V}, \tag{2.2}$$

$$\frac{\partial [\rho (e + \frac{V^2}{2})]}{\partial t} + \nabla \cdot [\rho (e + \frac{V^2}{2} \vec{V})] = \rho \dot{q} + k_{th} \nabla^2 T - \nabla (\vec{V} p) + \nabla (\vec{V} \tau) + \rho \vec{f} \cdot \vec{V}. \tag{2.3}$$

The Eulerian form of these equations is described above, and referenced from [3].

2.2. Turbulence and Modeling

Turbulence is a flow regime,characterized by unsteadiness, rotational motion, viscosity, and chaotic behavior, and is comprised of a wide range of length and time scales [90]. To indicate whether a flow has transitioned into such a regime, Reynolds number (Equation 2.4) is used, and computed as

$$Re = \frac{\rho V L}{\mu}. \tag{2.4}$$

The terms associated with viscosity, such as the non-linear diffusion term (Equation 2.2), are the mathematical expressions that include turbulence within a flow. As the Reynolds number for a flow increases, these non-linear processes dominate, and due to the influence of such terms, the Navier-Stokes equations is not trivial to solve [51]. Hence, there exists different methods to solve or approximate their solutions, which forms the basis of CFD models and turbulence modeling.

In a turbulent flow, the inertial forces dominate, with a continuous energy transfer. The largest scales in the fluid, also known as the macro-structures, generate turbulence energy, which is transferred to the smallest scales. This is the turbulence energy cascade and the process of vortex stretching, one of the drivers for three-dimensional turbulence, is responsible for this [51]. With turbulence production occurring at the macro-scales, the dissipation takes place at the smallest scales, also known as micro-structures. Based on scaling parameters, Kolmogorov scales for length (Equation 2.5), velocity (Equation 2.6), and time (Equation 2.7) are defined which represent the micro-structures present in a turbulent flow.

$$\eta_k = (\frac{\nu^3}{\epsilon})^{\frac{1}{4}}, \tag{2.5}$$

$$u_k = (\nu\epsilon)^{\frac{1}{4}}, \tag{2.6}$$

$$\tau_k = (\frac{\nu}{\epsilon})^{\frac{1}{2}}. \tag{2.7}$$

The scaling parameters, ν and ϵ, characterize the viscosity and amount of energy dissipation per unit time at the micro-scale level. Kolmogorov relation also states that the dissipation at the smallest scales directly scales (proportional) to the macro-structure.

2.2.1. Direct Numerical Simulation: DNS

Direct Numerical Simulation is able to capture all the length, time, and velocity scales in a flow, without any underlying assumptions or modeling of the turbulent flow, and accurately resolve the temporal evolution of the flow as well [51]. However, to completely resolve the flow field, the simulation domain must be large enough to contain the large-scale motions and have a sufficiently small spatial resolution to capture and resolve the smallest, i.e., Kolmogorov scales. For a 3-D simulation, the required number of grid points N_L, and the integration time step, N_T, to resolve the Kolmogorov length, and time scales is given by (Equation 2.8), and (Equation 2.9) respectively.

$$N_L^3 \approx \left(\frac{L}{\eta_K}\right)^3 \approx Re^{9/4}, \tag{2.8}$$

$$N_T = \frac{T}{\tau} \approx Re^{1/2}. \tag{2.9}$$

The total computational effort can thus be derived as

$$Cost_{DNS} = \mathcal{O}(Re^{9/4} \cdot Re^{1/2}) = \mathcal{O}(Re^{11/4}). \tag{2.10}$$

Given the order of the computational cost, DNS becomes unfavorable to produce results quickly, thereby limiting its applicability in the industry. Nonetheless, it remains a significant research tool in the academic field [12].

2.2.2. Large Eddy Simulations: LES

Unlike DNS, LES only resolves certain scales. It decomposes the flow into two different parts, based on a cut-off wavenumber (Equation 2.11), which determines the spatial scale of the fluid flow that must be resolved and those that should be modeled

$$\xi_C = \frac{\pi}{\Delta_C}. \tag{2.11}$$

Serving as a filter with width ΔC, all scales below this cut-off wavenumber (or larger than ΔC) are resolved, and the remaining scales modeled. The resolved scales, often referred to as the large scales of turbulence, carry most of the energy and are responsible for generating the Reynolds stresses, and thus, are computed [85]. The smaller scales, however, that are important for diffusion and molecular diffusion, contribute less to the Reynolds stresses, and are similar in all turbulent flows, hence, universal models are present. They are modeled in an LES simulation, and are not represented on the mesh.

As most of the dissipation occurs at these smaller scales, their neglect leads to a reduction in dissipation. The energy of the velocity scales is also lower due to filtering, which in turn leads to reduced

vortex stretching. Since vortex stretching is a key mechanism in turbulence production, its reduction causes a diminished generation of turbulent structures, resulting in LES producing less accurate results compared to DNS.

Nonetheless, although computationally less intensive than DNS, it still requires significant computational resources. In addition, implementing it can also demand a significant understanding and knowledge of discretisation, and placement of filter, among others features, making it difficult to be adopted easily.

2.2.3. Reynolds Averaged Navier Stokes: RANS

RANS decomposes the flow into a mean, and a fluctuating quantity, which when summed together, gives the actual flow field. The decomposition is stated below

$$u(\mathbf{x}, t) = U(\mathbf{x}) + u'(\mathbf{x}, t), \tag{2.12}$$

$$u = \bar{u} + u'. \tag{2.13}$$

where $\bar{u}$ (U) and u' ($u'(\mathbf{x}, t)$) are the mean and fluctuating quantities respectively. The above flow-quantity decomposition followed by Reynolds averaging procedure is applied through (Equation 2.1) to (Equation 2.3) to arrive at the averaged NS equations. It directly eliminates the need to solve or resolve for the turbulent structures, but attempts to model these structures on the time averaged flow [14]. This makes then computationally cheap when compared to DNS and LES solutions, and the widely adopted CFD methods in the industry as the averaged quantities contain sufficient information to perform flow analysis and compute quantities such as skin friction coefficient C_f and heat transfer.

In general, Reynolds averaging can take multiple forms involving an integral or a summation. The three common forms are time, spatial, and ensemble averaging [85]. Time averaging is an appropriate choice when the turbulent flow is statistically stationary, or the mean flow quantities do not vary with time such as the turbulent flow in the pipe driven a by a constant external source. The time averaged quantity based on the instantaneous flow variable $f(\mathbf{x}, t)$ is written as

$$F_T(\mathbf{x}) = \lim_{t\to\infty} \frac{1}{T} \int_t^{t+T} f(\mathbf{x}, t) dt. \tag{2.14}$$

Spatial averaging is the norm when the turbulent flow, on average, remains uniform in all direction, also referred to as homogeneous turbulence, and is described as

$$F_V(t) = \lim_{V\to\infty} \int\int\int_V f(\mathbf{x}, t) d\mathbf{V}. \tag{2.15}$$

Ensemble averaging, the most general type, is suitable for flows that decay in time, and defined as

$$F_E(\mathbf{x},\mathbf{t}) = \lim_{N\to\infty} \frac{1}{N} \Sigma_{n=1}^{N} f_n(\mathbf{x},\mathbf{t}). \tag{2.16}$$

Although the forms differ, there is no loss of generality as results for one type of averaging can also be valid for other kinds [85]. With steady state turbulent flows considered for this study, henceforth, time averaging will be considered, satisfying the relations

$$\bar{U}(\mathbf{x}) = \lim_{T\to\infty} \frac{1}{T} \int_t^{t+T} U(\mathbf{x}) dt = U(\mathbf{x}), \tag{2.17}$$

$$\bar{u'} = \lim_{T\to\infty} \frac{1}{T} \int_t^{t+T} [u(\mathbf{x}, t) - U(\mathbf{x})] dt = 0. \tag{2.18}$$

The NS equations, specifically continuity and momentum, after applying Reynolds averaging, can now be written as

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0, \tag{2.19}$$

$$\rho\frac{\partial\bar{u}_i}{\partial t}+\rho\bar{u}_j\frac{\partial\bar{u}_i}{\partial x_j}=-\frac{\partial\bar{p}}{\partial x_i}+\frac{\partial}{\partial x_j}\left[2\mu\bar{S}_{ij}-\rho\overline{u_i'u_j'}\right],$$
$$\text{where}\quad \bar{S}_{ij}=\frac{1}{2}\left(\frac{\partial\bar{u}_i}{\partial x_j}+\frac{\partial\bar{u}_j}{\partial x_i}\right) \tag{2.20}$$

The RANS equations now have an additional quantity in the form of $-\rho\overline{u_j'u_i'}$, which needs to be computed to evaluate the mean-flow quantities. The term is known as the Reynolds-stress tensor (RST), and forms the cause of all turbulent modeling. A transport equation for the RST can be derived and is

$$\frac{\partial u_i'u_j'}{\partial t}+K_{ij}=P_{ij}+T_{ij}+D_{\nu ij}+D_{ijp}+\Phi_{ij}-\varepsilon_{ij}. \tag{2.21}$$

The term such as $\mathbf{T}_{i,j}$ represents turbulent diffusion and is mathematically expressed as

$$T_{i,j}=\frac{\partial\overline{u_i'u_j'u_k'}}{\partial x_k}. \tag{2.22}$$

This further includes additional unknowns. Thus, deriving conservation equations for every unknown will lead to additional unknown quantities. The Reynolds-stress tensor (second order) is symmetric, resulting in six unknown quantities which is in addition to the unknown mean pressure and velocity components. On the other hand, there are 4 equations present (continuity, and three momentum equations). This is an under-determined system, and modeling equations are necessary to close the system.

2.3. Turbulence models for RANS

The following sections will explain the different turbulence modeling techniques and assumptions implemented for RANS.

2.3.1. Eddy Viscosity Hypothesis

The Reynolds stresses are modeled via the Eddy (Turbulent) viscosity hypothesis, referred to as the Boussinesq hypothesis. According to this, the deviatoric part of RST is proportional to the mean rate of strain. Mathematically, it can be expressed as (Equation 2.23)

$$-\rho\overline{u_i'u_j'}+\frac{2}{3}\rho k\delta_{i,j}=2\rho\nu_T\bar{S}_{i,j}. \tag{2.23}$$

The strain rate can be defined as

$$\bar{S}_{ij}=\frac{1}{2}\left(\frac{\partial\bar{u}_i}{\partial x_j}+\frac{\partial\bar{u}_j}{\partial x_i}\right). \tag{2.24}$$

The equations are referenced from [61]. ν_T acts as a proportionality scalar, and referred to as the eddy (turbulent) viscosity. The hypothesis assumes that the anisotropy of the Reynolds stresses,

$$a_{i,j}=-2\nu_T\bar{S_{i,j}}, \tag{2.25}$$

can be computed via the mean velocity gradients, and is proportional to the mean strain rate tensor. The RANS momentum equation, incorporating the Boussinesq hypothesis for turbulence closure, is expressed as

$$\frac{\partial\bar{u}_i}{\partial t}+\bar{u}_j\frac{\partial\bar{u}_i}{\partial x_j}=-\frac{\partial}{\partial x_i}\left(\frac{\bar{p}}{\rho}+\frac{2}{3}k\right)+\frac{\partial}{\partial x_j}\left[2(\nu+\nu_T)\bar{S}_{ij}\right], \tag{2.26}$$

It provides a straightforward closure to the RANS equations with one scalar unknown (ν_T), but the relation does not hold valid for different flow regimes. For example, in a wind tunnel with a straight section, followed by an axisymmetric contraction and a downstream straight section again, the mean strain in the downstream straight section is negligible. This implies that the Reynolds stress anisotropies

must also be negligible there, however, experimental data indicates otherwise [80]. A sketch of the wind-tunnel is shown below.

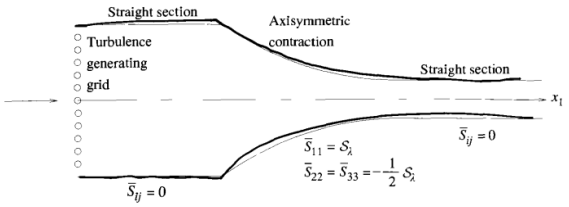


Figure 2.1: A similar wind tunnel sketch of the experiment [61]

The proposed hypothesis, however, becomes reasonable for simple shear flows, wherein slow evolution of local mean velocity gradients imply a relation between stresses and strains [61]. This is in contrast to turbulent shear flows, flows with significant curvature, and swirling flows where such an exact linear relation is difficult to establish [83] [2].

A number of different models are based on the mentioned Eddy viscosity hypothesis with improvements to tackle the mentioned issues.

2.3.2. Prandtl Mixing length models

These are further divided into zero, one and two equation models. In zero and one equation models, the mixing length, l_m, has to be empirically specified beforehand.

Zero- and One- Equation Models

The zero equation model, for a 2 dimensional flow, computes eddy viscosity as

$$\nu_T = l_m^2 \left| \frac{\partial \bar{U}}{\partial y} \right|, \tag{2.27}$$

Multiple generalizations have been applied to this model to make it applicable for free shear flows. It is one of the most simplest models, however, it lacks completeness as the mixing length, l_m needs to be specified. Furthermore, it is based on boundary layer assumptions, and this makes it difficult to apply for complex flows or away from the wall. Some equations to determine the l_m are

$$\begin{cases} 0.2\kappa u_\tau \delta_{99}, & y \geq 0.2\delta_{99}, \\ \kappa y \left(1 - e^{-y^+/A^+}\right), & y < 0.2\delta_{99}. \end{cases} \tag{2.28}$$

Here, κ and A^+ are von Kármán and van Driest damping constants respectively.

An improvement to this was the one equation model which includes a transport equation for turbulent kinetic energy, k in the turbulence model in the form of

$$\nu_T = c k^{1/2} l_m. \tag{2.29}$$

At high Reynolds number, dissipation scales with u^3 and $1/l$, hence, it can be modeled as

$$\varepsilon = C_D k^{3/2} / l_m. \tag{2.30}$$

Here C_D is a model constant. The transport equation for k reads as

$$\frac{\partial k}{\partial t} + \underbrace{\bar{u}_j \frac{\partial k}{\partial x_j}}_{\substack{\text{Convective} \\ \text{Term}}} = \underbrace{\tau_{ij} \frac{\partial \bar{u}_i}{\partial x_j}}_{P_k \text{ (Production)}} + \underbrace{\frac{\partial}{\partial x_j}\left(\left[\frac{\nu}{Re} + \frac{\nu_T}{Pr_k}\right]\frac{\partial k}{\partial x_j}\right)}_{D_k \text{ (Diffusion)}} - \underbrace{C_D \frac{k^{3/2}}{l_m}}_{\varepsilon \text{ (Dissipation)}}. \tag{2.31}$$

The left hand side of the equation represents the temporal change of turbulent kinetic energy and its transport by the mean flow (convection). Production refers to the transfer of energy from the mean

flow to turbulence, and dissipation refers to the loss of turbulent energy due to viscous effects, thus dominant at the small scales. The diffusion term represents the distribution of turbulent kinetic energy due to molecular diffusion or viscous effects (ν/Re) and turbulent diffusion (ν_T/Pr_k).

Although an improvement over the zero equation model, it still remains incomplete due to the need to specify l_m beforehand.

Two-Equation Models
These models involve transport equations for two variables, and eliminate the need to specify l_m beforehand, making them "complete". The formulations of the commonly used models are explained below.

The $k-\varepsilon$ model solves two transport equations; one for the turbulent kinetic energy, k and the other for turbulence dissipation rate, ε [34]. The equations to compute the eddy viscosity, and the transport equations used are

$$\nu_T = C_\mu \frac{k^2}{\varepsilon}, \quad C_\mu = 0.09, \tag{2.32}$$

$$\frac{\bar{D}k}{\bar{D}t} = \underbrace{\tau_{ij}\frac{\partial \bar{u}_i}{\partial x_j}}_{P_k \text{ (Production)}} + \underbrace{\frac{\partial}{\partial x_j}\left(\left[\frac{\nu}{Re}+\frac{\nu_T}{\sigma_k}\right]\frac{\partial k}{\partial x_j}\right)}_{D_k \text{ (Diffusion)}} - \underbrace{\varepsilon}_{\text{Dissipation}}, \tag{2.33}$$

$$\frac{\bar{D}\varepsilon}{\bar{D}t} = \underbrace{C_{\varepsilon 1}\frac{P_k \varepsilon}{k}}_{P_\varepsilon \text{ (Production)}} + \underbrace{\frac{\partial}{\partial x_j}\left(\frac{\nu_T}{\sigma_\varepsilon}\frac{\partial \varepsilon}{\partial x_j}\right)}_{D_\varepsilon \text{ (Diffusion)}} - \underbrace{C_{\varepsilon 2}\frac{\varepsilon^2}{k}}_{\text{Dissipation}}, \tag{2.34}$$

The standard values of the constants involved are $C_\mu = 0.09$, $C_{\varepsilon 1} = 1.44$, $C_{\varepsilon 2} = 1.92$, $\sigma_k = 1.0$, and $\sigma_\varepsilon = 1.3$

The model by itself however, is not completely accurate. It often requires modifications in the form of wall functions to make it accurate for near-wall regions. Furthermore, the model frequently struggles with flows that have strong pressure gradients, stream lime curvature or separation, as it can over-predict turbulence in such cases.

Another variation of the two equation model is the standard $k-\omega$ model wherein a transport equation for the specific turbulence dissipation rate, ω is solved [84]. The transport equation is written as

$$\omega \equiv \varepsilon/k, \tag{2.35}$$

$$\begin{aligned}\frac{\bar{D}\omega}{\bar{D}t} = &\underbrace{(C_{\varepsilon 1}-1)\frac{P_k \omega}{k}}_{P_\omega \text{ (Production)}} + \underbrace{\frac{\partial}{\partial x_j}\left(\frac{\nu_T}{\sigma_\omega}\frac{\partial \omega}{\partial x_j}\right)}_{D_\omega \text{ (Diffusion)}} - \underbrace{(C_{\varepsilon 2}-1)\omega^2}_{\text{Dissipation}} \\ &+ \underbrace{\frac{2\nu_T}{\sigma_\omega k}\nabla\omega\cdot\nabla k}_{\text{Cross-Diffusion}}.\end{aligned} \tag{2.36}$$

The $k-\omega$ model gives more accurate results in the near-wall regions and in areas of streamwise pressure gradients as well, when compared to $k-\varepsilon$. However, a treatment of non-turbulent free-stream boundaries can pose a concern, along with the over estimation of turbulence production at stagnation points. Thus for this reason, $k-\varepsilon$ maintains its superiority over external aerodynamic flows.

A blend of the above two, $k-\varepsilon$, and the $k-\omega$, into a new model was then proposed [48]. Incorporated with a blending function, it switches to $k-\varepsilon$ when away from the wall regions (wake regions and shear layers), while $k-\omega$ when modeling the flow close to the wall (boundary layers). This gives the best of

both the models without being relatively computationally expensive. The $k-\omega$ SST model is described as

$$\frac{D\rho k}{Dt} = P_k - \beta^* \rho\omega k + \frac{\partial}{\partial x_j}\left((\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_j}\right), \tag{2.37}$$

$$\frac{D\rho\omega}{Dt} = \frac{\alpha}{\nu_t}P_k - \beta\rho\omega^2 + \frac{\partial}{\partial x_j}\left((\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_j}\right) + 2(1-F_1)\rho\sigma_{\omega 2}\frac{\nabla k \cdot \nabla\omega}{\omega}, \tag{2.38}$$

where $F_1 =$

$$\tanh\left[\left(\min\left[\max\left(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\sigma_{w2}k}{\mathsf{CD}_{k\omega}y^2}\right]\right)^4\right]. \tag{2.39}$$

The eddy viscosity is computed as

$$\nu_t = \frac{a_1 k}{max(a1\omega, \Omega F_2)} \tag{2.40}$$

with F_2 as

$$\tanh\left[\left(\max\left(\frac{2\sqrt{k}}{\beta^*\omega d}, \frac{500\nu}{d^2\omega}\right)\right)^2\right], \quad \mathsf{CD}_{k\omega} = \max\left(2\sigma_{w2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}, 10^{-10}\right). \tag{2.41}$$

The numerical values of the constants are defined below.

$$\begin{aligned}\alpha_1 = 0.553,\ \alpha_2 = 0.44,\ \beta_1 = 0.075,\ \beta = 0.0828,\ \sigma_{k1} = 0.85,\ \sigma_{k2} = 1.0,\\ \sigma_{w1} = 0.5,\ \sigma_{w2} = 0.856,\ \beta^* = 0.09,\ \kappa = 0.41,\ a_1 = 0.31\end{aligned} \tag{2.42}$$

The model coefficients, for instance ψ are obtained by blending the coefficients between $k-\omega$ (ϕ_1) and $k-\varepsilon$ (ϕ_2) and accordingly, as

$$\psi = F_1 \cdot \phi_1 + (1-F_1)\phi_2. \tag{2.43}$$

An additional model that is also commonly used in the industry is the Spalart-Allmaras model which is based on one equation [69].

2.3.3. Reynolds Stress Models: RSM's

In contrast to mixing-length models outlined in Subsection 2.3.2, Reynolds Stress Models (RSM) solve the model transport equations for the individual Reynolds stresses. Thus, they directly rule out the limitations of Boussinesq hypothesis. The exact transport equation for the Reynolds stresses are

$$\frac{\bar{D}\overline{u_i' u_j'}}{\bar{D}t} + \underbrace{\frac{\partial T_{k,ij}}{\partial x_k}}_{\text{Diffusion}} = \underbrace{P_{ij}}_{\text{Production}} + \underbrace{R_{ij}}_{\text{Pressure-Strain}} - \underbrace{\varepsilon_{ij}}_{\text{Dissipation}}. \tag{2.44}$$

$T_{k,i,j}$, $R_{i,j}$ and $\varepsilon_{i,j}$ represent Reynolds stress flux, pressure rate of strain, and dissipation tensor respectively. These quantities also require models, while the other quantities are in closed form [61].

For high Reynolds number flows, the dissipation as a consequence of local isotropy, can be modeled as

$$\varepsilon_{i,j} = \frac{2}{3}\varepsilon\delta_{i,j} \tag{2.45}$$

For moderate Reynolds number flows, the isotropic modeling may not hold valid, but the effect is negligible.

The pressure rate of strain tensor redistributes energy among Reynolds stresses and can be decomposed into three components, namely the rapid, slow, and the harmonic pressure. Analogously, the pressure rate of strain tensor can also be decomposed into three parts as

$$R_{i,j} = R_{i,j}^{(r)} + R_{i,j}^{(s)} + R_{i,j}^{(h)}. \tag{2.46}$$

A generic model for the slow and rapid term was developed and is

$$\Phi^S_{i,j} = -\varepsilon(C_1 a_{i,j} + C_2(a_{i,k}a_{k,j} - \frac{1}{2}P_{i,j}\delta_{i,j})), \tag{2.47}$$

$$\Phi^R_{i,j} = k\frac{\partial u_k}{\partial x_l}(X_{kjli}(a_{nm}) + X_{kilj}(a_{nm})). \tag{2.48}$$

Based on this, two different modeling approaches were derived and proposed, namely the LRR RSM (Launder, Reece and Rody), and the SSG RMS(Speziale, Sarkar and Gatzki) [41] [24].

These are some of the fundamental models that are used in reference to the RSM's.

2.4. Comparison of RANS models

The zero and one equation models are limited to simple flows, and not complete as they require the specification of the mixing length. Spalart-Allmaras is efficient for aerodynamic flows, but similarly involves near wall distances, making it not the ideal model. The $k-\varepsilon$ model is widely regarded as being easy and computationally inexpensive. However, the source terms in these equations close to the walls become large, and thus without wall functions, this model can become inaccurate. This can be mitigated to a large extent by the $k-\omega\ SST$ model which combines the $k-\omega$, and $k-\varepsilon$ models. Inheriting the advantages of both, it is able to balance near-wall and far-field accuracy, while only being slightly more expensive.

Comparing the RSM, despite giving superior results for flows with strong anisotropy and curvature, it is more computationally expensive as there are seven turbulence equations to be solved for, instead of two [61]. There exist algebraic stress models as alternatives to avoid the cost and difficulty of RSM, however, these benefits are not realized generally. This is in addition to the reduced numerical stability effects when compared to $k-\omega\ SST$, as it is sensitive to modeling of the pressure rate strain tensor.

2.5. Limitations of RANS modeling

As explained earlier in Subsection 2.3.1, the zero, one, and two-equation models are based on the Eddy viscosity hypothesis, assuming that the anisotropy of the Reynolds stress tensor is linearly dependent on the strain rate tensor. This only holds for limited flows. Although the two equation models tend to correct for this up to some extent by including two transport equations, the models are still inaccurate due to the underlying Eddy viscosity hypothesis assumption. Hence, the closure of these Reynolds stresses is a well known problem.

Many industrially tested flow fields and setups have a well defined averaged and a unique solution. However, RANS models often fail to accurately predict the correct mean flow solutions, such as in flow separation cases. Furthermore, many ad-hoc fixes are proposed to the existing RANS models to flow phenomena such as high speed cold-wall heat transfer, and mixing layers with density or temperature variations. Although these ad-hoc fixes give acceptable results, they are not easily generalizable [9].

In addition, RANS models are also affected by the averaging operator. Although the Reynolds averaged equations, the flow decomposition (into average and fluctuating values), and the Reynolds stresses are valid, the modeled stresses or their formulations are only local as it involves the turbulence variables and derivatives at one point only [71]. Particles are involved in different turbulent eddies and span across regions of the flow. The turbulent motion at one point may be influenced by remote areas as well. This highlights the issue of non-local effects with RANS modeling, in addition to the already existing closure problems. For instance, in the wake of a cylinder, the averaging process often groups together instances when a vortex is passing through, with instances when irrotational flow is passing between the vortices. This averaging can be considered as unnatural. However, the localization of turbulence by RANS is not an absolute effect. The use of wall distance d in the Spalart-Allmaras and the $k-\omega\ SST$ (Menter) models perform well, despite the usage of the wall distance being argued as deviating from purely local properties.

Moreover, the conventional Reynolds stress transport equations do not make use of second or higher order derivatives of for instance, velocity ($\frac{\partial^2 U_i}{\partial x_j \partial t}$, $\frac{\partial^2 U_i}{\partial x_j \partial x_k}$). They can be used to incorporate the history effects in an empirical manner. Although complex and may lead to incompatibility with discretisation

schemes when used, stability problems were not reported when applied to SARC (Spalart-Allmaras with Rotation and Curvature model) [72].

2.6. Other Methodologies

Aside from the explained models, hybrid approaches exist to balance computational cost, and accuracy. They couple LES and RANS with a distinct interface. However, there are challenges because RANS is not able to provide the correct instantaneous fluctuations for LES to resolve large scales. Thus, LES cannot compute the correct velocity gradients, leading to under prediction of RST in the logarithmic region of the boundary layer. This is generally overcome by injecting artificial fluctuations. A blended hybrid approach, referred to as Detached Eddy Simulation (DES), uses unsteady NS equations with eddy-viscosity models but suffers from significant RANS to LES transition regions due to inconsistencies caused by grid spacing resulting in a premature transition to LES [70]. Although more accurate than RANS, these methods have their own set of own challenges and will not be discussed in the present study.

An illustration of the different CFD methods that exist is shown below, along with a visualization of the differing results between RANS, LES and DNS for the same simulation setup.

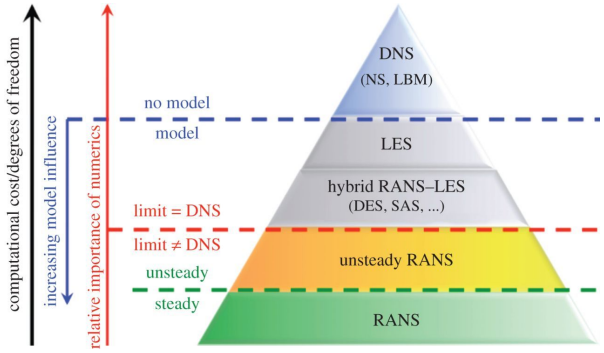


Figure 2.2: A visual comparison of different CFD methods in terms of cost and accuracy [16]

Figure 2.2 shows us that while DNS has the least model influence, it is also the most computationally expensive. RANS on the other hand, relying significantly on modeling but the lower computational cost makes it a favorable choice.

With this, the k-ω SST model is selected as the optimal model for the current study. The focus of the following chapters will therefore, be on enhancing the modeling capabilities of it through the development of data-driven turbulence closures, via techniques explained in later sections.

2.7. Uncertainty Quantification

It is established that the closure terms of RANS to model the Reynolds stresses result in non-reliable predictions, and inaccurate mean flow quantities. However, these errors can be quantified in probabilistic terms using uncertainty quantification techniques. In addition, using machine learning and optimization algorithms, these model closure terms can be re-calibrated to give improved results.

While constructing a RANS model, several assumptions are made which result in information being lost and introducing errors in the overall model. The four levels of simplification required to formulate a RANS closure are mentioned below [19].

- Level 1 (L1) uncertainties are introduced due to the averaging operator applied on the Navier-Stokes equations. Since RANS is based on this averaging operator, it is unavoidable but at the same time, the loss in information is irrecoverable as well. With many distinct realizations of a flow quantity compatible with the averaged field, it results in a hysteresis phenomena.
- Uncertainties introduced in the development of closures for the representation of Reynolds stresses comprise Level 2 (L2) uncertainties. Eddy viscosity models, and algebraic stress models are com-

mon examples, which assume that the Reynolds stress tensor is a function of local, averaged flow quantities.

- Further simplifications to denote physical processes without having to derive their differential models to reduce computational cost results in Level 3 (L3) uncertainties. Some processes in the NS equations include convection, and diffusion among other contributors.
- Once the complete model form has been established, the coefficients that are involved, such as those shown in Subsection 2.3.2 constitute the final uncertainties, known as Level 4 (L4). Their calibration is essential to ensure the proper contribution of different terms in the closure equation, which otherwise can lead to significant errors.

These combined 4 levels make it difficult for a RANS model to deliver consistent results for different type of flow fields, and different methods have been used to quantify these uncertainties. Background flow technique makes use of bounds for the unclosed terms. These bounds are based on theoretical arguments, and represent the extreme values a term can take during a flow simulation. They can be estimated via variational approaches to obtain a generic framework [17] [32]. However, it is practically challenging to define bounds for different quantities of interest such as dissipation rate, and turbulent kinetic energy [66]. An attempt to quantify L2 uncertainties was made by introducing realizable, physics-constrained perturbations to the RST [22]. This was based on the eigen value decomposition of the RST. Additional attempts include wherein the Reynolds Stress (model form) uncertainties were modeled in the form of a probabilistic description [87]. A maximum entropy distribution was made use of to indicate the uncertainties.

2.8. Data Driven Turbulence Modeling

Validation of different CFD models has usually been carried out with experimental data, or high-fidelity data has been used to gather insights on the modeling of RANS closures. Recently, with efficient algorithms and the possibility to store large data, the same data is now being used to calibrate RANS models and inform them, to improve closure and obtain physically more correct results. The choice of coefficients or optimal parameters has an effect on the results, and this leads to L4 uncertainties. The impact of it can be studied via varying the choice of coefficients, and finding the optimal distribution of parameter values. However, a more effective approach is to use high fidelity data to tune or inform these coefficient values and propagate the resulting expressions into simulations to obtain improved modeling of flow physics. A related example is the influential work by Kennedy and O'Hagan wherein a discrepancy (correction) term was added to the model prediction to obtain a more accurate model output of computer models, using Bayesian calibration [36].

Given Section 2.7 where the causes of uncertainties, and briefly the different methodologies to quantify them were highlighted, works that involve high-fidelity data to improve the overall performance of RANS and reduce the uncertainties will now be focused on, which is also the objective of the current research.

Bayesian Optimization

One of the first works where high fidelity data (DNS) was used, dates back to 2011 [11]. In this study, Bayesian inference was used to assign posterior probability distributions to model parameters to obtain an optimal range of values for plane channel flows. This was a way to correct for L4 uncertainties. In addition to this, the same group (in a different study) proposed the use of Reynolds stress discrepancy tensor to account for overall uncertainties [53]. A random field defined by differential equations that were simpler than the Reynolds stress transport equations was used, and their study enjoyed success for plane channel flows at different Reynolds numbers. Another work involved the use of a complete DNS velocity field to compute the viscosity field of a $k-\omega$ turbulence model, and the discrepancy was modeled via a Gaussian process [18]. Furthermore, sparse velocity fields were also used in a physics informed Bayesian framework to infer the structure of the Reynolds stress magnitude and its anisotropy. Even with sparse data, the simulation data had better observations to the benchmark model when compared to the baseline model [88]. Moreover, another study made us of limited observations of high fidelity simulation and experimental data and compared it with RANS models to infer the discrepancies in their source terms [67] [75]. This information in a Bayesian setup (field inversion) was utilized and applied to channel flow, shock-boundary layer iterations, and flows with curvature and separation. It was concluded that the accuracy of the propagated solution was improved over the entire domain, by

addressing the connection between physical data and model discrepancies. Some of the results of their work are presented below.

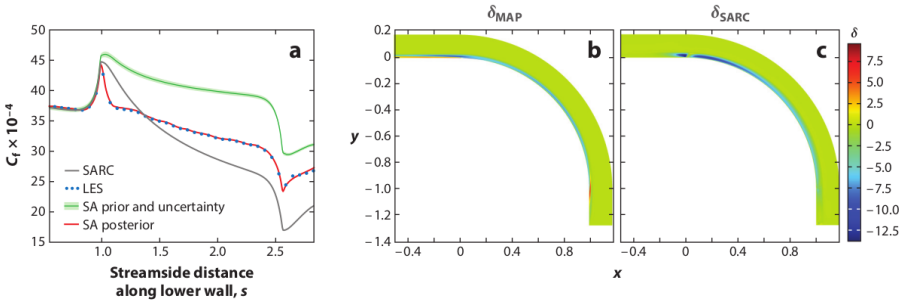


Figure 2.3: An illustration of the model improvement [67]

In Figure 2.3, δ_{MAP}, and δ_{SARC} represent the correction terms. The Spalart-Allmaras, after optimization, gives improved results and matches with LES, unlike the prior Spalart-Allmaras and Spalart-Allmaras with rotation and streamline curvature models (SARC). The quantities illustrated are, from left, the skin friction coefficient predictions, correction terms. In similar work, inverse modeling was coupled with the adjoint method, to build a statistical model for the uncertainties in the $k-\omega$ model [79]. The difference in the turbulent viscosity by the inverse RANS problem, and that of baseline RANS model was described, in comparison to DNS data. This model, further developed by maximum likelihood estimation, was used to propagate uncertainties to different quantities of interest. Elsewhere, a posteriori error estimate was developed using Bayesian calibration with experimental data [21]. This was then followed by Bayesian Model Scenario Averaging (BMSA) to gather the posteriors to provide an estimate for the quantity of interest. It was concluded that the mean of the estimate was more accurate than the individual model predictions.

2.8.1. Applications of Machine Learning

The challenge of the current study is to similarly leverage high fidelity data-sets and improve RANS modeling. However, this will not just be limited to one type of flow such as the literature reviewed above, but rather, to generalize the data or evidence to a class of flows sharing similar features, and understand the relation between different flow quantities.

Machine learning (ML) has gained popularity due to recent advancements in data storage capacities, data availability and efficient algorithms. It can be categorized into supervised and unsupervised learning. Since the present study deals with constructing a mapping between a set of given input vectors to accurately model the output, the focal point will be around supervised learning. Specifically, regression approaches of supervised learning are of particular importance in view of turbulent modeling. Unsupervised learning, on the other hand, deals with the discovery of patterns and to reduce the complexity of data, and will not be touched upon. Works involving supervised learning algorithms to effectively correct for the errors in RANS modeling are reviewed in the following segment.

Some of the earliest works that used ML techniques for CFD were related to turbulence and combustion modeling [74]. High fidelity (DNS) data was used to construct a stochastic model of the error for the $k-\omega$ SST model. The technique was applied to model combustion in a turbulent mixing layer, and to predict the anisotropy of the Reynolds stress tensor in a non-equilibrium boundary layer flow. Addressing the L3 uncertainties, the methodology is generalizable and applicable to any turbulence model. The local error model was derived based on kernel regression and was applied to RANS model. Better approximations on the prediction of source term in the combustion test case, and improved prediction of turbulent anisotropy as a function of various mean flow properties. An additional advantage of applying ML techniques is that the fundamental properties of modeling are not compromised, such as invariance. This was achieved by using tensor invariants of the Reynolds stresses, based on eigen decomposition.

Neural Networks

Recently, neural networks have been wide adopted due to their ability to process complex data. They are able to analyze and make functional forms for complex nonlinear models and can be used for high dimensional data as well [46]. The group of Duraisamy made use of neural networks, with a single hidden layer, for turbulence model development [75]. In this case, the Spalart-Allmaras model was improved using DNS data, via neural networks. It was observed that the trained network was able to improve the baseline model for two-dimensional flat plate boundary layers, and three-dimensional transonic wings. In another work, spatial and temporal dependent terms were introduced to the RANS model equation to compensate for the errors in the baseline model [92]. Fast Artificial Neural Networks and inversion techniques were used to construct correction terms in channel flows, and bypass transition. Elsewhere, a similar methodology of inversion technique coupled with neural networks was used to infer the functional form of deficiencies in closure models [20]. The inverse modeling step, in itself, can give valuable insights and information, which is then converted into modeling knowledge via ML techniques.

A significant work is that of Ling et al., where they incorporated multiple hidden layers and ensured invariance [42]. Invariance is important to make the turbulent flow independent of a frame of reference, and their methodology also provides the basis for the present work. By decomposing the anisotropy Reynolds stress tensor and forcing it to be composed of a basis of isotropic tensors, Galilean (rotational) invariance was embedded into the system. The mentioned invariance is defined as if the coordinate frame is rotated by a certain degree, the anisotropy tensor also rotates by the same. This was accomplished by constructing an integrity basis of the input tensors. With rotational and strain rate tensors taken as the inputs, the anisotropy tensor, b, with Galilean invariance, was represented as a linear combination of of 10 isotropic basis tensors as

$$\mathbf{b} = \Sigma_{n=1}^{10} g^{(n)}(\lambda_1, .., \lambda_5) T^{(n)}. \tag{2.49}$$

If interested, further literature on the decomposition can be reviewed from [62]. The objective of the TBNN framework was to determine the scalar coefficients, $\lambda_1, .., \lambda_5$ and then compute the anisotropy stress tensor. Two cases were tested: flow over wavy wall and a square duct setting. Comparing the TBNN framework with a generic multi layer perceptron (MLP) network without the invariance, the former was more accurate, and also produced improved mean velocity profiles (a posteriori setting) when compared to eddy viscosity models.

Although neural networks have been used extensively to improve turbulence modeling, they have several tunable parameters such as activation functions, number of layers, and regularization schemes. Furthermore, they can also be quite computationally intensive to train on a large (and a non-linear complex) dataset. Moreover, it is their black box nature that results in a lack of interpretability of the modeling process.

Random Forest

Building on TBNN, and to incorporate Galilean invariance, the tensor basis random forest (TBRF) was then introduced for RANS modeling [35]. The random forest algorithm was modified to accept a tensor basis, and used to improve the predictions of anisotropy stress tensor by leveraging DNS/LES data. Stabilization was achieved via a continuation method and a modified k-equation. The algorithm was easier to train, and had few hyper-parameters, making it less computationally intensive in comparison to neural networks [26]. The resulting model was able to provide more accurate mean flow fields. Some results from the work are shown below.

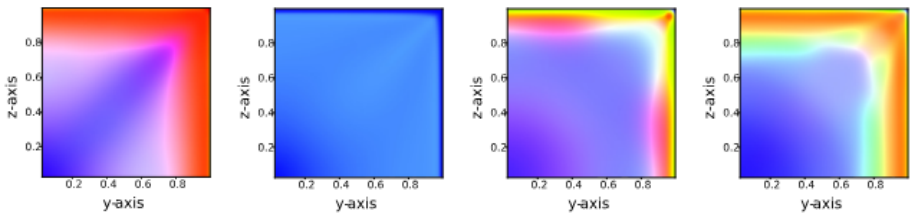


Figure 2.4: A comparison of TBRF with other methods
[35]

Contours of turbulence anisotropy state in the square duct ($Re = 3500$) are visualized for DNS, followed by baseline $k - \omega$, TBNN, and then TBRF, from left to right.

Analogously, random forest regressors was used to model jets in cross-flow, using high-fidelity data [43]. The algorithm, being robust and able to handle noise, was computationally cheap, and produced significantly improved anisotropy tensor predictions.

Both neural networks, and random forest have their disadvantages. It was reported that while the former is good in interpolations and extrapolations, they are susceptible to oscillatory behavior close to the wall boundaries and when used to predict turbulent eddy viscosity [76]. The latter, on the other hand, does not extrapolate well to unforeseen circumstances. Thus, both the techniques were combined to implement a novel neural network random forest model (NNRF) which was able to improve the pressure, skin-friction and velocity fields with respect to the baseline Spalart-Allmaras model and other ML models [76]. Flows around bump were considered as test cases.

Physics Informed Machine Learning
Recently, physics informed machine learning (PIML) approaches have also been introduced. They learn the functional form of discrepancy in the Reynolds stress tensor. The improved Reynolds stresses that are predicted, are then propagated to quantities of interest such as velocities and improve the RANS predictions for a flow [78]. Based on the discrepancy field calculated from the baseline RANS model and the high fidelity data, random forest regressors were used to create regression functions for the Reynolds stress discrepancies, with an input feature space constructed from 57 invariants. It was concluded that PIML showed excellent predictive performances in both the Reynolds stresses and the propagated velocities as well. The mentioned study is also similar to an earlier work that was able to obtain an improvement in the Reynolds stress tensor, by utilizing a set of 10 invariants [86]. The same research group also applied PIML's to obtain excellent predictions for the Reynolds stresses for fully developed turbulent flows at different Reynolds numbers in a square duct, and periodic hill flow cases, by leveraging DNS data [77].

Sparse Regression of Turbulent Stress Anisotropy
Another novel strategy to infer the stress models is Spare Regression of Turbulent Stress Anisotropy (SpaRTA), and can be treated as a deterministic symbolic regression tool [65]. A library of candidate functions is built, from which potential candidate models are derived. It was concluded that the predictions of the discovered models were a significant improvement over the baseline $k - \omega\ SST$ model. The algorithm selects an appropriate model using sparse-regression techniques. Elastic-net regularization, and least squares regression was used to promote sparsity, and stability in the CFD model. The augmented constitutive relation was defined as

$$b_{i,j} = -\frac{\nu_t}{k} s_{i,j} + b^{\triangle}_{i,j}, \tag{2.50}$$

where $b^{\triangle}_{i,j}$ denotes the data-drives correction for the anisotropy in RST. An additional correction, R, was implemented and provided local information to correct the transport equation for turbulent kinetic energy, and was modeled as

$$R = 2k b^{R}_{ij} \partial j U_i. \tag{2.51}$$

Velocity profiles were compared from the improved model to the high fidelity data. For the flow cases, it was observed that corrections for both $b^{\triangle}_{i,j}$, and k were required to generate consistently improved results. Three different cases of flow separation: periodic hill, converging-diverging channel and curved backward-facing step were used.

An example of periodic hill test case, $Re = 10595$ is shown below. The LES data in their works is referenced from [8].

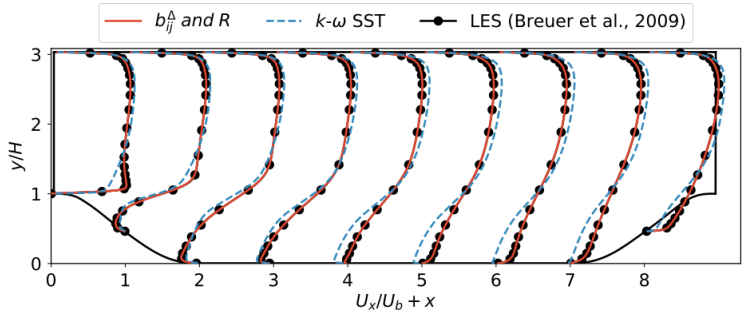


Figure 2.5: A visualization of the performance of SpaRTA
[65]

Evolutionary Algorithms: Gene Expression & Genetic Programming
In related work, evolutionary algorithms have gained momentum. They are based on the principle of *survival of the fittest*, and perform genetic operators such as mutation and crossovers on the individuals to converge to the optimum solution. By using symbolic expressions, explicit algebraic models for the Reynolds stresses are also directly attainable from high fidelity data. The optimization method has been used to find the optimal value of coefficients in the $k-\varepsilon$, $Spalart-Allmaras$ and $k-\omega\ SST$ models [23]. Testing was performed on different flows: impinging jet, backward facing step, and a nozzle, and significant improvement in the performance of the optimized models was observed. They reported that the method was numerically very robust, and did not require any $a\ priori$ knowledge of the solution space. However, the fitness evaluation of the individuals made the algorithm computationally very expensive, which was a major drawback to recalibrate the coefficients. An illustration of the performance over a test case is shown below.

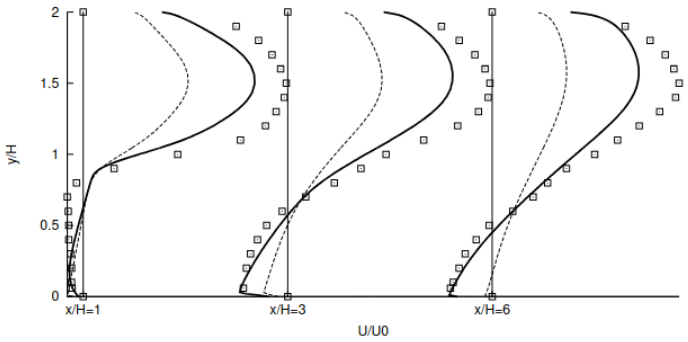


Figure 2.6: A visualization of the performance of the genetic algorithm framework
[23]

The velocity profiles are indicated at different downstream positions for a backward facing step, using standard $k-\omega$ model in dashed format. The bold line is the result after optimizing parameters, and the rectangles represent the reference data obtained from [64].

This methodology, combined with symbolic regression, has also applied to improve hybrid RANS/LES models on coarse meshes, by taking the benefit of available DNS data of a turbulent pipe flow [82]. The Gene Expression Framework was adopted due to its suitability with regression problems. In their work, a new damping function was built which essentially governs the contribution level of a RANS model. Considering tandem cylinders and periodic hill as test cases, the model with the new damping function was able to accurately capture the flow phenomena and be on par with DNS data. This further highlights the applications of Gene Expression Framework. An important advantage of such an algorithm is its interpretability. The final outputs in the above works were tangible algebraic equations which could easily be incorporated into CFD codes to propagate the effects of improved modeling.

Gene expression programming was also then directly used for tensor modeling and improving existing RANS models [81]. The anisotropy stress tensor was defined as

$$a_{ij} = a_{ij}(k, \varepsilon, S_{ij}, \Omega_{ij}). \tag{2.52}$$

A function of dissipation and turbulent kinetic energy, in addition to the rotation and strain rate tensors, the models were able to extract non-linear constitutive stress-strain relationships. They were described as 'implementation ready' as the generated individuals were in the form of an algebraic equation that could be easily inserted into the system. A linearly independent basis for a_{ij} was determined via the Cayley-Hamilton theorem. The fitness function of an individual over n data points, defined as

$$Fit(T_{ij}^{guess}) = \Sigma_{k=1}^{n} \frac{T_{ij,k} T_{ij,k}^{guess}}{T_{mn,k} T_{nm,k} T_{pq,k}^{guess} T_{qp,k}^{guess}}, \tag{2.53}$$

influenced the evolution of individuals to convergence. T_{ij}^{guess} is considered to be a combination of tensors and scalars, also referred to as plasmids. The algorithm was tested over backward facing step and periodic hill flows. An expression for the anisotropy tensor on of the test cases was

$$a_{ij}^{x} = 2\nu_t V_{ij}^{1}[0.01 - 0.0324\tau_1^2(3I_1 - I_2)] - 0.018\nu_t\tau_1[V_{ij}^{2} + V_{ij}^{3} + 2V_{ij}^{4}], \tag{2.54}$$

where V_{ij}, & I represent the tensor and scalar invariants respectively. Verification of different quantities such as τ_{ij}, C_f and velocity profiles was performed. A much improved agreement was found between the modified RANS model and the DNS data sets, barring an exception to a few profiles such as the performance in the favorable pressure gradient regions.

A posteriori analysis - Gene Expression Programming

The group from the previous study expanded their methodology and presented an $a\,posteriori$ approach that evaluated the fitness of individuals by running real time RANS calculations in an integrated way [93]. This gave the flexibility that the objective function could now be defined for any variable. By integrating RANS calculations, a robust, stable and a straightforward Reynolds stress closure could now be adopted. As the author of the work in discussion states, the motivation to include CFD calculations was to avoid the discrepancies that may exist between the high fidelity data and RANS environment, such as those in turbulent dissipation rate. It was explained that these discrepancies could exist due to time-averaged high fidelity data, which tend to produce good models for anisotropy stress tensor, but not so for other quantities. The high fidelity data was now only used to evaluate the fitness of converged RANS results. With the model trained on a high-pressure turbine cases, testing was done for distinct turbine nozzle flows. It was observed that the wake mixing profiles of turbomachines were much improved in the $a\,posteriori$ setting. However, some discrepancies in the high fidelity data and improved RANS models could be observed, which was attributed to the former's ability to resolve complex flow physics, while RANS may not be able to model it, such as low-frequency deterministic unsteadiness. Moreover, since the model equation was explicitly derived and available, it also helped in deducing the physics behind such wake profiles, and to understand the dominant relations for such profiles. Nonetheless, it was stated that the computational cost of such an integrated CFD-GEP framework limits its applicability and usage in the industry. This was because for each generation, CFD calculations have to be performed, and there may be hundreds or thousands of candidate models in a given generation.

A primary advantage of using such an algorithm is its open-box learning approach since it makes use of an evolutionary algorithm. In addition, its interpretability is also highly beneficial as the resulting expressions can easily be incorporated into CFD solvers. However, as it is computationally expensive to run the CFD calculations, and the GEP process in general, this poses a huge limitation to its widespread applicability.

A result from one of the test cases is presented below.

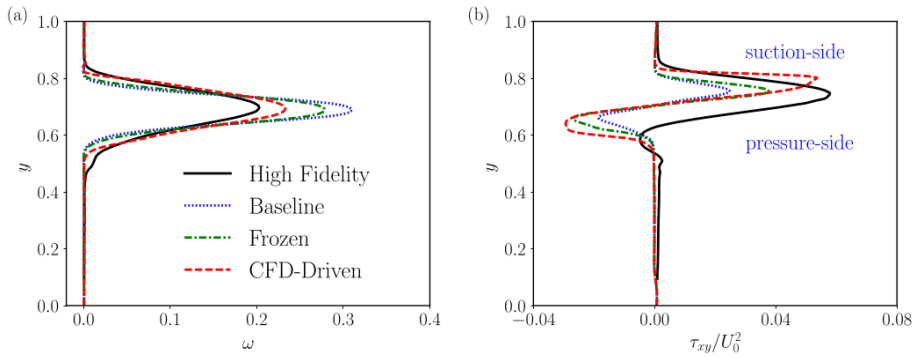


Figure 2.7: A visualization of the performance of the CFD-driven genetic algorithm framework [93]

Frozen data refers to using time-averaged high fidelity data. It can be observed that the proposed CFD-driven algorithm is much more accurate in this test setup. Further insights into the performance can be gained from [93].

Kolmogorov-Arnold Networks

Recently, Kolmogorov-Arnold Networks (KANs), inspired by the Kolmogorov-Arnold theorem. Rather than having learnable weights with fixed activation function, KANs have learnable activation functions themselves, and the weight parameters are replaced by univariate parametrized functions represented in the form of splines [45], defined as

$$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right). \tag{2.55}$$

A visual representation of the KAN network, along with the mathematical formula, is presented below.

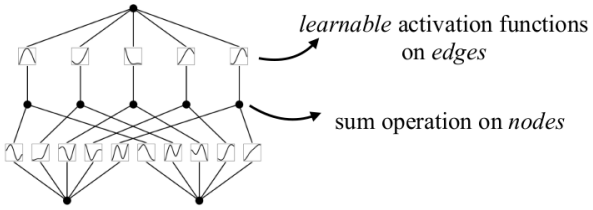


Figure 2.8: A visualization of the KANs network [45]

An upper hand that KANs possess is its scalability. It is illustrated, through several examples, that KANs require fewer parameters than much deeper MLPs. It was claimed that KANs are more effective at representing functions for regression, and solving Partial Differential Equations. While KANs showed a higher order for the reduction of testing root mean square errors with increasing parameters ($l \propto \mathcal{N}^{-\alpha}$) where l represents the test RMSE, $\mathcal{N}$ the number of parameters, and α as the scaling exponent, MLPs scaled much slowly and reached a plateau quickly. This is shown below.

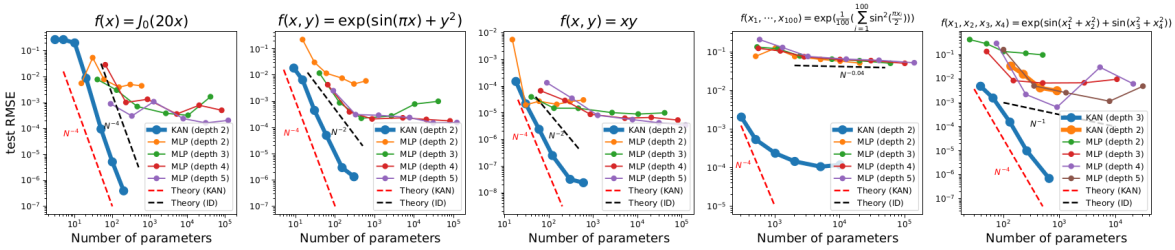


Figure 2.9: A visualization of scaling performance of KANs [45]

It can be seen that KANs can achieve a scaling $\propto N^{-4}$ while MLPs are quick to slow down and plateau. However, they also stated that for the same number of parameters, KANs can be much slower to train.

Furthermore, since KANs are based on splines, they exhibit the concept of fine graining. In MLPs, increasing the width and depth are the ways to improve the performance which can lead to slow scaling performance (shown earlier) and expensive. However, one can first train with very few parameters, and if required, can extend it to more parameters by making the spline grids finer, without the need to retraining a large model from scratch. This results in a more accurate spline to approximate the function space, and is much cheaper as well. The method is able to generate interpretable symbolic models, and also allow for effective pruning to reduce the functional space. Pruning is the process wherein neurons, edges, and inputs below a certain threshold are automatically removed from the network and it focuses only on the essential features and parameters and optimizes them to achieve the required results. This simplifies the KAN architecture and makes them more interactive and interpretable.

In light of the detailed literature, the current study will focus on the implementation of an open-box framework, i.e., Genetic Programming (GP). Henceforth, discussions will be made in reference to GP and literature will be discussed to make this algorithm more efficient in terms of computational cost.

In addition, given the advantage of KANs to generate a symbolic model, and initial proofs showing its high interpretability and improved scaling, they will also be researched upon.

2.8.2. Review and Analysis of Genetic Programming Methods and Applications

There have been several attempts to implement methodologies to accelerate the convergence of conventional genetic programming method, and their application has not been limited to CFD-related problems.

A popular method is the implementation of probabilistic techniques to guide the evolutionary process of genetic algorithms. In the works of Hauschild et al., the concept of Estimation of Distribution Algorithms (EDA) was introduced to explore the space of potential solutions by building probabilistic models [27]. They stated that the use of explicit probabilistic models provided an enhancement in the search efficiency over other types of population-based evolutionary algorithms. Based on the probabilistic models that are used, the EDA's can be categorized such as Extended Compact Genetic Algorithms, and Probabilistic Incremental Program Evolution among others. Building on this, a linear genetic programming structure was adopted, coupled with a probability model [68]. This captured the structure of the fit(ter) individuals and used it to sample new individuals. Each rule that governed the selection of a parameter (operator or feature) set at a node had an associated (independent) probability distribution. Two different approaches were tested - one where the individuals were sampled at every generation from the probabilistic model, and another in which the resampling was done only at "s" generations with conventional genetic programming used as reference. Their study indicated that resampling at every generation resulted in high success rates (determined by Mean Absolute Error and Median Absolute Deviation), followed by hybrid approaches, and then followed by conventional linear genetic programming. They tested their algorithms on symbolic regression problems, such as *nguyen1, nguyen2*, and *keijzer4* among others. The problems and their datasets can be referenced from [68]. Moreover, beam designing was also achieved by implementing probabilistic-based natural selections in a genetic algorithm [58]. Individuals were assigned probabilities based on fitness scores which resulted in good

combinations likely to propagate, all the while maintaining diversity. Their study showed that they were able to achieve lower mean errors, and narrower confidence intervals with the modified approach, thus producing more accurate and reliable results.

Elsewhere, mimetic algorithms have also been popular. In the field of Computer Science, combining evolutionary algorithms with local exploitation techniques are also referred to as mimetic algorithms [33]. For example, a tree based structure was adopted and instead of biasing the creation of individuals using probabilities, genetic evolution operators such as crossover were modified, via back propagation [47]. Treating a GP individuals similar to a neural network, the partial derivative of each node was calculated and this denoted the change a node/sub-tree can make towards the context of the tree. Ranking nodes based on the values, they predicted the impact of sub-trees and concluded that if a small change in the tree's behavior was to be made, a low impact site can be selected for crossover, and vice-versa. This prevented the random crossover wherein information regarding the tree structure was lost, and was accomplished by incorporating derivatives. On several symbolic regression problems, they were able to prove that their proposed method of redefining the crossover operation gave lower testing and training errors than other methods. However, despite achieving superior performance, incorporating back propagation meant evaluating derivatives and doing significantly higher tree (function) evaluations.

In related work, geometric semantic GP (GSGP) was adopted with gradient-based optimizers to to achieve the same [59]. Unlike tree based GP where the highlight is on the structure, GSGP focuses on the semantic aspect, i.e., how it will behave for given inputs. The individuals, programmed in the form of list, were subjected to Adam and Stochastic Gradient Descent Algorithm (SGD). Adam was able to outperform SGD and improved fitness results were attained after a few evaluations when compared to the converged results of conventional GP. However, instabilities were present and the performance was dependent on the selection of hyper-parameters. Although the algorithm was mainly tested on regression problems (up to 600 variables), the authors' claimed that it was difficult to generalize for complex datasets. The optimization was performed with respect to semantic operators such as α, that is defined below.

- Geometric Semantic Crossover :

$$T = (T_1 \cdot \alpha) + ((1 - \alpha) \cdot T_2) \tag{2.56}$$

where $0 \leq \alpha \leq 1$ and T_1, T_2 are the parent trees.

Another notable work is related to the tuning of random constants (leaf coefficients) of a tree individual by gradient descent [73]. The results of local learning (gradient optimization) were coded back to the genotype (individual) referred to as Lamarckian learning in their study. Applying algorithmic differentiation to compute gradients with arithmetic operators (primitive set), they performed regression analysis. They noted that crossover usually generated individuals with worse fitness levels than their parents. However, once the coefficients were tuned, the resulting individuals from genetic operation were appropriate. They concluded that local learning rates could create a bias mote towards more readily adaptable and fitter individuals. Hence, accelerating convergence to a given point in function space. In addition, a substantial improvement in final fitness and speed was also observed. However, similar to the previous study, the cost was increased due to the additional function evaluation (cost of differentiation). The numerical coefficients were updated in their study, by the following rule.

$$c_k \rightarrow c_k - \alpha \frac{\partial MSE(\mathbf{c})}{\partial c_k} \tag{2.57}$$

Similar work is related to parametrization of GP trees via gradient descent to improve the capabilities of symbolic regression [60]. This was unique in the sense that they included learnable parameters (multiplicative factors) for the operands. Hence, instead of having learnable terminal coefficients, they now had learnable weights for the different operations in an individual. They developed two distinct approaches - one where each operator would have the same weight for each instance, while the other wherein a different weight is assigned to each individual operator. Moreover, they also studied the effect of applying gradient based optimization after each generation, thus impacting the search at subsequent generations, or only applying at the last generation, after the evolutionary algorithm ends. Using Adam

with a learning rate of 0.01, it was concluded that GP coupled with gradient optimization was significantly better than conventional GP. They were able to achieve faster convergence of the evolutionary search process while testing on complex real-life regression datasets with greater than 5 features. A visual representation of their individual structure is shown below.

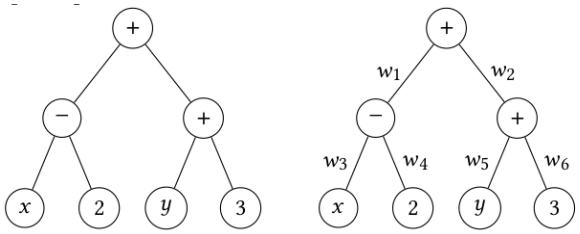


Figure 2.10: Parametrized GP with learnable weights
[60]

While the individual can be computed as $(x-2)+(y+3)$ from the right tree, it would now be computed as $w_1 \cdot (w_3 \cdot x - w_4 \cdot 2) + w_2 \cdot (w_5 \cdot y + w_6 \cdot 3)$

Another novel approach to incorporate gradient based optimization into genetic programs was making the use of differentiable symbolic trees to relax the discrete structure (function space) into a continuous form [91]. They devised two different matrices to do so. A node matrix, which consisted of probabilities of a node taking certain operators or variables (features). They were learnable and softmax activation was applied to derive the matrix. The adjacency matrix represented the strength of connection between nodes. It was used in sampling step and essentially, it dictated the contribution of multiple child nodes to a parent node. A common issue with mimetic algorithms is that the local search method can force the solution to converge to a local optima and avoid large explorations. The authors' addressed this by sampling from the optimized differentiable trees. They tested their algorithm on real-world benchmarks (100+ features) and synthetic (regression) cases. They were able to observe that their proposed method outperformed all GP-based and neural network based symbolic regression methods. They concluded that they were able to achieve superior results because differentiable symbolic trees allowed for a continuous search in localized space, while diversification helped escape local optima.

The literature above gives us an overview of recent developments in genetic programming, to converge to superior results quickly. Given strong evidence that this results in improved and consistent performances when tested, the present thesis will focus on developing a mimetic algorithm to achieve superior results.

2.8.3. Review of Applications of KANs

Explained in Subsubsection 2.8.1.7, KANs have been applied for different problems. An example is the application of KANs for predictive modeling of flexible electro hydrodynamic pumps [57]. Leveraging the ability of having learnable activation functions, they were able to learn the nonlinear relationships with greater flexibility. Using it for pressure and mass flow rate predictions, they utilized two hidden layers and cubic splines with 5 input features and 2 output features. In addition, they made use of the LBFGS optimizer, and pruning to streamline to architecture. They were able to extract the symbolic formula which provided them with further interpretability and understand how different quantities behave and their relations. Furthermore, on comparison with Random Forest and MLP's, they were able to obtain better predictive accuracies. Another similar work follows the application of KANs to time series forecasting [89]. They proposed two variants, namely the T-KAN and MT-KAN. The former was used to detect concept drift and essentially capture the non linear relationships between predictions and previous time steps through symbolic regression. The latter, on the other hand, was used to improve predictive performance by effectively uncovering and leveraging the complex relationships (temporal dependencies) among variables in a multivariate time series. From their results, they were able to conclude that T-KAN, only with a single hidden layer with 5 neurons, achieved efficiency and robustness comparable to other models making use of additional hidden layers (and neurons). Moreover, T-KAN only made use of 193 parameters, as compared to Long Short Term Memory Network (along with Multi

Layer Perceptrons and Recurrent Neural Networks) model that made use of roughly 11,671 parameters to achieve the same level of accuracy.

KANs have also been utilized in conjunction with evolutionary algorithms [25]. In their study, they used KANs as a surrogate model to evaluate the fitness of individuals that were generated by a genetic program. The fitness was approximated based on previously evaluated data and predicted the performance of the offspring. It aided in making informed decisions about which solutions to carry forward. The surrogate was constructed using a set of already-evaluated solutions to learn the underlying patterns. In addition, they were also train on the top $x\%$ of the individuals to assist in choosing the best solution from the offspring. They showed that such a combination yielded quick convergence when compared to MLPs, Bayesian Optimization, and other methods.

Elsewhere, KANs have also been coupled with Graph Neural Networks (GNN) [15]. They propose GKAN, combining both the techniques and showed that it outperformed state-of-the-art GNN models in node classification, link prediction, and graph classification tasks. In addition to this, they state that GKANs provide clear insights into the model's decision-making process, enhancing interpretability.

The works mentioned above demonstrate the capability of KANs and their superior performance is clearly visible on multiple datasets. Hence, this thesis will involve implementing KANs on CFD datasets and determine if it can efficiently learn the symbolic form of the corrective terms.

2.9. Research Questions

With the detailed literature review done above that gives us a thorough background on CFD models, their limits, and the application of machine learning techniques in improving them, the research questions are summarized below.

The focus of the present thesis is to develop (and utilize existing) algorithm(s) to construct interpretable models for correcting different quantities such as RST and the Turbulent Production. It needs to be determine whether they can achieve acceptable correlations while ensuring reduced calls to the objective function (CFD solver) during the optimization process. The methods selected for this research are Genetic Programming, and Kolmogorov Arnold Networks.

Once accomplished, it will be investigated if the best-performing method can be used in an $a\ posteriori$ **framework to improve the turbulence modeling which involves differentiating the CFD code during the optimization process.**

It can be sub-divided into the following objectives and questions that reflect the literature.

- Can genetic programs (algorithms) that are incorporated with information from derivatives (gradients, and hessians) achieve quicker convergence, with reduced function evaluations, and acceptable correlations?
- How and which parameters (population size, evolution probabilities, number of generations) would need to be optimized, in addition to the frequency and optimizer used?
- Does the local search overshadow the exploratory ability of genetic algorithms to observe large function spaces, causing the solution to converge to a local optima?
- Can KANs be used on CFD datasets and converge to superior results with reduced function evaluations? Can the learnable activation functions capture the complex (non linear) relation between different quantities?
- Can the superior method maintain its performance when applied in an $a\ posteriori$ setting and if a generalizable model be obtained that is applicable for different flow cases without encountering overfitting and stability issues? Is the symbolic formula physically meaningful?

3

Methodology

This chapter explains the working of the selected algorithms - Genetic Programming and KANs. It describes the methodology adopted to generate corrective fields for turbulent production and RST, and how it is ensured that the flow physics do not change with the reference frame (invariance). The chapter then presents the development of the $a\ posteriori$ framework using the algorithm found to be most effective, with details on the implemented turbulence model.

3.1. Working of Genetic Programming

In the work of Mr. Koza, genetic programming initially enabled computers to solve problems without being programmed [38]. This was further developed by John Holland's genetic algorithm that generated programs to solve various optimization problems, also laying the foundation for the present work [31].

Genetic Programs are a population based optimization algorithm. Population refers to a set of individuals, and these individuals, over a given number of iterations or generations, are evolved through evolutionary operators to converge to the optimum solution. An individual will be referred to as a chromosome, while the individual terms (symbols) that make up an individual will be called as genes in the framework of genetic programming, for the present study.

A general representation of chromosomes, genes and a population is shown below.

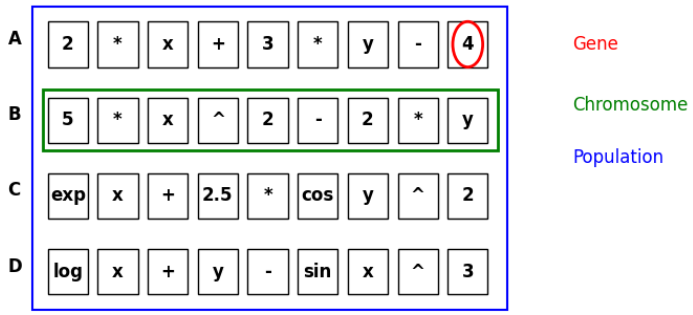


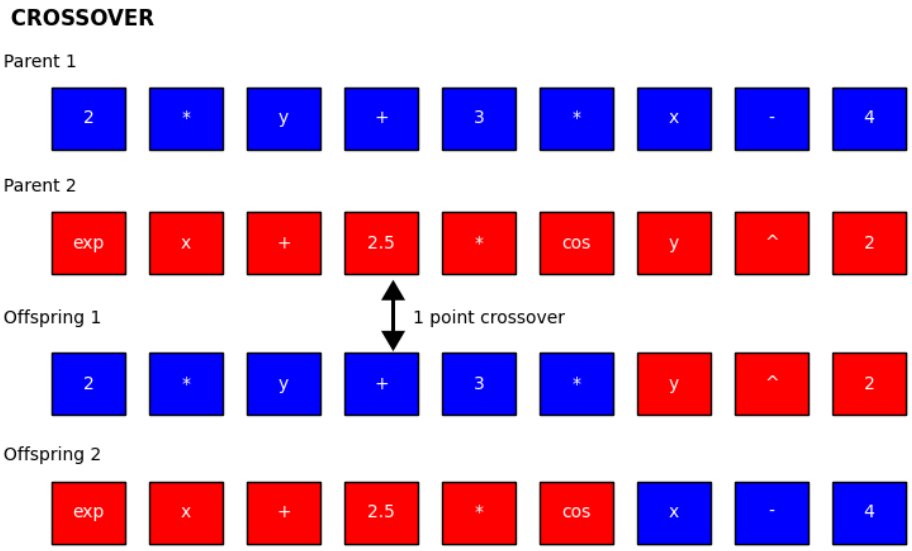
Figure 3.1: Genetic program representation

Each individual is assigned a fitness value that corresponds to how well it performs in optimizing the objective function [1]. Based on natural selection, chromosomes with higher fitness values have a higher chance to be propagated to the next generation and influence the evolutionary process. Such algorithms need to evolve the individuals such that there is diversity in the population to allow exploration of large functional spaces, thus preventing premature convergence to a local optima, and is achieved via crossover and mutation. Crossover refers to the selection of two individuals and swapping them at

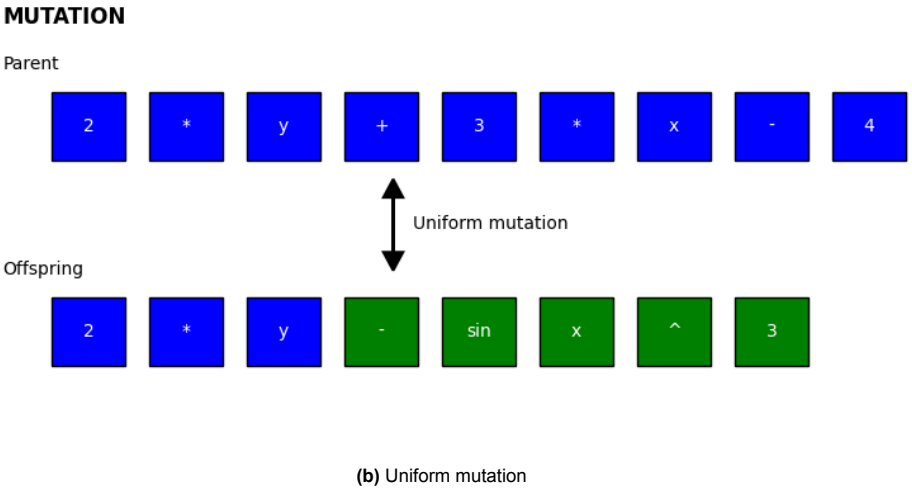
a point. This point is chosen at random in a way such that the syntactical structure of the chromosome remains intact. Mutation is another tool which selects a random node in an individual and replaces it with a randomly generated expression. While the aim of such operators is to also modify the individuals in the direction of achieving a global optima, however, since they are random, it is often observed that useful combinations of terms may be broken, and an individual's fitness may worsen due to such evolutionary tools.

The ability of genetic programs to efficiently explore the function space and optimize the objective is influenced by population size, crossover and mutation probabilities, and total number of generations among others.

Visualization of the genetic operators is shown below.



(a) One point crossover



(b) Uniform mutation

Figure 3.2: Genetic operators: (a) One point crossover, (b) Uniform mutation

Figure 3.2a shows us the process of one point crossover wherein two parents are selected and a random point for interchanging their genes is chosen such that the overall syntax of the individual remains valid. Figure 3.2b shows the process of uniform mutation, implemented by the **DEAP** package in Python.

The main steps in a genetic algorithm are mentioned below.

- Initialize the population using parameters such as probabilities of evolutionary operators, number of generations and the number of individuals. The individuals have to be created with a certain syntax (structure). There is no standard for the parameter values.
- Evaluate the individuals of the population and assign them numerical fitness scores based on a chosen metric for evaluation such as root mean square error, and R_2 scores. The fitness measure gives an indication to the algorithm on what needs to be performed further to achieve the optimal objective function value.
- Selecting individuals for the next generation. In addition to selecting the best individual (elitism), individuals may be subjected to selection procedures such as tournament, roulette wheel, stochastic methods, and so on. This is done to selectively pick individuals to either directly clone to the next generation, or further undergo evolutionary operations. For further details on the selection methods, the reader can refer to [1]
- Applying evolutionary operations to maintain diversity and evolve the individuals in the direction of achieving an optimal objective value. Crossover and mutation are the two main techniques, which have been explained earlier.
- Once the solution attains convergence, the algorithm may be terminated. This can be done in three distinct ways. The number of generations can be specified beforehand, or the algorithm can be terminated once tolerance levels in the fitness scores or the pre-set optimal requirement has been met. The third way is to terminate after the Pareto-optimal solution (multi-point objectives), with no better results further achievable [56]. Both, the number of generations in total, and the desired tolerance level will be specified in our codes. The required data is recorded as well to then evaluate the performance of the algorithm.

A pseudo algorithm of the genetic programming concept used for this thesis is also shown below, for better understanding.

Algorithm 1: Genetic Algorithm Framework

```
1  Procedure Genetic_Algorithm;
   Input: Define an initial parent population based on parameters, described earlier
   Result: Optimal (elite) individual after optimization

2  for g = 1 to total generations do
3      fitness = evaluate fitness of individuals in parent population;
4      elite = optimal individual in parent population;
5      offspring = TournamentSelection(parent population);
6      foreach i in offspring do
7          if random.random() ≤ mutation probability then
8              1 = SelectParent(parent population);
9              1 = Mutate(1);
10             Insert(1, offspring);
11     foreach i in offspring do
12         if random.random() ≤ crossover probability then
13             1, 2 = SelectParent(parent population);
14             1, 2 = Crossover(1, 2);
15             Insert(1, 2, offspring);
16     parent population = elite + evolved offspring;
17     if fitness of elite satisfies tolerance then
18         Break;
19 return elite
```

This algorithm is followed for the implementation of conventional genetic programming, without any fur-

ther optimization methods. The elite individual is returned and used to see how the algorithm performs. Note that tournament selection is used as the mechanism to pick out individuals for the offspring as it is one of the most significant selection methods in GA's, due to being highly effective and easily implementable [1]. In this approach, **X** individuals are selected from a population of size **Y**. These selected individuals are subjected to a tournament and the one with optimal fitness scores is added to the offspring. This method gives an opportunity for all individuals, good and bad, to be selected and helps in retaining diversity thus not hindering the global search capabilities of genetic algorithms, provided the tournament size is not very large.

A pseudo algorithm of the tournament selection mechanism can be referenced from Appendix Section A.1.

From the reviewed literature in Subsection 2.8.1, individuals have been coded in the form of a tree when genetic program has been applied on CFD datasets [81] [93]. This is known as a tree-based genetic programming (TGP). In such a structure, there is a tree (root) node that consists of functions such as $+, -, /, \times$, while the terminal (leaf) node consists of a variable. They offer the advantage of high interpretability when the final model is obtained. In addition, TGP has found its use in different domains such as quantum computing, solving complex optimization problems, search problems, and pattern recognition among others [6] [13]. An individual of TGP form is shown below.

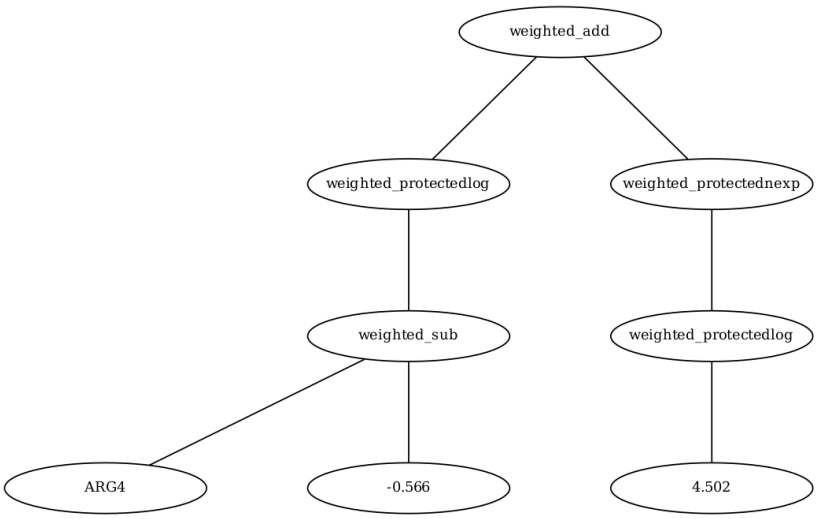


Figure 3.3: An individual in tree format

An individual can be read via a bottom-up approach, and the above individual in Figure 3.3 is equivalent to

$$TGP_{individual} = \text{log}(ARG4 - (-0.566)) + \text{exp}(\text{log}(4.502)) \tag{3.1}$$

Note that the operators have different names as compared to their conventional operator names such as add, it is prefixed by ***weighted_***. In addition, some operators such as $\log, \exp,$ div have $protected$ in the names as well. This is to indicate that the basic operation of these operators has been modified to avoid instabilities or overflowing values, such as $\log(x)$ where $x < 1$, or $\exp$ of a very large number. This convention of naming the operators via this method is followed throughout the current study for the author's convenience. Moreover, nodes with an $ARG\_$ are used to represent the input features.

3.2. Gradient Optimized Genetic Programming

The gradient based genetic program (mimetic algorithm) developed for this thesis couples genetic program (algorithm 1) with a gradient optimization technique. From the population, a biased set of individuals is selected via the tournament selection method. By implementing tournament selection, diversity is ensured among the individuals as it gives a fair chance to the bad individuals to be selected as well. Continuing, another subgroup is created from this biased distribution set via random sampling.

These randomly samples individuals are then selected for optimization via the gradient method of an appropriate choice. The reason to create such sub-sets of individuals within the population is to prevent applying gradient optimization to all the individuals in a generation. If done so, only optimized individuals would be propagated to the next generation, leaving minimal opportunity for variability and diversity. This could then result in the algorithm achieving premature convergence without exploring enough functional spaces to reach the global optima. Moreover, the current thesis is focused on reducing the function evaluations. Optimizing each individual will unnecessarily increase the function evaluations. Hence, to avoid large number of function calls, and ensure good results, the above procedure is followed. The selected individuals then undergo optimization to modify the terminal constants and the operator weights as well.

To optimize the individuals by a gradient method, using a TGP is not suitable as it is not trivial to modify the nodes. This is because the **DEAP** library in Python represents individuals in the form of objects with a unique identifier tag, such as of the form < 07xxxxx$>$. Although individual nodes may be accessible and optimized, inserting the modified constants (and functions) back into the tree may break this identifier tag, resulting in an error when an individual is evaluated or subjected to evolution. Hence, a linear genetic programming (LGP) is employed. It is a variant of GP in which the individual is expressed in the form of a linear list and the nodes are now stacked (appended) one over the other as shown in Figure 3.2. The decision to do so was the straightforward implementation of the nodes of the individuals with scipy or PyTorch based gradient methods to optimize the nodes (operators and terminals) when an LGP based individual was used. However, given that a tree based individual is much easier to interpret and that LGP has a higher compiler overhead than the TGP, the linear list was then converted back to a tree after optimization was performed [1]. In addition to this, implementation of evolutionary tools on tree based individuals can be achieved directly from the **DEAP** packages. Hence, the gradient based genetic programming algorithms made use of two different individual representations - a tree based structure at all times, and a linear based structure when the individuals were subjected to optimization.

An individual, in its tree and linear list format, is visualized below for further clarity on these concepts.

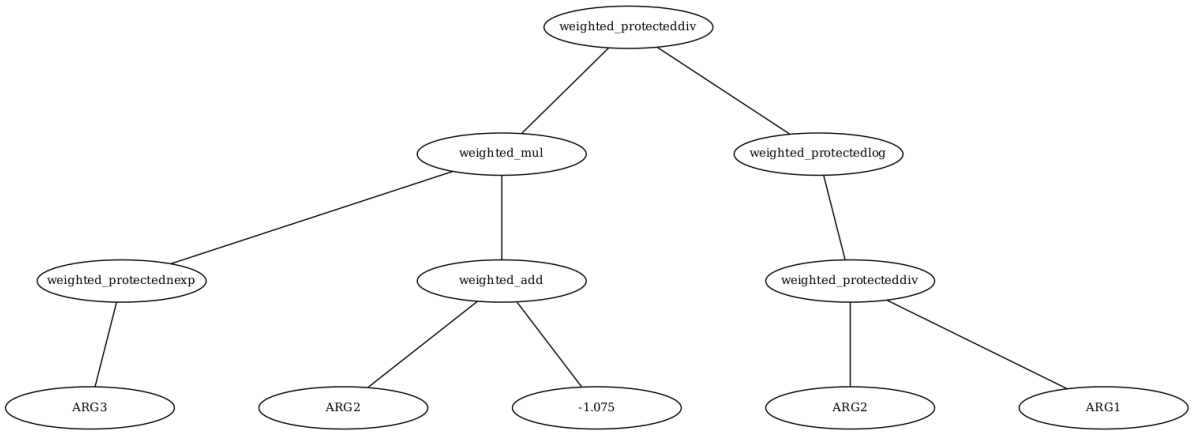


Figure 3.4: TGP individual

Linear Sequence

`ARG1`, `ARG2`, `weighted_protecteddiv`, `weighted_protectedlog`, `-1.075`,
`ARG2`, `weighted_add`, `ARG3`, `weighted_protectednexp`, `weighted_mul`, `weighted_protecteddiv`.

Figure 3.5: LGP individual

The following individual is equivalent to

$$\frac{\exp(ARG3) \times (ARG2 + (-1.075))}{\log(\frac{ARG2}{ARG1})} \tag{3.2}$$

There are two components of the individual to optimize - the operators and the terminal constants. After converting the tree based individual to a linear list based individual, it is then checked for how many constants and operators are present in the individual. These terms are then assigned weights that would be optimized by a gradient optimizer. The term at that node essentially becomes equivalent to the term multiplied by the optimized weight. For instance, a terminal constant 2.3, which is a part of a larger tree or an individual, has an initial weight of 1.0. With respect to a conventional genetic algorithm, the 2.3 is taken as it is, and at the beginning of the optimization process, the 2.3 is multiplied by 1.0, inherently reducing to a conventional algorithm. However, once these weights are optimized, it may change to -0.67. Thus, the resulting constant becomes equivalent to 2.3 $\times$ -0.67 which equals -1.541. This new constant is then inserted into the primitive set to avoid issues when converting the linear list back to a tree representation, and subjecting it to evolutionary operators. In a way, these weights represent how much a particular node (terminal or a function) is contributing to the prediction of a target variable. Similarly, the function or the operator set is also optimized. For example, the addition operator is acting on two variables, x_1, x_2 and the resultant is $R = x_1 + x_2$. This result is stored under the addition function set. Initially assigning a weight of 1, it is equivalent to $W \times R$, where W represents the weight. This weight is then optimized by the same gradient method. The optimized weight, for the sake of understanding, becomes 1.2. Thus, the new resultant of the addition operator will be equivalent to $1.2 \times R$, or $1.2 \times (x_1 + x_2)$. This is defined as a new function in the primitive set so there are no errors with the attributes of the individual and no connections are broken such as the identifier tags. This is procedure of optimizing the terminal and operator nodes of an individual. The modified individual (with a higher fitness value) is then inserted back to the population to guide the evolution process.

A graphical representation of the optimization process is shown below.

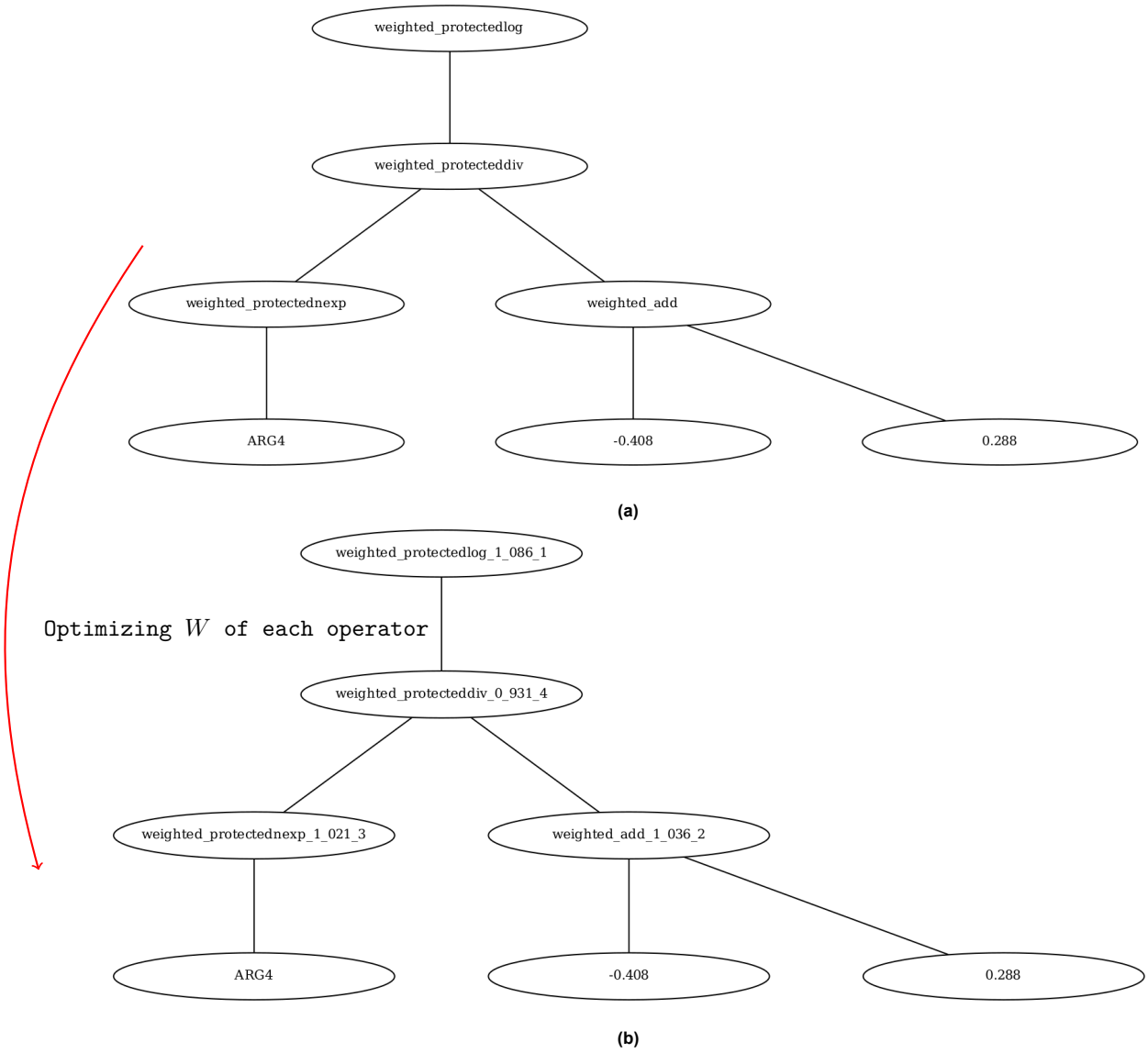


Figure 3.6: Comparison of the original and optimized individuals during the optimization process.

Figure 3.6b shows how the weights of each operator are optimized, and gives a clear illustration of the process. Initially set as 1 when an individual is created, the learnable weights (associated with the operators) are then optimized by the gradient method. For instance, observing the right sub-tree, it was computed initially as $R = 1 \times (-0.408 + 0.288)$. However, after optimization, it will now be evaluated as $R = W \times (-0.408 + 0.288)$, where $W = 1.036$. The number following the weight is a unique identifier, given to each optimizer operator when added to the pset.

During the optimization process, the objective is defined as the root mean square error. Thus, at each iteration of the optimizer when the weights are changed or optimized, the linear sequence of the individual is evaluated. The resulting loss function then further guides the optimization process and serves as the cost (objective) function which is to be minimized via these weights.

As the tree structure is converted to a linear list, the identifier tag is broken and hence the evaluation of the individuals cannot occur via the **DEAP** package as it does not have the fitness attribute. In order to tackle this, an explicit function is written which evaluates the resulting linear sequence of the individual in a proper manner. However, once the optimization is completed, the weights are added in the primitive set which restores the connection.

Once the optimization of all the individuals are completed, they are converted back to trees. The rest of the algorithm follows as algorithm 1, wherein an offspring is created via tournament selection, which is then subjected to crossover and mutation operations. Thus in this manner, the individuals are optimized and are inserted back into the population to guide the evolution process. This is how advantage of both local search (gradient) methods, and global search (genetic programming) is leveraged to achieve accelerated convergence (reduced function calls) to the global optima by incorporating the extra information and reflecting it back onto the population.

A pseudo algorithm explaining the working of the gradient based genetic program is presented below.

Algorithm 2: Genetic Algorithm with gradient optimization Framework

```
1  Procedure Genetic_Algorithm;
   Input: Define an initial parent population based on parameters, described earlier
   Result: Optimal (elite) individual after optimization
2  for g = 1 to total generations do
3      fitness = evaluate fitness of individuals in parent population;
4      elite = optimal individual in parent population;
5      biased set = TournamentSelection(parent population);
6      optimization set = RandomSampling(biased set);
7      foreach i in optimization set do
8          original individual = SelectIndividual(optimization set);
9          optimized individual = GradientOptimize(individual 1);
10         Delete(original individual from parent population);
11         Insert(optimized individual into parent population);
12     statistics = RecordStatistics(parent population);
13     gradient optimized elite = optimal individual in parent population;
14     offspring = TournamentSelection(parent population);
15     foreach i in offspring do
16         if random.random() ≤ mutation probability then
17             1 = SelectParent(parent population);
18             1 = Mutate();
19             Insert(1 into offspring);
20     foreach i in offspring do
21         if random.random() ≤ crossover probability then
22             1, 2 = SelectParent(parent population);
23             1, 2 = Crossover(1, 2);
24             Insert(1, 2, into offspring);
25     parent population = elite + gradient optimized elite + evolved offspring;
26     if fitness of gradient optimized elite or elite satisfies tolerance then
27         Break;
28 return gradient optimized elite or elite
```

In this case, 2 elites are selected. One before optimization and one after optimization. This is taken as a precaution in the case that the optimization fails to improve the fitness of the elite, the default elite will still be present to guide the evolution process.

Pseudo algorithms for the different steps in the gradient based optimization process are shown below.

Algorithm 3: Initialization of weights and individual pre-evaluation

Input: Individual represented as a tree structure
Output: Optimized individual with weighted operators and terminals

```
Procedure Optimize_Terminals_and_Operators(individual);
    Convert TGP Individual into a linear representation (LGP Individual);
    check = 0;
    Initialize empty lists: weights;
    foreach node element in LGP Individual do
        if element is a constant and not an operator or feature then
            Assign a learnable weight (requires_grad=True);
            Store its position;
            check = 1;
        else if element is an operator then
            Assign a learnable weight (requires_grad=True);
            Store its position;
            check = 1;
        else
            Assign a fixed weight (requires_grad=False);
        Store weights and positions for the sequence;
    if check = 0 then
        return LGP Individual // No optimization needed
    Perform optimization using gradient-based approach:
    foreach position in optimized positions do
        if node element is a weighted operator then
            Compute weighted operator name and update primitive set;
        else if node element is a constant then
            Compute new constant value and update primitive set;
    return LGP Individual with optimized weights
```

Algorithm 4: Gradient-Based Optimization for Terminal and Operator Weights

Input: Linear sequence(s) $\mathcal{L}$, inputs X_{nondim}, X_{dim}, initial weights $\mathcal{W}$ indices to optimize $\mathcal{I}$
Output: Optimized weights and final loss

Procedure Optimization($\mathcal{L}, X_{nondim}, X_{dim}, \mathcal{W}, \mathcal{I}$);
  Initialize $\mathcal{W}_{overall}$ with ones for each node element in $\mathcal{L}$;
  foreach *index* i *in positions* **do**
    Assign a random initial learnable weight in the range [0.9, 1.1] to $\mathcal{W}_{learnable}[i]$;
  Define Objective_Function($\mathcal{W}_{learnable}$);
    Copy $\mathcal{W}_{overall}$ to $\mathcal{W}_{updated}$;
    Assign $\mathcal{W}_{learnable}$ to corresponding *positions* in $\mathcal{W}_{updated}$;
    Compute predicted outputs Y_{pred} using
     Evaluate_Linear_Sequence($\mathcal{L}, X_{nondim}, X_{dim}, \mathcal{W}_{updated}$);
    Compute loss as the mean squared error: $\mathcal{L}oss = \frac{1}{N}\sum(Y_{train} - Y_{pred})^2$;
    return $\mathcal{L}oss$;
  Flatten $\mathcal{W}_{overall}$ for optimization;
  Solve optimization using preferred gradient method with Objective_Function ;
  foreach *optimized index* k **do**
    Update $\mathcal{W}_{overall}$ with optimized weights;
  Store function evaluation counts $nfev, njev$;
  return *Final loss and optimized* $\mathcal{W}_{overall}$;

3.2.1. Gradient based optimization methods

Given the wide variety of optimizers available, 4 were chosen to be implemented. Newton's method is one of the most popular optimization algorithms that is based on using second order information (Hessians). It constructs a local quadratic approximation, based on the 2^{nd} order Taylor series, and minimizes that. It can find the global minima of a quadratic function immediately. However, if the Hessian is not positive definite, the solution diverges, in addition to the evaluation of Hessian being impractical as well, for a large dataset. Thus, it is not robust. Given that this thesis is related to CFD simulations, achieving converged stable solutions quickly is a primary focus to avoid the CFD solver blowing up, therefore, implementing a model that is not robust will not be helpful, and hence, not used. Alternatives to Newtons methods are the *trust region*, and *quasi-Newton methods* (1^{st} order) that rely on approximating the step size based on a trust region, or approximating the Hessian by utilizing only first-order derivative information to do so. The former will be touched upon first, and then the latter will be discussed.

Trust Region Method

Newton's method, which is computationally cheapest on the Rosenbrock function in terms of the number of function calls, can further be made robust by applying the trust region approach [55]. The trust region methodology makes use of the concept that the model approximation around the current point (iterate), of the objective function, is bounded by a trust region wherein it is optimized. The trust region essentially limits the search space to ensure that the local quadratic approximation also remains locally valid and remains a good model. It is based on approximation quality, and is updated depending on how well the model is able to estimate the objective function. At each iteration, a subproblem is solved to find the optimum step size to minimize the model approximation. The methodology is formulated as

$$\min_{\Delta x} \left(f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T H(x) \Delta x \right).$$

$$\text{subject to: } \|\Delta x\| \leq \varepsilon$$

where ε defines the trust region.

The *scipy.optimize* package offers the implementation of the trust region reflective, *trf*, method to optimize a nonlinear problem using the first-order optimality condition. It is based on a second order

approximation by utilizing the Gauss Newton method via the Jacobian. The implementation of the method can be found in [7].

L-BFGS optimization
Moving to quasi-Newton methods, a first order Taylor approximation of the gradient is used to approximate the Hessian. The BFGS (Broyden, Fletcher, Goldfarb, Shannon) method is a Quasi-Newton method, and is more robust and able to tackle the issue of computing Hessians at each step. However, for large problems that involve multiple features, it stores a dense matrix that approximates the Hessian. This makes it memory intensive and unsuitable for multi-dimensional problems. Hence, the Limited-memory BFGS (L-BFGS) is a strong candidate to be used along with GP, as it uses a limited amount of computer memory [44].

It should be noted that the quasi-Newton optimizers make use of the Wolfe conditions to perform the unconstrained minimization (loss function). It is used to determine a sufficient step size, α during the line search to proceed to the global solution in an appropriate manner [52]. This is mainly because the Hessian is not used directly, rather, an approximation of the inverse Hessian is, thus, the information for a suitable step size is lost. Hence, line search is implemented, via iteratively testing candidate step sizes, which further involves function and gradient calls to ensure the step is appropriate by checking the curvature condition. The *scipy.optimize* library in Python provides the implementation of the LBFGS method and its implementation can be referenced from [10].

Levenberg-Marquardt Method
The problems in Newton's methods can also be tackled by the *Levenberg-Marquardt* method. It redefines the Hessian into a modified version (approximation) given by

$$\tilde{H} = H + \beta I. \tag{3.3}$$

This is done so that $\tilde{H}$ remains positive definite at all times. The Hessian approximation, for function curvature insights, is based on the information from Jacobians. *scipy.optimize* provides the implementation of the Levenberg-Marquardt method for nonlinear problems. It applies the method formulated as a trust-region type algorithm to update the parameter β, and is very robust. If interested, the reader can find the implementation in [50].

Adam: Adaptive Moment Estimation
Another common gradient based optimizer that has been used with GP is *Adam* [59]. It is a first-order based method, efficient, and easily implementable. In addition to the inputs, *Adam* also requires the specification of the learning and exponential decay rates. However, the learning rate for each parameter is adjusted based on the gradients by the optimizer. The reader can find further information on the optimizer from [37].

Based on preliminary results from symbolic benchmark problems, procedure explained in Section 3.4, an optimizer and its settings were then finalized.

This completes the methodology on the developed mimetic algorithm as a part of this thesis, which is able to successfully implement the global search method - genetic programming, and leverage the advantages of a local search method to exploit functional spaces - gradient based optimization.

3.3. Working of Kolmogorov-Arnold Networks

This section will focus on the parameters of KANs. To implement the method, the source codes were used from the developers, available at `https://github.com/KindXiaoming/pykan`. Building up on (Equation 2.55), a general output of a KAN network, with *L* layers, can be written as

$$\text{KAN}(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_1 \circ \Phi_0)x. \tag{3.4}$$

wherein Φ_L, Φ_{L-1} represent the linear and nonlinear transformations (activation functions). In the implementation, a residual activation function is added such that the activation function $\phi(x)$ is equivalent to the sum of the spline function, and the residual function. A *silu* function is taken as the residual basis function, $b(x)$. The activation can thus be written as

$$\phi(x) = w_b b(x) + w_s spline(x), \tag{3.5}$$

In the context of KANs, it is the $spline(x)$ that is learnable.

A *silu* function is smooth everywhere and allows for expressive representations, and improves the overall stability of the model. In addition, for a large input, it behaves like an identity function (linearly) and can introduce non-nonlinearities for small or negative inputs as well. It acts similar to a residual connection, hence, facilitates training of deeper networks, and prevents performance degradation. Moreover, it helps in preserving gradient flow, which can be affected due to the vanishing gradient problem, further affecting the learning process [4].

The splines can be represented as a linear combination of B-splines which are further a combination of Bezier curves. In space domain, these curves can be represented via Bernstein basis polynomials, given by

$$B(x) = \sum_{i=0}^{n} \binom{n}{i} (1-x)^{n-i} x^i P_i = \sum_{i=0}^{n} b_{i,n}(x) P_i.$$

The points to interpolate between are referred to as control points, and each Bernstein basis, $b_{i,n}$ gives information about the contribution of each point. The B-splines can be formulated in a similar sense to the Bezier curves, as

$$S(x) = \sum_{i=0}^{n} N_{i,n(x)} P_i \Rightarrow spline(x) = \sum_{i} c_i B_i(x), \tag{3.6}$$

where c_i is the control point, and $B_i(x)$ represents the basis function. It is this B-spline that corresponds to the $spline(x)$ part in the activation function $\phi(x)$ that is learned during training. For simplicity, a spline of order $k = 3$ was fixed with 5 number of grid intervals, *G*. Moreover, a cubic spline ensures C^1 level of continuity at the points where the underlying Bezier curves are joined, or at the *knots*. Hence, this makes sure that the function (spline) and the its gradients remains continuous at these points, thus avoiding potential discontinuities. The number of control points can be computed as

$$control\ points = G + k. \tag{3.7}$$

This subsequently fixes the basis functions as well. The concept of control points, B-splines, and the basis function is illustrated and can be referred from Appendix Section A.2.

The remaining parameter, control points (c_i), can be treated as weights or coefficients that when multiplied with the B-spline basis functions, define the shape of the activation. Moreover, the effect weights w_b, and w_s can be absorbed by $b(x)$ and $spline(x)$ respectively, but to better control the overall magnitude of the activation function, avoid divergence, and irregular or oscillatory activation functions (overfitting/underfitting), these factors are also trainable. Furthermore, some noise is also added to the initialization of the weights, c_i, w_b, w_s, in order to make the training robust and avoiding sensitivity to initialization. If interested, the reader can see the initialization of these weights from the source codes, the link to which (git repository) has been provided earlier. These weights are the parameters of the KAN network.

An oscillatory activation function due to non-uniform grid initialization is shown below.

(a) Oscillatory activation function

(b) Smooth activation function

Figure 3.7: Activation function behavior

Although at different edges, Figure 3.7 shows how an activation function can be represented or approximated in different ways due to improper training or grid initializations. This then has implications on the final model which can be unstable.

Once the model is initialized, it undergoes optimization of the parameters via the LBFGS optimizer where the objective function to minimize is the loss function (root mean square error). This is followed by pruning when the training is completed. First, the inputs are pruned. This is based on the attribution scores of the input, and if they are below a certain threshold, the inputs are discarded from the network. A threshold hyperparameter of value **2** $\times 10^{-1}$ is chosen for our datasets. When inputs are pruned, we iterate from the output layer to the input layer. Initially, the attribution scores are initialized as an identity matrix, thus assigning unit scores, for each output. Based on the connectivity of each output node to the number of sub-nodes in the preceding layer, the scores are distributed among them, also referred to as *sub-node scores*. The key step is the calculation of the edge scores. Edge here refers to the number of connections that a given node has. Based on these sub-node scores which can essentially be treated as weights, Einstein summation is performed with the standard deviation of the activation functions over that edge, referred as *edge_actscale*, and then normalized by the standard deviation of all the activation functions over that sub-node, referred by *subnode_actscale*. This is performed over all the edges that are connected to a given node, and the mean of them gives the node score for a given neuron. This is back-propagated to the input layer, where subsequently, the node scores for each of the inputs are computed. The inputs which then have lower attribution scores that the threshold are discarded from the network.

Mathematically, the input pruning process is formulated below.

$$\text{edge\_score}_i = \sum_j \frac{\text{edge\_actscale}_{ij} \times \text{subnode\_score}_j}{\text{subnode\_actscale}_j + 10^{-4}} \tag{3.8}$$

where:

- edge_actscale$_{ij}$ represents activation scale factors for the edges between subnode j in layer $(l-1)$ and node i in layer l.
- subnode_score$_j$ is the attribution score from the previous step.
- subnode_actscale$_j$ normalizes the contribution of each edge.

In other words, edge_actscale is a representative of the variance of the activation function at that edge, and subnode_actscale correspondingly represents the variance in the output from a particular neuron. The node score is then obtained by averaging the following quantity.

$$\text{node\_score}_j = \sum_i \text{edge\_score}_{ij} \tag{3.9}$$

Once the inputs are pruned, the redundant nodes and edges are removed from the network, as well. In a similar sense to pruning the inputs, if the node attribution score is below a certain threshold, it is removed. The threshold value used for node is set to 1 $\times 10^{-2}$. The threshold scores of each node is calculated in the same way, followed by the initialization of tensors `overall_important_up`, and `overall_important_down` that indicate which nodes are important. Depending on the number of connections from a particular node, the attribution scores are distributed. This ensures that if a particular neuron is considered important, then all of its branches are also considered important and marked as active, to ensure that the information flows from one layer to another in the correct manner. Similarly, if a neuron is considered to be inactive, or unimportant, it is removed and the corresponding connections have their masks updated to 0, otherwise initialized as 1. *Masks* is a term in the context of KANs which essentially represents if the edge is active or not. With an activation mask of 0, there is no contribution of that edge in the model. The redundant neurons and edges are removed accordingly. If the attribution score of the edge is greater than the threshold, set to 3 $\times 10^{-2}$, the edge is then multiplied with its activation mask value. This ensures that even if the edge may have a higher threshold, if its corresponding neuron is inactive, that edge will also be inactive. This avoids the issue of hanging edges wherein an edge may be present due to a high attribution score, but may not be connected to any neuron.

A pseudo-algorithm for the pruning of inputs and nodes & edges is presented below.

Algorithm 5: Input Pruning Based on Attribution Scores

Input: Initialized model
Output: Model with pruned inputs

Prune Inputs Based on Attribution Scores;
Initialize attribution scores as an identity matrix (unit scores for each output);
for *each output layer node* **do**
    Calculate node scores based on connectivity to subnodes in the previous layer;
    for *each edge connected to the node* **do**
        Retrieve subnode scores (weights for each connection);
        Perform Einstein summation with standard deviation of the activations over the edge;
        Normalize the result by the standard deviation of the activation functions over the sub-node;
    Average the edge scores to get the final node score for the given neuron;
    Backpropagate the node scores to the input layer;

Calculate Final Input Scores;
for *each input node* **do**
    Compute node scores from backpropagated values (propagate score from output to input);

Prune Inputs Below the Threshold;
Define a pruning threshold value (e.g., 2×10^{-1});
for *each input node* **do**
    if *input node's attribution score is below the threshold* **then**
        Discard the input node from the network;

Update the Model;

Algorithm 6: Pruning of Nodes and Edges Based on Attribution Scores

Input: Initialized model, threshold values for node and edge
Output: Pruned model with redundant nodes and edges removed

Prune Redundant Nodes;
Define node pruning threshold value (e.g., 1×10^{-2});
for *each node in the network* **do**
    Calculate the node attribution score based on connectivity and edge weights;
    if *node attribution score is below threshold* **then**
        Remove the node from the network;
        Update masks for incoming and outgoing edges of the removed node to 0;

Identify Important Nodes and Update Connections;
Initialize tensors `overall_important_up` and `overall_important_down`;
For each active node, distribute its importance score across its connected nodes and edges;
for *each active node* **do**
    Mark connected branches as important based on node attribution score;

Prune Redundant Edges;
Define edge pruning threshold value (e.g., 3×10^{-2});
for *each edge in the network* **do**
    Retrieve the edge attribution score based on its connected nodes;
    if *edge attribution score is greater than threshold* **then**
        Multiply the edge with its mask value;

Update the Model;

With pruning completed, the activations functions were set to be symbolic. A library of unary functions

was given to KANs and each of the activation functions were approximated by the functions in the library. The best fit, in terms of R^2 score was chosen as the symbolic representative for that activation function. An advantage of KANs is that it can also perform affine transformations of the input library functions. Hence, if $\sin(x)$ is not a good fit, it can scale and shift the function to the form $a \cdot \sin(bx)$.

An activation mask is used to determine whether an edge is active or not. Similarly, a symbolic mask is used to determine if a certain connection is particularly valid, and if it can be represented via a mathematical function, remains interpretable, is physically and mathematically correct.

Once completed, the model was trained again, and then finally the symbolic formula was exported in the form of symbolic equations. If the network had multiple outputs, each output had its own symbolic equation. The equations represented the final output of the model after the training. To promote a simplistic model, basic algebraic and trigonometric functions were given in the library. This avoided overly complex equations for turbulent production and the Reynolds Stress Tensor, and a trained model with functions that could possibly cause bloating or divergence when put in the CFD solver, such as $\frac{1}{x^5}$, $\exp(x)$, and $\log(x)$ among others.

The pseudo-algorithm of KANs, outlining the important steps is shown below.

Algorithm 7: KAN Pruning and Symbolic Regression

Input: Input dataset
Output: Pruned and trained model with symbolic representation

Step 1: Model initialization with input data;
Step 2: Prune Inputs Based on Attribution Scores;
`prune_input()` $\rightarrow$ Algorithm 5;
Step 2: Prune Nodes and Edges;
`prune()` $\rightarrow$ Algorithm 6;
Step 3: Update the Model;
Make a copy of the original model;
Remove unnecessary neurons from each layer;
Update the model by removing redundant edges;
Step 4: Train the pruned Model;
Step 4: Convert Activations to Symbolic Form;
Define a library of unary functions for symbolic representation;
for *each activation function in the network* **do**
    Approximate using best fit function based on R^2 score;
    if *symbolic mask and activation mask is 0* **then**
        Set activation function to 0;
Step 5: Train the Model and Export Symbolic Formula;
Export symbolic formula for each output as equations;
Ensure mathematical correctness and interpretability;

3.4. Algorithm development procedure

An 80%-20% (training-testing) split is made for any dataset for the $frozen$ or any $a\ priori$ analysis. This helps to identify if the model is overfitting on the training data and behaving poorly on unseen data, or if it is able to train a sufficiently capable model and maintain the correlations. Based on the result, parameters such as crossover, and mutation probabilities, population size, or number of grid points can be changed for further refinement, if need be.

The development of the gradient based genetic algorithm was tracked by testing it against different symbolic regression benchmark problems, referenced from [63], and [39]. Given the inherent random nature of genetic programs, it is not possible to draw valid conclusions after one complete run with a given dataset, and using one particular random seed. Hence, to tackle this, multiple runs were

performed with the same dataset for the GP algorithms with different random seeds to be able to come to a deterministic conclusion. For KANs however, results from the paper that introduced them were reproduced using same (and similar) datasets and parameters [45]. When completed, they were then directly applied to the CFD cases. Metrics such as root mean sure errors, R^2 scores, median distribution of quantities, and scatter plots were used as a comparison between the existing and the developed methods. For the benchmark problems, the datasets had non-dimensional features and target variable. Hence, the features could be combined in any way with mathematical operators without having to worry about the dimensions of the resulting expression. Thus, the algorithm only produced one expression as its output, and this expression was used to regress for the target variable.

Once the developed algorithms were successful, the next step was the development of models for CFD quantities and perform *a priori* testing, as explained ahead.

3.5. Frozen (A priori) Model development

CFD datasets were used to see how well the developed methods were able to perform against real-time simulation data, and if they were able to overcome the limitations of existing method(s) in terms of computational cost (number of function evaluations), and predictive accuracy (correlations). Models for turbulent production and the RST were developed and tested. Unlike several runs on benchmark cases (GP), only a few iterations were performed when required as strong inferences were already made for the developed methods.

3.5.1. CFD cases

The two CFD cases that were used for *a priori* testing were the Periodic Hill (PH), and the Square Duct (SD) case setups, with LES data as the reference case. The Periodic Hill case setup is representative of flow over a series of hills separated by a certain distance. RANS models fail to accurately predict the separated flow and circulation regions, subsequently the reattachment point as well due to modeling errors largely related to turbulent production. Square Duct on the other hand, provides a good case where modeling errors due to RST are relevant.

The boundary conditions are and the geometries can be referenced from the Appendix Section A.4. Furthermore, these datasets and simulation setups have previously been used by the author's research group for academic studies and serve as benchmark cases for evaluating new methods. Hence, no validation has been performed to verify if the setup diverges. In addition, the quantities are from incompressible simulations, and the flow is driven by fixing the bulk velocity (mass flow rate) at $0.7\ m/s$ and $38.7\ m/s$ for PH and SD cases respectively.

3.5.2. Turbulent Production and RST modeling

The methodology to represent the target variables as a function of flow quantities was referenced from the works of [62], which has also been explained below. In it, the author deduces a more general general form of the RST (anisotropy component) by applying dimensional analysis, variance, and coordinate transformations. The RST is thus composed of isotropic $(\frac{2}{3}k\delta_{ij})$ and an anisotropic component a_{ij}. The isotropic component is absorbed into the pressure term resulting in a modified mean pressure, and responsible for pressure-like behavior of turbulence [65]. The anisotropy component however, represents the deviations from isotropy, and is responsible for the directional transfer of momentum in a turbulent flow. This makes it crucial in predicting separation, reattachment and secondary flows. The anisotropy component of the RST can be represented as a function of the strain rate tensor, s and the rotational rate tensor, ω, where each of the mentioned quantities can be scaled or normalized by the turbulent production, k and dissipation, ε. The mathematical definitions of these quantities for an incompressible flow are

$$a_{ij} = \overline{u_i' u_j'} - \frac{2}{3}k\delta_{ij}, \tag{3.10}$$

$$s_{ij} = \frac{1}{2}(k/\varepsilon)(\partial_j U_i + \partial_i U_j), \tag{3.11}$$

$$\omega_{ij} = \frac{1}{2}(k/\varepsilon)(\partial_j U_i - \partial_i U_j), \tag{3.12}$$

$$a_{ij} = a_{ij}(\mathbf{s}, \omega). \tag{3.13}$$

Note that the form of (Equation 3.10) can also be inferred from (Equation 2.25). a_{ij} can be represented as the sum of an infinite tensor polynomial, but with the application of the Cayley-Hamilton theorem, the number of linearly independent second-order tensors that may be formed from **s**, and ω are finite, with the independent coefficients, **I** a function of a finite number of invariants as well. Furthermore, the anisotropy component a_{ij} can be formulated as $2kb_{ij}$ where b_{ij} can further be represented as $-\frac{\nu_t}{k}S_{ij}$. The infinite tensor polynomial b_{ij} can now be expressed as

$$\mathbf{b}(\mathbf{s},\omega) = \sum_{\lambda} I^{\lambda}\mathbf{T}^{\lambda}. \tag{3.14}$$

It can be considered that the predicted anisotropy is forced to lie on a basis of isotropic tensors, hence, rotational invariance is also embedded into it [42]. Thus, by adopting this methodology for the current study, Galilean invariance is also ensured, which implies that the flow, or the physics is independent of the frame of reference.

For a 2-D flow, there exists only 3 linearly independent tensors (Pope tensors) and 2 non-zero independent variants. On the other hand, for a 3-D flow, there are 10 tensors, and 5 invariants, which are

$$\mathbf{T_1} = s,$$
$$\mathbf{T_2} = s\omega - \omega s,$$
$$\mathbf{T_3} = s^2 - \frac{1}{3}I(s^2),$$
$$\mathbf{T_4} = \omega^2 - \frac{1}{3}I(\omega^2),$$
$$\mathbf{T_5} = \omega s^2 - s^2\omega,$$
$$\mathbf{T_6} = \omega^2 s + s\omega^2 - \frac{2}{3}I(s\omega^2),$$
$$\mathbf{T_7} = \omega s\omega^2 - \omega^2 s\omega,$$
$$\mathbf{T_8} = s\omega s^2 - s^2\omega s,$$
$$\mathbf{T_9} = \omega^2 s^2 + s^2\omega^2 - \frac{2}{3}I(s^2\omega^2),$$
$$\mathbf{T_{10}} = \omega s^2\omega^2 - \omega^2 s^2\omega.$$

and $\mathbf{s^2}, \omega^\mathbf{2}, \mathbf{s^3}, \omega^\mathbf{2}\mathbf{s}, \omega^\mathbf{2}\mathbf{s^2}$, where $I = \delta_{ij}$.

If interested, the reader can further learn about them from [62]. These Pope tensors are further functions of rotational and strain rate tensors, that make use of higher order velocity gradients. Hence, often, it may cause instability issues and the model developed may diverge. Thus, to tackle this, only the first 6 tensors will be used for some cases.

Thus, with this it can be concluded that the Reynolds stresses are a function of a limited number of tensors (basis), and scalars. Moreover, in addition to the invariants, additional scalar features are also added that are assumed to be important to the flow, and are explained below.

$$q_Q = \frac{||\omega^2|| - ||s^2||}{2 \cdot \max(||s^2||, 1 \times 10^{-10})},$$
$$q_T = ||S||_F \cdot \frac{k}{\max(\varepsilon, 10^{-10})},$$
$$q_\gamma = \frac{||\nabla U|| \cdot k}{\max(\varepsilon, 1 \times 10^{-10})},$$
$$q_{Re} = 2 - \min(\frac{\sqrt{k} \cdot y_{wall}}{50\nu}, 2),$$
$$q^{(B)}_{\tau/k} = \frac{||\tau^B|| \cdot k}{\max(k, 1 \times 10^{-10})}$$
$$q^{(B)}_{C/P} = \frac{U \cdot \nabla k}{\max(|s : \omega\tau^{(B)}|, 1 \times 10^{-4})},$$

These are non-dimensional ratios of local flow quantities which result in the final correction field being physically meaningful, and aligning itself with the flow characteristics. The physical significance of the terms is explained below.

Table 3.1: Physical interpretation of turbulence quantification metrics

Notation	Physical Interpretation
q_Q	Dimensionless Q-criterion evaluating vorticity dominance (ratio of rotation rate tensor norm to strain rate tensor norm)
q_T	Turbulence intensity ratio: relative magnitude of turbulent fluctuations compared to mean flow strain rate
q_γ	Shear production-dissipation balance: measures whether turbulence production by shear dominates over viscous dissipation
$q_{\mathbf{Re}}$	Local Reynolds number scaling: characterizes turbulence development based on velocity and length scales
$q^{(B)}_{\tau/k}$	Boussinesq stress anisotropy: ratio of modeled turbulent stresses to normal Reynolds stress components
$q^{(B)}_{C/P}$	Energy ratio: compares convective transport to production of turbulent kinetic energy

The need to correct for turbulent production arises from b_{ij}'s correction. The discrepancy in a_{ij} can be obtained by subtracting the high-fidelity anisotropy component and that obtained from the Boussinesq hypothesis. The augmented b_{ij} with the data driven correction can be formulated as

$$b_{ij} = -\frac{\nu_t}{S_{ij}} + b^{\Delta}_{ij} \tag{3.15}$$

However, ν_t needs to be computed for this which requires ω. To obtain it, the k-corrective Frozen RANS is used by the author's research group where the ω equation is iteratively solved for, keeping other high-fidelity variables (U, k, b_{ij}) as $frozen$, to obtain ν_t. It is made clear that even though b^{Δ}_{ij} alters the turbulent production P, solving the k equation will not result in the same k from the high fidelity data [65]. Thus, the residual of the k equation is also computed with the high fidelity data and the modeling of this field, R, is referred to as *Turbulent Production modeling* in the present study. In addition, both the target fields are treated together to ensure that energy remains conserved.

The field R will tend to correct the transport equations by adding or decreasing the local turbulent kinetic energy production. Thus, it is modeled in a similar way to turbulent production as

$$R = 2kb^R_{ij}\frac{\partial U}{\partial x_j}. \tag{3.16}$$

Thus, the tensor for turbulent production can be formulated as

$$G_i = 2k\sum_n \sum_{i,j} \nabla U_{ij} \cdot T^{(i)}_{ij}, \tag{3.17}$$

with the same invariants. The modeling of the *Augmented k-ω SST model* which involves the inclusion of these correction fields can be referenced from Appendix Section A.5

Once the basis are evaluated for a given flow, the objective of the developed algorithm(s) is to determine the scalar coefficients, which when plugged into (Equation 3.14) will solve for b, or for the correction in turbulent production. To do so, the approach of Tensor Basis Neural Networks is adopted from [42]. With regards to the gradient optimized genetic program, an individual will now contain λ expressions, where λ corresponds to (Equation 3.14). Each of these coefficients will correspond to each of the Pope tensors. The fitness will be evaluated by taking the dot product or the element-wise product of the coefficients with the tensors, and observing how well the correlation is to the reference data. Similarly, in the case of KANs, the dot product is then taken of each of the λ neurons in the output layer with the tenors. In this way, both methods will be structured and evaluated to optimize the coefficients.

The concept of adapting the TBNN structure to KANs and genetic programming is visually shown and can be referenced from Appendix Section A.3.

3.6. A posteriori Model Development

The superior method among the two (KANs) is now selected for *a posteriori* analysis. This involves running a flow simulation and differentiating the CFD code to correct for the turbulence modeling errors. The model produced by the algorithm is inserted into the solver, and a CFD run is performed with the correction field (β). This tells us how the correction field is influencing flow quantities such as velocities, shear stresses, and visualize regions of circulation, flow separation and reattachment. The error, or the objective $\mathcal{I}$, is now defined as the squared $\mathcal{L}2$ norm between the velocities produced from the CFD run, and the high-fidelity data. This information is used to evaluate how the objective is changed by the correction field, which gives us $\frac{d\mathcal{I}}{d\beta}$. The information is transferred to KANs, which uses it to compute the quantity - $\frac{d\mathcal{I}}{d\theta}$. Given that the β field is produced by KANs, it is a function of the parameters, θ. Hence, by direct chain rule, the computation is as

$$\frac{d\mathcal{I}}{d\theta} = \frac{d\beta}{d\theta} \cdot \frac{d\mathcal{I}}{d\beta}. \tag{3.18}$$

The L-BFGS optimizer uses this gradient to update its parameters accordingly which changes β to optimize for the objective function. Once the model produces a new β, its symbolic formula is obtained for better interpretability and to understand which flow quantities are influencing the correction field, and subsequently the overall solution. The associated algorithm is presented below, for better clarity.

Algorithm 8: A Posteriori Optimization of turbulence modeling via KANs

Procedure `A posteriori_CFD_Optimization`;
Input: Flow invariants from converged CFD solution
Result: Optimized correction field β minimizing objective $\mathcal{I}$ and interpretable symbolic formula

Initialize parameters θ of KAN model and give it the inputs;
while *not converged* **do**
  $\beta \leftarrow$ `ComputeBeta(`θ`)`;
  Obtain symbolic form of β;
  Insert β into CFD solver;
  CFD velocity field $\mathbf{U}_{\text{cfd}} \leftarrow$ `RunCFD(`β`)`;
  Objective $\mathcal{I} \leftarrow$ `ComputeError(`CFD velocity field $\mathbf{U}_{\text{cfd}}$, High-fidelity velocity field $\mathbf{U}_{\text{true}}$`)`;
  $\frac{d\mathcal{I}}{d\beta} \leftarrow$ `DifferentiateCFD(`Objective $\mathcal{I}$ *w.r.t.* β`)`;
  $\frac{d\mathcal{I}}{d\theta} \leftarrow \frac{d\beta}{d\theta} \cdot \frac{d\mathcal{I}}{d\beta}$;
  $\theta \leftarrow$ `L_BFGS_Step(`θ, $\frac{d\mathcal{I}}{d\theta}$`)`;

The above explanation briefly gives an overview about the framework, and how $a\ posteriori$ analysis is performed. The working of differentiating the CFD code, followed by the derivation of scalar invariants and how they are inserted into the code is explained in detail below.

To begin with, the author uses **DAFoam v4.0** to differentiate the CFD code (turbulence model), i.e., the objective function with respect to the correction field [29] [28]. It should be noted that at the time of starting $a\ posteriori$ analysis, DAFoam v4.0 was not complete, and unable to provide the required derivatives, or differentiate the turbulence model, also referred to as **field inversion** modeling. The author completed DAFoam v4.0, and successfully incorporated the field inversion models capable to differentiate two turbulence models - $k-\omega\ SST$, and $Spalart-Allmaras$. Moreover, the author also developed the interface between DAFoam and KANs, as the former was written in C++, while the latter in Python. This ensured that information could be easily transferred between the differentiation and the machine learning frameworks.

Ideally, we need to compute the quantity $\frac{d\mathcal{I}}{d\beta}$ directly so that the change the production field, β, can be done accordingly to minimize $\mathcal{I}$. However, that is a computationally expensive process as it would involve observing how the velocity field changes with small perturbations in β, which would further

invoke multiple CFD runs. Thus, we resort to the adjoint method to obtain the quantity $\frac{d\mathcal{I}}{d\beta}$, followed by chain rule to compute $\frac{d\mathcal{I}}{d\theta}$. Python's **OpenMADO** framework is used to facilitate the adjoint computation.

This involves first initializing the $\frac{d\mathcal{R}}{d\mathcal{W}}$ matrix, the adjoint vector, adding the state variable $(\mathcal{U})$ as an output to the OpenMDAO framework to track its residuals, and β as the input field. After the flow (non-linear problem) has been solved, the Jacobian $\frac{d\mathcal{R}}{d\mathcal{W}}$ is computed using PETSc, where $\mathcal{R}, \mathcal{W}$ refers to the residuals, and state vector respectively. Accuracy, and memory usage is balanced by using a pre-conditioner approximation, and lower-bound thresholding $(< 10^{-16})$. Coloring is further used to accelerate the finite differencing $(\delta = 10^{-5})$ by perturbing multiple states at once, and hence computing (and extracting) multiple Jacobian columns simultaneously. The adjoint vector ψ is then computed via the linear system as

$$\left(\frac{\partial \mathcal{R}}{\partial \mathcal{W}}\right)^{\top} \psi = \frac{\partial \mathcal{I}}{\partial \mathcal{W}}, \tag{3.19}$$

where

- $\frac{\partial \mathcal{I}}{\partial \mathcal{W}}$ is the sensitivity of the objective $\mathcal{I}$ to state variables $\mathcal{W}$,
- ψ is the adjoint vector.

The state vector is the velocity, $\mathcal{U}$, in our case. Krylov method (GMRES) is used to iteratively approximate ψ. State variable normalization ensures numerical stability, and is performed as

$$\begin{aligned} \mathcal{U} &\rightarrow \mathcal{U}_o \quad \text{(Bulk velocity normalization)}, \\ p &\rightarrow \frac{\mathcal{U}_o^2}{2} \quad \text{(Pressure scaling)}, \\ \tilde{\nu} &\rightarrow 10\tilde{\nu_o} \quad \text{(Modified viscosity scaling)}. \end{aligned}$$

ψ is then used to compute the derivative of the objective with respect to the correction field as Jacobian-transposed vector product given by

$$\frac{d\mathcal{I}}{d\beta} = \underbrace{\frac{\partial \mathcal{R}}{\partial \beta}^{\top}}_{\substack{\text{Residual-Jacobian} \\ \text{w.r.t. } \beta}} \cdot \underbrace{\psi}_{\substack{\text{Adjoint} \\ \text{vector}}}, \tag{3.20}$$

where $\frac{d\mathcal{R}}{d\beta}$ is given by OpenMDAO. Essentially, substituting the quantities, the total derivative becomes

$$\begin{aligned} \frac{d\mathcal{I}}{d\beta} &= \left(\frac{\partial \mathcal{R}}{\partial \beta}\right)^{\top} \psi \\ &= \left(\frac{\partial \mathcal{R}}{\partial \beta}\right)^{\top} \left[\left(\frac{\partial \mathcal{R}}{\partial \mathcal{W}}\right)^{-\top} \frac{\partial \mathcal{I}}{\partial \mathcal{W}}\right]. \end{aligned} \tag{3.21}$$

This information is transferred to KANs to compute $\frac{d\mathcal{I}}{d\theta}$ as per (Equation 3.18) and continue the optimization process.

The case selected for $a\ posteriori$ is the Periodic Hill. It was observed that the $k-\omega\ SST$ turbulence model was unstable on coarse meshes, and required turning of parameters such as relaxation factors to allow for a stable optimization process. Thus, the $Spalart-Allmaras$ turbulence (1-equation) model was selected for optimization with $k-\epsilon$ (2-equation model) taken as the reference. The turbulence model is described as

$$\begin{aligned} &\underbrace{\frac{\partial(\phi\rho\tilde{\nu})}{\partial t}}_{\substack{\text{Unsteady} \\ \text{Term - } U_{\text{ns}}}} + \underbrace{\nabla\cdot(\phi\rho\mathbf{U}\tilde{\nu})}_{\substack{\text{Convection} \\ \text{Term - } C_{\text{onv}}}} - \underbrace{\nabla\cdot\left(\phi\rho\frac{\nu+\tilde{\nu}}{\sigma}\nabla\tilde{\nu}\right)}_{\substack{\text{Molecular + Turbulent} \\ \text{Diffusion - } D_{\text{iff}}}} \\ &\qquad - \underbrace{\frac{C_{b2}}{\sigma}\phi\rho\|\nabla\tilde{\nu}\|^2}_{\substack{\text{Cross-Diffusion} \\ \text{Term - } C_{\text{ross}}D_{\text{iff}}}} = \underbrace{C_{b1}\phi\rho\tilde{S}\tilde{\nu}}_{\substack{\text{Production} \\ \text{Term - } P_{\text{rod}}}} - \underbrace{C_{w1}\phi\rho f_w\frac{\tilde{\nu}^2}{d^2}}_{\substack{\text{Destruction} \\ \text{Term - } D_{\text{est}}}} \end{aligned} \tag{3.22}$$

The transport equation is solved for a modified turbulent viscosity $(\tilde{\nu})$ which is later used to compute the turbulent viscosity. The model is a good candidate for optimization as it computationally cheap, and over-predicts the separation region due to its inability to accurately model strong anisotropic effects (recirculation, shear layer) as compared to $k - \varepsilon$.

The simulation parameters for training, testing and reference case are tabulated below. The geometry and mesh details for the cases are presented later.

Table 3.2: Input Parameters for DAFoam Simulation

Category		Value
Physical Parameters		
	Reference velocity (U_0)	0.02
	Reference viscosity ($\tilde{\nu_0}$)	1×10^{-4}
Solver Options		
	Solver name	`DASimpleFoam`
	Primal minimum residual tolerance	1.0×10^{-8}
	KANs (optimization) tolerance	1.0×10^{-4}
fvSource Options		
	Type	`meanVelocityForce`
	Value	$\bar{U}$ = 0.02
	Direction	`[1.0, 0.0, 0.0]`
Adjoint Options		
	PC fill level	`1`
	GMRES relative tolerance	1.0×10^{-8}

Table 3.3: Boundary Conditions for PH CFD case

Variable	bottomWall	topWall	inlet	outlet
U	`noSlip`	`noSlip`	`cyclic`	`cyclic`
p	`zeroGradient`	`zeroGradient`	`cyclic`	`cyclic`
$\tilde{\nu}$	`fixedValue` $\tilde{\nu} = 0$	`fixedValue` $\tilde{\nu} = 0$	`cyclic`	`cyclic`
nut	`nutLowReWallFunction`	`nutLowReWallFunction`	`cyclic`	`cyclic`

Table 3.4: Fluid Transport Properties Used

Parameter	Description	Value
`transportModel`	Constitutive model	Newtonian
ρ	Density [kg/m^3]	1
ν	Kinematic viscosity [m^2/s]	5×10^{-6}
Pr	Prandtl number (molecular)	0.7
Pr_t	Turbulent Prandtl number	1.0

First order `(bounded Gauss upwind)` schemes were used for divergence, with `GaussSeidel` smoother and `smoothSolver` used.

Based on the Relative Term Importance Analysis (RITA), new invariants were derived from the converged $Spalart - Allmaras$ flow field as the selected model did not have an explicit equation for k and

b_{ij}^{Δ}, hence the invariants or tensors from the $k-\omega\ SST$ model where not applicable. RITA technique was developed by Monica et al. to understand the relative importance of different terms such as diffusion and production in regions of circulation, separation, and shear layer and use them as invariants [40]. For the Spalart-Allmaras model, they were defined as

$$\text{RITA}_{C_{onv}} = \frac{|C_{onv}|}{|C_{onv}| + |D_{iff}| + |C_{ross}D_{iff}| + |P_{rod}| + |D_{est}|}, \tag{3.23}$$

$$\text{RITA}_{D_{iff}} = \frac{|D_{iff}|}{|C_{onv}| + |D_{iff}| + |C_{ross}D_{iff}| + |P_{rod}| + |D_{est}|}, \tag{3.24}$$

$$\text{RITA}_{C_{ross}D_{iff}} = \frac{|C_{ross}D_{iff}|}{|C_{onv}| + |D_{iff}| + |C_{ross}D_{iff}| + |P_{rod}| + |D_{est}|}, \tag{3.25}$$

$$\text{RITA}_{P_{rod}} = \frac{|P_{rod}|}{|C_{onv}| + |D_{iff}| + |C_{ross}D_{iff}| + |P_{rod}| + |D_{est}|}, \tag{3.26}$$

$$\text{RITA}_{D_{est}} = \frac{|D_{est}|}{|C_{onv}| + |D_{iff}| + |C_{ross}D_{iff}| + |P_{rod}| + |D_{est}|}, \tag{3.27}$$

$$\tag{3.28}$$

Unsteady term is not taken into account as steady state is assumed. Since these are ratios of local flow quantities, they also highlight regions where production is relatively dominant (shear layer), or where destruction is (circulation region) in the flow. The correction field is thus easier to interpret and physically meaningful. Moreover, as the same turbulence model is used to solve the flow, it was observed that the distribution of relative importance of these terms across different periodic hill geometries remains similar, a finding also supported by [40]. This also makes these ratios generalizable, and applicable to different geometries, and flow conditions. In addition to the above, an invariant based on the viscosities is also used, and formulated as

$$q_{\nu} = \frac{\nu_t}{100 \cdot \nu}. \tag{3.29}$$

This invariant is representative of the turbulent and mean flow quantities, and is a function of the flow Re. Thus, the values can be very large for a high Re due to lower ν causing instabilities in the model. Hence, it is scaled by a factor of 0.01.

The invariants were given to a KANs model and the output was a correction field, β, with one value for each mesh cell. This was then inserted into the production term of the turbulence model as

$$\begin{aligned} \underbrace{\frac{\partial(\phi\rho\tilde{\nu})}{\partial t}}_{\substack{\text{Unsteady}\\ \text{Term}}} + \underbrace{\nabla\cdot(\phi\rho\mathbf{U}\tilde{\nu})}_{\substack{\text{Convection}\\ \text{Term}}} - \underbrace{\nabla\cdot\left(\phi\rho\frac{\nu+\tilde{\nu}}{\sigma}\nabla\tilde{\nu}\right)}_{\substack{\text{Molecular + Turbulent}\\ \text{Diffusion}}} - \underbrace{\frac{C_{b2}}{\sigma}\phi\rho\|\nabla\tilde{\nu}\|^2}_{\substack{\text{Cross-Diffusion}\\ \text{Term}}} \\ = \underbrace{C_{b1}\phi\rho\tilde{S}\tilde{\nu}(1+\beta)}_{\substack{\text{New Production}\\ \text{Term}}} - \underbrace{C_{w1}\phi\rho f_w\frac{\tilde{\nu}^2}{d^2}}_{\substack{\text{Destruction}\\ \text{Term}}}. \end{aligned} \tag{3.30}$$

Thus, the correction field, taken as the multiplicative factor to the production field, directly influences the distribution of turbulent energy in the flow. It is to note that ideally, an invariant would need to be Galilean and rotational invariant (independent of frame of reference, and rotation of coordinate axes) and Re independent. This ensures the flow physics remains the same regardless of the frame reference, and the features do not diverge at high Re such as q_{ν}. Features such as velocity gradients, RST, and transport scalar quantities are essential, however, as noted in a prior study, such features are challenging to derive which can also demonstrate meaningful correlations. Therefore, while these features will be used KANs pruning and mutual information will be implemented to systematically remove non-informative features where feasible.

4

Preliminary Results of GP & KANs

This chapter presents the results of the algorithms against benchmark cases. Based on these results, a firm conclusion is then deduced. The procedure has been explained in Chapter 3.

4.1. Preliminary Testing of Genetic Programming

The current section presents the benchmark results and conclude with the selection of an optimizer and its parameters, to be subsequently used in CFD cases and discuss the limitations as well.

4.1.1. Initial Testing of GP with author defined problems

Prior to the testing against benchmark data, the ability of the GP algorithms was checked against certain functions, defined arbitrarily by the author. They were implemented and served the purpose as an initial indication of the performance of the GP algorithms and if optimizing the weights of terminals and operators can generate good correlations.

The problems are tabulated below, and functions illustrated following it.

Function	Expression	Domain
$f_1(x_1, x_2)$	$-7\sin(10x_1) + 12.5\cos(10x_2) - 2.5\log(x_1^2 + x_2^2)$	$x_1 \in [-1, 1], \quad x_2 \in [-2, 7]$
$f_2(x_1, x_2)$	$-2.5\sin(10x_1^{0.5}) + 4.5\cos(10x_2^2) - 3.5\exp(x_2 + x_1^{-0.8})$	$x_1 \in [3, 10], \quad x_2 \in [-2, 7]$
$f_3(x_1, x_2)$	$6\sin(x_1)\cos(x_2) + \log(x_1) + \exp(x_2) + x_1^{-0.6}$	$x_1 \in [3, 6], \quad x_2 \in [-10, -3]$

Table 4.1: Functions and Their Domains - Terminal Optimization

The functions were designed with a mix of exponential, logarithmic, power, and trigonometric operations to ensure that the devised algorithms can work against a wide variety of functions. In addition, the input features were initialized with a random distribution, 250 sample points, within the domains. The functional space of these problems is visualized below.

$$f(x_1, x_2) = -7\sin(10x_1) + 12.5\cos(10x_2^2) - 2.5\log(x_1^2 + x_2^2)$$

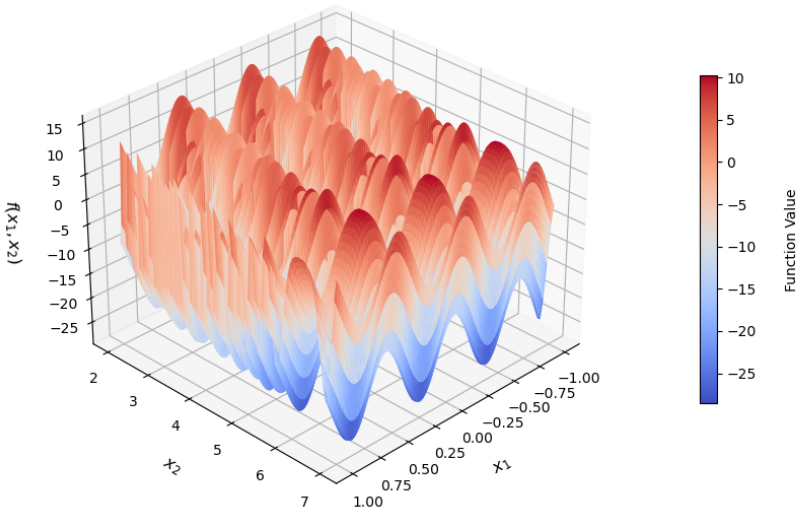


Figure 4.1: Demo function - I

$$f(x_1, x_2) = -2.5\sin(10\sqrt{x_1}) + 4.5\cos(10x_2^2) - 3.5\exp(x_2 + x_1^{-0.8})$$

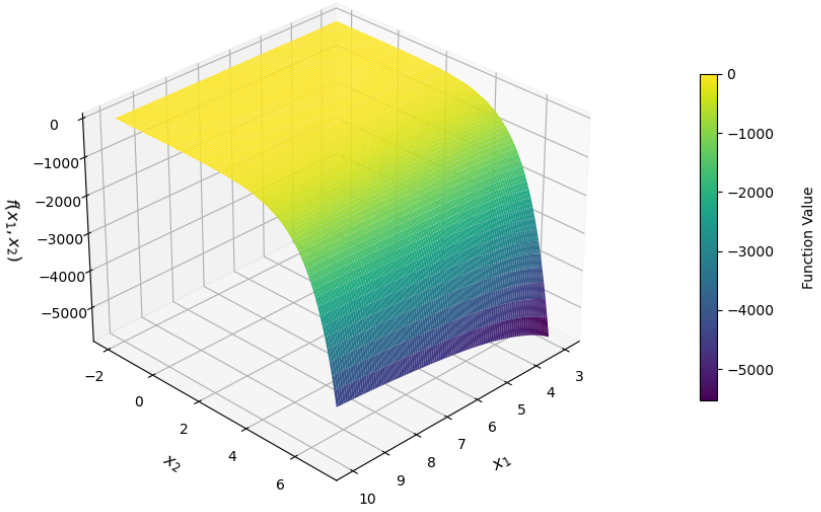


Figure 4.2: Demo function - II

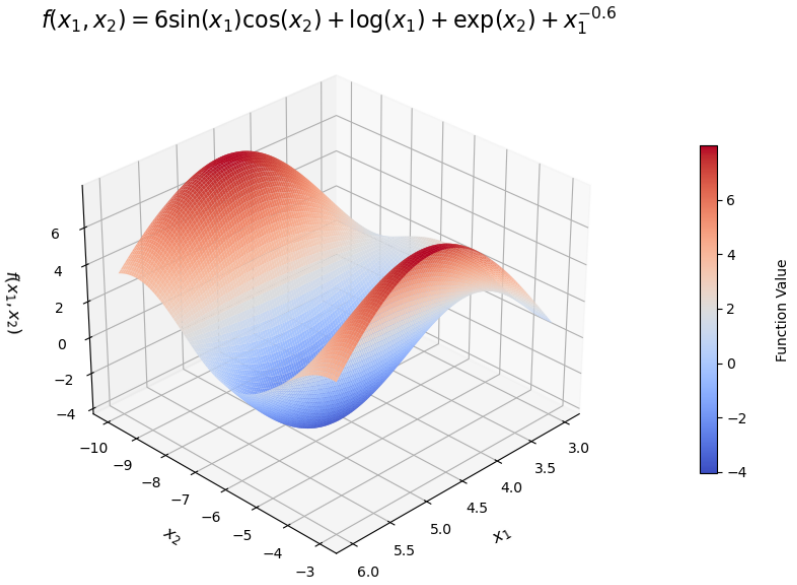


Figure 4.3: Demo function - III

Terminal Optimization

Optimization of the individuals was performed at every 2^{nd} generation, and only of the top 40% individuals in the population, to avoid excessive computational cost (function calls) and maintain a balance of both diversity and localization. The results, along with the details on the parameters of the optimizers, are presented below. Since we are interested to observe if the gradient optimization can result in convergence to superior quality individuals, only the testing correlations are presented. The evolution of metrics through the generations did not exhibit any informative trends, offering limited insights. Hence, the training R^2 and the Elite RMSE behavior of the algorithms is presented in Appendix Section B.1.

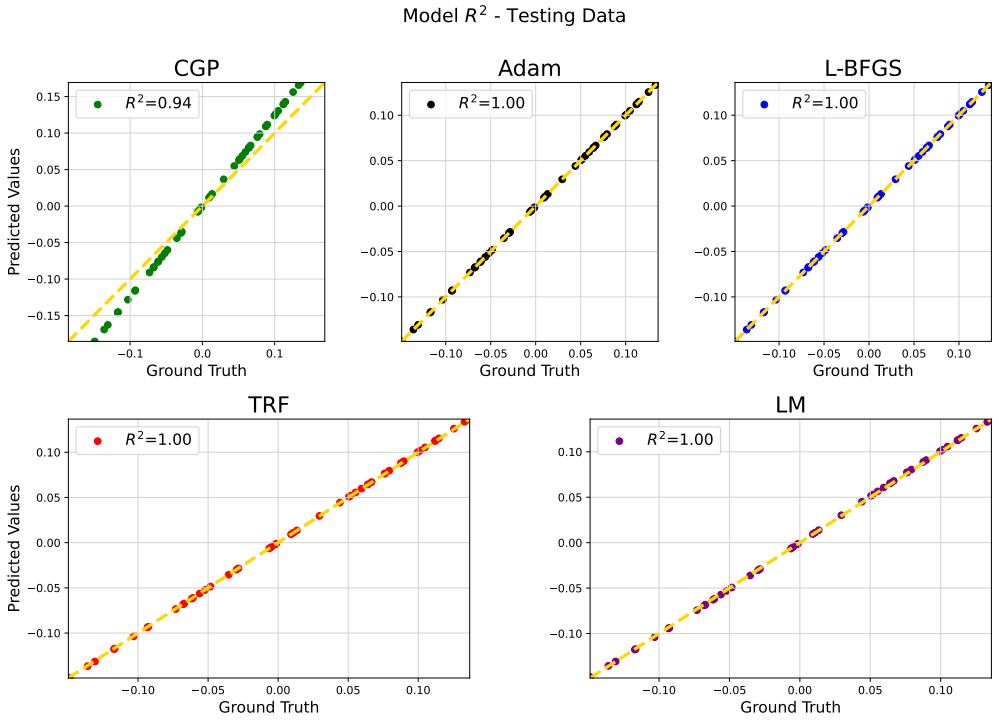


Figure 4.4: Scatter plot comparing predicted values and ground truth for the elite individual of $f_1(x_1, x_2)$.

The underlying function is $f_1(x_1, x_2)$, and can be referenced from Table 4.1. The testing correlations for all the optimized based GP were perfect scores. We can observe that CGP found it difficult to generate as good a model and converged to a local optima when compared with the optimizer based algorithms. This is also evident from the Elite RMSE where it can be seen that CGP struggles to reduce the error (Appendix Section B.1).

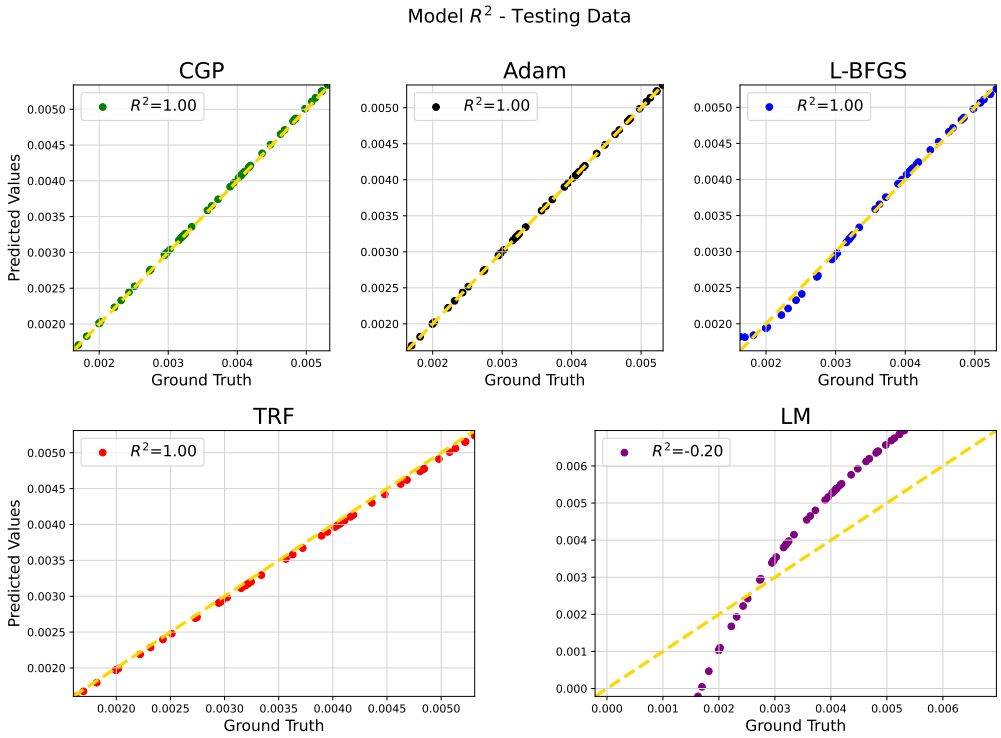


Figure 4.5: Scatter plot comparing predicted values and ground truth for the elite individual of $f_2(x_1, x_2)$.

The underlying function is $f_2(x_1, x_2)$, and can be referenced from Table 4.1. Given the function space is not very challenging, it can be observed that all the methods were able to converge to very good final models. However, LM struggled to find a good correlation. This shows us that the optimizers act differently and it is not always certain that using gradient methods can result in a good model. Depending on the function and parameter space, the optimization is susceptible to divergence as well.

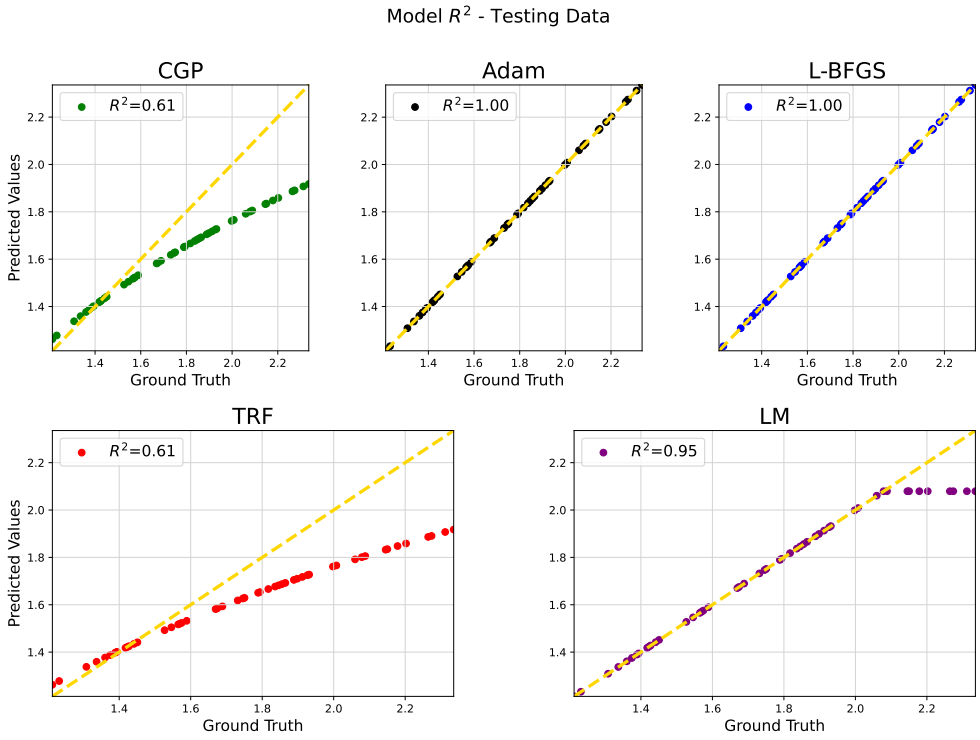


Figure 4.6: Scatter plot comparing predicted values and ground truth for the elite individual of $f_3(x_1, x_2)$.

The underlying function is $f_3(x_1, x_2)$, and can be referenced from Table 4.1. Similar to previous results, the $L-BFGS$, and $Adam$ give superior results in terms of R^2 scores, when compared to conventional genetic programming. Furthermore, the trust region algorithm achieves similar performance metrics as compared to the conventional method, both unable to converge to good models.

The algorithms were evaluated between 5-10 times over these author-defined regression problems, and a consistent behavior was observed in the performance of these algorithms when evaluated multiple times with these functions. However, given the inherent random nature of these algorithms, firm conclusions cannot yet be drawn. Nonetheless, the results are in good agreement with literature, and give a good indication that optimizing the terminals at every 2^{nd} generation enhances the performance of genetic programs. For the optimizers that often converge to a local optima quickly, it would imply that the effect of local optimization is very strong and it is limiting the algorithm to explore diverse functional spaces. One way to tackle this issue to further reduce the frequency of applying gradient optimization between generations.

The hyper parameters of the genetic program and that of the optimizers are tabulated below.

Optimizer	Parameters
Adam	`learning_rate`=0.01, `epochs`=50
L-BFGS	g_{tol}=f_{tol}=x_{tol}=1×10^{-4}, `eps`=1×10^{-2}, `max_iter`=20
Trust Region	g_{tol}=f_{tol}=x_{tol}=1×10^{-4}, `loss`=huber
Levenberg-Marquardt	g_{tol}=f_{tol}=x_{tol}=1×10^{-4}, `loss`=linear

Table 4.2: Optimizer settings for terminal optimization

Hyperparameter	Value
pop_size	50
generations	100
tournament_size	3
n_elites	1
function_set	add, sub, mul, protected_div, protected_nexp, protected_log
rc_ub	5
rc_lb	-5
pb_cx1p	0.3
pb_cx2p	0.2
pb_mut_node_rep	0.05
pb_mut_uniform	0.05
metric	rmse
tolerance	1e-9
optimizing frequency	2
num_optimizing_ind	20 - tournament selection

Table 4.3: Genetic Programming Hyperparameters

The paramaters rc_ub, and rc_lb are the upper and lower bounds for the terminals, when they are generated initially by **DEAP**.

Operator Optimization
Once an indication was obtained that optimizing for terminals can give better correlations, and scores, the next step was to incorporate the optimization of operators into the method, and observe if similar superiority, at the least, could be obtained over the conventional genetic programming method. Similar to the previous procedure, author-defined functions were implemented as an initial check. The functions are tabulated below.

Function	Expression	Domain
$f_1(x_1, x_2)$	$6\sin(x_1)\cos(x_2) + \log(x_1) + \exp(x_2) + x_1^{-0.6}$	$x_1 \in [3, 6], \quad x_2 \in [-10, -3]$
$f_2(x_1, x_2)$	$-7\sin(10x_1) + 12.5\cos(10x_2) - 2.5\log(x_1^2 + x_2^2)$	$x_1 \in [-1, 1], \quad x_2 \in [-2, 7]$
$f_3(x_1, x_2)$	$-2.8x_1^3 + 1.7x_2^2 + 5x_1 + 25\sin(x_2 - x_1) - 2.25\sin(x_1^2)$	$x_1 \in [-1, 0], \quad x_2 \in [0, 1]$

Table 4.4: Functions and Their Domains - Operator Optimization

The functional space of $f_1(x_1, x_2)$ can be referenced from Figure 4.8. The functional space of the remaining datasets is visualized below.

$$f(x_1, x_2) = -7\sin(10x_1) + 12.5\cos(10x_2^2) - 2.5\log(x_1^2 + x_2^2)$$

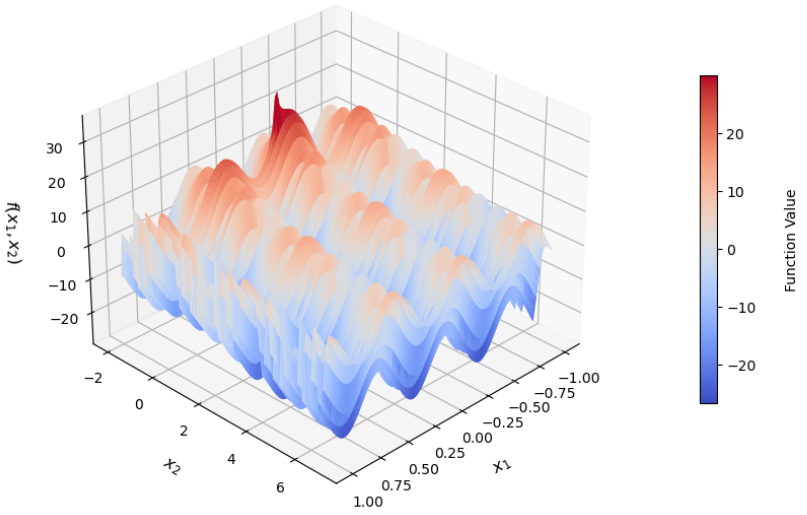


Figure 4.7: Demo function - II

$$f(x_1, x_2) = -2.8x_1^3 + 1.7x_2^2 + 5x_1 + 25\sin(x_2 - x_1) - 2.25\sin(x_1^2)$$

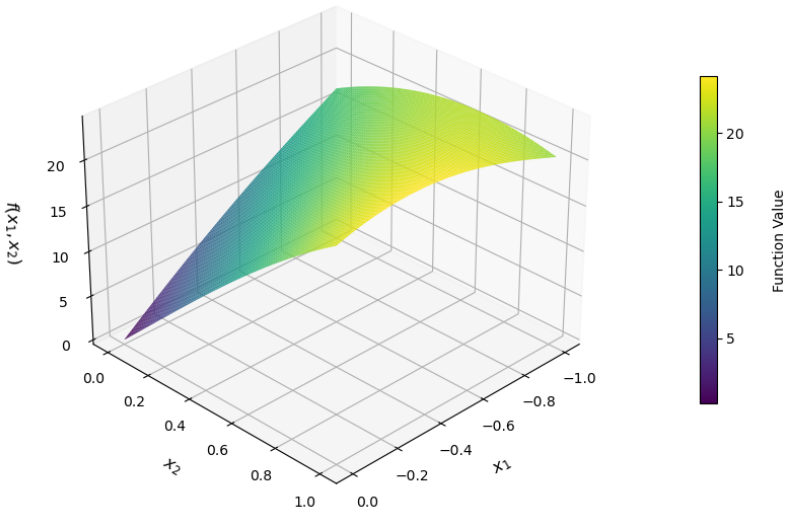
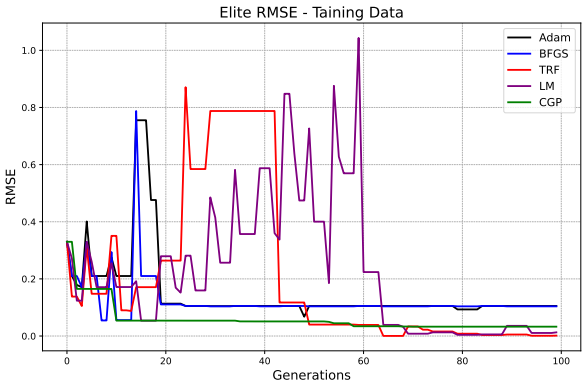
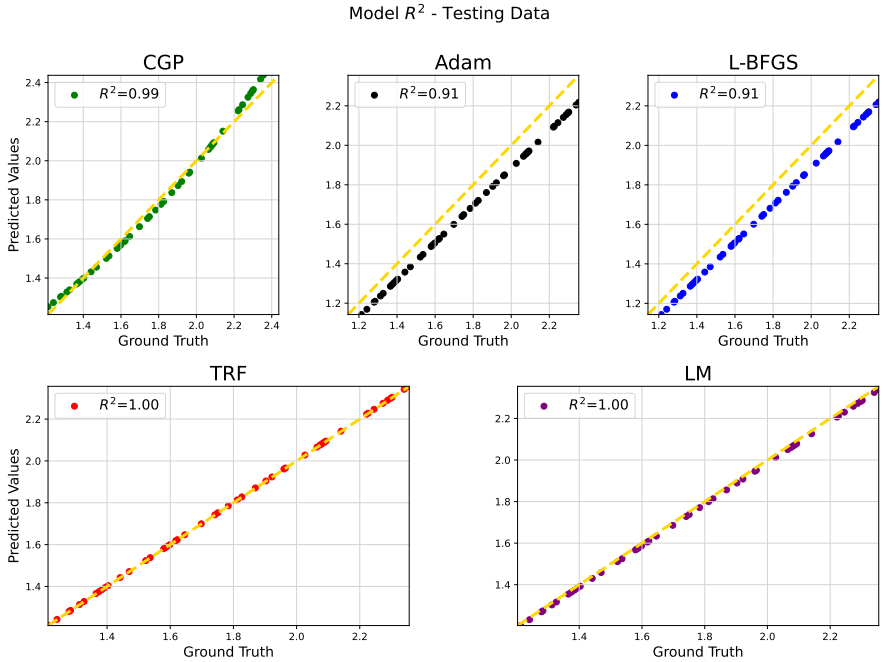


Figure 4.8: Demo function - III

Once again, the developed algorithms were only run a few times against these datasets to have an initial justification to move forward. Unlike terminal optimization, the convergence evolution of these algorithms demonstrated useful trends, providing valuable information on the optimization process. Hence, the testing correlations and error convergence is presented, with the training R^2 presented in Appendix Section B.2. The results are presented below.



(a) RMSE of elite individual

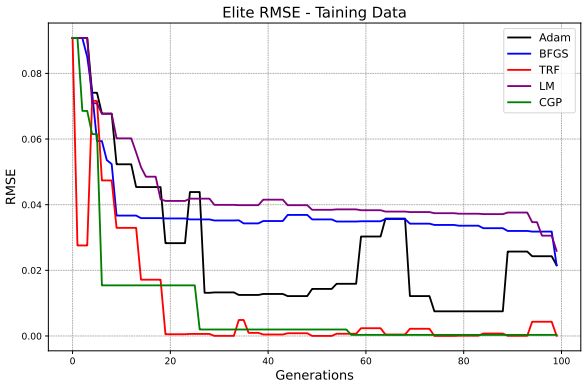


(b) R^2 of elite individual

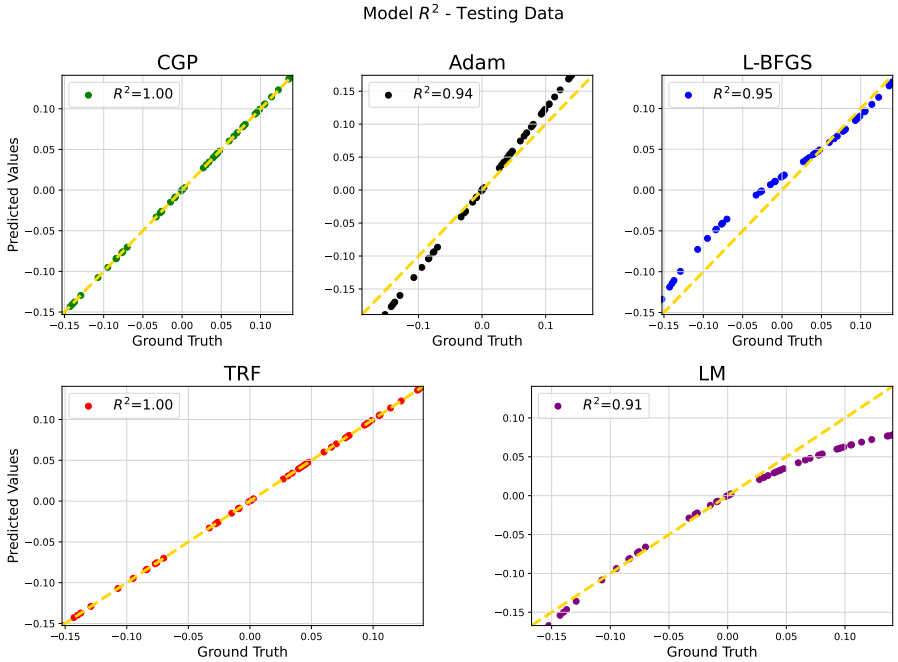
Figure 4.9: Performance metrics (RMSE and R^2) for the elite individual of $f_1(x_1, x_2)$.

The function is $f_1(x_1, x_2)$, and can be referenced from Table 4.4.

Over the course of a few iterations, the presented trend was observed to be consistent. Although the methods exhibit good correlations with testing data, when observing the evolution of these algorithms, we can see that the gradient based algorithms exhibit fluctuations, or jumps, as well, implying that the balance of exploitation and exploration is not very well maintained. Although, the gradient effect is strong and optimizes the solution, the effect of genetic operators often destroys these combinations, resulting in abrupt destruction of fitter individual structures. This makes these algorithms more prone to oscillations and divergence. An early conclusion can be made that the effect of optimizing operators is more impactful and needs to be carefully balanced with the aggressiveness of genetic operators. This is in contrast to when applying only terminal optimization wherein the algorithms were quite stable, and the effect of exploitation was often overshadowed by exploration. In addition, it was observed that $Adam$ often diverged when optimizing an individual, making it unstable and unfavorable at this point.



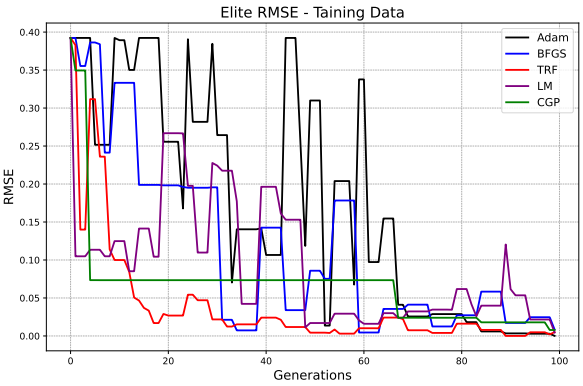
(a) RMSE of elite individual



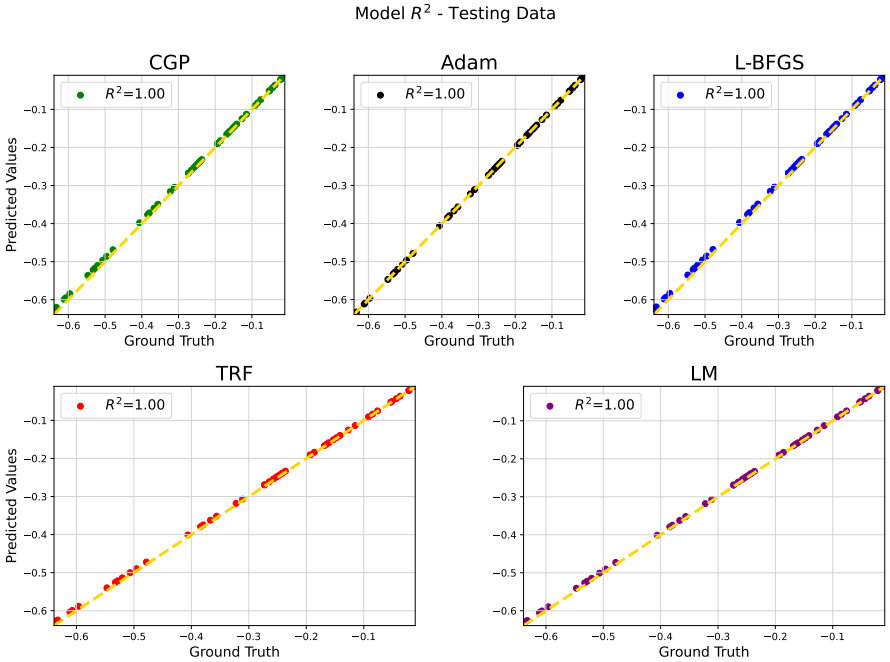
(b) R^2 of elite individual

Figure 4.10: Performance metrics (RMSE and R^2) for the elite individual of $f_2(x_1, x_2)$.

The tested function is $f_2(x_1, x_2)$, and can be referenced from Table 4.4. $Adam$ optimizer exhibits oscillations in this case highlighting its sensitivity. Although TRF converged to the same individual quality as CGP, the convergence was not stable, further supporting our previous argument that the frequency of operator optimization and genetic evolution needs to be carefully balanced.



(a) RMSE of elite individual



(b) R^2 of elite individual

Figure 4.11: Performance metrics (RMSE and R^2) for the elite individual of $f_3(x_1, x_2)$.

The function regressed is $f_3(x_1, x_2)$, and can be referenced from Table 4.4. It can be observed that once again, all gradient based methods exhibited instabilities and only converged to the final model at the last generation. This is an undesirable behavior as the whole motivation of applying this method is to accelerate convergence.

Conclusions from testing with author-defined problems

Although strong conclusions cannot be yet drawn by the results presented from optimizing terminals, and operators individually, due to lack of repeated testing, and not using benchmark datasets yet, certain findings are evident and give us an indication regarding the devised algorithms.

- Both terminal and operator optimization can produce individuals with superior results, lower RMSE and higher R^2 scores, aligning with literature.
- Operator optimization has a tendency to cause instabilities and the algorithm may produce an individual of inferior quality with respect to the evaluated metrics. Hence, a proper balance between the frequency of optimization, number of individuals to optimize and the probabilities of genetic evolution would need to be carefully set.

- Essentially when only terminal optimization was applied, the algorithms could converge to such superior results quickly (lower RMSE and higher correlations). This is a favorable outcome as it implies a potential decrease in the function calls when compared to the conventional implementation.
- Of all the optimizers implemented, $Adam$ can be said to have a higher tendency to diverge or overshoot from the optimal solution. However, this is not surprising. The adaptive learning rate can make it susceptible to noise, and lead to convergence to a local optimum. Furthermore, it has also been stated that

  > Adam can enter a state in which the parameter update vector has a relatively large norm and is essentially uncorrelated with the direction of descent on the training loss landscape, leading to divergence

  [49]. Given the large-scale datasets associated with our problems, this behavior hence, can be expected to occur frequently.

The same settings were implemented, as presented in Table 4.3 and Table 4.2.

With these insights, the study now progresses to deduce firm conclusions about these optimization methods by implementing both terminal and operator optimization together, and deciding on the most effective optimizer.

4.1.2. Benchmark Testing - phase I

3 benchmark symbolic regression problems were used to test the conventional genetic programming and the devised gradient optimized GP. The characteristics of these datasets are tabulated below.

Dataset	# observations	# features	# targets
210_cloud	108	5	1
537_houses	20,640	8	1
344_mv	40,768	10	1

Table 4.5: Symbolic benchmark regression problems

The datasets can be referenced from [54] and were chosen keeping in mind that CFD datasets potentially also contain $\geq$ 10,000 observations with $\geq$ 5 features. Furthermore, 100 iterations were performed, each with a distinct random seed to consider the effect of the inherent random nature of the algorithms and be able to make conclusions with a proper basis.

Furthermore, given our insights from Subsection 4.1.1, the devised algorithms have the potential to reach optimal results in fewer iterations. Hence, the challenge for these methods will be if they can produce superior (or at least the same) results as the conventional methods in half the generations when terminal and operator optimization is combined. Therefore, for the benchmark datasets, while the conventional algorithm was run for a 100 generations (as is implemented by the author's research group currently), the developed algorithms were run for 50 generations. In addition, the optimization was performed at every 2^{nd} and 5^{th} generation for the operator and terminals respectively. The selection of optimal hyper-parameter setting was done in a systematic manner, via benchmark testing and to begin with, all the individuals of a given population were subjected to gradient optimization.

The results of these benchmark regression problems, along with the parameter settings are presented below.

Hyperparameter	Value
pop_size	50
generations - CGP	100
generations - Optimized GP	50
tournament_size	3
n_elites	1
function_set	add, sub, mul, protected_div, protected_nexp, protected_log
rc_ub	5
rc_lb	-5
pb_cx1p	0.3
pb_cx2p	0.2
pb_mut_node_rep	0.05
pb_mut_uniform	0.05
metric	rmse
tolerance	1e-9
optimizing frequency	2^{nd} generation - operators 5^{th} generation - terminals
num_optimizing_ind	Population

Table 4.6: Genetic Programming Hyperparameters - Symbolic Benchmark Regression Datasets

Optimizer parameter settings can be referenced from Table 4.2. Note that there are multiple quantities that can be plotted. However, only the testing correlations and function counts are presented here to demonstrate whether the devised algorithms can meet the objectives. The behavior of remaining quantities can be referenced from Appendix Section B.3

Dataset - 1: 210_cloud
For the above dataset, when $Adam$ was implemented, the solution diverged 16.7% of the times, *1 in every 6 iterations*. Hence, it was decided to not implement the said algorithm further for any optimization process. The diverging tendency could also be observed in the initial (preliminary) analysis with author-defined functions. This reduces the number of optimizers to 3 : $L-BFGS, TRF, \& LM$.

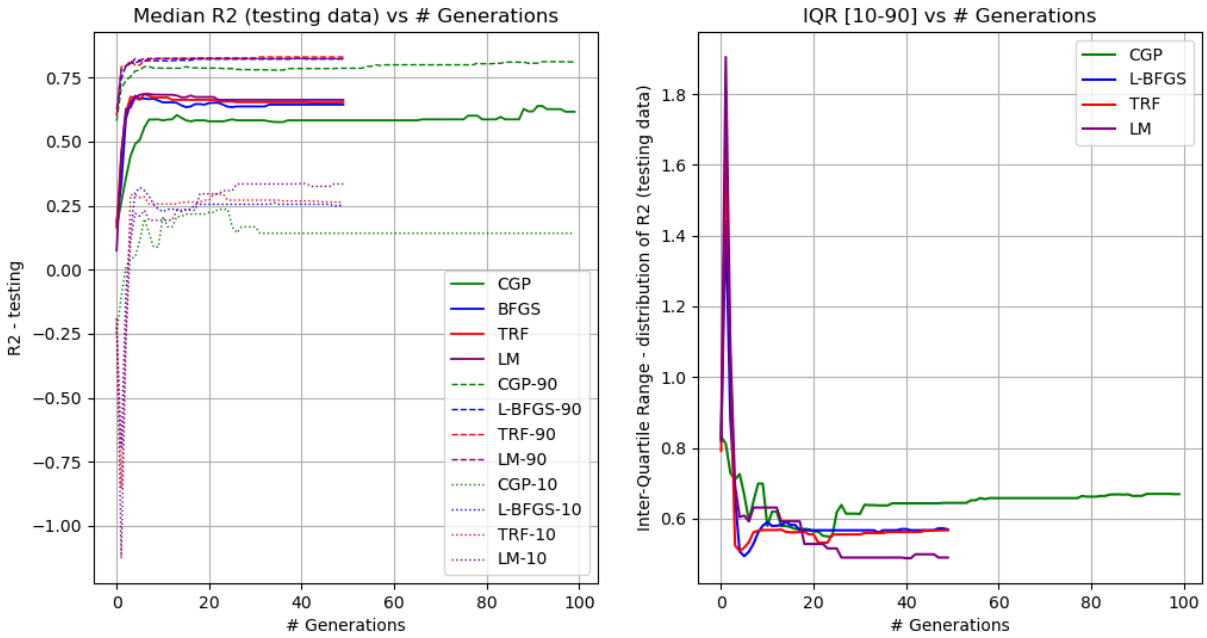


Figure 4.12: Testing R^2: Dataset-1

It is evident that the gradient-based algorithms achieve higher median scores compared to the conventional approach in just half the generations. There is a significant improvement in the testing R^2 values, although the median RMSE and training R^2 scores are superior only by a fine margin as seen from Figure B.8, Figure B.7. This demonstrates the capacity of the algorithms to leverage the abilities of both exploration and exploitation of functional space.

Nonetheless, the 10^{th}, and 90^{th} percentile distributions reveal the effect of exploitation (localization). It can be observed that the conventional genetic program exhibits greater variation in the quality of elite individuals across different random seeds. Although it has the potential to reach lower RMSE values, and higher R^2 scores, at the same time, it is also vulnerable to converging to poorer solutions.

This is also supported from the inter-quartile distribution which directly shows the variation between the top 10% and the bottom 10% of the elite individuals. CGP has higher values and makes it is clear that it has more diversity and potential to explore functional spaces, whereas genetic program coupled with localized gradient optimization can lead to most of the individuals being converged around a certain optimum.

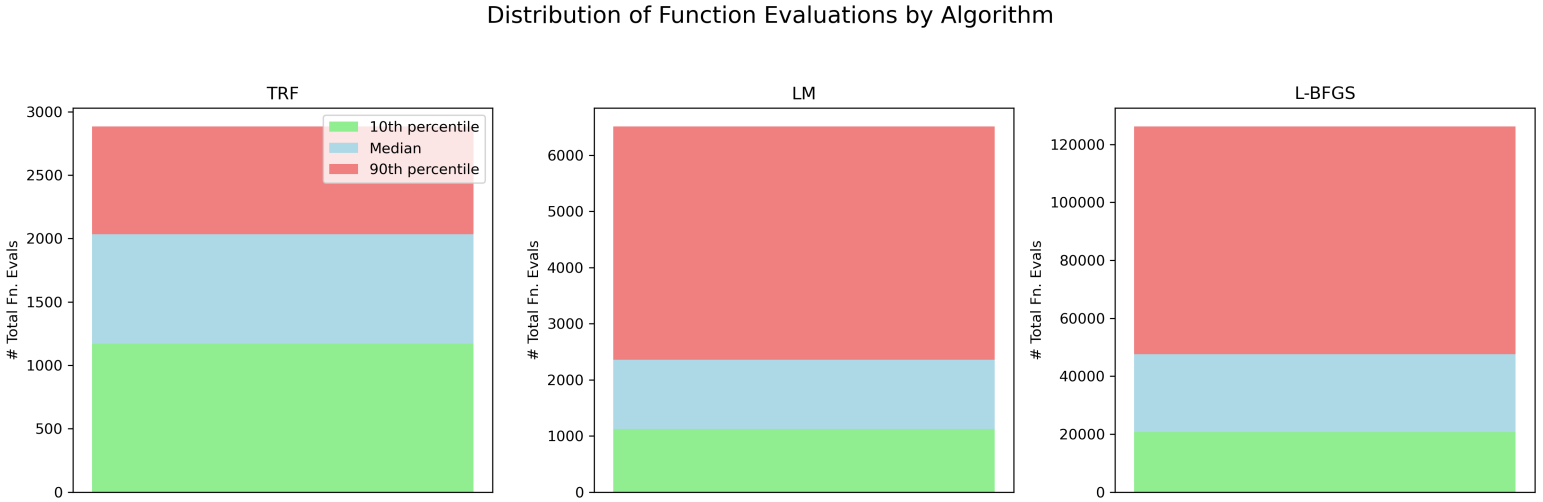


Figure 4.13: Median distribution of function evaluations: Dataset-1

Figure 4.13 is critical as it illustrates the distribution of function count, i.e., the number of times an optimizer calls the objective function to further optimize the loss function, $\mathcal{L}$. This is also the deciding factor for the selection of the optimum algorithm, to be applied to CFD datasets.

It is noticeable how the *TRF* method is computationally the most cheapest algorithm for the given dataset, with most of the iterations requiring the objective function to be evaluated in the vicinity of 2,000 counts. Looking at the same distribution for the *Levenberg-Marquardt* method, most of the iterations required a significantly higher function count. There is a large gap the median distribution and the 90^{th} percentile. This can be explained by the way the methods, *TRF & LM* are implemented. The former acts on a scalar loss function, and minimizes it by a trust region framework, explained in Subsection 3.2.1. It can efficiently balance the step quality and Jacobian computation via Singular Value Decomposition (SVD) and trust region radius. On the contrary, the latter works directly on the residual vector. Although it can potentially lead to faster convergence because of this, however, the gradient (Jacobian) computations can be computationally expensive if the residual vector is large. Hence, for a problem with large number of residuals or parameters, the approximation of Jacobian can invoke multiple function calls, making it computationally expensive.

Moving to the *L-BFGS* method, it is the most expensive algorithm. This can be explained by the fact that it makes use of information from both first and second order derivatives (approximations) which in itself can require multiple function calls. In addition to this, the said method makes use of line search to determine a suitable step size. Thus, the gradient history storage and line searches add to the computational cost, when compared with the *TRF* and the *LM* methods. Although the approximated Hessian can guide the optimization process in which direction to proceed to converge to the global optima, the line search involves evaluating the objective function several times which can add to the computational cost of the algorithm.

It is to be noted that *L-BFGS* diverged, or crashed twice out of the 100 iterations due to very large (complex) expression, thereby giving syntax errors. The remaining algorithms did not report such an issue. Moreover, *LM*, and *TRF* methods had the shortest expressions, and frequently converged to similar individuals, followed by *L-BFGS*, and then *CGP*. This demonstrates another useful ability of the gradient based algorithms - to produce shorter expression length individuals of superior quality.

Dataset - 2: 537_houses
Similar to the previous case, the results for this dataset are presented below.

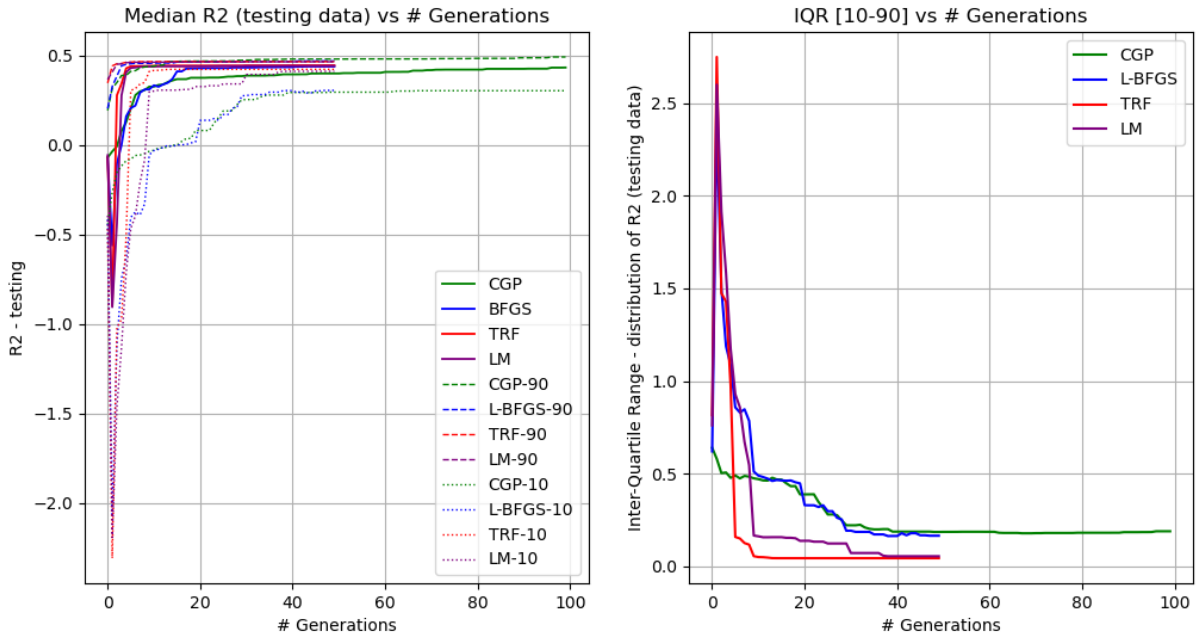


Figure 4.14: Testing R^2: Dataset-2

The above reinforces the effectiveness of gradient-based optimization algorithms in consistently converging to superior optima in fewer generations. This further strengthens the answer to the research question - if genetic programs coupled with the derivative information can achieve quicker convergence, and have acceptable correlations. All three performance metrics — RMSE, training R^2, and testing R^2 — show noticeable improvements with the proposed methods (Figures 4.14, B.11 & B.12). Notably, the *LM* and *TRF* methods exhibit minimal variation in the performance of elite individuals across different random seeds, as reflected in their narrow inter-quartile ranges. This indicates a strong tendency of these methods to rapidly converge toward a common optimal region in the functional space, regardless of initial population diversity. This can be also be correlated to the fact that both *LM*, and *TRF* often converged to similar expressions. Although the *L-BFGS* method demonstrated slightly higher variability, its median performance remains comparable to, or better than, that of the conventional approach.

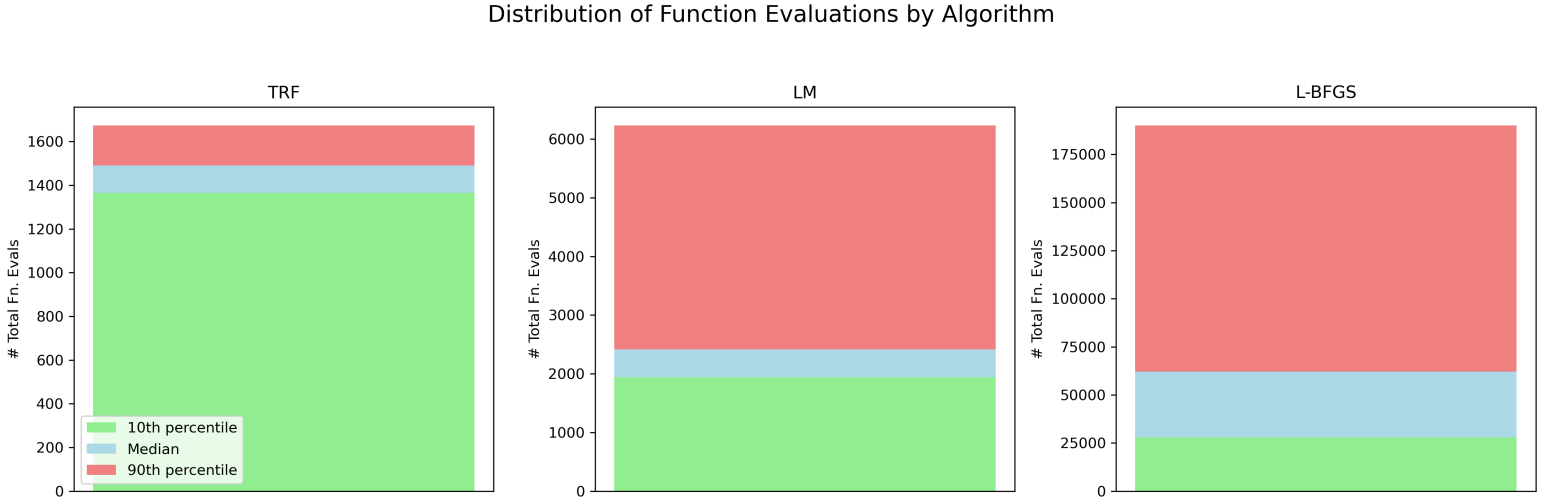


Figure 4.15: Median distribution of function evaluations: Dataset-2

The above figure further substantiates our previous findings that the *TRF* method is the most computationally efficient alternative, in terms of the number of function evaluations. A consistent pattern in the number of function calls are observed to occur across multiple iterations indicating that the algorithm is quite stable in terms of exploiting the functional space effectively. In addition, a very small region beyond the 10^{th} percentile is present, indicating minimal variation in the optimization cycles across the random seeds.

Elsewhere, *LM*, and *L-BFGS* show an increased number of function calls during the optimization in that order. There are instances wherein both the methods require the objective function to be evaluated significantly more number of times. Similar findings were also observed for Dataset-1 (Figure B.9). This results in a higher computational cost and reflecting that the methods may find it difficult, at instances, to optimize the individuals, and maintain a balance between exploration and exploitation. *TRF* on the other hand, is able to maintain this balance, making it a favorable choice, at this point.

With analysis now completed on 2 of the 3 symbolic benchmark datasets, certain conclusions can be drawn and are mentioned below.

- All 3 optimizers can efficiently exploit the functional space when both terminal and operator optimization is performed on the individuals, with the former taking precedence. *Adam* is quite unstable, and oscillating, as also observed in Subsection 4.1.1. Hence, not being considered further.
- The *TRF* method is consistent with regards to the number of times it calls the objective function to optimize over different random seeds for a particular dataset. Thus, it is quick to converge to the optimum irrespective of the initial population.
- Both *LM*, and *TRF* are consistent in converging to an optimum point in the functional space, with minimal inter-quartile values. *L-BFGS*, on the other hand, exhibits some variability in the quality of elite individuals. Thus, the effect of global exploration is much more stronger in *L-BFGS*.
- In terms of computational cost (function evaluations), *TRF* emerges as a robust and a favorable candidate at this point. Another merit of this method is its ability to generate compact expressions. This results in simpler models with good correlation values, which make such expressions more convenient to insert into CFD models.

Dataset - 3: 344_mv
This was the biggest dataset to be tested thus far, and consequently, consumed more computational power than the previous 2 datasets. The results are presented below.

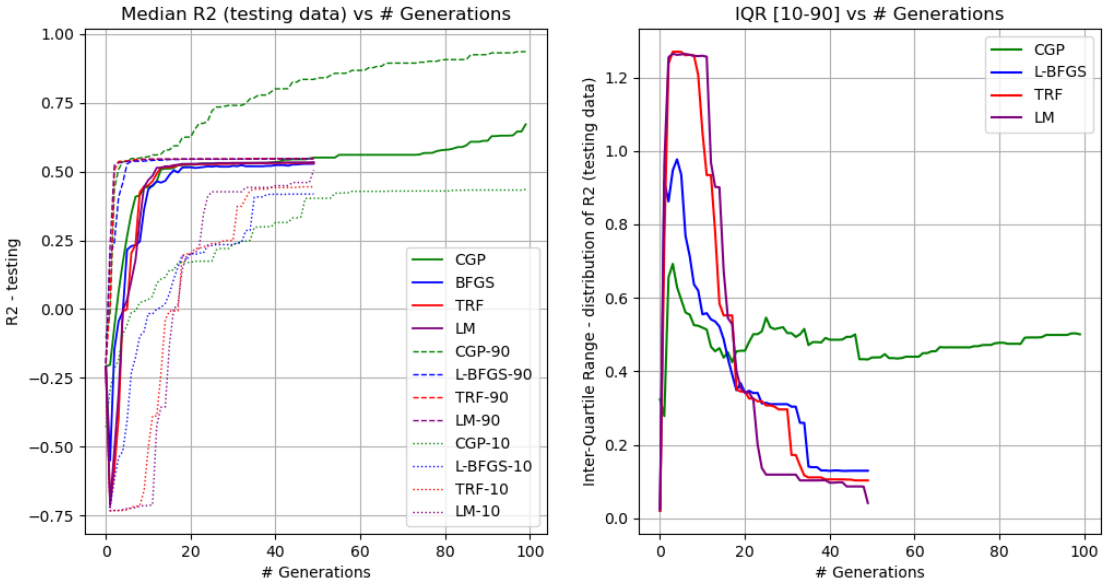


Figure 4.16: Testing R^2: Dataset-3

This dataset distinctly exploits the limitations associated with a gradient based genetic framework. It is clearly visible that the conventional algorithm outperforms all the 3 optimizers. Although the risk of converging to a local optimal individual is slightly higher, as reflected in the high variance of the percentile plots, it also demonstrates a much greater chance of converging to a superior individual, and by a significant margin.

This establishes that the extent of gradient optimization can be far too strong in certain function spaces. Despite having the same probabilities of genetic operations on an individual, the gradient optimization can dominate and suppress the effective exploration of the function space by prematurely guiding the individuals toward a local optimum. Each optimization iteration is strong enough to force the search for the global optimum to be confined within a narrow region of solution space (local optimum), thereby causing difficulties in navigating the optimization landscape, and reducing the robustness of the global search.

Similar to previous findings, the inter-quartile range for the gradient based algorithms remains notably low when compared to that of the conventional algorithm. This additionally emphasizes the lack of diversity among the individuals, in a given generation of a certain random seed.

The crux of implementing a genetic program is its inherent random (stochastic) nature, that enables it to perform a form of random walk over the functional landscape. This capacity gives it the ability to effectively traverse the function space, explore regions with multiple local optima, hence increasing the likelihood of discovering globally optimal solutions. However, it can be observed that when this is coupled with gradient optimization, while beneficial in some function spaces, it can hinder the exploratory capacity. This lack of variability further highlights the tendency of these methods to converge rapidly but locally.

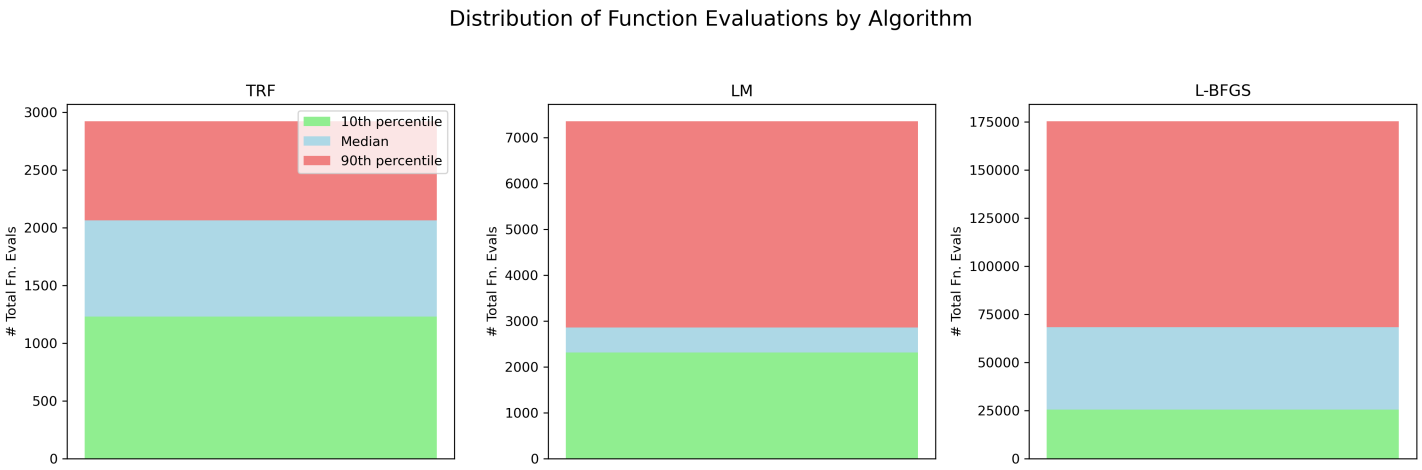


Figure 4.17: Median distribution of function evaluations: Dataset-3

Turning to the function evaluation metrics, findings remain consistent with our previous results. The *TRF* method, is once again able to demonstrate its effectiveness by requiring the least function evaluations when compared to the other methods. We see that most of the random seeds required nearly 2,000 function evaluations, touching 3,000 occasionally. The *Levenberg-Marquardt* method on the other hand had higher function calls with some counts going as high as 8,000. This was then followed by the *L-BFGS* method, which was consistently the most expensive algorithm on all 3 datasets.

This once again indicates the robust nature of *TRF*. Thus, *TRF* can effectively exploit the functional space with minimal computational overhead, even in the presence of random initialization and evolutionary variation. The remaining methods were however susceptible to this, and often required significantly more optimization efforts than the average or the median counts.

Conclusions - Benchmark Testing Phase I

This concludes the first phase of testing the proposed algorithms on the symbolic benchmark regression datasets. With comprehensive analysis on the performance metric and supporting visualizations now completed, definitive conclusions can now be made with regards to the performance of each of these methods. These findings are summarized below.

- A genetic programming (global search) algorithm incorporated with gradient descent optimization (local search) is, in general, able to have an enhanced performance (lower errors, and higher correlations) while accelerating the convergence to a global optimum. It can be said firmly that half the iterations are in fact sufficient to reach the same results, if not superior, attained by the conventional algorithm.
- When comparing the performance of the optimizers, from Subsection 4.1.1, *L-BFGS* appeared to be the most superior, consistently converging quickly to high-quality individuals without any noticeable oscillations, or signs of divergence. However, given the implementation of the algorithm wherein it makes use of an approximated Hessian and continuous line searches, it proves to also be the most expensive algorithm with significantly more function calls compared to other algorithms when tested against repeated runs of benchmark datasets. It also exhibits a relatively large variation in the quality of individuals, with some of them being quite lengthy. This would further make it inconvenient to insert the expressions into the CFD solver with certainty that the solution will not diverge. Hence, although able to give improved results, it does not present itself to be the best candidate.
- The remaining two optimizers, *Levenberg-Marquardt*, and *Trust Region Reflective* show comparable performances, with regards to individual qualities, and function calls. It was observed that in most of the cases, both of them converged to similar expressions which were simpler and compact, with identical variations in the overall distribution of individual quality, and scores. Interestingly, even the CPU times remained alike for them. However, *TRF* was deemed to be more efficient due to its reduced number of function calls, thus emerging as the favorable optimizer.

- Furthermore, for the same population, it can be observed that *LM* may require significantly more function calls for convergence. This highlights a potential issue of improper initial guesses, or the need for a proper balancing of the damping parameter, β. Although one would argue that performing increased function evaluations within the same time frame as *TRF* makes it more smarter and efficient, when these function calls are replaced by a CFD run in *a posteriori* setting, it would make it inefficient.
- There is a risk that the gradient based optimization can be too powerful and it dominates the global search for a global optimum. The gradient based optimization is only used for refining the individuals by locally exploiting the search space. However, as observed with Dataset-3, it was found that it was guiding the search process itself. This can be harmful as it would imply a premature convergence to a sub-optimal optima, with genetic evolution not able to perform its role. This needs to be addressed, and a distinction needs to be made between the two different search methods. These methods are explained below.
  - One way to mitigate this is by reducing the number of individuals that undergo gradient based optimization. A selected group of individuals can be chosen, as done in Subsection 4.1.1. This would additionally, reduce the number of function evaluations as well. Hence, potentially providing a solution to an issue, and improving the performance.
  - Another alternate is to increase the mutation and crossover rates, or their strengths. This would make the occurrence of genetic operations more frequent, allowing for effective exploration of the search space.
  - A third method is to further reduce the frequency of optimizing the individuals to potentially accumulate more randomness in the individual before subjecting them to a strong optimization cycle.
  - An additional possibility is to limit the number of function calls or the optimization iterations performed. Although this can reduce the extent of optimization, it can however, lead to convergence issues.

With an optimizer now chosen and these conclusions in hindsight, the *TRF* method is further tested against the benchmark regression datasets, with the implementation of these modifications, to determine if such adjustment can indeed enhance the performance of it. The next phase of testing to establish the optimum conditions for implementing *TRF*, referred to as Phase-2, commences below.

4.1.3. Benchmark Testing - phase II

Multiple runs of symbolic benchmark datasets were implemented to ensure the reliability of the quantitative results describing the overall performance of these algorithms. The datasets are presented below, and were referenced from [54].

Dataset	# observations	# features	# targets
210_cloud	108	5	1
344_mv	40,768	10	1

Table 4.7: Symbolic benchmark regression problems - *contd.*

The modifications done with the hyper-parameters of the algorithm are tabulated below.

Hyperparameter	Value
pop_size	50
generations - CGP	100
generations - Optimized GP	50
tournament_size	3
n_elites	1
function_set	add, sub, mul, protected_div, protected_nexp, protected_log
rc_ub	5
rc_lb	-5
pb_cx1p	0.5
pb_cx2p	0.3
pb_mut_node_rep	0.2
pb_mut_uniform	0.5
metric	rmse
tolerance	1e-9
optimizing frequency	3^{rd} generation - operators 5^{th} generation - terminals
num_optimizing_ind	Initially, select 50% of population based on tournament selection followed by further selecting 40% of individuals from this subset via random sampling

Table 4.8: Genetic Programming Hyperparameters - Symbolic Benchmark Regression Datasets:2

Optimizer	Parameter Settings
Trust Region	
g_tol	1×10^{-4}
f_tol	1×10^{-4}
x_tol	1×10^{-4}
loss	huber
max_nfev	40

Table 4.9: Optimizer settings for optimization

It can be seen that the probabilities of the genetic operations have been increased to allow for frequent occurrence, thereby mitigating the effect of strong local optimization. Note that for the conventional algorithm, the same probabilities were implemented as mentioned in Table 4.6. This modification is done to only enhance the performance of the optimization. In addition, the number of function evaluations have also been limited. Moreover, now instead of optimizing every individual in a given generation, a new selection mechanism was devised to keep a balance between randomness, and optimization. With these implemented changes, the results were gathered and are presented below.

Dataset - 1: 210_cloud
The results are visualized below.

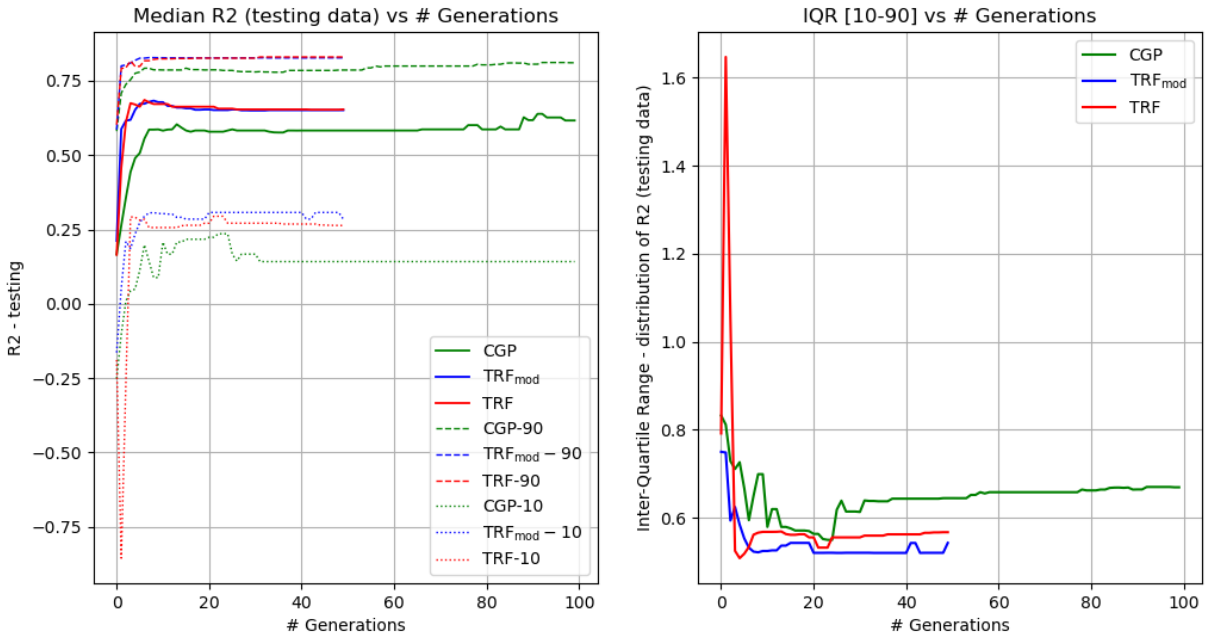


Figure 4.18: Median Testing R^2: Dataset-1

From the above, improvements can be observed in the behavior of the *TRF* algorithm. However, they are very slight and the performance does not enhance significantly. The $RMSE$ decreases marginally and when considering the testing R^2 scores, we can see that the modified algorithm has an improved probability to produce less optimum individuals. This signals some improvement towards our goals, but also requires further analysis of other metrics.

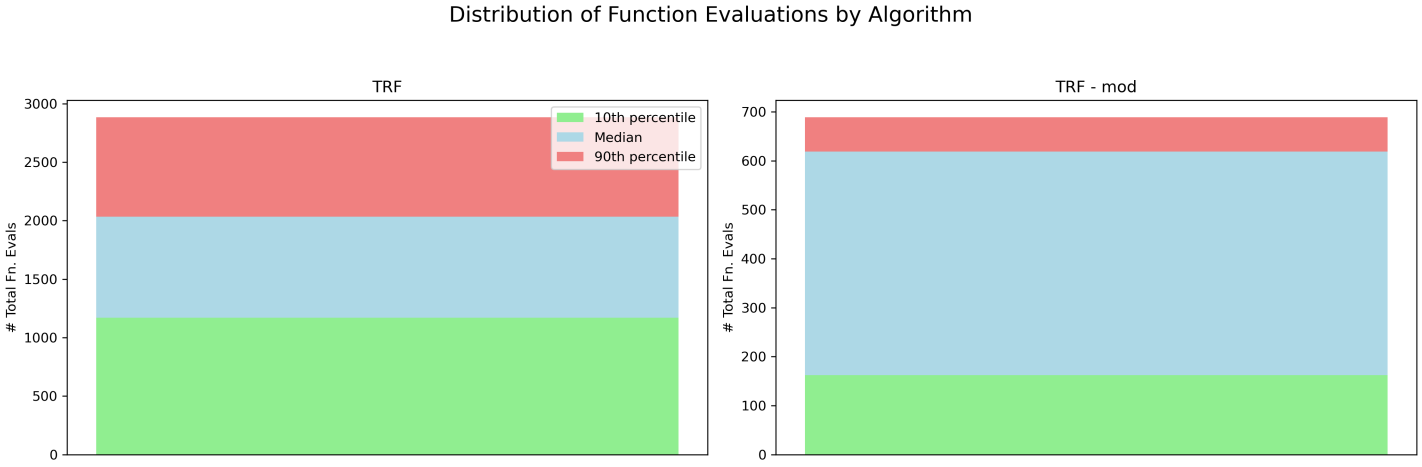
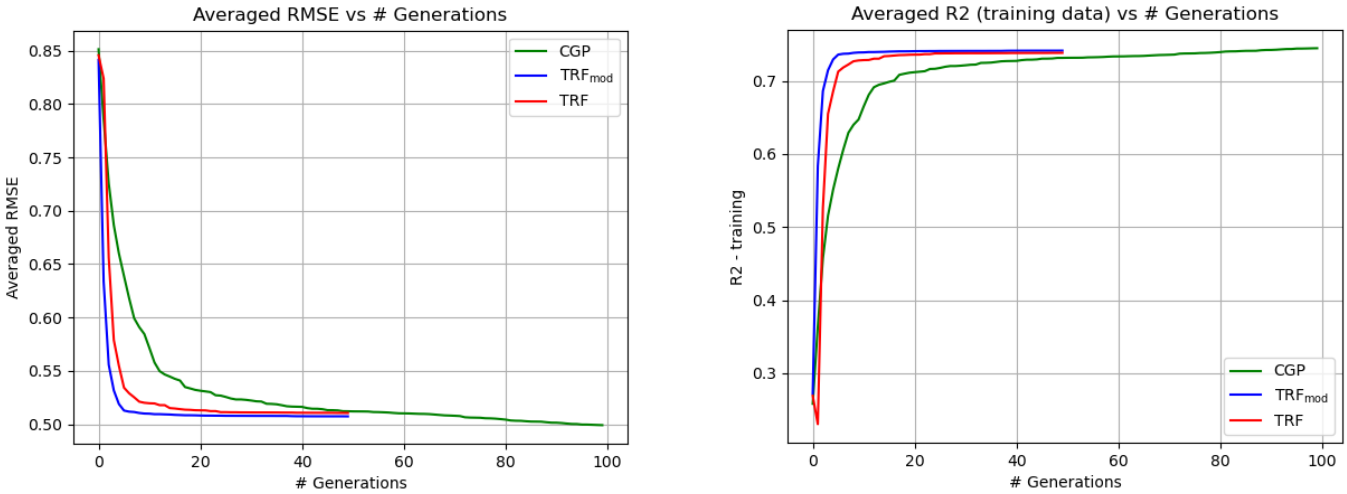


Figure 4.19: Median distribution of function evaluations: Dataset-1

The above imply significant performance in terms of function calls. It can be seen that the total number of function evaluations required for optimization have reduced by roughly 75%, which is a substantial improvement over the current optimizer settings, even if the results remain the same. This implies that even with just 25% of the total function calls, we can essentially converge to the same results without any convergence issues, when compared to the original implementation of the optimizer. It can be said that majority of the random seed iterations required the function to be evaluated in the

range of $\approx [200,\ 600]$ with very few iterations exceeding this range and is mainly due to less number of individuals now undergoing the optimization process coupled with the reduced optimization frequency. This is a key metric to determine the effectiveness of the proposed algorithm, and the modifications look promising.



(a) Averaged RMSE

(b) Averaged Training R^2

Figure 4.20: Comparison of RMSE and Training R^2: Dataset-1

To better understand the performance enhancement, the averaged plots of metrics are plotted to be able to obtain more information. Interestingly, it can be seen that the *TRF* optimizer, with the modifications, has a further acceleration in its convergence. It is able to obtain identical training R^2 scores as the conventional method, but more quickly as compared to the already implemented *TRF* optimizer. Nonetheless, the improvement on the metric values remains rather small. It can be seen that although there is significant improvement in the reduction of function calls, the overall results remain similar. Thus, not proving to be effective in terms of the quality of the final individual obtained. However, given one more dataset, it remains to be seen whether these modifications are robust and can cause further improvements.

Dataset - 2: 344_mv

Similar to the previous case, the results for this dataset are presented below.

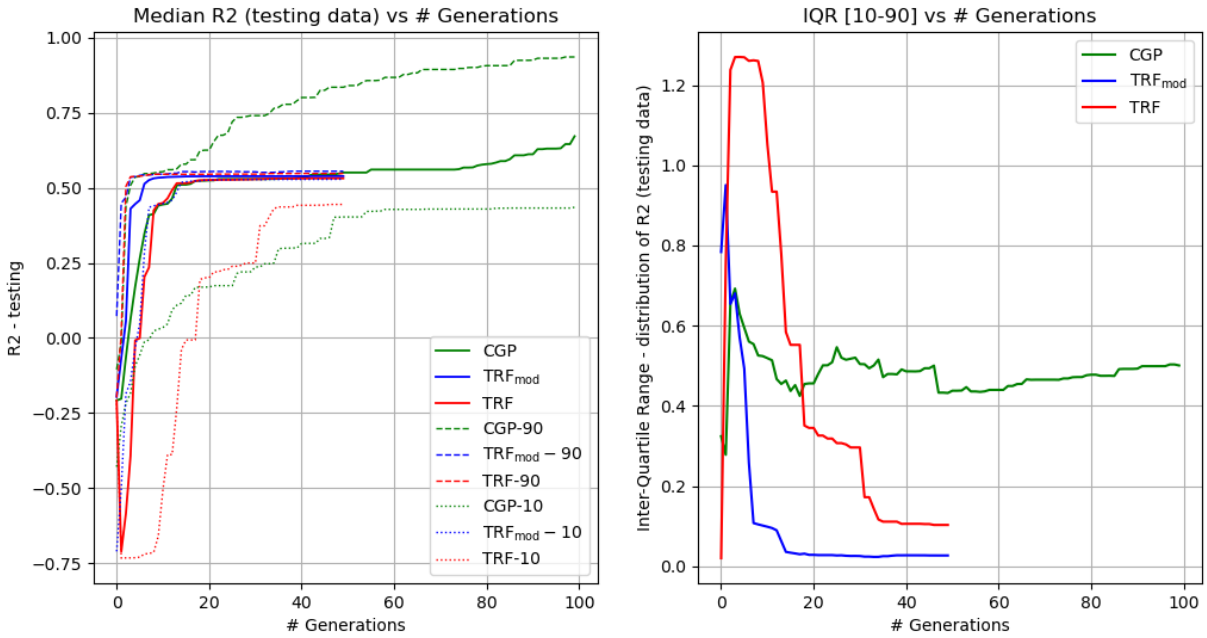


Figure 4.21: Median Testing R^2: Dataset-2

Unfortunately, as can be seen, there are no notable improvements in the results obtained from the optimizer with the modifications. The conventional method still maintains its superiority in this dataset, and is able to achieve far superior results. It can be said that the effect of local optimization is still far too strong, and the increased probabilities of genetic operations are still not able to inculcate, into the algorithm, the necessary randomness required to explore the functional space effectively. Despite the implementation of modifications, the behavior of the optimized algorithm, in terms of RMSE and R^2 metrics, remains consistent to when there was no modifications. It shows us that the optimized algorithms still largely have similar solutions (elite individuals).

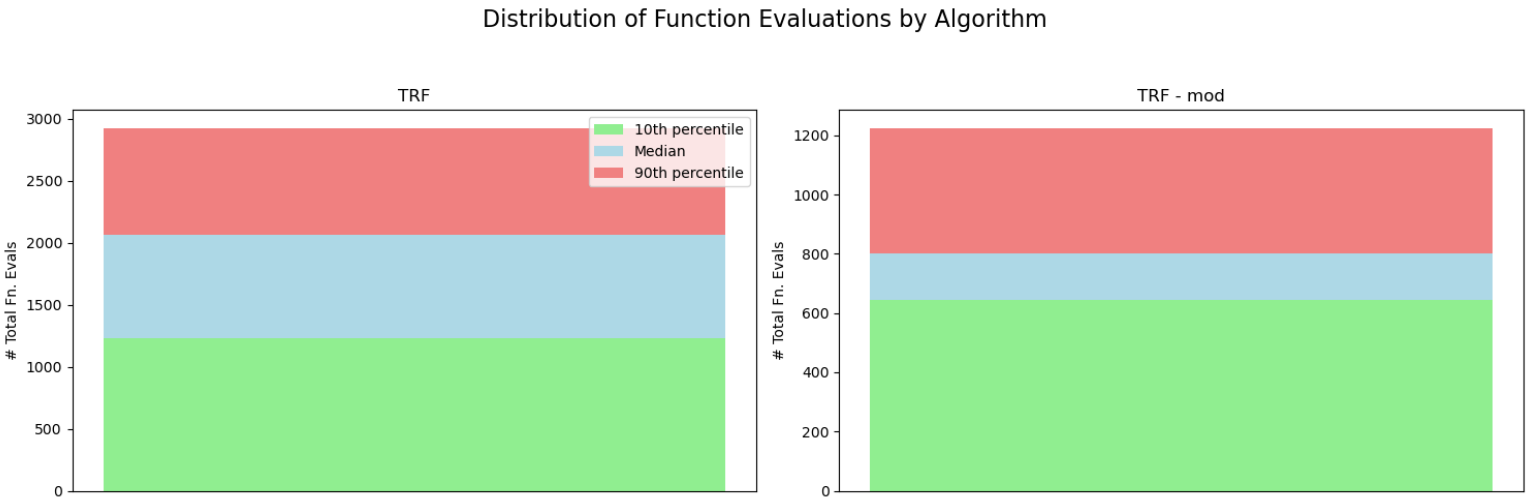
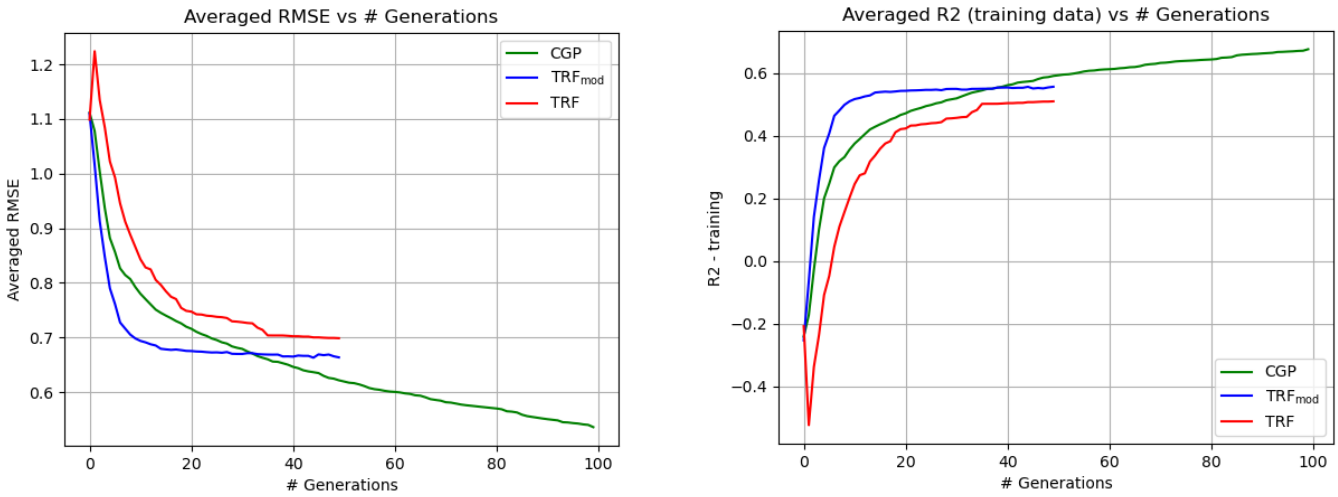


Figure 4.22: Median distribution of function evaluations: Dataset-1

Despite the inability of the modified *TRF* to converge to optimal results, another consistent trend observed was the reduced number of times the objective function was evaluated. The total function calls

reduced by more than 50%, which in itself is a very positive takeaway.

This forces us to consider that these algorithms may potentially be effective in the long run with some further modifications, if required, to tackle the issue of local optimum convergence, as the number of function calls have reduced by an order of magnitude. Nonetheless, these are only benchmark datasets, and not real CFD cases, hence, these findings cannot be extrapolated to CFD datasets with surety. Even so, the behavior makes the author to consider such an algorithm as a promising choice.



(a) Averaged RMSE

(b) Averaged Training R^2

Figure 4.23: Comparison of RMSE and Training R^2: Dataset-2

From the above visualization, significant improvements can be observed in the scores obtained by the modified *TRF*, unlike with the previous dataset. The modified algorithm is now able to illustrate to us that on average, it is able to converge to a superior optimum, although still not the global one. The R^2 values are also improved for both training and testing data, with improved correlation from the scatter plot observed.

This shows us that even though the modified *TRF* is not able to achieve a very significant improvement and match the superiority of the conventional algorithm, it is still able to outperform the optimizer that was initially implemented. It can be inferred that the average performance indicates that there is a broader distribution of improved solutions, even though the central value remains largely similar. Consequently, this implies that the variance in the converged solution quality increases, or a frequent presence of superior solutions that push the algorithm to converge to a global optimum, thereby also affecting the average distribution.

Conclusions - Benchmark Testing Phase II

This marks the end of the second phase of testing which focused on evaluating the performance of both the original implementation of the optimizer, and its modified version to assess any potential improvements. Through a comprehensive analysis on the performance metric and the use of quantitative visualizations, conclusive insights can now be drawn with regarding the effectiveness of these modifications. These findings are summarized below

- Altogether, from the analysis of the 2 symbolic benchmark datasets, it can be inferred that improvements are obtained. The enhancement in the average performance, and the slight improvement in the median scores make it clear that the modified *TRF* is able to alleviate the issues that were highlighted in the previous testing phase, with 3 symbolic benchmark problems.
- However, what is of prime importance is the ability of the modified *TRF* to be able to converge to the same results, if not improved, when compared to the original implementation by calling the objective function a significantly reduced number of times. This directly translates to reduced

calls to the CFD solver in the *a posteriori* setting. Furthermore, with the benchmark datasets, this reduction is more than two-folds. This makes the modified *TRF* as the ideal replacement to the original optimizer.

- In addition, it can be deduced that the average performance suggests a wider range of improved solutions, despite the central tendency remaining largely unchanged. This indicates an increased spread of the data, with a more frequent occurrence of superior solutions that guide the algorithm towards convergence at a global optimum, consequently influencing the average distribution.
- It can be said that the impact of the modifications have not been strong and does not make the proposed algorithm to directly rival the performance of the conventional method, nevertheless, it is still able to deliver improved results with quicker convergence as well, when compared to the initial implementation of *TRF*.
- It can be seen that with the 2^{nd} dataset, the gradient based algorithm struggles to converge to the global optimum, and settled for a sub-optimal value. This implies that the effect of exploitation is still dominant over the exploration of functional space. The reduction in the number of individuals undergoing optimization and increment in the probabilities of genetic operations have helped mitigate this problem, however, the issue still remains. However, it is also noted that out of the 3 tested benchmark datasets, the proposed algorithm struggles against this particular dataset only (*344_mv*), which is not enough to deem the algorithm as being completely ineffective. On the remaining 2 datasets, the proposed algorithm was able to converge to at least slightly better results in comparison to the conventional method, if not greatly superior.
- Given the large number of parameters to potentially manipulate aside from the ones already done, the author does not consider it to be effective to repeat this symbolic benchmark testing by further altering these parameters and determine the optimum settings for each one. This is because it is clearly visible that even if the proposed algorithm struggles against one dataset, it can at the same time, converge to much improved results on other datasets with the same parameters.
- Therefore, moving forward, the current settings will be implemented, as presented in Table 4.8, unless seen as necessary to change.

This completes the testing process of genetic programming, making the algorithm implementation-ready on CFD datasets. Further changes will only be implemented if found suitable, and after a thorough analysis.

4.2. Preliminary Testing of Kolmogorov Arnold Networks

The present section will deal with the early testing of Kolmogorov-Arnold Networks before moving to the CFD datasets. KANs are already well defined and established and been applied to model different relationships for a variety of regression problems, as also seen in Subsection 2.8.3. That said, the initial testing phase of KANs will, hence, not involve evaluating the algorithm against symbolic benchmark problems to establish its performance and ability to capture complex relations between variables. The already presented literature is enough to convince the author of its effectiveness. Therefore, the initial testing will be only to understand the working of KANs, the effect of network architecture, and determine if there can be any loss of information when generating the symbolic formulae.

The examples considered for the preliminary testing, also defined as *supervised toy datasets* in the work that describes the implementation of KANs [45], and are presented below.

Function	Expression	Domain
$f_1(x_1, x_2)$	$\exp(\sin(\pi x_1) + x_2^2$	$x_1 \in [-1, 1], \quad x_2 \in [-1, 1]$
$f_2(x_1, x_2)$	$x_1 x_2$	$x_1 \in [-1, 1], \quad x_2 \in [-2, 7]$
$f_3(x_1, x_2, x_3, x_4)$	$\sqrt{(x_1 - x_2)^2 + (x_3 - x_4)^2}$	$x_1 \in [-1, 0], \quad x_2 \in [1, 2], \quad x_3 \in [3, 4], \quad x_4 \in [5, 6]$

Table 4.10: Functions and Their Domains - KANs

The functional space of the datasets is visualized below.

$$f(x_1, x_2) = \exp(\sin(\pi x) + y^2)$$

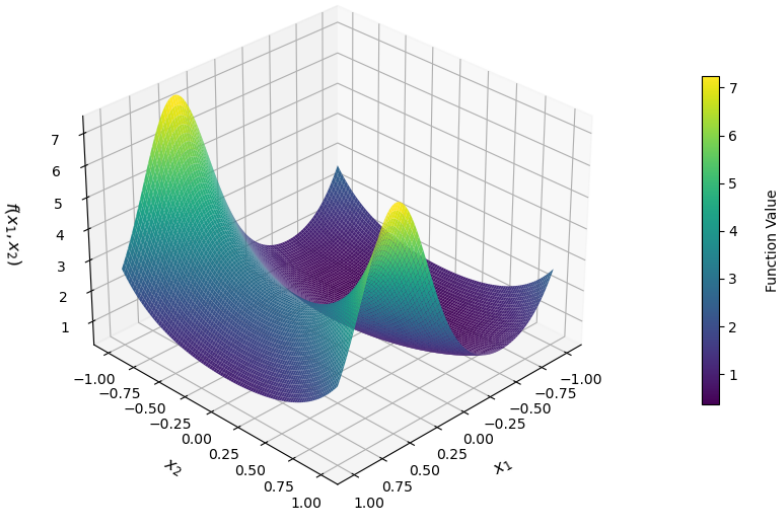


Figure 4.24: Demo function - I

$$f(x, y) = xy$$

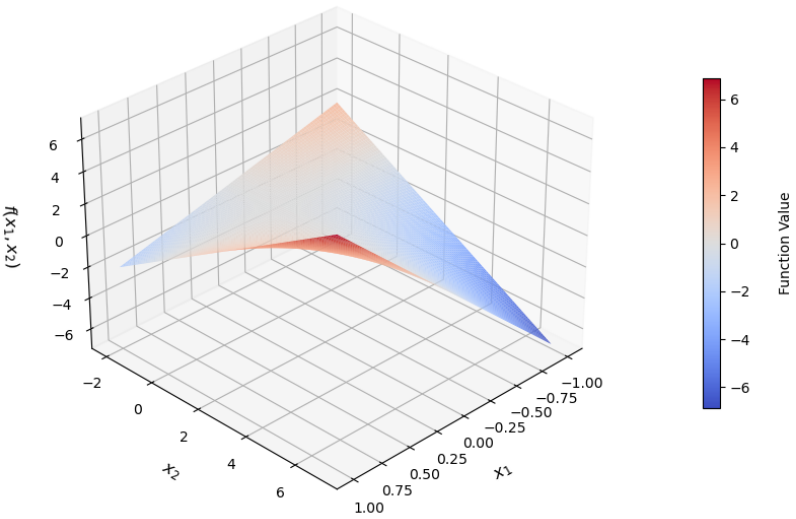


Figure 4.25: Demo function - II

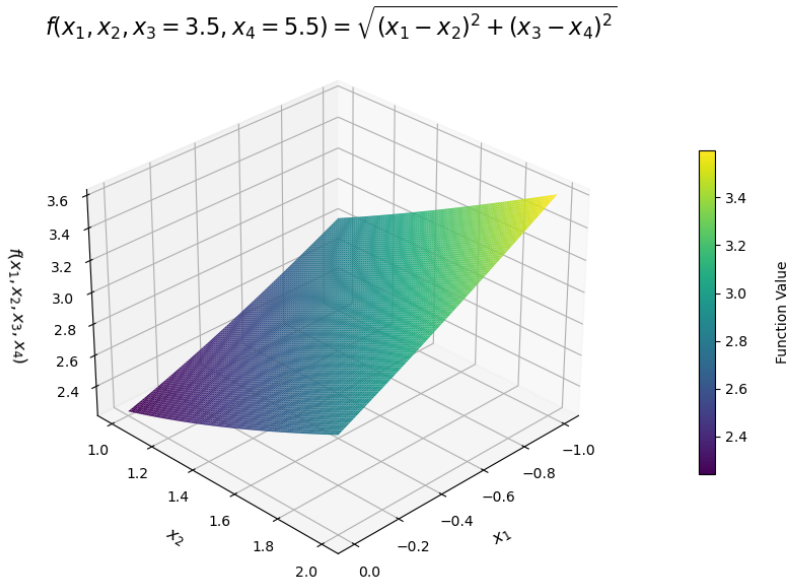


Figure 4.26: Demo function - III

The library of functions that is given the model to generate the symbolic formula is also presented below.

$$\mathcal{L} = \{x,\ x^2,\ x^3,\ x^4,\ \exp(x),\ \log(x),\ \sqrt{x},\ \tanh(x),\ \sin(x)\}$$

The pruning of inputs is not applied to these cases, unless stated otherwise. Nonetheless, pruning of nodes and edges is performed here. Moreover, although the functions are taken directly from the publishers of KANs, the functional spaces are not. In the original implementation, all the input variables were defined within the range [-1, 1]. However, keeping in mind the distribution of our CFD datasets and to provide some variability in the input variables, the domains were redefined.

The parameters of the optimizers can be referenced from Appendix Section B.5.1. The model's convergence history in terms of $RMSE$ and R^2 is plotted, along with a scatter plot is against the testing data. The final symbolic formula is also written, along with the R^2 scores.

4.2.1. Function 1

`[2, 5, 1]` Network

For the given function, a `[2, 5, 1]` KAN network is adopted. Later, it is pruned to remove any redundant neurons or edges. The results are presented below.

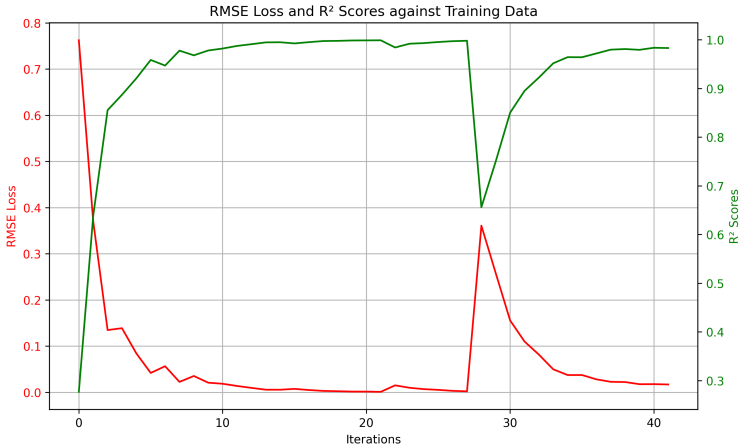


Figure 4.27: RMSE and R^2 - Function 1: `[2, 5, 1]`

We can see that the model is quickly able to reduce its $RMSE$ and get to excellent R^2 scores in $\approx$ 10 iterations, and then converges. This shows the ability of the model to quickly learn the activation functions and converge to a model with good quality. Even after pruning, which occurs at the 15^{th} iteration, the model retains its quality. This indeed shows us that the removal of edges and neurons that do not contribute to the overall model behavior does not cause any instability issues.

However, it can be seen that at $\approx$ 28 iterations, there is a significant decrease in the overall quality of the model. The $RMSE$, and the R^2 values deteriorate significantly by over 30%. This is the part where the learnt activation functions are now converted to a symbolic form via the library. It is to note that not all numerically active activations may be symbolically active as well. In other words, there may be some activations that cannot be represented or modeled via the given library. As a result, these activations become symbolically inactive, and do not contribute further to the final model. Furthermore, this loss can also be explained as the library is only intended to model the activation functions, and not act as a true representation of them. This is a form of approximation that also causes the loss of information when obtaining the symbolic formula. However, when this symbolic is further trained, the KAN parameters are optimized to achieve excellent scores again, and reach to a similar state after pruning,. This makes training after obtaining the symbolic formula necessary as well.

In addition to this, depending on the type of functions learnt, and the library given, this drop in the model may or not be substantial. Nonetheless, we can observe that towards the end, the model is able to converge to very good scores in only a few iterations, highlighting the potential of KANs to model different relationships quickly.

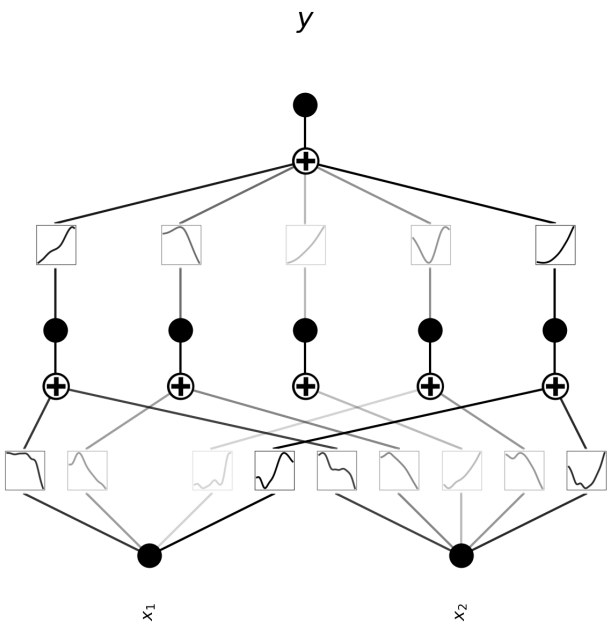


Figure 4.28: Pruned network - Function 1: `[2, 5, 1]`

From the pruned network, it can be seen that although not many, one connection is absent from the first neuron in the first layer (1^{st} input) to the third neuron in the second layer. The edges are also colored differently. This indicates the overall contribution of that edge to the overall model. If a learnt activation function contributes significantly to the overall quality of the model, it is ”black”, and fades as its importance diminishes. In a way, it can be though that if the pruning threshold was higher, the light colored edges would be the first to be pruned away. The calculation of the edge scores has been explained in Section 3.3.

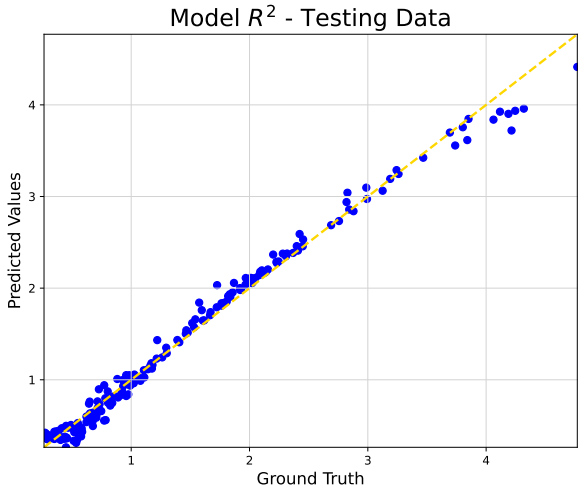


Figure 4.29: Scatter plot - Function 1: `[2, 5, 1]`

Table 4.11: Model Performance: R^2 Scores

Dataset	R^2 Score
Training	0.967
Testing	0.955

The above plot indicates the performance of the final model and can be observed that it behaves quite well and does not exhibit any overfitting issues. The final R^2 values for both training and testing are excellent and the model maintains its consistency.

The final symbolic formula can be referenced from Appendix Section B.5.2.

`[2, 5, 5, 1]` **Network**
To study the effect of depth, a `[2, 5, 5, 1]` layered network is also implemented, with the same settings. The results are presented below.

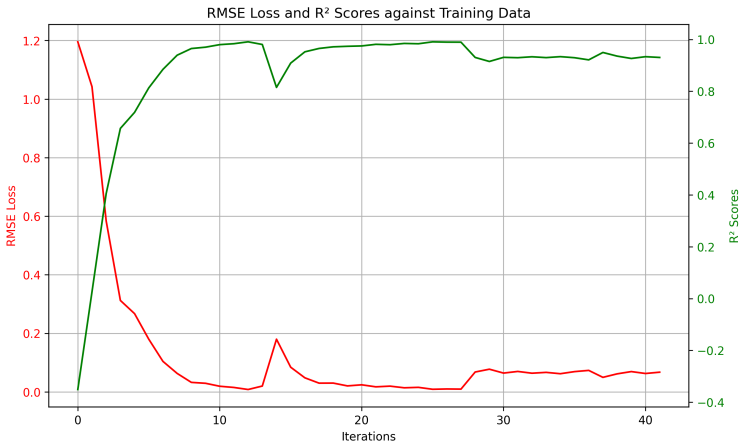


Figure 4.30: RMSE and R^2 - Function 1: `[2, 5, 5, 1]`

It can be clearly seen that with a deeper model, the model does not experience the drastic drop in quality occurring at the 28^{th} iteration. With only slight slumps at a few instances, the model is able to

retain its good quality throughout the training process, without any fluctuations. Thus, it can be said that a deeper network may be inherently more stable.

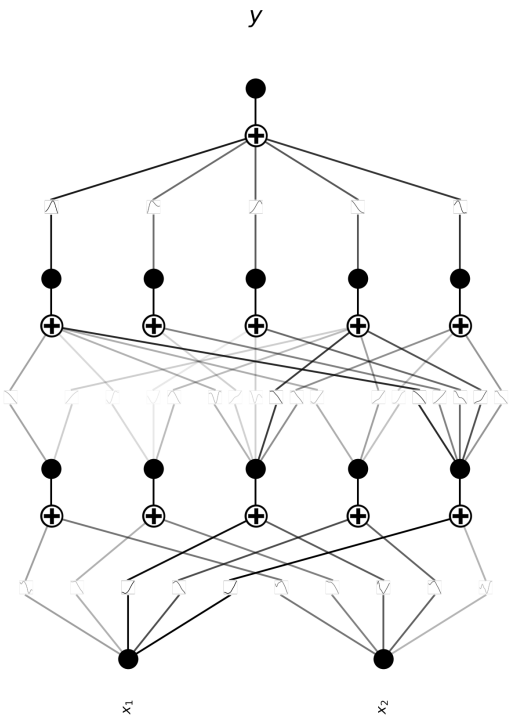


Figure 4.31: Pruned network - Function 1: `[2, 5, 5, 1]`

Unlike the previous case, it can be noted that none of the connections from the first layer are pruned. Rather, only the second layer experiences some removals, such as the first neuron in the second layer which only has 2 edges in what would be 5 edges representing 5 activation functions otherwise. Hence, the behavior of the model completely changes, with the current model appearing to be more complex. This is also expected as now it has to learn more activation functions.

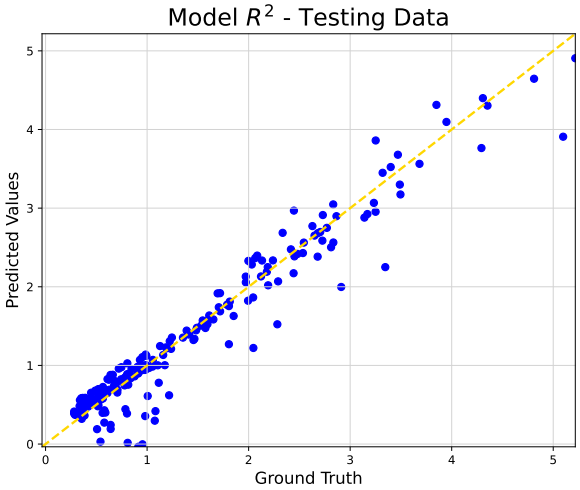


Figure 4.32: Scatter plot - Function 1: `[2, 5, 5, 1]`

Table 4.12: Model Performance: R^2 Scores

Dataset	R^2 Score
Training	0.935
Testing	0.928

These plots and values are more insightful. Although a deeper network can be expected to model even more complex relations appropriately, the converse is observed here. Aside from the model retaining more edges, and consequently more activation functions, it does not prove to be beneficial as the R^2 values drop by quite some margin when compared to a simpler network. This is also in line and reported by the publishers of KANs wherein when the number of parameters in the KANs model increased, the model accuracy dropped as well. Hence, not always can a deeper architecture model achieve improvements over a shorter version of it.

The symbolic formula can be referenced from Appendix Section B.5.2 and it can be seen that the resulting symbolic equation is severely more complex.

From the above dataset, it can be said that a deeper network is not favorable at all times. Although exhibiting more stability and robustness, it not only results in more unknowns, but produces a significantly complex model without further improvements in the R^2 values as well.

4.2.2. Function 2

`[2, 5, 1]` **Network**

The findings are visualized below, in the same manner as for the previous case.

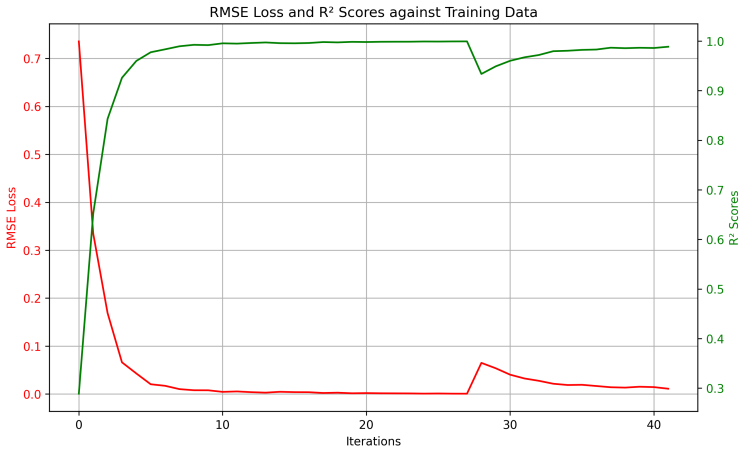


Figure 4.33: RMSE and R^2 - Function 2: `[2, 5, 1]`

Identical to the previous case with a 1-layered network, a dip in the model performance can be noted at the 28^{th} iteration, when the model undergoes training after assigning the symbolic functions to the activations. However, for this function, the model appears to be more robust as the drop in the performance is not very substantial, and quickly converges to the optimal results it was at, prior to assigning symbolic functions.

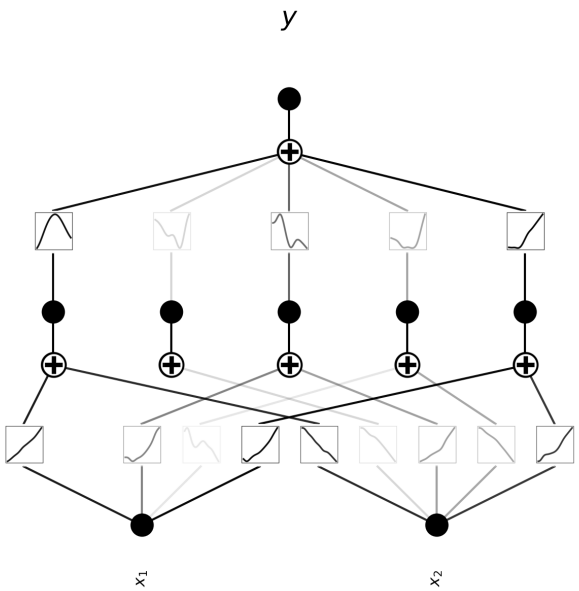


Figure 4.34: Pruned network - Function 2: `[2, 5, 1]`

A very similar model can be observed as to the previous dataset. One of the connection from the first neuron is removed, while others maintain their presence. In addition, the model has a mix of black and faded edges, implying the presence of both of highly critical, and less significant activation functions in the predictions made by the final model.

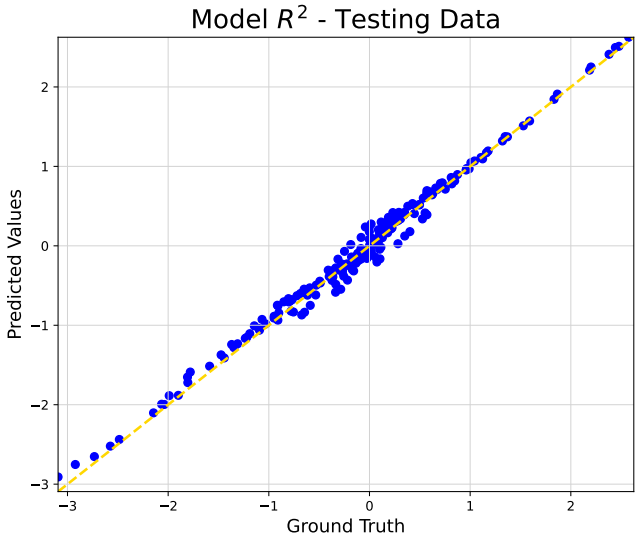


Figure 4.35: Scatter plot - Function 2: `[2, 5, 1]`

Table 4.13: Model Performance: R^2 Scores

Dataset	R^2 **Score**
Training	0.998
Testing	0.987

We can see that the model behaves excellently with both training and testing data. With a nearly linear plot on testing data, and getting an almost perfect training R^2 of 1, it can be said that the model with this architecture is easily able to regress for the formula of Function - 2. Perhaps it can be said that the function is trivial, nevertheless, it is still impressive that the model is able to attain such high scores on a dataset with 800 training points, and only one hidden layer.

The symbolic formula can be referenced from Appendix Section B.5.3.

`[2, 5, 5, 1]` Network

To analyze the impact of network depth as done in the previous case, a `[2, 5, 5, 1]` architecture was also tested. The results are shown below.

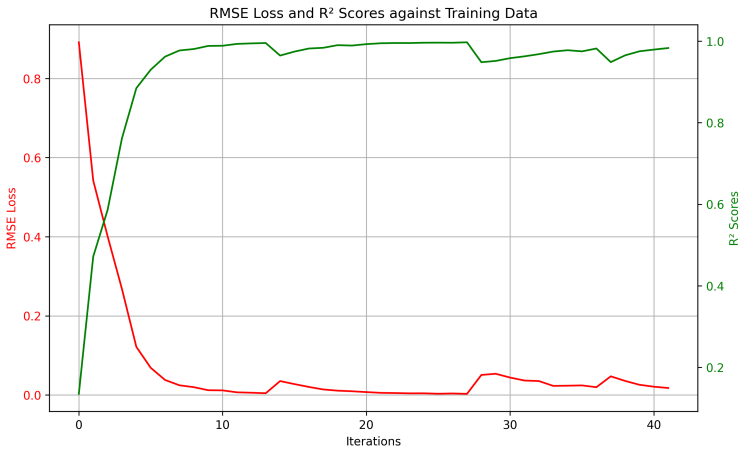


Figure 4.36: RMSE and R^2 - Function 2: `[2, 5, 5, 1]`

Similar to our previous findings with a deeper network, a deeper network for this function appears to be more robust as well. The model only suffers from a minor decrease in its $RMSE$, and R^2 when pruned or when the activation functions are being approximated. This further strengthens our inference that deeper models, with greater number of parameters to optimize for and activation functions to learn, are more robust, and can handle the loss of information more efficiently.

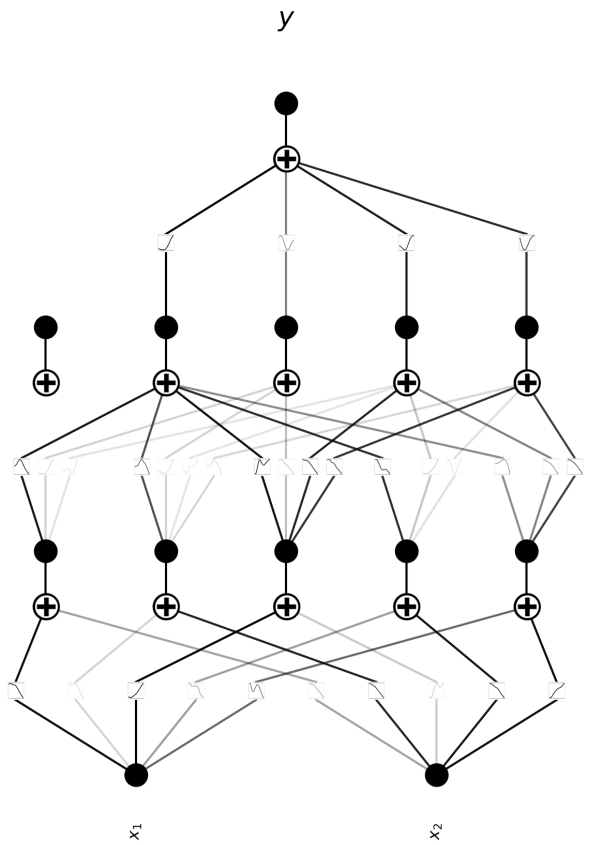


Figure 4.37: Pruned network - Function 2: `[2, 5, 5, 1]`

In this case, the pruning is more effective than with the previous function. Although no edges are removed from the first layer, the second layer is subjected to the complete removal of one of its neurons. The first neuron of the second layer does not receive or send out any information. This can contribute further in obtaining a relatively simpler symbolic formula, as compared to when the pruned neuron would also be involved and have the need to be approximated.

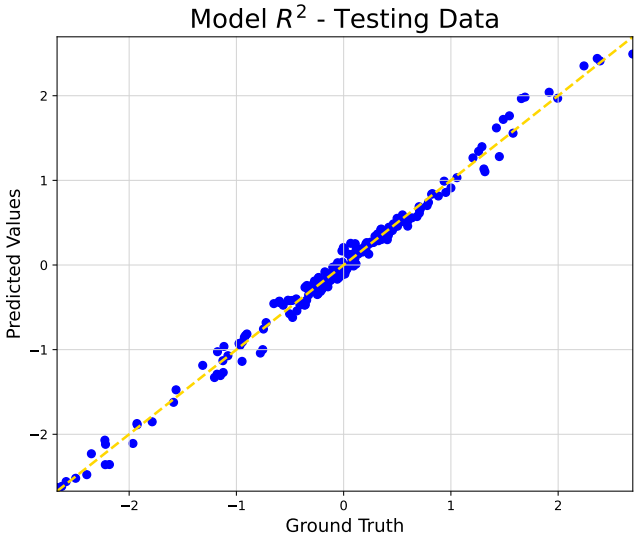


Figure 4.38: Scatter plot - Function 2: `[2, 5, 5, 1]`

Table 4.14: Model Performance: R^2 Scores

Dataset	R^2 Score
Training	0.995
Testing	0.992

The R^2 values and plots further describe the quality of the model. For the current case, a deeper network is able to obtain excellent correlations, and maintain its superiority in having a better testing R^2 over a simple architecture, although only by the faintest of margins. The $RMSE$ error, however, remains identical across the two architectures.

The symbolic formula can be referenced from Appendix Section B.5.3. As we need much simpler expressions that can be inserted into the CFD solver and evaluated with ease without causing divergence issues, a 2 layered network, or a deeper network in general, at this point, becomes unfavorable and not the right architecture to be implemented for large datasets such as the CFD cases with 5,000+ data points.

4.2.3. Function 3

This formula is also used to compute the Euclidean distance in a two dimensional space. Given the results from the previous functions and the greater mathematical depth of the formulae, partly due to a complex library as well, a different and a much simpler library is given to KANs for this function. This also serves as a test to determine whether KANs can effectively adapt simpler functions to model or represent the complex interactions between the variables in the functional space. The library is presented below.

$$\mathcal{L} = \{x,\ \sin(x),\ \ cos(x)\}$$

The library was also chosen as such because the previous symbolic formulae predominantly consisted of $\sin$, $\cos$, and linear algebraic terms.

`[4, 3, 3, 1]` Network

For the given function, a `[4, 3, 3, 1]` KAN network is applied, identical to the publishers of KANs. The corresponding results are detailed below.

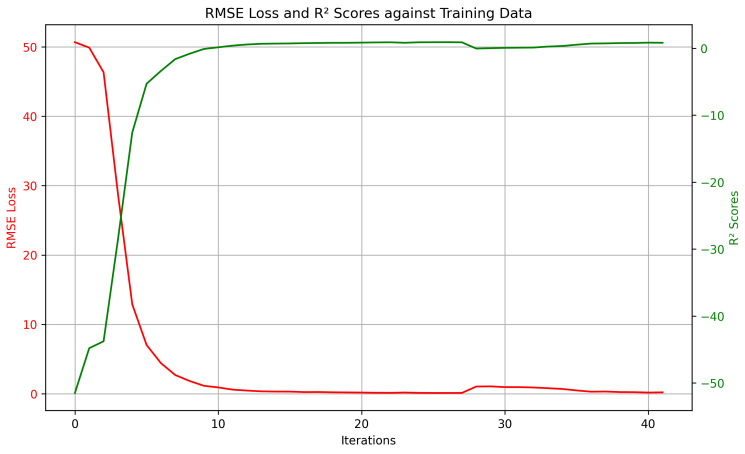


Figure 4.39: RMSE and R^2 - Function 3: `[4, 3, 3, 1]`

It can be seen that the model finds it very difficult and struggles in the initial 10 iterations. With high $RMSE$, and very low R^2 scores, the model requires at least more than 10 iterations to generate scores in the acceptable range. However, given the rate of change, it also tells us about the ability of a deeper

architecture to learn the bulk of the mapping almost immediately. This can prove to be beneficial when mapping complex interactions between variables in the functional space such as in CFD cases where there can be more than 15 scalar invariants, as it can very swiftly converge to at least acceptable scores, if not excellent.

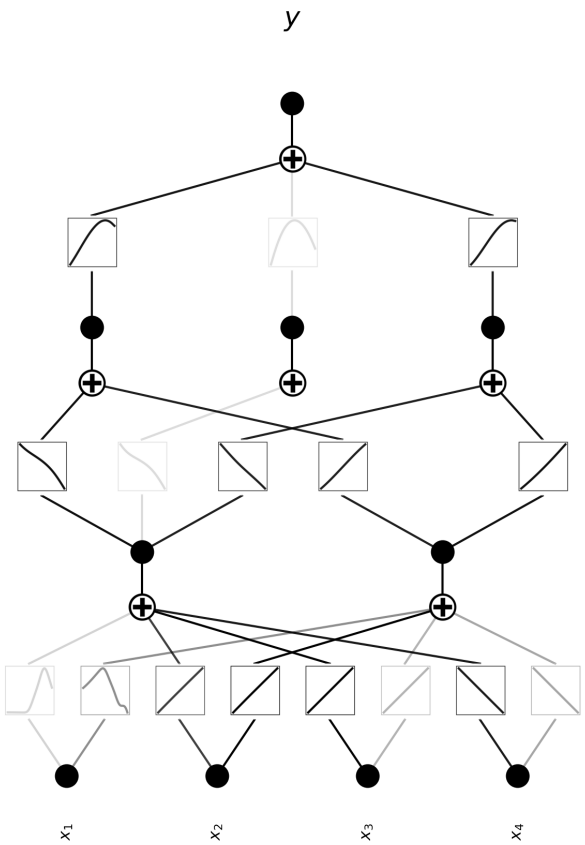


Figure 4.40: Pruned network - Function 3: `[4, 3, 3, 1]`

We can see that pruning has has been very effective in this case. Starting with a `[4, 3, 3, 1]` network, is is reduced to a `[4, 2, 3, 1]` architecture. Moreover, the second layer is also subjected to many edge removals, which in turn imply the complete removal of the associated activation functions as well. Hence, the model only considers the crucial inputs, and the nonlinearities, discarding the rest. This would also subsequently produce a simpler symbolic formula as well.

Once again, at iteration 28, the concerned metrics worsen before eventually recovering with training. This represents the point where the activations are approximated via the library of the given functions, resulting in too large a learning and possibly instabilities.

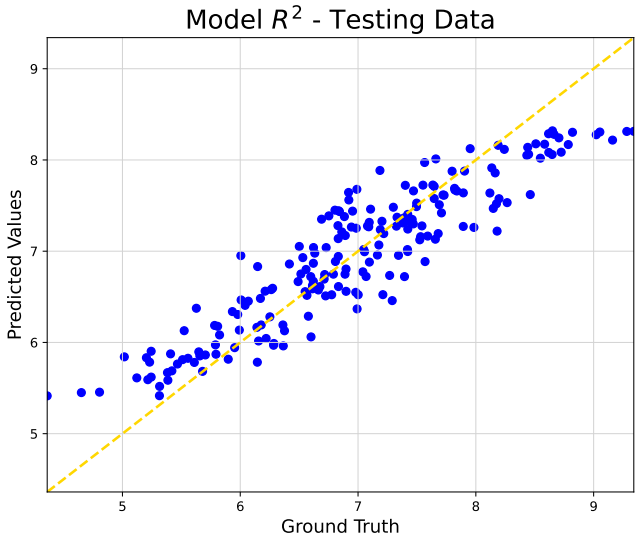


Figure 4.41: Scatter plot - Function 3: `[4, 3, 3, 1]`

Table 4.15: Model Performance: R^2 Scores

Dataset	R^2 **Score**
Training	0.838
Testing	0.827

With regards to the correlations, it can be said that the resulting model is not excellent or on par with the models that were produced for previous functions. It is able to obtain an R^2 only in early 0.80's. Despite its expressiveness, ability to quickly learn and model the relations, it can be said that KANs may require further training iterations to achieve exceptional scores in the range of > 0.90, the range that the present study also aims to obtain with the CFD datasets in the $a\ priori$ testing. However, this can also be contributed due to the limited library. By giving only a few functions, we are essentially limiting the expressive power of KANs. This can increase the loss of information, which already occurs when the approximation takes place, by the symbolic functions. Thus, having shorter library may not be able to produce a very accurate model.

The symbolic formula can be referenced from Appendix Section B.5.4. The resulting equation does not have any *power* terms, yet is able to learn the activation functions effectively. It can further be observed that although the underlying function is more complex, the resulting symbolic formula is quite short when compared to the previous 2-layered KAN implementations. The effect of pruning can be indirectly visualized here. With less number of unknowns to approximate and reduced functions to learn, the resulting model is also quite simpler.

`[4, 3, 1]` **Network**

Contrary to the previous cases, instead of implementing a deeper network than the choice of the authors of KANs, the network is shortened. A `[4, 3, 1]` layered network is now implemented to determine if a simpler network is also potentially able to model the relationships that a deeper network can. The performance is visualized below.

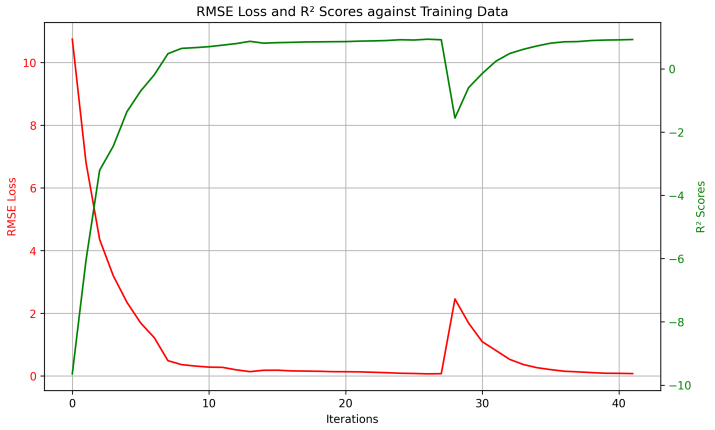


Figure 4.42: RMSE and R^2 - Function 3: `[4, 3, 1]`

It can be clearly observed that suddenly, with a shallower network, the capability of KANs to learn and model the functional spaces is also enhanced. Even at the very initial stages of optimization, the model performs considerably better. The worst $RMSE$ reported by this model, at the very beginning, is only slightly above 10, as opposed to being greater than 50 with the 2-layered network. The same goes with R^2 values.

Another consistent trend is the drop in the performance once the activations are approximated via the symbolic functions. However, it is quickly able to learn the updates, optimize the selected functions, and return back to its original optimized state. This establishes that simpler networks are more effective in optimizing its parameters θ to quickly learn the dependencies between variables, the target variable, and explore the functional space.

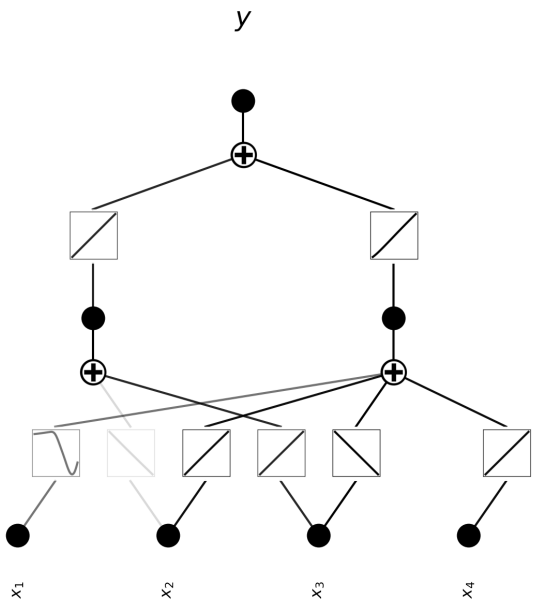


Figure 4.43: Pruned network - Function 3: `[4, 3, 1]`

We can observe that pruning is much more effective in this scenario. Similar to the deeper network, one of the neurons is removed in the hidden layer, thus reducing the `[4, 3, 3, 1]` to a `[4, 3, 1]` KAN. Moreover, it is also clearly evident that more edges are pruned here. In comparison to Figure 4.40 wherein each input variable had 2 edges, the present network only has half the inputs with 2 edges, or 2 association activation functions. This further makes learning quite fast as it has to approximate

relations or optimize a significantly reduced number of parameters. We can strongly now say that pruning is much more effective with short layered networks, and it makes the training process much quicker as well.

Figure 4.44: Scatter plot - Function 3: `[4, 3, 1]`

Table 4.16: Model Performance: R^2 Scores

Dataset	R^2 **Score**
Training	0.923
Testing	0.922

It is clearly evident how well the resulting model behaves with both training and testing data. With minimal drops between the datasets (training & testing), the shorter network is quickly able to produce excellent correlations, which the deeper network was not able to.

The symbolic equation can be referenced from Appendix Section B.5.4. With more effective pruning of neurons and edges, faster training as there are less number of functions and parameters to learn, and a straightforward symbolic equation easily evaluated, a 1-layered KAN would be the reasonable choice, moving forward.

Pruned Inputs - `[4, 3, 1]` Network

As discussed in earlier sections, specifically Section 3.3 wherein pruning of inputs was explained, the same is applied in this iteration of the dataset to further assess the ability of KANs if it can remove an unnecessary input(s) and still produce an excellent model. The graphs to illustrate the performance of the resulting KAN model are presented below. The order of graphs is changed here as it would be wise to first see the pruned model, and then proceed to visualize its performance.

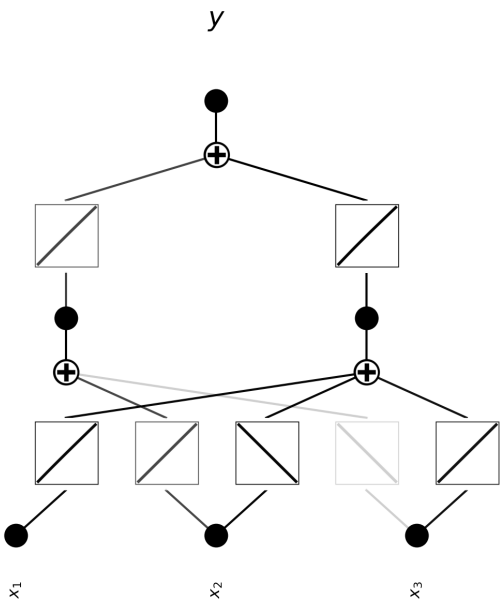


Figure 4.45: Pruned network (inputs) - Function 3: `[4, 3, 1]`

It was seen in Figure 4.43 that both x_1, and x_4 had only single edges coming from them, implying that for the chosen network, their contribution in approximating the overall target variable was less, as compared to the rest of the inputs. In this iteration, it is observed that x_4 has been pruned. Thus, the 4-dimensional space will now be approximated using only 3 inputs. This reduces the model complexity significantly, and makes the training a lot less computationally extensive. This ability of KANs to do so in itself describes the potential of them to not just effectively model, and quickly adapt and learn the functional space, but also reduce its dimensions.

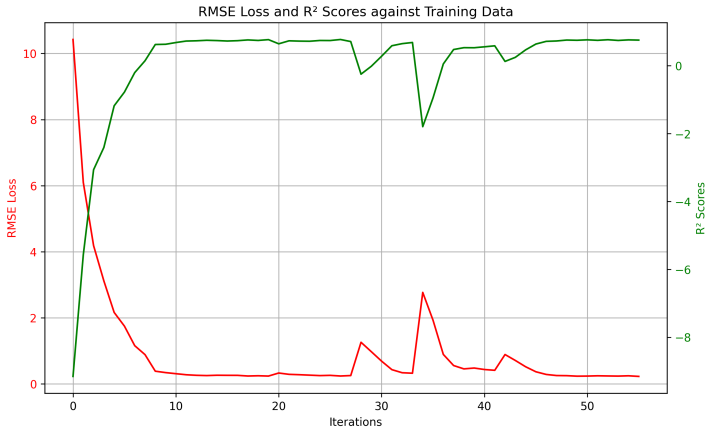


Figure 4.46: RMSE and R^2 (pruned inputs) - Function 3: `[4, 3, 1]`

The initial performance can be comparable to the network wherein pruning of inputs was not implemented. It quickly converged to the optimal results. However, it can be seen that the model is more sensitive now. It required additional training iterations to converge to a point in the solution space. This is also coupled with the few drops in its performance occurring between $\approx$ 28-42 iterations. This suggests that the model, with one input removed, has become more susceptible to potential divergence, and extremely sensitive to even small changes in the symbolic formula. Thus, it necessitates further training to make the model more robust, which would not be ideal when applied to the CFD datasets, as it would mean to run the CFD solver several more number of times for the model to achieve convergence.

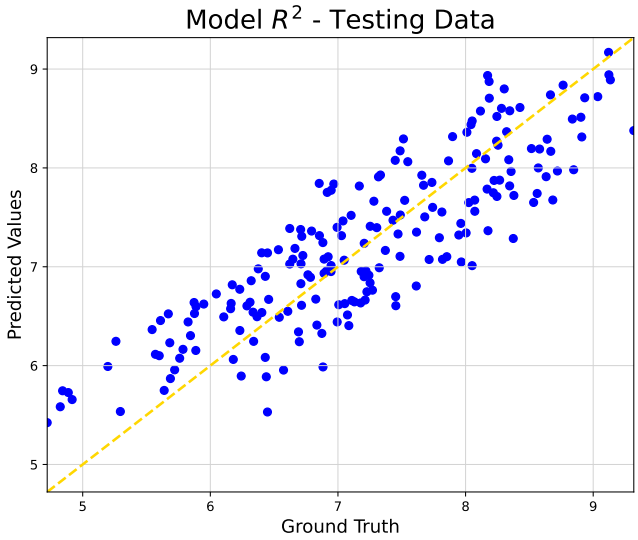


Figure 4.47: Scatter plot (pruned inputs) - Function 3: `[4, 3, 1]`

Table 4.17: Model Performance: R^2 Scores

Dataset	R^2 **Score**
Training	0.761
Testing	0.762

It is clearly evident that the model only manages to reach sub-optimal convergence, and requires further training to attain better correlations. With the reduced dimensions, it is not able to effectively capture the interactions between variables.

The resulting symbolic formula can be referenced from Appendix Section B.5.4. Although much more simpler to understand and evaluate, it is not ideal as it fails to give excellent correlations that the current research aims for. However, the idea of reducing the dimensionality of the functional space and using a select few variables to map the entire functional space resulting in simpler symbolic equations and reduced computational cost is far too appealing to not be tried and implemented in the CFD datasets.

It is also important to note that it achieves the correlations without using any square or root functions, despite the original function being primarily composed of such operations. This highlights the strong capability of KANs to approximate complex functional relationships, even with a limited function library. KANs are able to represent operations absent from the library by effectively mapping existing functions onto the functional and parameter space, forming efficient and expressive combinations. This demonstrates their potential to learn rich representations in constrained symbolic environments. Thus, we do not need to know $a\ priori$ the relationships between the different inputs and the target variables, as KANs have repeatedly displayed its ability to model them smartly with the limited options that are given to it.

4.2.4. Function 4

The testing on above 3 functions were sufficient to come to certain conclusions. This function only served as a supplementary test case for the author to become more familiar with the working of KANs and be confident of its performance. Hence, its results can be referenced from Appendix Section B.5.5.

4.2.5. Conclusions - Preliminary KANs testing

The preceding section concludes the preliminary evaluation of KANs. Their performance has been assessed across multiple datasets using varying architectures and pruning strategies. A comprehen-

sive analysis has been conducted, with results carefully examined and justified. To summarize the key findings, the following conclusions are drawn.

- One of the initial observations made regarding the performance is that there can be loss of information when converting or approximating the activation functions via to its symbolic form via the library given. However, training further over a few iterations restores the model's performance. Deeper networks, however, tend to be more robust and have a less significant information loss.
- Nonetheless, when taking pruning into account, it is less effective on a bigger network. In a deeper network with more number of neurons, an individual edge score can be lower, but relatively be similar to the rest of the edges. In other words, it can be difficult to distinguish important edges, and can potentially represent functions with overlaps. Hence, each activation function can have similar contributions to the final approximations. However, with shallower networks, it is easier to single out edges that contribute minimally, as they are forced to use fewer but meaningful connections.
- This has direct consequences in the final symbolic formula obtained. With pruning not as effectual as it could be, many more activation functions are learnt and approximated. This increases not just the length of the symbolic expression, but the complexity of the model as well. Hence, shorter networks need to be preferred for an interpretable compositional structure of the final model.
- KANs exhibit continual learning. This coupled with small models (small networks), and a simpler final function can enhance its accuracy and efficiency. This can also be seen with the comparison of correlations when a deeper and a shallower network were compared. The latter outperformed and gave excellent correlations on almost all the functions. This makes a 1 hidden-layered network an obvious and a favorable choice moving forward. Increasing or decreasing the number of neurons in this 1 hidden-layered network can enhance the expressive of power of KANs, and make training easier by quickly learning the activation functions and mapping them onto the symbolic functions. However, this would also depend on the dataset and the user, as too many neurons can once again also enhance the complexity of the model. In short, we can say KANs are extremely expressive, and demonstrates fast learning capabilities, making it well-suited for complex functional mappings in high-dimensional spaces.
- As seen in Subsection 4.2.3, KANs have the ability to prune inputs. This can prove to be very useful in CFD datasets where there can be up to 25 inputs (scalar invariants). The curse of dimensionality, which many machine learning algorithms suffer from, cam easily be tackled by KANs, as they reduce the number of parameters, and the dimension of search space significantly. However, a balance between pruning inputs and training iterations would be needed, to get excellent correlations.
- Another interesting observation is that it is not necessary to provide an extensive library to approximate the activations. Although having more functions available may potentially increase the expressive power, it also results in a symbolic formula not easy to comprehend. KANs have exhibited their capabilities to express a functional form by smartly selecting and making combinations of functions from a limited library, and then further optimizing them by affine transformations. Therefore, when implementing KANs on CFD datasets, a limited library of a few functions will be given to fully leverage this ability of KANs, when obtaining the functional form of the model.

This culminates the entirety of the preliminary testing of both the methods - genetic programming with gradient descent, and Kolmogorov-Arnold Networks. With parameter settings, optimizer, and the methodology of KANs studied extensively and finalized, the present work, until now, gives us a strong foundation of these methods, and evidence that they can potentially give excellent correlations on actual CFD datasets, and improve RANS modeling. The study now leads to the next chapter, wherein different CFD cases are studied in a methodical manner.

5

A priori CFD Results

The present chapter will implement the developed (and optimized) algorithms on different CFD datasets.

Initially, as an additional test or screening for these methods, the algorithms are applied to formulate the corrective term for turbulent production based on only one tensor: dissipation (ϵ). It is well known $a\ priori$ that the turbulent production is heavily influenced by the ϵ for cases such as Periodic Hill, and Square Duct. Essentially, this reduces to a least squares regression problem, wherein the coefficient of ϵ will be optimized which further depends on the scalar invariants. Hence, a comparison with standard least squares regression is also made where the coefficient is only a constant (scaling factor).

Furthermore, given the inherent random nature of GP's, multiple random seed iterations were performed to quantify the performance appropriately, and mitigate the effects of any randomness in the final results. For KANs however, only one-off, or at most a few runs were performed with a selected architecture to ensure the results remain consistent.

5.1. Genetic Programming with gradient descent for PH - ϵ

Identical to the previous methodology implemented in Subsection 4.1.3, 100 iterations were performed with distinct random seeds. The input space comprised of 17 scalar invariants, combining to form the coefficient of ϵ, used to regress for $k_{deficit}$. The results, along with the input space, are described below.

Table 5.1: Functional Space

Set Type	**Variables**
Non-dimensional Set	$\mathrm{S2_s}$, $\mathrm{W2_s}$, $\mathrm{S3_s}$, $\mathrm{W2S_s}$, $\mathrm{W2S2_s}$, $\mathrm{Ak2}$, $\mathrm{Ak2S_s}$, $\mathrm{Ak2S2_s}$, $\mathrm{Ak2WS_s}$, $\mathrm{Ak2WS2_s}$, $\mathrm{Ak2SWS2_s}$, q_Q, q_T, $q_{\mathrm{CP_B}}$, $q_{\mathrm{tauk_B}}$, q_γ, q_{Re}
Dimensional Set	ϵ
Target Variable	$\mathrm{k_{deficit}}$

The presence of subscript s in the scalars indicate its non-dimensionalisation by the mean flow time scale when these features are derived.

Note that we are most interested in the function count and the quality of the converged individual. Hence, correlation graphs and function counts are plotted, in addition to briefly discussing the convergence behavior. An analysis on the evolution of quantities such as the *Elite RMSE*, training and testing R^2 is performed and can be referenced from Appendix Section C.1.

5.1.1. Optimization - I

Optimization I refers to applying operator optimization at every 3^{rd} and terminal optimization at every 5^{th} generation, chosen as the final settings after preliminary testing in Section 4.1.

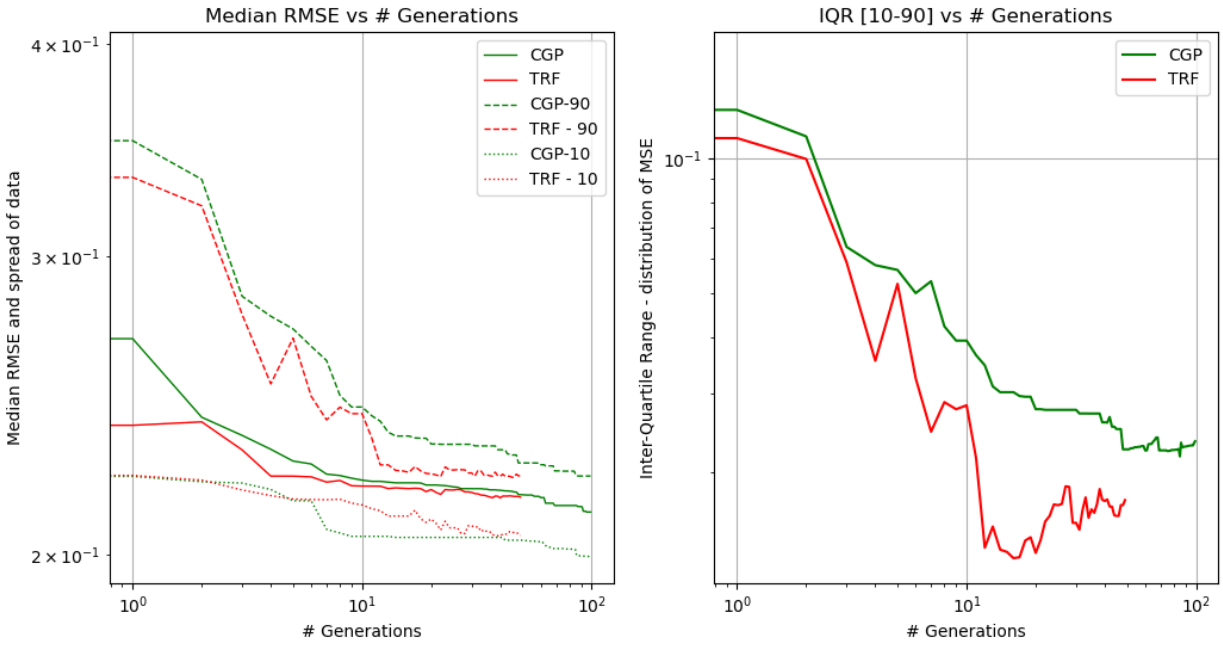


Figure 5.1: Log Median RMSE: $k_{deficit}$ for PH based on ϵ

To gain a better perspective on the convergence rates of the algorithms, the above plot was made on a logarithmic scale. It is seen that the mimetic algorithm exhibits superior convergence rates throughout. The convergence rate especially experiences a jump when an optimization cycle occurs, at the 3^{rd}, and the 5^{th} generation for terminals and optimizers, respectively. Notably, when going from the 2^{nd} to the 3^{rd}, the elite individual after terminal optimization at the end of the 3^{rd} generation is much more improved than the preceding elite individual. This can be observed throughout until the end.

However, it can also be seen that at certain generations the quality may deteriorate as well. For instance, when going from the 4^{th} to the 5^{th} generation, the bottom 10% of the individuals suffer from the operator optimization, and their quality worsens. Hence, a proper balance is required between exploration, and the extent of exploitation. The IQR plots shed further light on the capabilities of the optimizer. We can see that the variation in the quality of the individuals significantly drops when an optimization cycle occurs. Thus the optimizer is able to effectively guide the overall search into an optimal space, with randomness from genetic programming providing the ability to explore this optimal space only, rather than the whole functional space.

Nevertheless, as the algorithm proceeds and cycles of optimization keep on occurring, the key takeaway is that the proposed algorithm exhibits superior convergence rate to the optimal solution space, and a higher quality of individuals. This is attributed to its structured exploitation of the search space, and effective hybridization strategy.

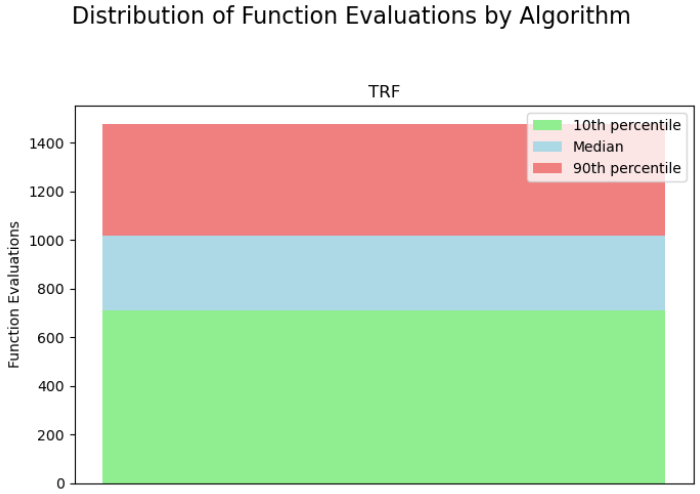


Figure 5.2: Function Evaluations: $k_{deficit}$ for PH based on ϵ

It can be deduced that the optimization process is not very expensive. With a majority of the random seeds requiring roughly 1000 and 90% of the runs requiring $\approx$ 1400 function evaluations over 100 runs, which translate to an equivalent number of CFD runs, the optimizer is interestingly very strong, efficient and cheaper in converging to an optimal solution. The performance is aided by the fact that the optimized algorithm is made to run for only half the generation.

Each generation requires the evaluation of the individuals in the population who's fitness is not valid. This is equivalent to performing a CFD run with the individual, and determining the fitness value with an appropriate metric. As a rough estimate with the implemented population size and number of generations, the conventional method performs **4900** function evaluations, keeping one elite per generation. The proposed algorithm on the other hand, performs $\approx$ **3500** function evaluations. Although only a crude estimate, the proposed algorithm is around **30%** more cheaper than the benchmark method. These are very encouraging results.

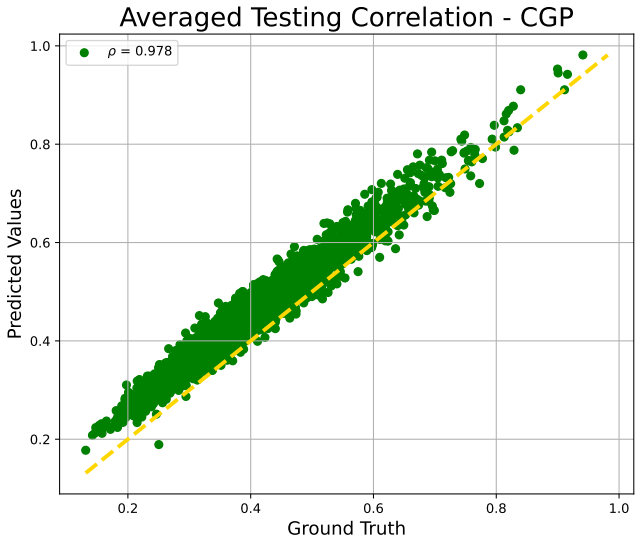


Figure 5.3: Averaged CGP correlations: $k_{deficit}$ for PH based on ϵ

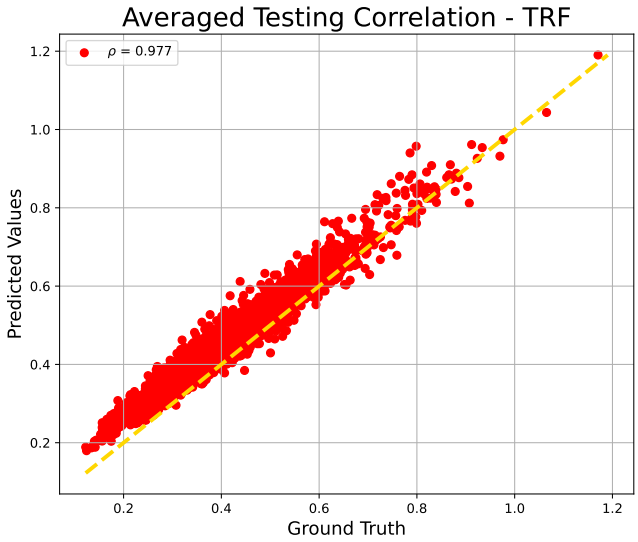


Figure 5.4: Averaged TRF correlations: $k_{deficit}$ for PH based on ϵ

The quality of the converged individuals can be further visualized via the scatter plot. The gradient based algorithm is able to achieve better correlations. Although only a slight improvement, it clearly demonstrates the ability of the proposed method to not only converge to higher quality individuals, but also do it in a significantly reduced number of function calls.

This simplified analysis on an actual CFD case is directly able to answer the primary question laid out in Chapter 2. *A genetic program that is incorporated with information from derivatives can indeed achieve quicker and a higher order convergence, with reduced function evaluations, and acceptable correlations*.

5.1.2. Optimization - II

It was now decided to further improve the performance of the gradient based algorithm. Instead of performing operator and terminal optimization at the 3^{rd}, and the 5^{th} generation, it was decided to combine both the terminal and operator optimizations at the 5^{th} generation with the aim to make the implementation easier and reduce the function calls made by the optimizer. However, to verify this concept and evaluate its performance, the same case of Periodic Hill, with 1 dimensional quantity - ϵ was tested over 100 random seeds, and a direct comparison was made with the existing optimal conditions. The results are presented below. An analysis on the evolution of quantities such as the *Elite RMSE*, training and testing R^2 is performed and can be referenced from Appendix Section C.2.

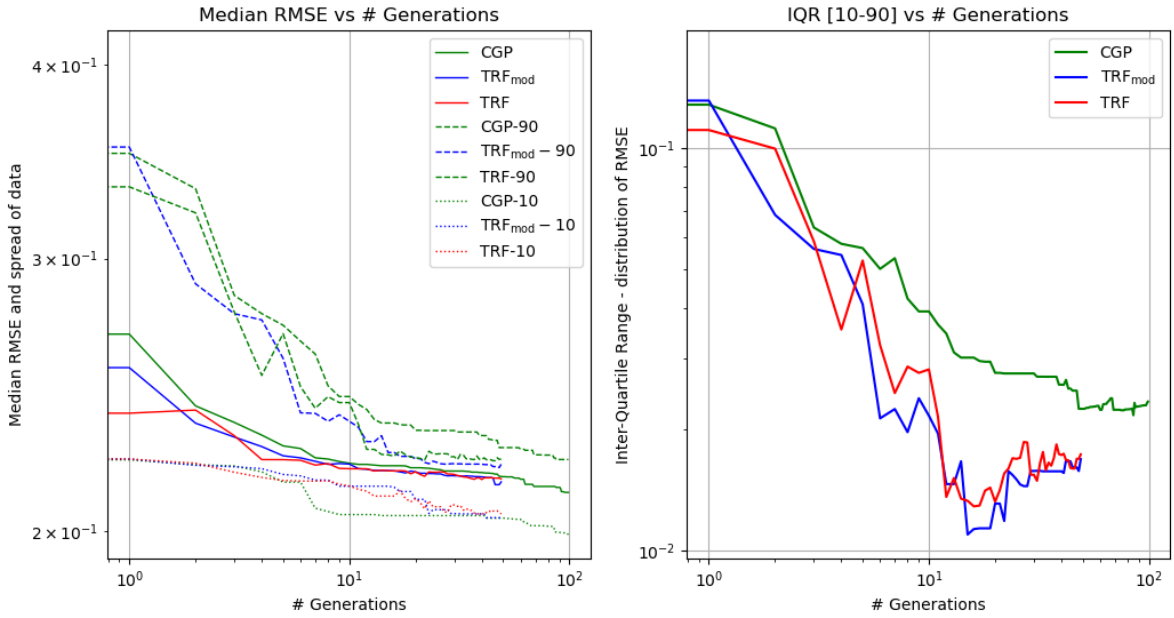


Figure 5.5: Log Median RMSE R^2: $k_{deficit}$ for PH based on ϵ - combined optimization

It is now clearly visible that the convergence rate, and its order also remains identical with the modified optimization settings. Interestingly, the occasional deterioration in the individual qualities observed with the previously implementation of the optimizer was no longer observed. The modified optimizer showed an improvement in the individual quality throughout, with minimal oscillations.

Thus, it can be said that the search trajectory is more stable. Simultaneous updates allow better coordination between terminals and the operators, preventing one of them from undoing improvements made by the other. This makes the whole optimization process more coherent, wherein the operators and terminals are adjusted in synchronously. Furthermore, since quality deterioration is not significant, the simultaneous optimization is more efficient as it likely spends less time in correcting suboptimal configurations generated earlier. In addition, the algorithm exhibits an improved ability to exploit promising regions without needing to backtrack due to poor operator-terminal compatibility. The information to explore and exploit the whole functional space is being propagated more effectively between successive generations with simultaneous optimizations.

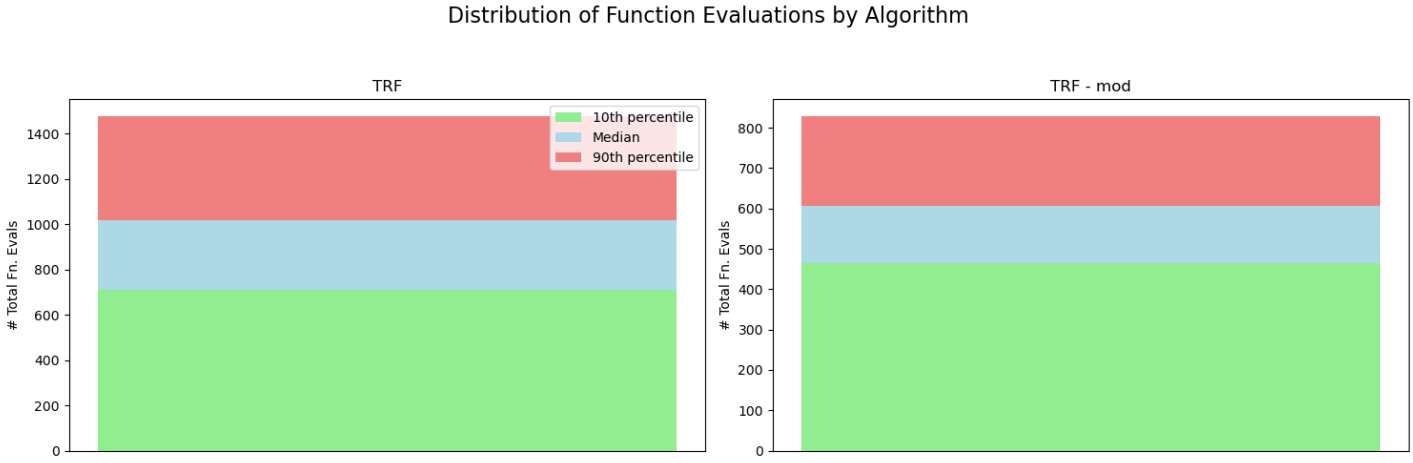


Figure 5.6: Median Function Evaluations: $k_{deficit}$ for PH based on ϵ - combined optimization

It can be seen that the median function calls have dropped by over 40%, with 90% of the runs experiencing a similar drop as well. This is a remarkable decrease, and as seen earlier, the algorithm is now able to achieve excellent solutions with significantly few function evaluations, indicating that the simultaneous optimization process makes better use of each function call. From an $a\ posteriori$ perspective, this translates to CFD simulations in the order of a few hundreds only. The scatter plots for CGP and TRF can be referenced from Figure 5.3 and Figure 5.4

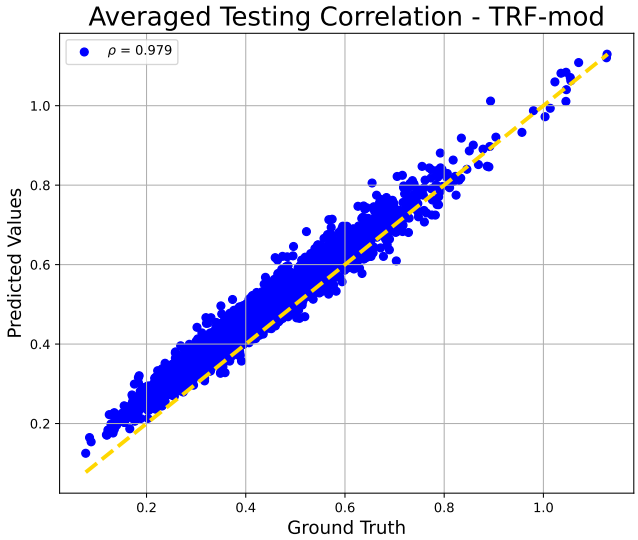


Figure 5.7: Scatter Plots: $k_{deficit}$ for PH based on ϵ - combined optimization

Table 5.2: Performance of different algorithms: Converged Training and Testing R^2 Scores

Method	**Training** ρ	**Testing** ρ
CGP	0.98026	0.97794
TRF	0.97953	0.97675
TRF - mod	0.97897	0.97857

The averaged correlation from the modified settings has improved. Hence, the quality of the individuals is not compromised at the expense of fewer function calls. But rather, reinforces our claim earlier that simultaneous updates allow a more coherent optimization preventing one of them from undoing improvements made by the other and consistently improving the quality of the individual.

With this, the idea of having simultaneous optimizations is proven to be more efficient in terms of improved metrics, and with regards to function calls as well. Thus, this setting will be used hereafter when all the dimensional quantities will be used to develop the completed corrective equation, unlike the partial corrective equation, based on only ϵ.

Observing the converged correlation numbers as well, it can be seen that the algorithm with simultaneous optimization is able to converge to ever superior correlations, although only by a small margin. However, it is still sufficient to further reinforce our claim that simultaneous optimization is better than the alternative optimizations.

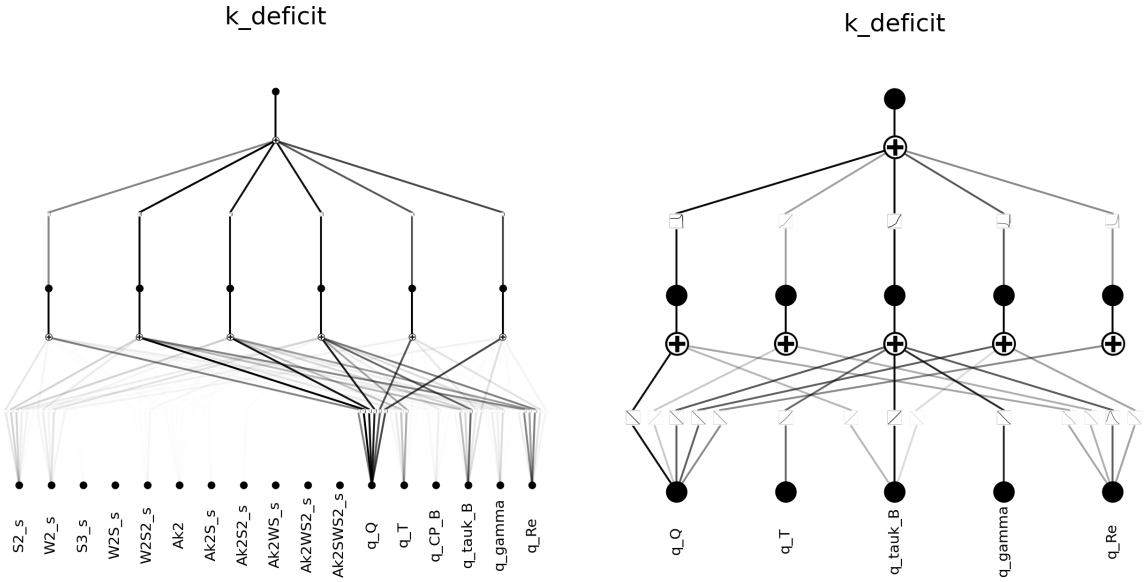
Before concluding this, the author also wanted to give the reader further insights into how the scores evolve over with repeated function calls, and how the algorithm proceeds (convergence). This can be referenced from Appendix Section C.2.1.

5.2. Kolmogorov Arnold Networks for PH - ϵ

The same set of 17 non-dimensional scalars will be used to optimize the coefficient of the dimensional quantity, ϵ, to deduce the corrective term for turbulent production.

We have seen that having 4-5 neurons in the hidden layer from benchmark cases has produced favorable results, Section 4.2. Moreover, the authors of KANs claim it can be convenient to start with a large enough network, which can then be pruned to reduce the parameter space. This was seen earlier wherein a `[2, 5, 1]` network was pruned to `[2, 2, 1]`, as reported in Subsection B.5.5. Keeping this in mind, a `[17, 6, 1]` network was implemented.

The functional space can be referred from Table 5.1, and the results are presented below.



(a) Initial network **(b)** Pruned network

Figure 5.8: Comparison of initial and pruned KAN structures: $k_{deficit}$ for PH based on ϵ

From the above figure, we can see the pruning of input space, and the activation functions. Only the essential ones are considered for further model training, and symbolization. What is most beneficial is not just its ability to prune one or two inputs, but rather 12 inputs, and significantly reduce the solution space constrained by 5 variables only. This is one of the many very beneficial features of KANs, as also has been demonstrated earlier.

Figure 5.9: Convergence History: $k_{deficit}$ for PH based on ϵ

The convergence history demonstrates that despite certain fluctuations, and not converging directly, the model is still able to attain very good metrics by the end. When compared with the proposed genetic programming algorithm, it is able to clearly outperform it. The achieved RMSE is significantly lower, and it is able to achieve an excellent correlation of $>$ 0.95, which is attained in a significantly lower dimensional solution space. This further reinforces the smart ability of KANs of not just pruning the input space, but also combining the remaining inputs in an effective manner, and then using a library of user-defined functions to obtain a symbolic formula.

Figure 5.10: Scatter Plot: $k_{deficit}$ for PH based on ϵ

Table 5.3: Model Performance: R^2 Scores

Dataset	R^2 **Score**
Training	0.983
Testing	0.979

It can be seen that excellent correlations are obtained on both the datasets, and the model does not suffer from issues such as overfitting, or underfitting, despite having only 1 hidden layer with 6 neurons and operating on $10,000+$ data points. This is a problem that is often encountered with neural networks when the number of parameters are significantly lower than the number of data points, which does not pose a problem in this case.

The symbolic formula can be referenced from Appendix Section C.3. It can be noticed the symbolic formula is only composed of linear algebraic operations, and trigonometric functions. Despite not providing an extensive library with polynomials, logarithms, and exponential functions, the model is still able to use the library and map the continuous space onto the unary functions.

In addition to this, a dominant factor is the total function count. For reference, roughly 4900, and $\approx$ 3200 function calls were required by the conventional, and the modified genetic programs respectively. KANs however, only required 848 function calls to train a model capable of converging to excellent scores, and correlations, in addition to also giving out an interpretable symbolic formula. This makes KANs heavily favorable as the method for further $a\ posteriori$ studies.

This concludes the implementation of KANs to regress for a corrective equation for $k_{deficit}$, based on one feature only.

5.3. Least Squares comparison

Until now, we have taken one dominant tensor or a dimensional quantity and optimize its multiplicative factor, which in turn is a function of a number of flow properties, referred to as the invariants. It can be considered as a simple least squares regression problem where the coefficient is optimized. Therefore, as a final validation, comparison of correlations is done with classical least squares regression algorithm (LSR).

The correlations with the testing, and training data for all the 3 methods are tabulated below. Note that median values are presented for genetic programming, while the obtained values from one-off runs via KANs, and LSR are presented.

Table 5.4: Performance with different methods: Training and Testing R^2 Scores

Method	**Training** R^2	**Testing** R^2
Mimetic Algorithm	0.954	0.954
KANs	0.959	0.956
LSR	0.952	0.951

The scalar, α, for LSR was 1.44. The capability of KANs to converge to models with superior correlations is clearly distinguishable. The predictive performance of the models remains identical for both training and testing datasets, followed by the mimetic algorithm which has an improved correlation when compared to the LSR. Although LSR can give an optimized coefficient instantaneously, it lacks any physical significance. KANs and the genetic programming on the other hand, optimize the coefficient composed of a combination of invariants, and dimensional quantities (tensors) which is essential to interpret the corrections in a physical way, and understand which flow quantities impact the overall CFD solution.

Nonetheless, the devised algorithms are able to outperform classical least squares regression method. This concludes the testing on a canonical flow case, with just one feature.

5.4. Periodic Hill Testing

The algorithms were applied to the Periodic Hill datasets. All the features and tensors were now utilized to formulate the completed correction term. One-off runs were performed, and the converged results are presented below (along with input space), for both GP $+$ gradient descent and KANs.

Table 5.5: Complete Functional Space for Correction Terms - PH

Set Type	**Variables**	**Target Application**
Non-dimensional Set	$\mathrm{S2_s}$, $\mathrm{W2_s}$, $\mathrm{S3_s}$, $\mathrm{W2S_s}$, $\mathrm{W2S2_s}$ $\mathrm{Ak2}$, $\mathrm{Ak2S_s}$, $\mathrm{Ak2S2_s}$, $\mathrm{Ak2WS_s}$, $\mathrm{Ak2WS2_s}$ $\mathrm{Ak2SWS2_s}$, q_Q, q_T, $q_{\mathrm{CP\_B}}$, $q_{\mathrm{tauk\_B}}$, q_γ, q_{Re}	Both Corrections
Dimensional Set	$\mathrm{G_{1s}}$, $\mathrm{G_{2s}}$, $\mathrm{G_{3s}}$, $\mathrm{G_{4s}}$, $\mathrm{G_{5s}}$, $\mathrm{G_{6s}}$, $\mathrm{G_{7s}}$, $\mathrm{G_{8s}}$, $\mathrm{G_{9s}}$, $\mathrm{G_{10s}}$, ϵ	$\mathrm{k_{deficit}}$
	$\mathrm{T_{1s}}$, $\mathrm{T_{2s}}$, $\mathrm{T_{3s}}$, $\mathrm{T_{4s}}$, $\mathrm{T_{5s}}$, $\mathrm{T_{6s}}$, $\mathrm{T_{7s}}$, $\mathrm{T_{8s}}$, $\mathrm{T_{9s}}$, $\mathrm{T_{10s}}$	$\mathrm{b_{ij}^{\Delta}}$

This is the complete functional space. However, the author's research group implies `mutual information` and statistical measures to reduce the dimension of the input space. During the feature definitions, certain invariants are culled depending on their standard deviations, or complexity. They are then subjected to `mutual information` where the invariants and tensors above a certain threshold only are selected as final inputs. Moreover, these tensors are derived from the velocity gradients raised to higher

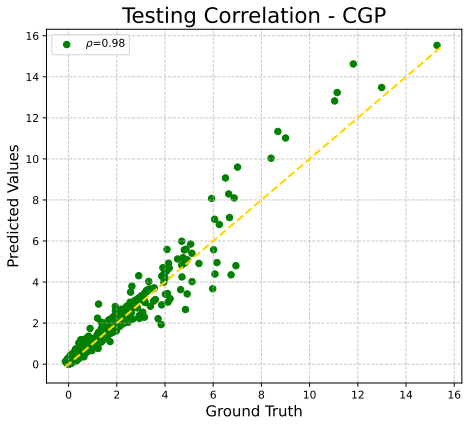
powers. These can potentially cause instabilities when inserted into the CFD solver. Hence, based on experience and as a rule of thumb, certain tensors are neglected. This helps in converging to good models and having a stable optimization process. The table presenting the mutual information thresholds can be referenced from Appendix Section C.4, Table C.1. Note that for KANs, its own pruning feature was used to reduce the dimension of the input space (invariants), as explained in Section 3.3. To demonstrate the effectiveness and the quality of the converged model, scatter plots are presented with testing data. Statistical comparison is presented in Section 5.6.

5.4.1. Turbulent Production

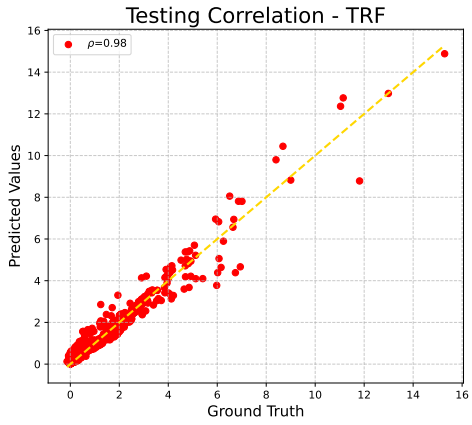
Table 5.6: Reduced Functional Space for PH Correction in Turbulent Production - GP

Set Type	Variables
Non-dimensional Set	$\mathrm{S2_s}$, $\mathrm{W2_s}$, q_T, $q_{\mathrm{CP_B}}$, $q_{\mathrm{tauk_B}}$, q_γ, q_{Re}
Dimensional Set	G_{1s}, G_{6s}, ϵ
Target Variable	$\mathrm{k_{deficit}}$

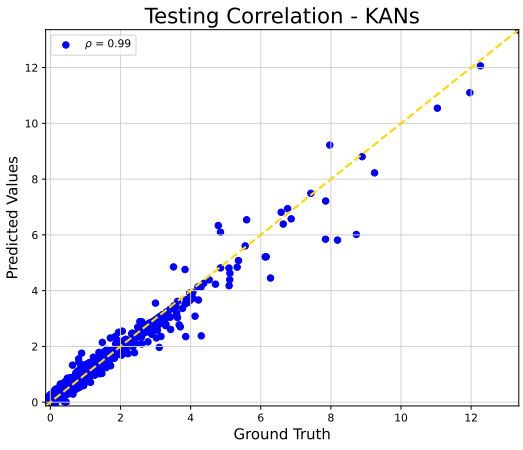
The symbolic formula and the model can be referenced from Appendix Section C.6.1.



(a) CGP Scatter Plot



(b) TRF Scatter Plot



(c) KANs Scatter Plot

Figure 5.11: Comparison of CGP, TRF, and KANs Scatter Plots - k_{deficit} PH

5.4.2. Reynolds Stress Tensor

Table 5.7: Reduced Functional Space for PH Correction in RST - GP

Set Type	Variables
Non-dimensional Set	$S2_s$, $W2_s$, $S3_s$, $W2S_s$, $W2S2_s$, $Ak2$, $Ak2S_s$, $Ak2S2_s$, $Ak2WS_s$, $Ak2WS2_s$, $Ak2SWS2_s$, q_T, q_{CP_B}, q_{tauk_B}, q_γ, q_{Re}
Dimensional Set	T_{1s}, T_{2s}, T_{3s}, T_{4s}, T_{5s}, T_{6s}
Target Variable	b_{ij}^{Δ}

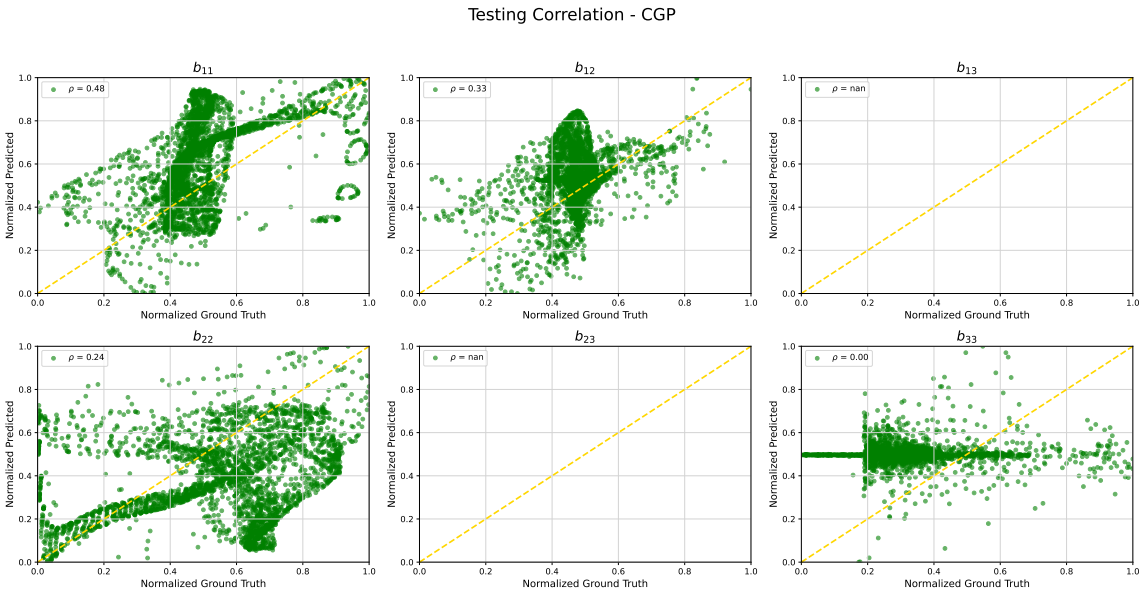


Figure 5.12: CGP Scatter Plot - b_{ij}^{Δ} PH

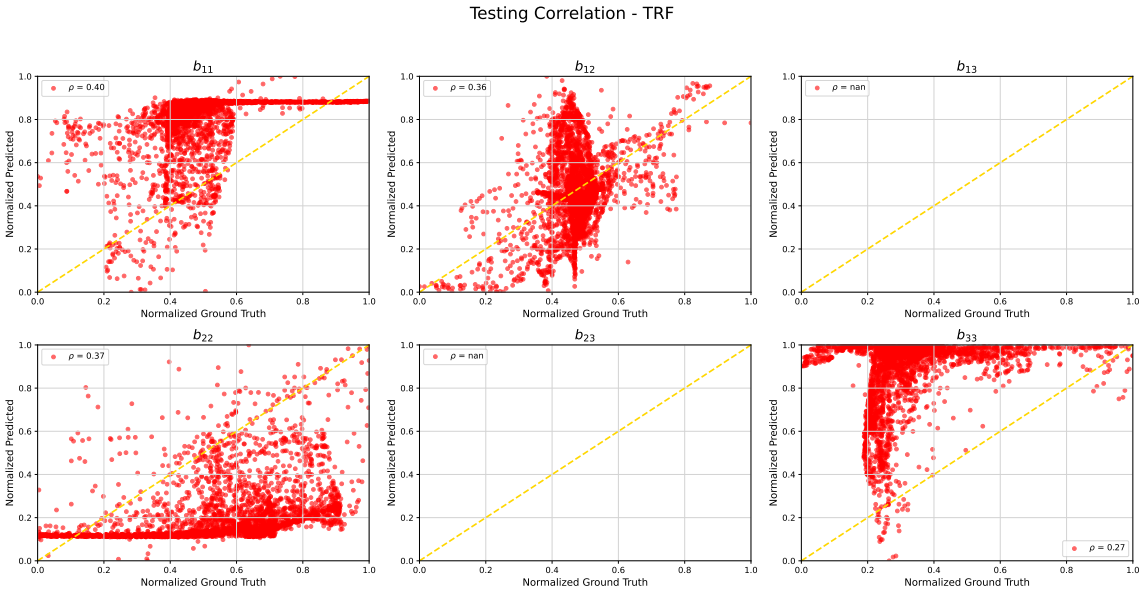


Figure 5.13: TRF Scatter Plot - b_{ij}^{Δ} PH

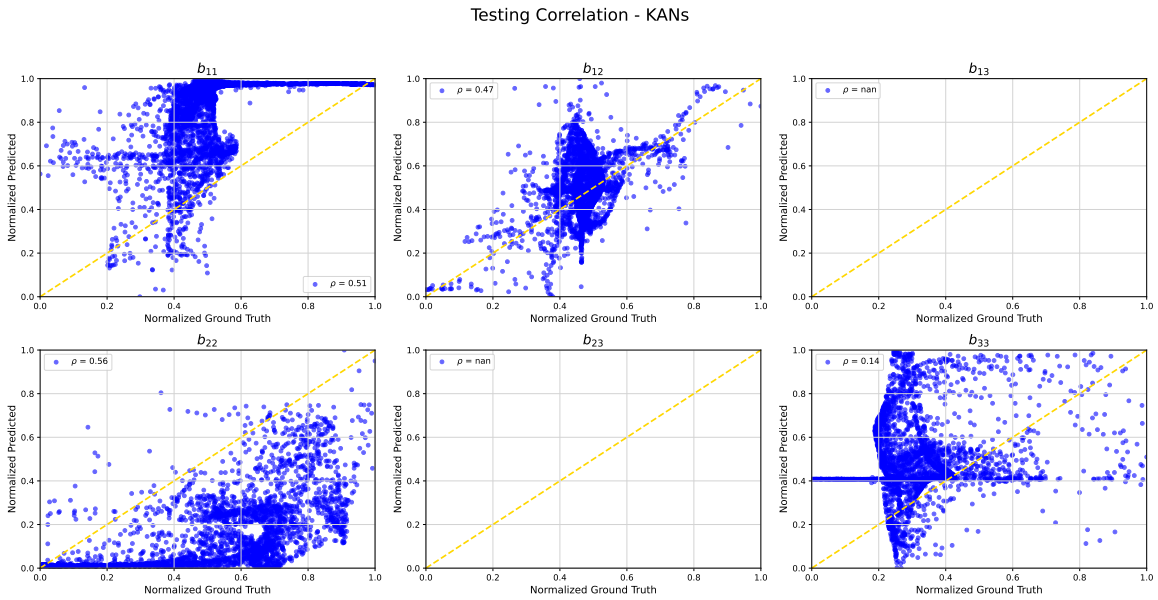


Figure 5.14: KANs Scatter Plot - $b_{ij}^{\triangle}$ PH

The model and the Symbolic formula of all the methods along with the overall correlation of the RST can be referenced from Appendix Section C.6.2. Conclusions are drawn after implementing the algorithms on the Square Duct setup as well.

5.5. Square Duct Testing

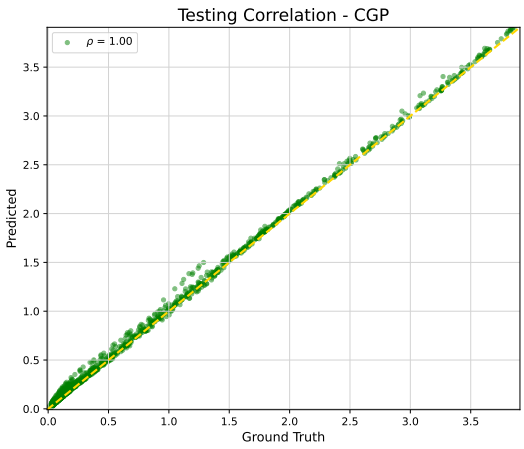
The algorithms were now applied to the Square Duct datasets. The complete function space for this CFD case can be referenced from Appendix Section C.5.

5.5.1. Turbulent Production

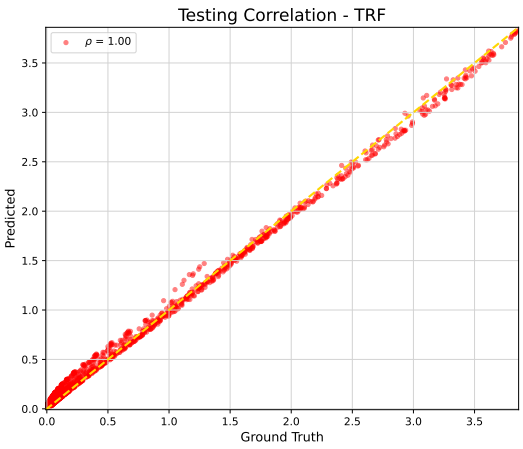
As done for PH, similar pruning of the input space is performed. The reduced space, comprised of the selected invariants, is tabulated below.

Table 5.8: Reduced Functional Space for SD Correction in Turbulent Production - GP

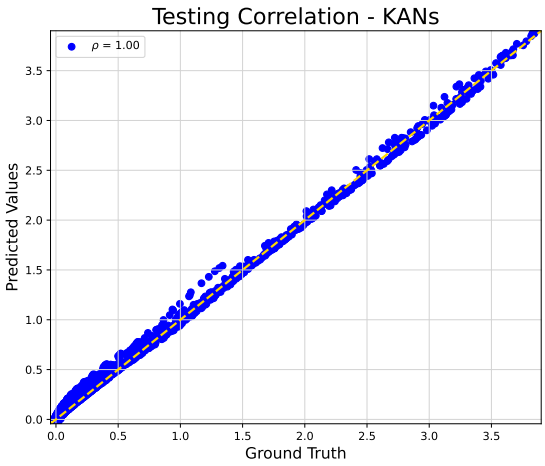
Set Type	Variables
Non-dimensional Set	$\mathrm{S2_s}$, $\mathrm{W2_s}$, $\mathrm{S3_s}$, $\mathrm{W2S_s}$, $\mathrm{W2S2_s}$, Ak2, $\mathrm{Ak2S_s}$, $\mathrm{Ak2S2_s}$, $\mathrm{WAk_s}$, $\mathrm{WAkS_s}$, $\mathrm{WAkS2_s}$, $\mathrm{Ak2WS_s}$, $\mathrm{W2AkS2_s}$, $\mathrm{Ak2WS2_s}$, $\mathrm{Ak2SWS2_s}$, q_T, q_Q, $q_{\mathrm{CP\_B}}$, q_γ, q_{Re}
Dimensional Set	G_{1s}, G_{6s}, ϵ
Target Variable	$\mathrm{k_{deficit}}$



(a) CGP Scatter Plot



(b) TRF Scatter Plot



(c) KANs Scatter Plot

Figure 5.15: Comparison of CGP, TRF, and KANs Scatter Plots - k_{deficit} SD

The Symbolic formula of the methods can be referenced from Appendix Section C.6.3.

5.5.2. Reynolds Stress Tensor

Table 5.9: Reduced Functional Space for SD Correction in RST - GP

Set Type	Variables
Non-dimensional Set	$\mathrm{Ak2}$, $\mathrm{Ak2S2_s}$, $\mathrm{Ak2WS_s}$, $\mathrm{Ak2WS2_s}$ q_γ, q_{Re}
Dimensional Set	T_{2s}, T_{3s}, T_{4s}
Target Variable	$\mathrm{b}_{\mathrm{ij}}^{\Delta}$

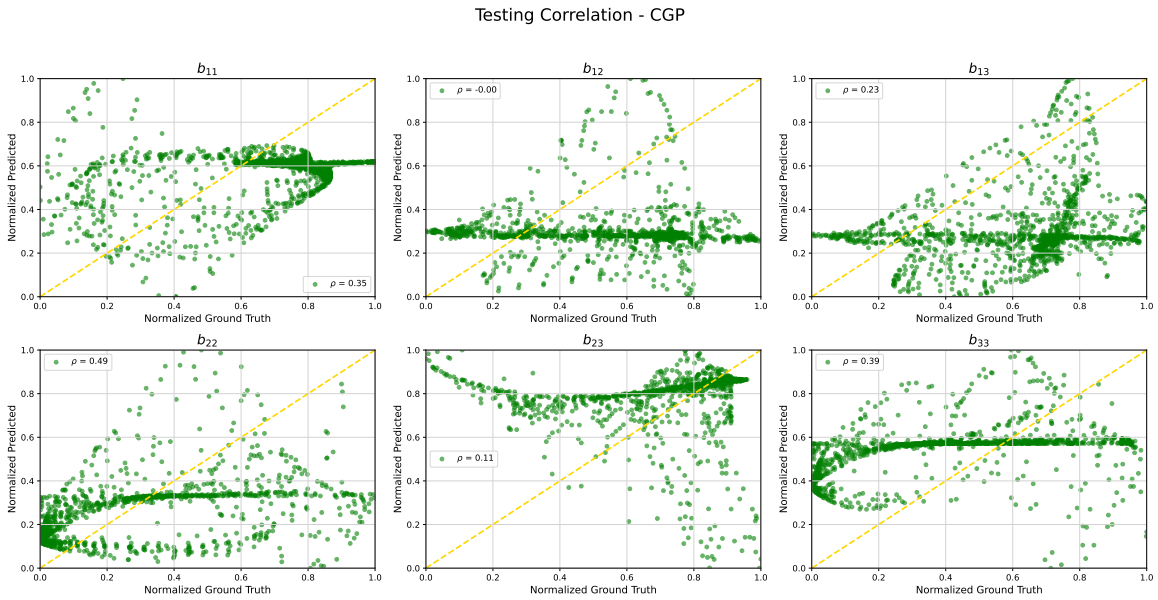


Figure 5.16: CGP Scatter Plot - b_{ij}^{Δ} SD

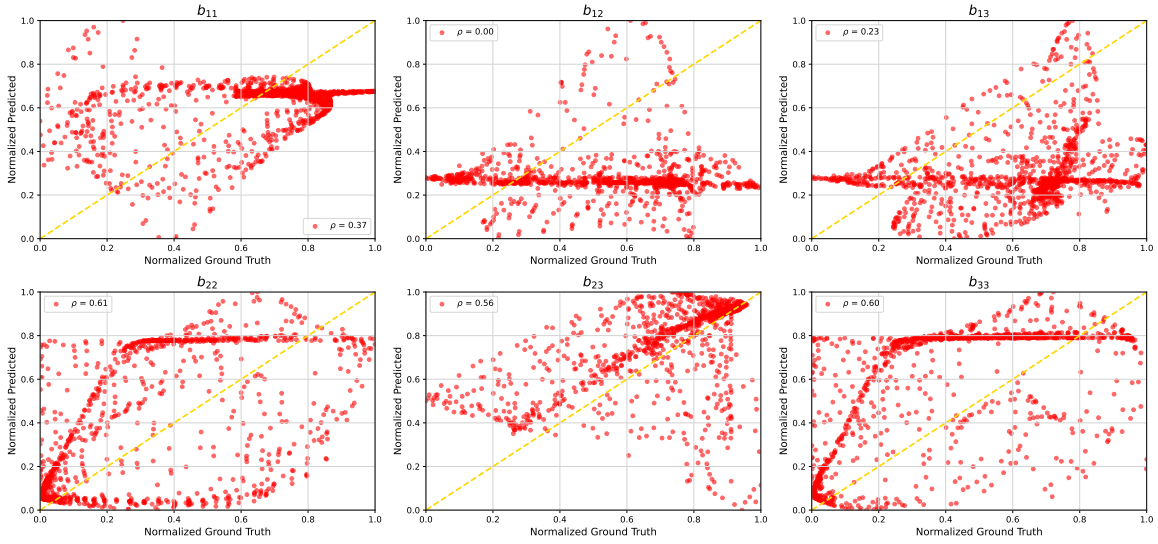


Figure 5.17: TRF Scatter Plot - b_{ij}^{Δ} SD

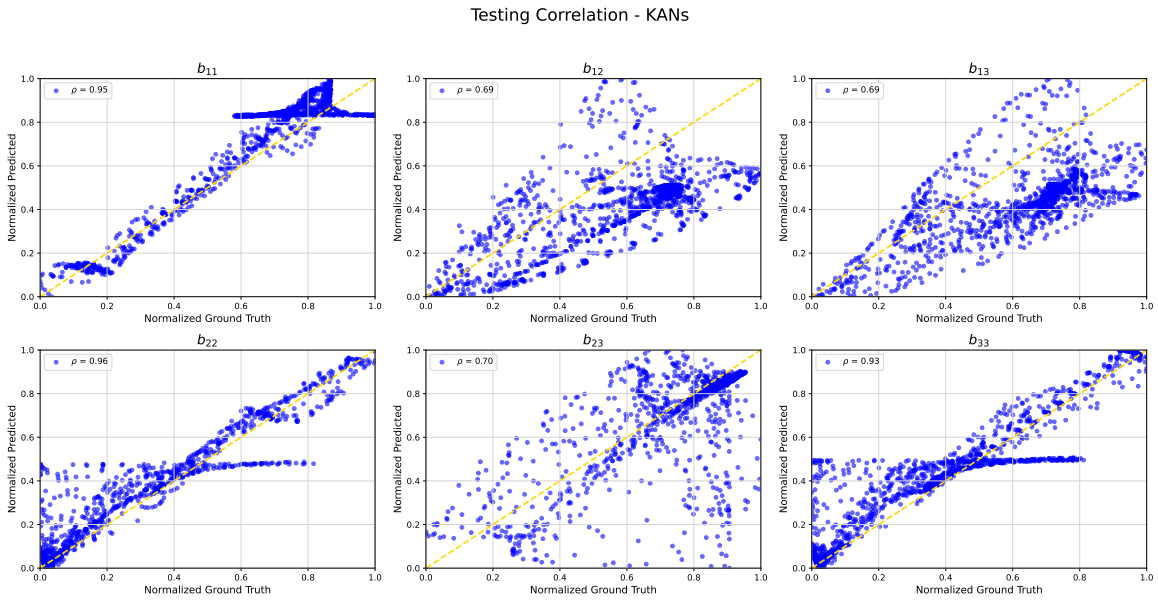


Figure 5.18: KANs Scatter Plot - b_{ij}^{Δ} SD

The KANs model and the symbolic formula of the methods can be referenced from Appendix Section C.6.4.

5.6. Performance Comparison

The following tables summarize the performance of the algorithms. The correlations with testing data and the function count is reported, as they directly relate to the research questions mentioned in Section 2.9.

Table 5.10: Performance Comparison of Different Methods - PH

Case Setup	Quantity	KANs	GP + TRF	CGP	Metric Type
Periodic Hill	k_{deficit}	0.988	0.979	0.981	Correlations
	b_{ij}^{Δ}	0.782	0.739	0.721	
	k_{deficit}	1651	2871	4229	Function Evaluations
	b_{ij}^{Δ}	2793	3386	4889	

Table 5.11: Performance Comparison of Different Methods - SD

Case Setup	Quantity	KANs	GP + TRF	CGP	Metric Type
Square duct	k_{deficit}	0.998	0.998	0.999	Correlations
	b_{ij}^{Δ}	0.994	0.962	0.958	
	k_{deficit}	612	2746	4118	Function Evaluations
	b_{ij}^{Δ}	1245	3016	4130	

Table 5.12: Percentage Improvement Comparison of Different Methods - PH

Case	Quantity	KANs (%)	GP+TRF (%)	Metric Type
Periodic Hill	k_{deficit}	+0.71	-0.20	Correlation
	b_{ij}^{Δ}	+8.46	+2.49	
	k_{deficit}	+60.9	+32.1	Function Evaluation
	b_{ij}^{Δ}	+42.87	+30.74	

Table 5.13: Percentage Improvement Comparison of Different Methods - SD

Case	Quantity	KANs (%)	GP+TRF (%)	Metric Type
Square Duct	k_{deficit}	-0.1	-0.1	Correlation
	b_{ij}^{Δ}	+3.75	+0.42	
	k_{deficit}	+85.13	+33.31	Function Evaluation
	b_{ij}^{Δ}	+ 69.85	+26.97	

In the tables, $+$ signs indicate an improvement and a $-$ sign indicates a deterioration for that metric.

5.7. $A\ priori$ Conclusions

The testing of the algorithms is now completed and the results produced by them certainly outperform the benchmark case (CGP). Among the tested algorithms, KANs are the best-performing algorithms and able to produce consistently improved results. A detailed conclusion will be made for each of the algorithms, highlighting their capabilities and limitations. Keeping in mind the research questions for this thesis, Section 2.9, and further work related to $a\ posteriori$ studies, the following conclusions are made.

- Genetic programs incorporated with information from gradients indeed exhibit accelerated convergence to excellent correlations. The number of times the objective function is evaluated is reduced significantly for all the quantities.
- While the generations are reduced by half, a reduction in the function count $>\approx 33\%$ is not observed. Although the genetic program itself calls the objective function only $\approx 50\%$ of the times, the added function calls are made by the TRF optimizer to tune the coefficients and the operator weightings of the individual, thus accounting for nearly $1/3^{rd}$ of the total function calls by the $GP + TRF$ method.
- The converged individuals from the mimetic algorithms, though exhibiting strong correlations, are weakly dependent on the flow-derived feature set. Most of the trees were only scalars as opposed to conventional GP which had a stronger dependence on the invariants. This is not favorable as the individual contributions of the tensors now lack any physical meaning.
- KANs are able to produce improved results for each quantity and each test case. Unlike the $GP{+}TRF$ method, correlations for RST are also improved by 3.5% at the least. Even for turbulent production, while $GP + TRF$ only managed convergence to nearly identical correlations, KANs were able to get better results. This guarantees an improvement in the turbulent flow modeling where either of the modeling errors are dominant.
- The decrease in function count is remarkable. Although using LBFGS proved to be expensive for GP, KANs were able to effectively use it and converge rapidly. Especially for turbulent production, it requires only 15% of the function evaluations to that of CGP.
- This can be explained by the input pruning. It is more effective than `mutual information` at selecting out features which the target variable would significantly depend on. Its capability can be seen from the Square Duct case. With an original input space of 25 invariants, it selects only 2 and 4 inputs for k and b_{ij}^{Δ}, respectively. This reduces the search space significantly and helps converge to excellent solutions very quickly. Thus, it can be said that as many features can be

given, but pruning allows only the most important features to be used and form an efficient solution space to model the target variable. The curse of dimensionality (input space) does not pose a problem here.

- The activation functions are able to capture the complex (non-linear) relations between different quantities. In addition, the resulting expressions are strongly reliant on the input space which give it the physical interpretability advantage, as opposed to $GP + TRF$.
- However, the symbolic expressions are very lengthy and composed of `sin` and `cos`, with the possibility of the coefficients lying on extreme ends. This can produce some high frequency functions and make it susceptible to oscillations and divergence when inserted into a CFD solver. However, when visualizing the symbolically active KANs, it can be seen that such high frequency activations are absent. Nonetheless, the risk still remains.
- With regards to flow physics, correlations for the correction in TKE were nearly perfect. For RST and its components however, all the model struggled. It was known beforehand from the author's research group that regressing for b_{ij}^{Δ} as a whole for PH is challenging. In addition, although the overall correlations of the tensors for SD case remains quite high, the component correlations are not.
- For a PH, it is a 2D case setup. Hence, the tensor components b_{13} and b_{23} are zero, and give NaN correlations. Each of the models show good correlation with the turbulent normal stress (anisotropy) in the streamwise direction. KANs was however, also able to capture the trend in the turbulent normal stress in the wall-normal direction and has a higher correlation. In addition, they also have a better correlation with the diagonal components of the tensor, thus they can further improve the turbulent energy distribution than $GP + TRF$. The correlations for off-diagonal terms in PH are also more improved in KANs. It can be expected that they better improve the shear stresses and momentum transfer in the streamwise-normal direction. Thus, It further improves the production of the turbulence model and helps in more improved modeling of the separated flow region.
- SD is a 3D setup and exhibit secondary flows due to anisotropy in the RST. Secondary vortices and corner effects are better captured if the RST modeling (momentum transport) is improved. Once again, the anisotropy distribution or structure in x, y, z direction is significantly better captured by the KANs model. In this case, b_{13} and b_{23} is more important for the out of plane momentum transfer, subsequently also for secondary vortices. The KANs model is significantly better at capturing this effect.

This concludes the 1^{st} part of the thesis where the frozen or $a\ priori$ study was performed. The thesis began with the objective to develop or use a method that can accelerate quickly (reduced function calls) to excellent correlations while also produce a physically meaningful symbolic formula for the corrections in Reynolds Stress Tensor and Turbulent Production. It is now completed and comparing the methods, KANs emerge as the superior candidate. It is shown that given a high-fidelity data, KANs can be used to very efficiently develop models (that can be inserted into CFD models) to improve flow modelling. Although the models developed were corrections for k and RST, they can effectively be used for any other flow (turbulent) quantity as well.

Note that Pearson's correlation coefficient was used to evaluate each method's ability for the quantities. We were interested in the model's ability to reproduce the spatial distribution and directional alignment of the RST components as it measures how the variables are linearly aligned. R^2 can be negative in poor models making it difficult to interpret if the model is even slightly better or not. The difference between ρ and R^2 for turbulent kinetic energy was not significant 10^{-2}.

6

A posteriori CFD Results

This chapter of the thesis explains the application of the $a\ posteriori$ framework with KANs to develop these correction terms while taking information by differentiating the CFD code. The objective will be to determine if they can be used to improve flow modeling without compromising the stability of the CFD solver and produce a generalizable model. The methodology has been explained in Section 3.6.

6.1. Initial $a\ posteriori$ testing

The Spalart-Allmaras Field Inversion model was used that incorporated the correction field from KANs into the turbulence model. Equations (3.23) to (3.29) describe the invariants. Note that pruning was not implemented during the optimization process. Hence, to adopt the most simplest network, pruning of the inputs was performed beforehand. All the 6 invariants were given to a KAN network utilizing 1 hidden layer of 4 neurons and 1 run of optimization was performed followed by pruning. The resulting pruned network consisted of the 4 invariants from the converged $Spalart - Allmaras$ solution field: RITA$_{D_{est}}(q_{dest})$, RITA$_{D_{iff}}(q_{diff})$, RITA$_{P_{rod}}(q_{prod})$, and q_ν. Therefore, only these 4 invariants were used for the entire training process and other cases.

Ideally, we would want to have only 1 neuron in the hidden layer capable of using the inputs to generate the appropriate corrective field, thus having the least number of parameters to optimize. However, it was often diverged. With 4 neurons in the hidden layer, the $a\ posteriori$ process was found to converge most of the times, hence it was decided to go ahead with it. After completing the differentiation & machine learning frameworks, the initial $a\ posteriori$ testing was performed on a 3500 Periodic Hill mesh, which also serves as the training case for the study. This is also used by the publishers of DAFoam for their own optimization of $Spalart - Allmaras$ to $k - \epsilon$ using the `OpenMDAO` framework where the correction field β was defined as a spatially varying field (scalar). The mesh can be referenced from Appendix Section D.1 and its characteristics are presented below.

Table 6.1: y^+ Statistics - initial $a\ posteriori$ testing testing

Statistic	Value
Minimum	0.014
Maximum	0.700
Mean	0.332

The y^+ is shown for the bottom wall which is that actual region of improvement for the turbulence model. Tolerances of 10^{-8} and 10^{-3} were selected for the CFD solver and the optimization process respectively.

The distribution of the invariants is shown below.

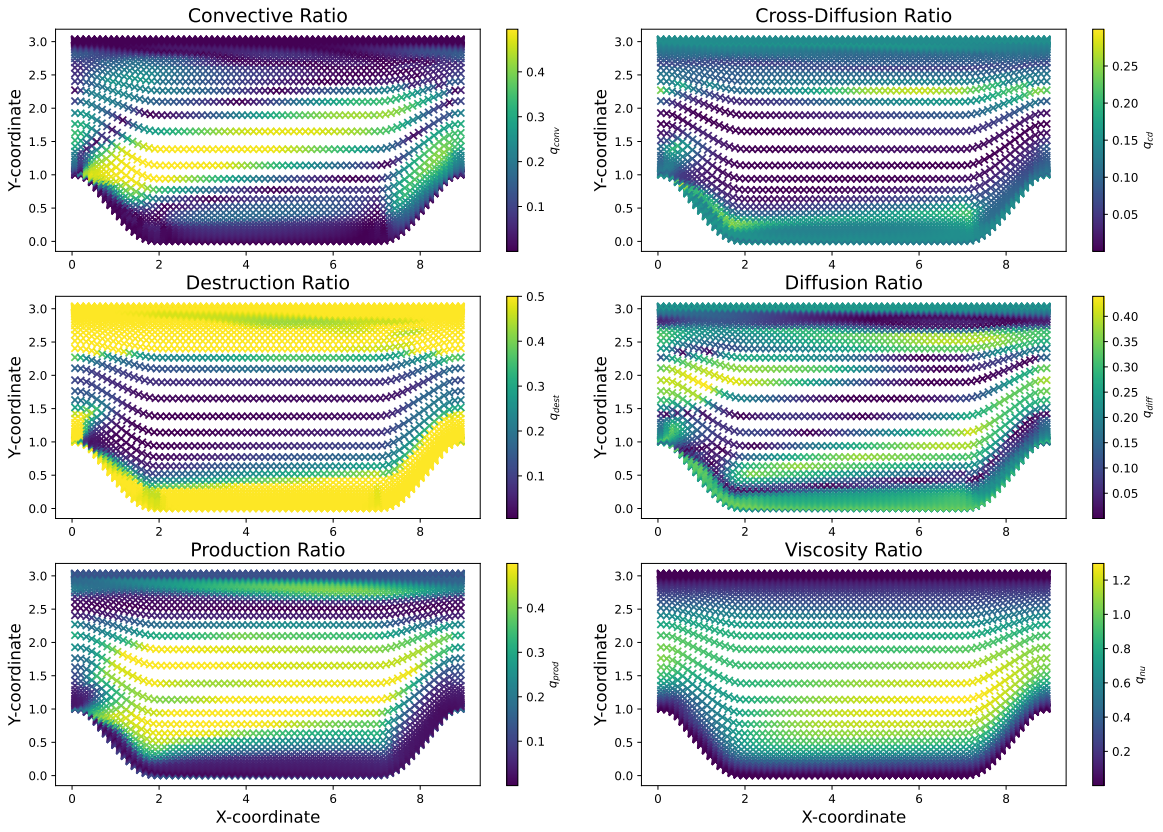


Figure 6.1: 3500 mesh Periodic Hill - Invariant Distribution

The convective and diffusion ratios represent transport mechanisms. Soon after the inlet the fluid accelerates and in the separated shear layer, the velocity is higher resulting in the convective transport becoming relatively dominant. Similarly, the fluid again begins to accelerate upstream of the second hill causing higher convection. Close to the solid walls (top & bottom), the velocity vanishes due to the no-slip condition thus making convective ratio the weakest in that region. In the separated region, the convective ratio also reaches minimal values due to low mean velocities.

Both the diffusion and cross-diffusion ratio are relatively higher at the walls. Close to the walls (boundaries), there is a sharp increase in the velocity gradient which increases molecular diffusion and cross-diffusion due to mixing and the spread of turbulence. In the core of the flow and inside the separated bubble, the ratios have lower magnitudes because of reduced turbulence variation, lower eddy viscosity and velocity gradients. The diffusion ratio near the outlet again enhances due to flow reattachment occurring that again enhances the velocity gradients and turbulence generation. The cross-diffusion region highlights interactions between momentum & scalar transport, and depends on the gradient. Hence, it becomes prominent in regions of strong acceleration and high curvature.

At the top wall, the flow remains attached and fully developed. The turbulent viscosity is not increasing, the mean velocity gradients remain low and a boundary layer is developed. This results in the relative dominance of destruction ratio, and lower production. The production ratio reaches high values at the shear layer and reattachment regions that exhibit strong velocity gradients, thereby increasing turbulence. Inside the separation (recirculation) region downstream of the periodic hill, the flow does not exhibit strong shear and velocity gradients. Hence, destruction is stronger and production is weaker. It was interesting to note the region of high production and low diffusion close to the top wall.

The viscosity ratio is minimal at the top and bottom wall as ν_t approaches 0 close to the walls due to the no slip & boundary conditions, resulting in low turbulence generation. However, in regions of separation and reattachment, turbulence production is strong, which in turn also increases the turbulent viscosity, and hence the viscosity ratio as well.

These ratios can be explained by flow physics and give insights into how the different turbulent quantities are distributed, and helps in identifying the dominant physical processes occurring in the flow. Thus, when used to develop the β field, it provides a more physically meaningful correction that also aligns with the flow and can be explained by these invariants.

6.1.1. Results of Initial *a posteriori*

Before visualizing the optimized velocity field and flow quantities, the optimization process is analyzed. The convergence of the objective function is shown below.

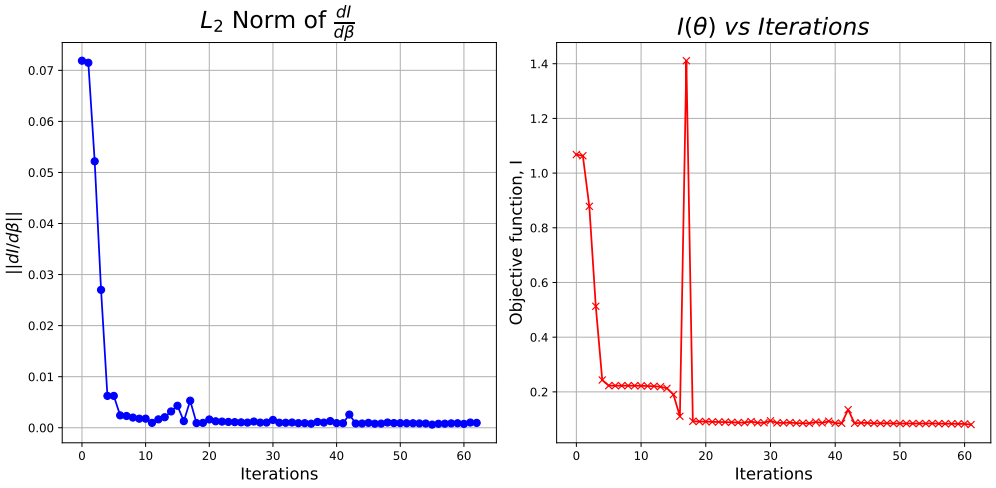


Figure 6.2: Optimization Convergence

The left figure shows how the velocity field depends on the correction field, while the right figure shows how the objective value is minimized over the optimization process. The optimization process is quick to converge. In 63 iterations, a convergence of 10^{-4} is attained. Although the objective value does deteriorate at one or two instances, the overall optimization process is stable and quick. From the left figure, initially, the $\mathcal{L}2$ norm is higher indicating that the velocity field that needs to be optimized is sensitive to the correction field and that it can be easily improved. As the optimization progresses, the $\mathcal{L}2$ norm reaches 9.5×10^{-3} indicating that the optimized correction field is no longer able to significantly influence the velocity field and that it has already been optimized.

To visualize the flow field and the extent of optimization, velocity (U_x) profiles, coefficient of friction and the correction field is presented below. The converged correlations, flow streamlines and model details can be referenced from Appendix Section D.1

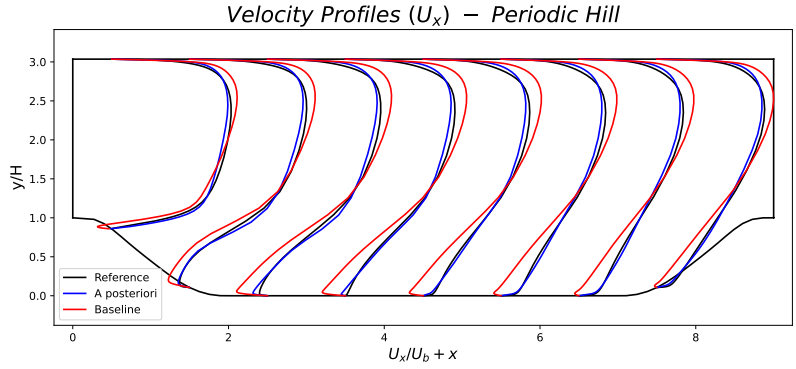


Figure 6.3: Velocity Profile Comparison across the PH domain

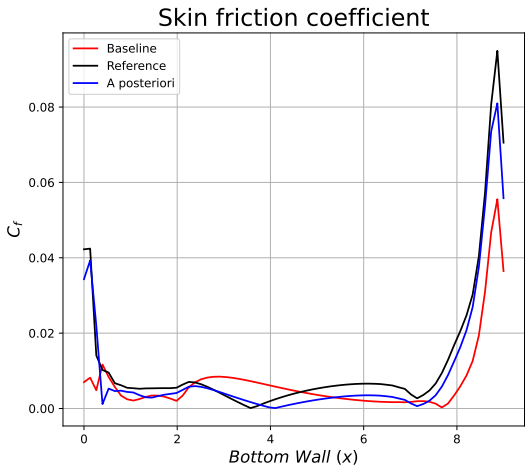


Figure 6.4: Skin friction coefficient Comparison across the PH bottom wall

Figures 6.3 and 6.4 illustrate the axial velocity profiles and C_f in the regions of interest. U_x is divided by the bulk velocity to provide an improved visualization of the profiles. The same turbulence model, after inserting the correction field, is now able to accurately capture the flow separation region which was being over-predicted by the baseline model and this is also supported by the C_f plot. It can be observed that the $a\ posteriori$ model closely follows the reference C_f curve and the the point of reattachment is now very close to the reference case. C_f is calculated as

$$C_f = \frac{|\tau_w|}{\frac{1}{2}\rho U_b^2}. \tag{6.1}$$

The initial decrease in C_f is due to flow deceleration and a strong adverse pressure gradient causing the flow to separate from the wall $(\mathbf{x} \approx 0)$. The $a\ posteriori$ model is now able to accurately predict it. In the recirculation zone downstream of the hill at $(x \approx\ 0.2\ -\ 2)$, the C_f remains relatively constant. The prediction of the reattachment point from the $a\ posteriori$ model is significantly improved, now at $x = 4.1$ compared to $x = 3.57$ from reference. This is in contrast to the baseline model where the reattachment location is predicted at $x \approx\ 7.67$. After this first reattachment location, there is favorable pressure gradient and the boundary layer develops. This is predicted by the $a\ posteriori$ and reference cases where the C_f shows a hump for a short distance until $x \approx\ 6$, but is completely missed by the baseline model. As the flow approaches the second hill, adverse pressure gradient develops and the boundary layer decelerates. Both the $a\ posteriori$ and reference have a local minimum at $x \approx\ 7.14$ trying to capture the secondary incipient separation region which would have been captured by a higher-fidelity model such as LES, as seen in [40]. Nonetheless, the baseline model again completely fails to capture this. As the flow reaches just upstream of the second hill, it accelerates which is shown by the sharp increase in C_f, with $a\ posteriori$ modeling it accurately.

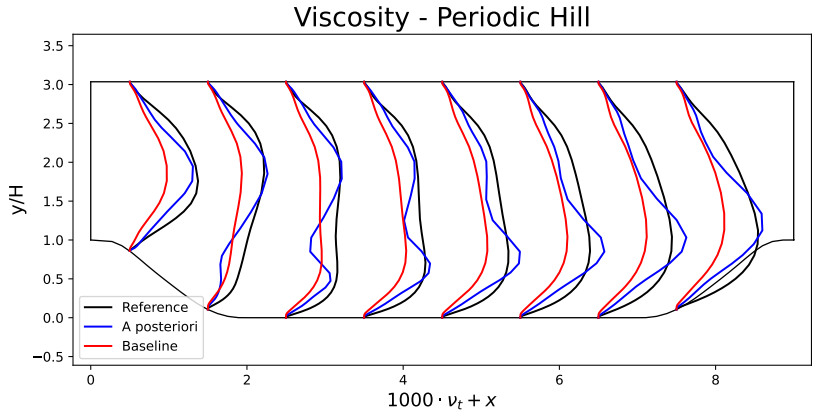


Figure 6.5: Viscosity Profile Comparison across the PH bottom wall

The viscosity profiles are a better representative of the turbulence modeling and it can be seen that they are not as improved as the velocity profiles. Across the bottom wall, there is improvement in the viscosity implying that it is able to better model turbulence (production, destruction & mixing) and enhances momentum transfer near the wall, which is essential in predicting the separation and reattachment points. Along the top wall, the first half has an improved viscosity while the latter half sees almost no improvement. The improved model also suffers in the shear layer, downstream of the first periodic hill where it exhibits large fluctuations. However, upstream of the second periodic hill, the modeling improves. This suggests that the *a posteriori* model is able to capture the near-wall turbulence up to some extent, but still struggles to model it in regions of strong velocity gradients and away from the wall. Thus, although the model is able to get the correct mean flow suggesting improved turbulence modeling, the viscosity profiles indicate otherwise. It can be inferred that the mean flow improvements may be due to the correction field dominating the turbulence modeling and that turbulence is still not accurately resolved as would be preferred.

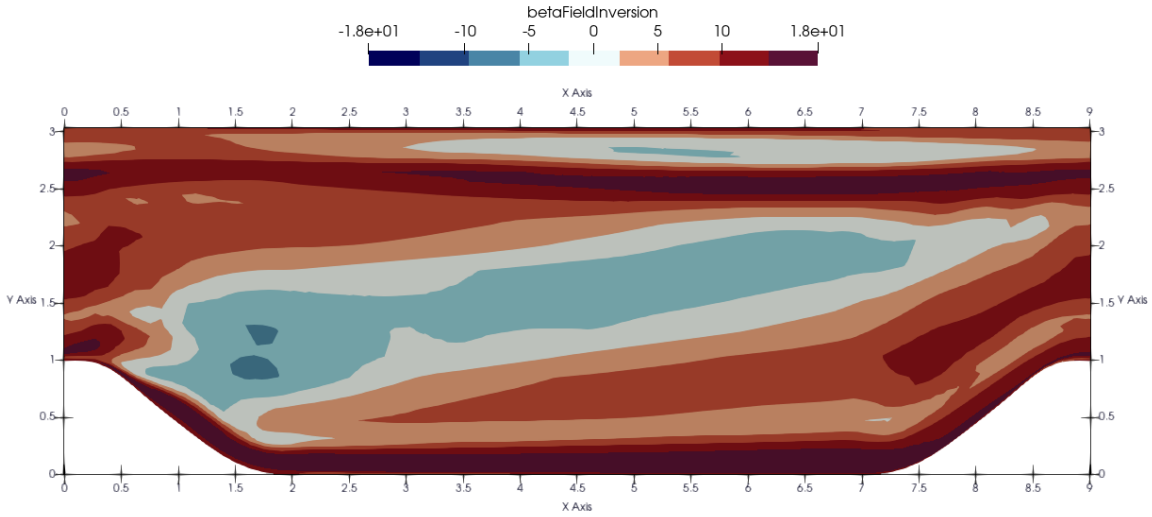


Figure 6.6: β field

The values of the β field imply that there are regions where production is being enhanced by as much as 18 times and reduced significantly as well. The range strongly indicates an overfit solution. It can be said the turbulence modeling is dominated by this correction field and not the transport equation itself.

At the inlet and outlet regions, the turbulence production increases, but is more significant along the bottom wall. As seen from Figure 6.5, the baseline model under-predicts ν_t close to the inlet. The

strength of the shear layer is weak resulting in too long a separation bubble. The β field corrects this by enhancing production locally, thus matching the velocity profiles. In the baseline model, the reattachment point is located close to the second hill. To correct this, there is a large production increase as this results in increased ν_t. With this, the diffusion increases and there is more effective momentum transfer thus reducing the size of the separation bubble. This is also supported by Figure 6.5 where the viscosity along the bottom wall is much improved. In the attached flow above the bottom wall, the baseline model does sufficiently good in modeling the flow, hence the corrections are light. However, in some regions, the correction modifies the production unnecessarily. In the recirculation region, β is negative implying more destruction and is the region where the β field struggles to improve turbulence modeling. The production is reduced such that the resulting shear stresses are able to improve the velocity profiles while compromising the viscosity, thus not producing physical results. Similar behavior can be observed at the top wall.

Overall, the $a\ posteriori$ shows very good agreement of the velocity profiles and C_f with the reference case. The prediction of reattachment point is much more accurate now, and also correctly capturing the flow acceleration (& deceleration). Although the representation of the near-wall flow is improved up-to some extent as seen from the viscosity profiles, when observing the complete profiles, it indicates that turbulence modeling itself is not significantly improved. From the β field, it can be said that the optimization is very strong and only focuses on matching the velocity fields while the turbulence modeling is compromised. The β modifications are more localized and counteract each other with the extreme values. Even though these effects are strong enough to correct the velocities, the net effect on ν_t overall remains small.

The next section explores solutions to mitigate this overfitting issue and have a β field that lies in acceptable ranges.

6.2. Regularization Analysis

This section gives details on the regularization penalty that was implemented in the objective. To penalize large values of β and get a correction field that can be easily generalizable, squared $\mathcal{L}^2$ norm of the β field was added in the objective function. The objective function can now be written as

$$\mathcal{I} = \underbrace{\sum_{i \in \Omega} \left\| \mathbf{u}_i - \mathbf{u}_i^{\text{ref}} \right\|^2}_{\text{Velocity } \mathcal{L}^2 \text{ norm}} + \underbrace{\sum_{i \in \Omega} \beta_i^2}_{\beta\ \mathcal{L}^2 \text{ norm}}$$

This has now become a multi-objective optimization problem with the parameters of KANs (θ) being optimized so as to match the velocity field with reference data by using the numerically smallest β field possible. However, according to this implementation, one of the costs can easily dominate the other cost function and produce unfavorable results. Hence, to control the effect of the β $\mathcal{L}2$ norm, a regularization parameter λ was used, which modifies the objective as

$$\mathcal{I} = \underbrace{\sum_{i \in \Omega} \left\| \mathbf{u}_i - \mathbf{u}_i^{\text{ref}} \right\|^2}_{\text{Velocity } \mathcal{L}2 \text{ norm}} + \underbrace{\lambda \cdot \sum_{i \in \Omega} \beta_i^2}_{regularized\ \beta\ \mathcal{L}2 \text{ norm}} \quad ,$$

where λ controlled the contribution of β in the overall objective function. It was not known beforehand which value of λ would be optimal. Hence, by selecting values in the range $[10^{-4}, 10^{0}]$, a regularization study was performed where an appropriate value of λ was determined. The $a\ posteriori$ process remains the same with modifications only in the objective function. The results are presented below.

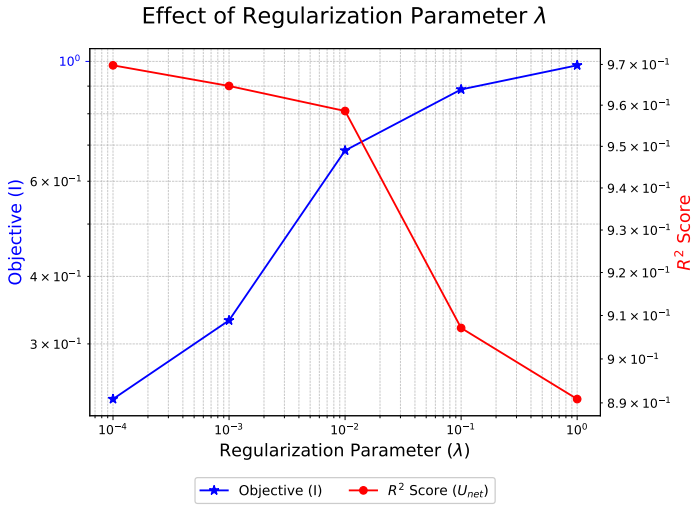


Figure 6.7: Effect of regularization of β field on the velocity field

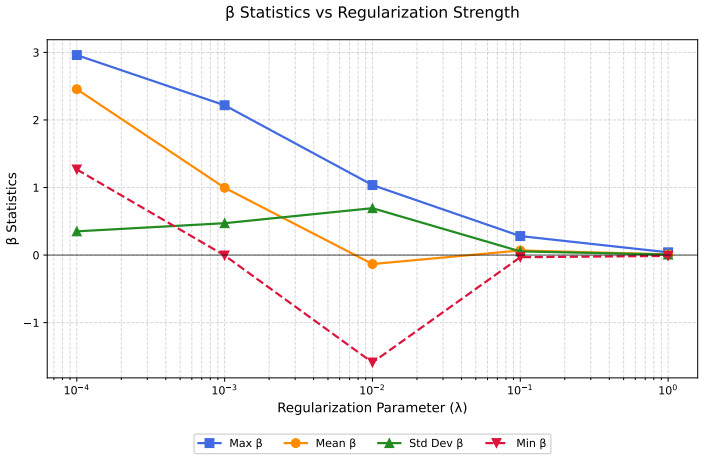


Figure 6.8: β field statistics

Figure 6.7 tells us about the effect of penalizing the correction field, β. When the regularization is high, $(\lambda = 1)$, the optimizer finds it difficult to optimize the velocity field and converge to a correlation > 0.9. The regularization error dominates the objective and the optimizer prefers smaller β values even at the cost of a higher velocity field error. KANs exhibit a very limited ability or parameter space to reduce the velocity error and the β is confined within a small space as attempts to explore other spaces get penalized. This can effectively be seen from Figure 6.8 as well. At $\lambda = 1$, all the statistical quantities of β converge to a single point.

However, at lower λ values, the optimizer has more freedom to adjust β and explore large values. This reduces the error between the velocity fields and the objective drops due to reduced penalization. This is also supported by Figure 6.8. At lower λ values, β starts to take extreme values and has a higher mean value resulting in overfitting issues and lack of robustness. The error between velocity fields starts to dominate the objective.

Looking at Figure 6.7, $\lambda = 1$ is where the optimizer struggles to find a good β field and $\lambda = 10^{-4}$ is where the risk of overfitting becomes significant as the training correlations become very high. Ideally, $\lambda = 10^{-2}$ is the optimal regularization value as it strikes a good balance between overfitting and underfitting, without compromising generalization. However, the optimizer at this regularization penalty diverges. Hence, the next best is chosen as the most suitable regularization parameter, $\lambda = 10^{-3}$

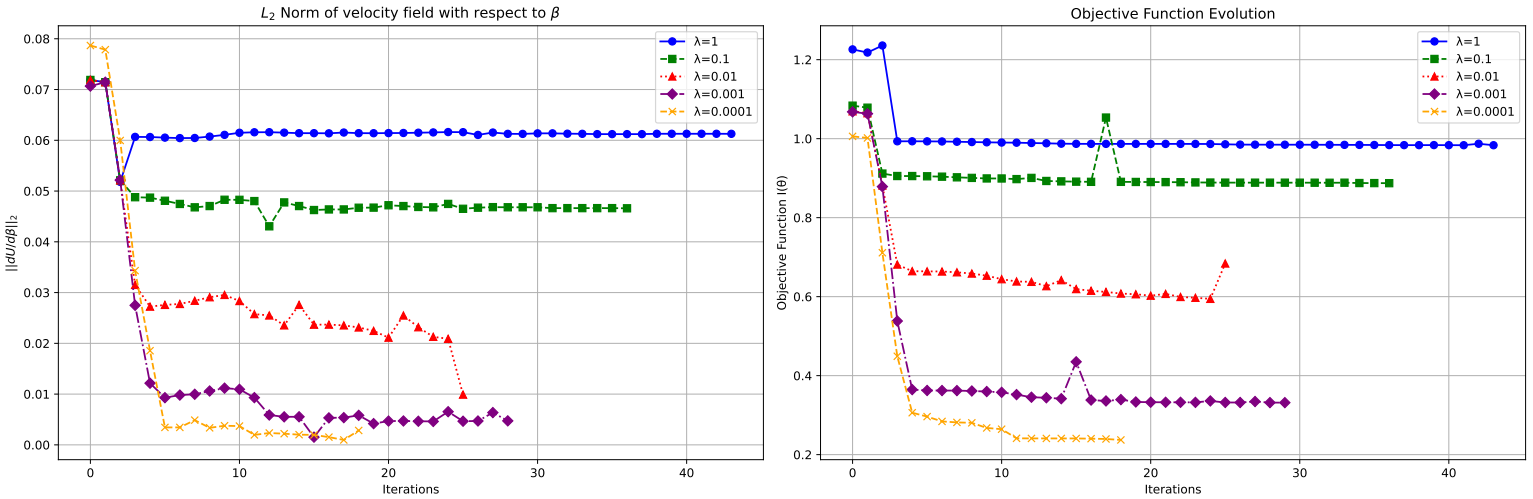


Figure 6.9: Effect of regularization on overall optimization process

Figure 6.9 describes the convergence history of the optimization process. It can be seen that at higher regularization, the optimization is stable as the search is conservative. KANs are penalized heavily to change β drastically and remain in a confined parameter space. The functional space is such that the sensitivity of velocity with the β field remains large and constant. Although the $\mathcal{L}^2$ norm remains higher thus implying that the velocity field can change significantly if the correction field changes, KANs inability to produce a good β field due to regularization limits this.

At $\lambda = 10^{-2}$, divergence is observed. At iteration 24, the optimizer takes a step that generates a β field that reduces the velocity error. However, the optimizer is penalized by the regularization and unable to recover, the optimization process diverges. The unstable $\mathcal{L}^2$ norm indicates that β can cause potentially non-physical adjustments to the flow. At lower λ values, the $\mathcal{L}^2$ norm also converges to a lower value. This indicates that a correction field is learnt such that the flow is not strongly dependent on the β field anymore. Ideally, this is desired as this means the velocity field will not change with further changes in β and reaches a converged state. However, this is also where the optimization risks overfitting the correction field. In general, a lower converged sensitivity is preferred as the correction becomes efficient and the velocity field is not very sensitive to it. This further supports our decision to select $\lambda = 10^{-3}$ as the velocity field has a lower converged sensitivity to the β field, in addition to smooth convergence.

It is also seen at lower regularization, the $\mathcal{L}^2$ norm of velocity field is no longer as smooth as β now moves into regions where the flow is easy to change. It is seen at instances that the $\mathcal{L}^2$ norm may fluctuate while the objective remains stable. $\mathcal{I}$ can remain stable as it is a sum of two errors, but it can also cover the instability in the sensitivity. The optimizer now prefers fitting the data over small values of β which can lead to less stable changes in the sensitivity when mapping β to u. In addition, it was observed that there is no unique solution. There can be more than one β field capable of producing the same velocity field overall. However, depending on the magnitudes, the β field is then penalized which guides the optimization.

6.3. Final *a posteriori* training and testing

With the regularization parameter implemented, a correction field was formulated by optimizing the same periodic hill geometry, thus serving as the training case. When completed, the correction field (from the symbolic formula) was computed and used for different PH geometries at the same mass flow rates. These served as test cases to verify if our model can be generalizable and improve turbulence modeling.

6.3.1. Training Case

Similar to the presentation in Section 6.1, the velocity and viscosity profiles are presented & discussed along with C_f and the β field. The symbolic formula, converged model and the streamlines can be referenced from Appendix Subsection D.2.1.

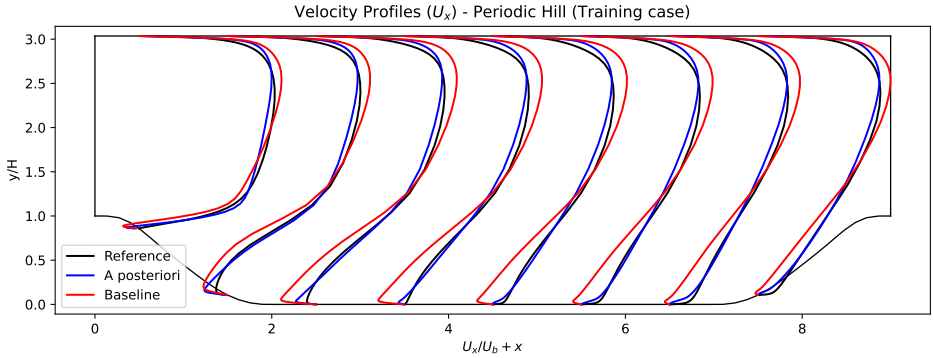


Figure 6.10: Velocity Profile Comparison across the PH domain - training with λ

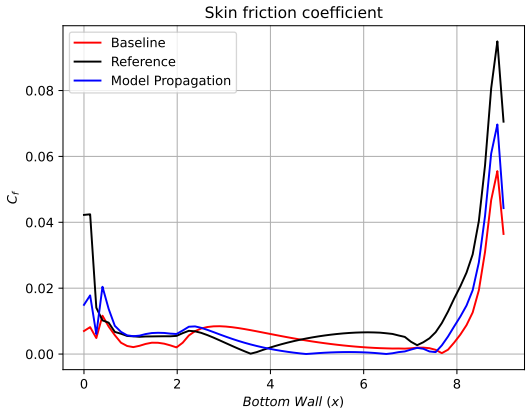


Figure 6.11: Skin friction coefficient Comparison across the PH bottom wall - training with λ

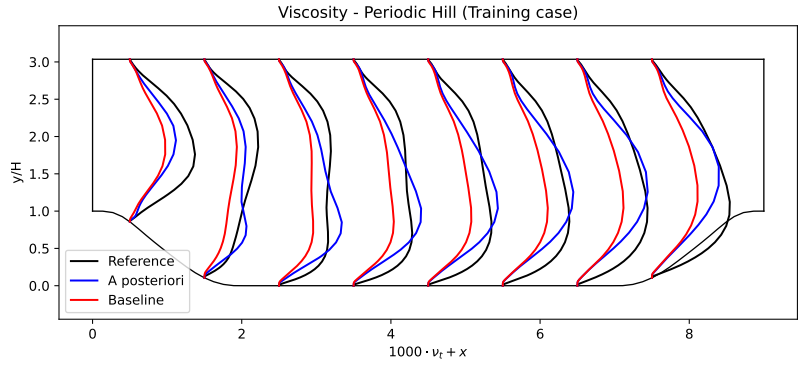


Figure 6.12: Viscosity Profile Comparison across the PH domain - training with λ

Although the improvements in profiles from Figures 6.10 and 6.12 are not as significant as in Section 6.1, the results are better. The velocity profiles are improved the most. In the separation region, the velocities match accurately and are no longer under-predicted. However, as we move towards the

top wall, slight deviations can be observed from the reference profiles. Even with regularization, a correlation of 0.965 was obtained for U_{net}. With regards to C_f, the model is unable to capture accurately the adverse pressure gradient causing the flow to decelerate and separate. However, the prediction remains accurate in the recirculation zone until $x \approx 2$. The prediction of reattachment point is improved with $a\ posteriori$ predicting it at $x = 4.76$ compared to $x = 3.57$ from reference. After attachment the improved model is unable to completely capture the effect of favorable pressure gradient where the boundary layer develops before the flow again decelerates (adverse pressure gradient) due to the presence of the second hill. The C_f due to acceleration of the flow, upstream of the second hill, is much improved.

The viscosity is much improved and no longer exhibits oscillations. Very close to the boundaries, the model struggles to improve the viscosity. But significant improvements can be observed in the core region of the flow. Although viscosity is over-predicted in the shear layer that develops due to flow separation, the behavior becomes accurate as we move downstream of the first hill. The model still finds it difficult to optimize the viscosity in the region close to the top wall where flow is slightly disturbed from the flow at the bottom wall, if at all. The trends are consistent with Figure 6.5 except now the profiles are more smoother and physically meaningful.

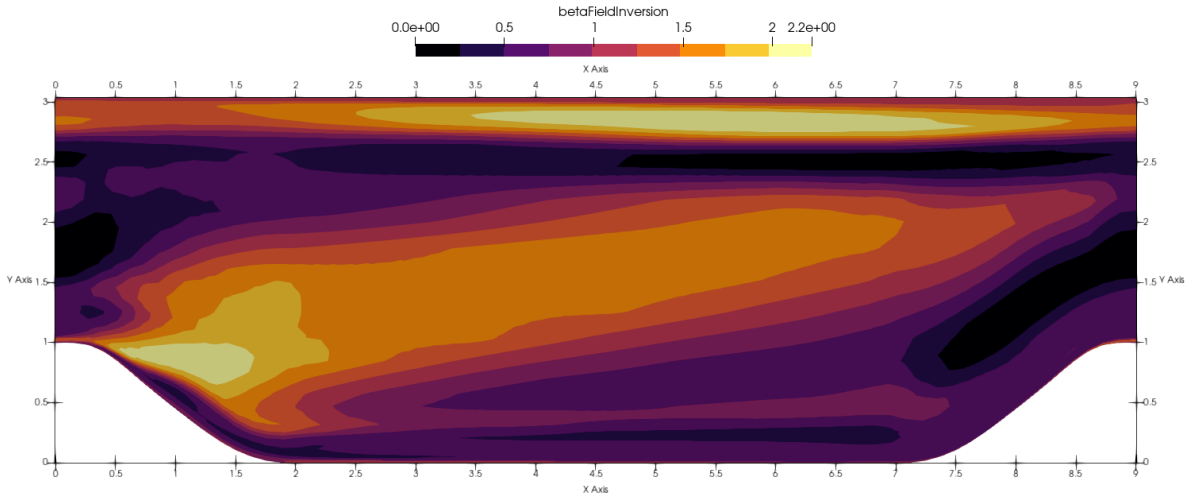


Figure 6.13: β field - training with λ

The β field is much more acceptable now, not very aggressive and regions of modifying production significantly in opposing manners is not observed. the correction physically makes sense as the baseline model significantly over-predicts the size of the separation bubble and reattachment point. The β field correct this by increasing the production in the shear layer due to separation and in the circulation zone as well. This increases the turbulent diffusion and mixing in the flow that help the flow overcome the adverse pressure gradient. With improved momentum transfer, the separation bubble becomes smaller and reattachment occurs sooner, as seen. Corrections in other regions are weaker. The strong correction along the bottom wall is likely an effect of the input invariants. This patch along the top wall is also seen in production, diffusion and slightly in the destruction ratios, Figure 6.1. When the β field is formulated, it strongly depends on the diffusion and production ratios. Thus, it can be thought of a correction that should not be present, ideally.

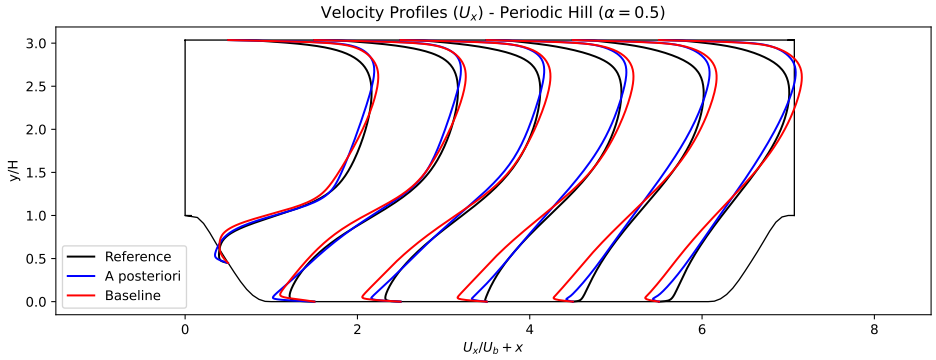
With this, the training is completed. Improvements are substantial and the β field does not exhibit overfitting now. The symbolic formula is exported and needs to be applied to testing cases to verify if it is generalizable. This is performed in the next section.

6.4. Testing Cases

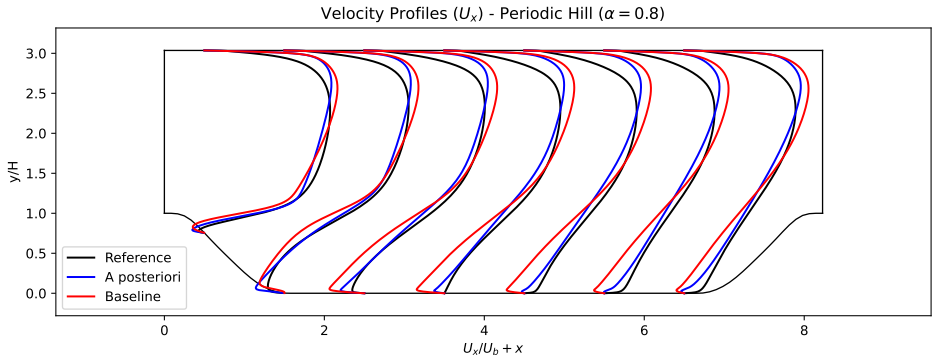
The β derived from the symbolic formula was applied to 3 different geometries that were characterized by different slopes of the hills. The geometries were used from the author's research group and have

been used already in other studies for validation purposes. Hence, no further studies were done to verify if the mesh refinement and setup needed improvements. Moreover, as stated in Section 3.6, the distribution of invariants remains similar for different geometries as they are representative of the turbulent phenomena. Hence, the mesh along with the y^+ values and the distribution of invariants can be referenced from Appendix Section D.2.2.

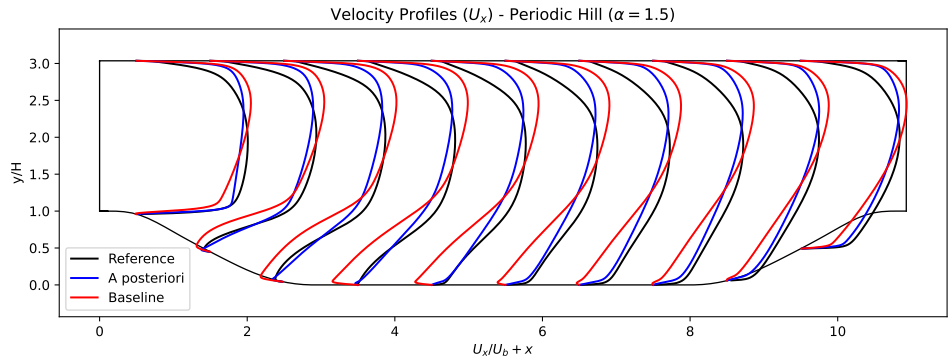
The velocity & viscosity profiles, C_f and the β field is presented below. The streamlines can be referenced from Appendix Section D.2.2.



(a) $\alpha = 0.5$

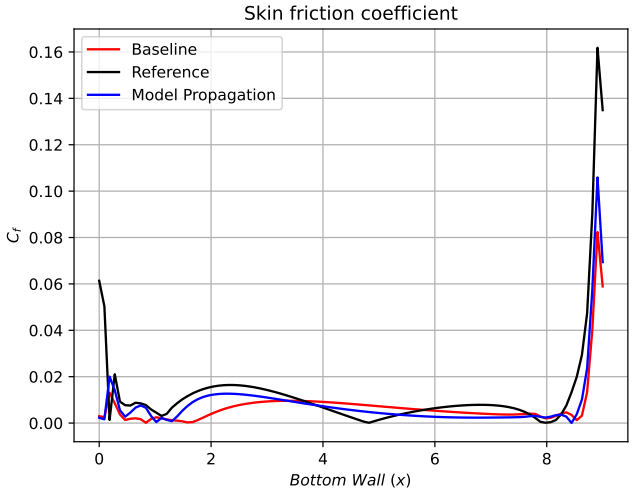


(b) $\alpha = 0.8$

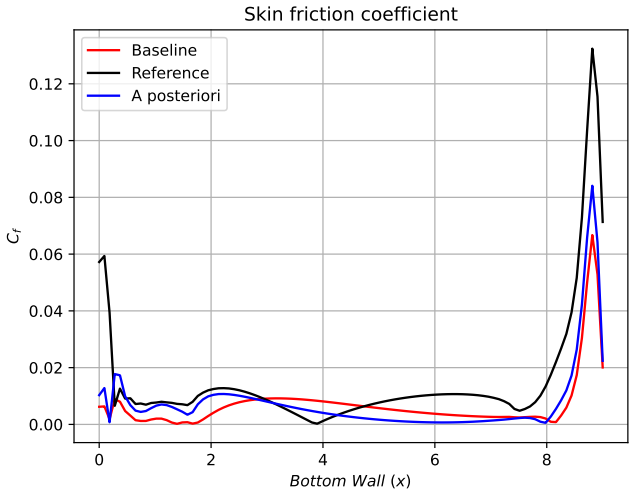


(c) $\alpha = 1.5$

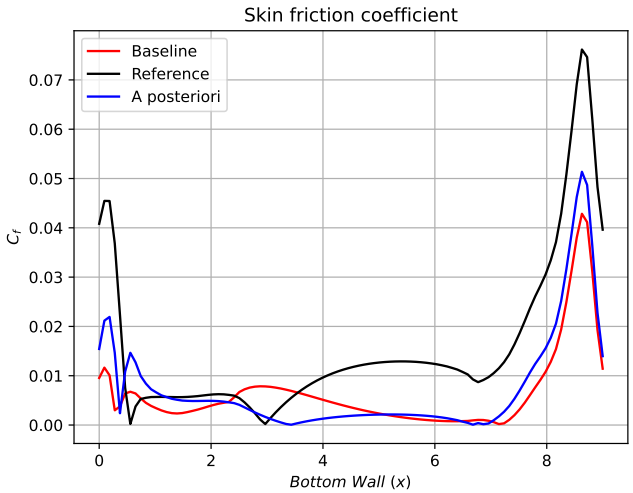
Figure 6.14: Velocity profiles - Baseline, Reference and A posteriori



(a) $\alpha = 0.5$

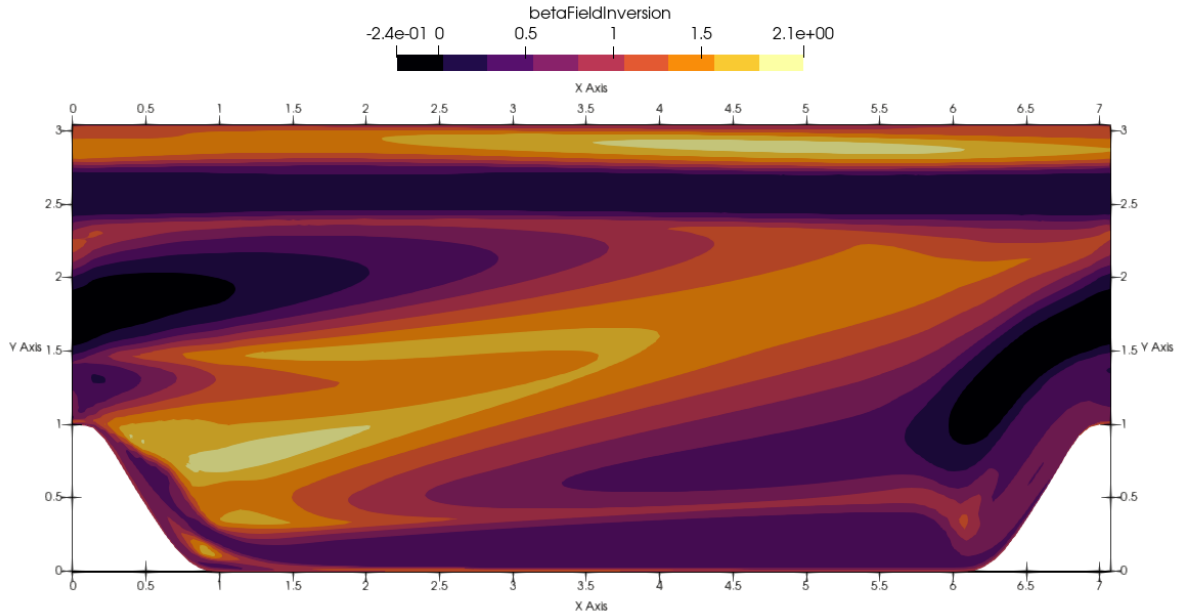


(b) $\alpha = 0.8$

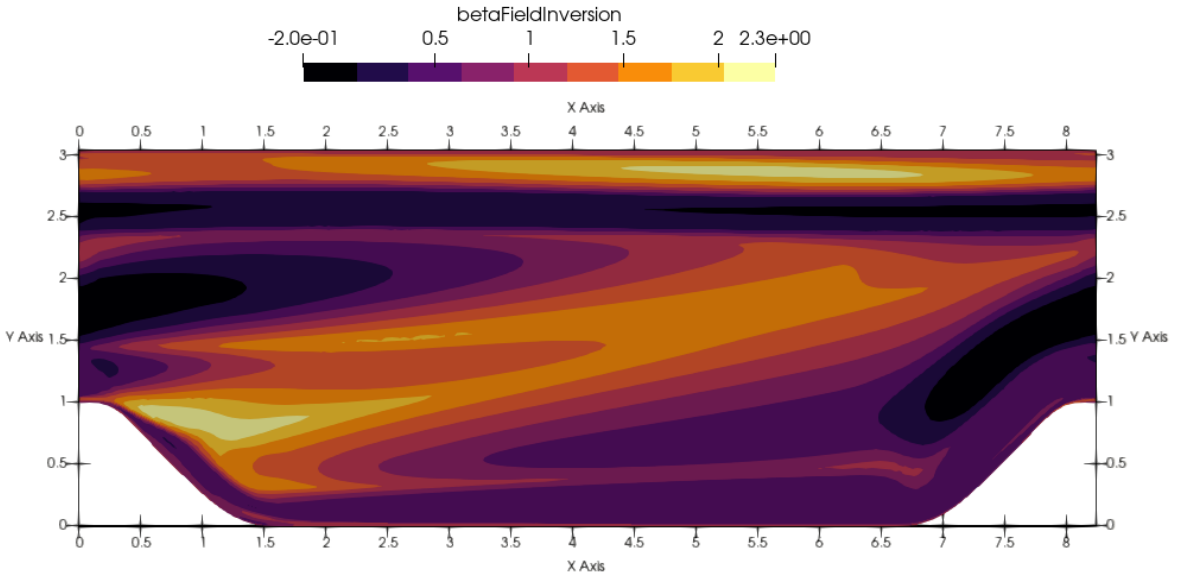


(c) $\alpha = 1.5$

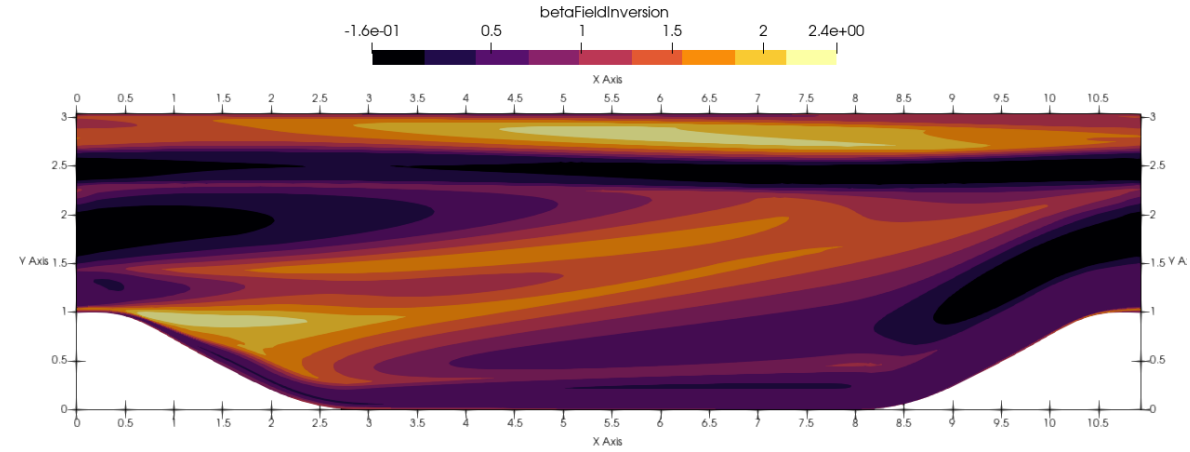
Figure 6.15: Skin friction coefficient - Baseline, Reference and A posteriori



(a) $\alpha = 0.5$

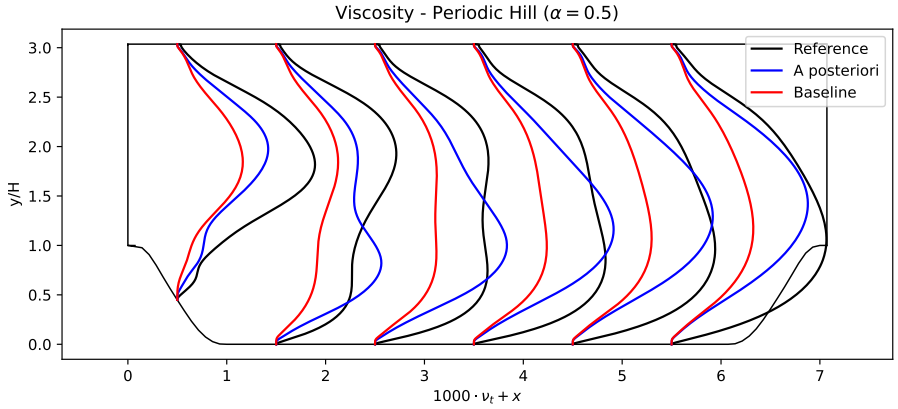


(b) $\alpha = 0.8$

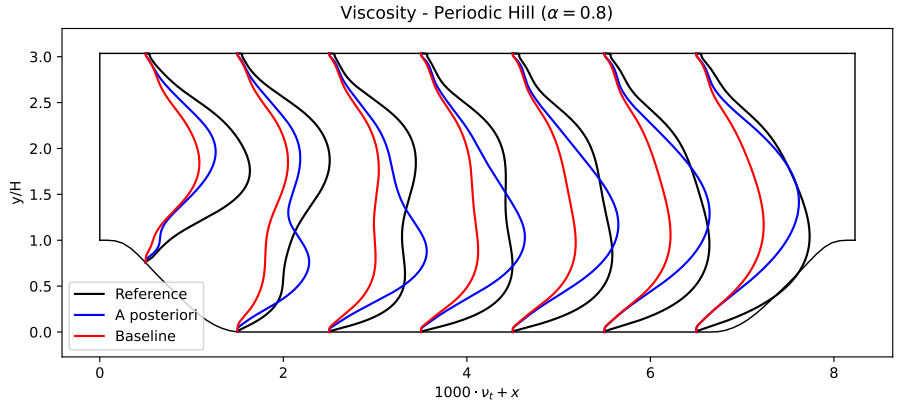


(c) $\alpha = 1.5$

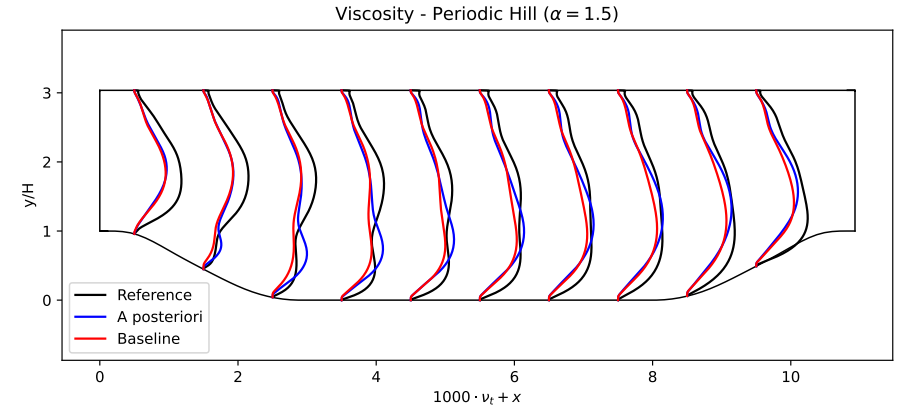
Figure 6.16: β fields



(a) $\alpha = 0.5$



(b) $\alpha = 0.8$



(c) $\alpha = 1.5$

Figure 6.17: Viscosity profiles - Baseline, Reference and A posteriori

It can be concluded that the model is robust as all the testing cases show an improvement. When considering the velocity profiles from Figure 6.14, most significant improvements can be observed for $\alpha = 1.5°$. The velocities at the recirculation regions and along the entirety of the bottom wall show a good match with the reference profiles. However, close to the top wall, the velocities are over-predicted as was also observed in the training case. This can be directly related to the β field as shown in Figure 6.16. The region of high production at the top wall is causing the over-prediction of the velocities which is believed to be derived from the invariants. The β field is also physically meaningful and aligns with the flow physics. For all 3 cases, there is significant increase in production in the shear layer to correct for the larger separation bubble predicted by the baseline model. Corrections in other regions remain

mild. Moreover, for $\alpha = 0.5°$ specifically, there is also a small region of strong correction immediately downstream of the first periodic hill. This can be explained by the velocity streamlines presented in Appendix Section D.2.2, Figure D.15. The baseline model predicts a secondary separation region which is absent from the reference case. This is corrected by the strong production, and hence absent from the $a\ posteriori$ model.

With regards to C_f as shown in Figure 6.15, improvements are not as significant. The $a\ posteriori$ model is not able to capture the flow deceleration and acceleration from the periodic hills as the β field only weakly enhances the production in those regions. Nonetheless, the recirculation region along the bottom wall is accurately captured by the $a\ posteriori$ model. Close to the reattachment point predicted by the reference case, the results are improved. For the case of $\alpha = 1.5°$, there is an over prediction of C_f at $x = 0.57$ which can be explained because of the *more stronger than required* correction which causes the size of separation bubble to in fact be reduced from the reference case. For the case also, the model had a better prediction of the reattachment point at $(x = 3.4)$ when compared to $x = 2.96$ from reference. However, after this reattachment point (reference case), the $a\ posteriori$ model struggles, performing worse than the baseline model in some regions. It is not able to capture the region of favorable and adverse pressure gradient post reattachment. However, as the second hill approaches, the model is better able to capture the flow acceleration (favorable pressure gradient).

The viscosity profiles from Figure 6.17 also show improvements. Starting from the bottom wall, the recirculation region and shear layer, viscosity is improved for both $\alpha = 0.5°, 0.8°$. As the top wall approaches however, the effect of corrections on ν_t diminishes and the $a\ posteriori$ behaves similarly to the baseline model, as issue also faced in the training case. For $\alpha = 1.5°$ however, the improvements are not significant. In the recirculation region, the model tends to over damp the solution by increasing ν_t more than the reference, and in other regions, it struggles for even the slightest improvements.

6.5. $A\ posteriori$ Conclusions

A short summary of the current chapter is provided, before moving to the detailed conclusions of the complete thesis in the next chapter.

KANs continue to demonstrate strong performances in the programmed $a\ posteriori$ framework. Considering the initial testing, it took just 63 CFD runs to get a correlation ≈ 0.97. This shows that the optimization is very strong and also stable as none of the iterations diverged. Although the quality deteriorated occasionally, the optimizer was quick to learn the functional space and adjust its parameters to optimize the objective.

However, with the initial testing, the β field was more of an ad-hoc correction to get accurate velocity fields rather than an improvement in turbulence modelling. This is undesirable and highlights that the optimization effect needs to be controlled. Given this, regularization was introduced to allow for a more robust and a generalizable model. As seen with the overfit solution, high frequency oscillations could be observed in the symbolic model. Such oscillations were absent with regularization, thus converging to a much more stable model. The resulting model produced improved results on the testing cases. On 3 different hill slopes, the velocity profiles showed the most improvements. Improvements in viscosity profiles and the correction field now aligned with flow physics and consistent with how turbulence modeling should be improved. Moreover, the quality of the correction field reflects the input variants. The β field struggled to improve the top wall region as the invariants exhibited unusual behavior as well, which the author believes are due to the limitations of the $Sapalart - Allmaras$ model to model turbulence.

Overall, it can be concluded that the programmed $a\ posteriori$ framework, when subjected to a representative training case, can produce a generalized symbolic formula to improve turbulence modeling. The resulting β field is able to provide corrections to not just match the velocity fields, but improve flow quantities as well, thus, answering the last research question mentioned in Section 2.9.

7

Conclusion

Based on the detailed research and reflecting each sub-question and main question presented in Section 2.9, they are answered here. The $a\ priori$ study is first addressed and then the $a\ posteriori$ study.

Sub-question 1: Can genetic programs that are incorporated with information from derivatives achieve quicker convergence, with reduced function evaluations, and acceptable correlations?

Yes. A genetic program is a heuristic method that depends on evolutionary operators to converge to the optimal solution. These genetic operators (crossover and mutation) depend on probabilities and they occur at random nodes of an individual. This can eliminate useful combinations and often makes the algorithm behave similar to a random search method. Gradient information to optimize the weights of operators and terminals helps the algorithm to quickly learn the functional space. As seen, the convergence rate and the individual quality increases significantly as soon as gradient information is incorporated. It refines the search by exploiting the space and makes the algorithm converge quickly. The optimal solution also exhibits improved correlations with a reduced functional count by up-to 30%.

Sub-question 2: How and which parameters (population size, evolution probabilities, number of generations) would need to be optimized, in addition to the frequency and optimizer used?

Sub-question 3: Does the local search overshadow the exploratory ability of genetic algorithms to observe large function spaces, causing the solution to converge to a local optima?

These are important parameters that need to be carefully adjusted. A population size of 50 was chosen to provide sufficient diversity in the initial sample space. A larger population can also be used but makes the process more expensive as there are more individuals to evaluate. The optimizer and the frequency of optimization are the most important as it can make the algorithm computationally very expensive if it calls the objective function repeatedly. It was seen that simultaneous optimization of terminals and operators at every $5^{th}\ generation$ is more effective than doing it separately. As seen with benchmark testing in Section 4.1, the evolution probabilities were modified in phase I and phase II and were also critical to the algorithm's performance.

Linking this to the next sub-question, gradient based optimization and optimization by genetic operators needs to be carefully balanced. The rate of occurrence of these operations needed to be increased as gradient-based optimization can confine the algorithm's search space causing it to converge to a local optimum. When optimization was performed separately, the developed algorithm struggled on some benchmark cases. However, with the increased probabilities and simultaneous optimization now, the local search did not overshadow the exploratory ability of conventional genetic program. The correlations remained similar, if not improved.

Sub-question 4: Can KANs be used on CFD datasets and converge to superior results with reduced function evaluations? Can the learnable activation functions capture the complex (non linear) relation between different quantities?

Yes. KANs can be used on CFD datasets and reduce the computational cost (function calls) by as much as 85% for some cases. KANs, by pruning, select only the most relevant inputs and reduce the functional space significantly. The learnable activation functions can be dynamically adapted to modify the shape of the basis functions and are capable to efficiently map the relations between different quantities in the reduced space. They are able to converge to improved correlations.

This completes the $a\ priori$ research questions and its objectives. Both methods, $GP + TRF$ and $KANs$ have significantly lower computational costs (function calls). They are able to converge to superior correlations and provide meaningful symbolic formula for the regressed quantities - RST and Turbulence Production. However, KANs are the better candidate for the $a\ posteriori$ study as it has much lower computational costs, a symbolic formula strongly dependent on the pruned functional space and individuals of a higher quality as well (correlations).

Sub-question 5: Can the superior method maintain its performance when applied in an a posteriori setting and if a generalizable model be obtained that is applicable for different flow cases without encountering overfitting and stability issues? Is the symbolic formula physically meaningful?

KANs can maintain is performance when incorporated with the differentiation framework. It is able to adjust its parameters to modify the correction field to optimize the objective function. Although applied to improve the $Spalart-Allmaras$ model which is computationally cheaper than $k-\omega\ SST$, it takes only 63 iterations to achieve a correlation of ≈ 0.97. However, when applied as is, the optimization is too strong and the solution over fits. A generalizable model for the correction field that aligns with flow physics cannot be obtained. Thus, regularization is needed to limit this effect and balance the optimization of the objective function while converging to a stable model as well. An appropriate regularization parameter is selected by incorporating the squared $\mathcal{L}^2$ norm of the correction field. KANs can now produce a generalizable symbolic formula when subjected to a representative training case. The correction field obtained from the symbolic formula is interpretable and aligns with how turbulence modeling must be improved.

It is also important to mention that the quality of the correction field depends on the invariants. The $Spalart - Allmaras$ model tends to over predict the production ration at the top wall (close upstream of the second hill) and this influences the corrections as well. The β field exhibits strong production correction in that region as well, thus compromising the overall improvement in turbulence modeling.

This answers the last research question of the thesis. The investigation of KANs in an $a\ posteriori$ framework is completed and this completes the study.

Note that there are a lot of code files with $1000+$ lines in each file, especially for the *a posteriori* framework. Thus, instead of presenting it here, all the codes can be obtained from the git repository: `https://github.com/affu5154/MSc_Thesis_Codes.git`

8

Recommendations

Based on the research performed, some improvements can be made if decided to continue with the study and the following recommendations are made:

- The tree expressions of $GP + TRF$ method are weakly dependent on the input features. Often, they are just scalars which does not provide them with the physical interpretability advantage, as exhibited by $KANs$ or CGP. To resolve this, a tree syntax can be enforced such that every individual has at least one input feature in its node.
- KANs can generate a symbolic formula but is often very complex or composed of multiple `sin` and `cos` that can cause instability. This can be resolved by providing a more interpretable library of functions or further increasing the correlation which serves as the threshold to accept a symbolic function to represent the activation function.
- $GP+TRF$ was allowed to evolve for 50 generations and was able to converge to superior results. However, it was seen from benchmark cases that convergence could also be attained much earlier for some cases. Hence, the number of function calls could further be reduced by reducing the number of generations and analyzing the effect.
- The quality of the converged individuals from the developed methods was superior in the $a\ priori$ setting. However, often the results are not as improved when the symbolic model is inserted into a turbulence model (CFD solver). As also reported by [30], the *a priori* model required further optimizing the model coefficients to produce improved results. Hence, to be sure of the model's quality, it can be inserted into the CFD solver and checked if turbulence modeling does improve.
- Regarding *a posteriori* study, $Spalart - Allmaras$ was chosen as it is computationally cheap to evaluate. However, to further continue this study, $k - \omega\ SST$ can be chosen as the model to improve with LES or DNS taken as reference. This will also improve the quality of the invariants as it does not suffer from the peak in production anomaly that $Spalart - Allmaras$ model predicts. $k - \omega\ SST$ model is better suited for such internal flows.
- It was seen that KANs exhibit loss of information and individual quality when the continuous space is mapped using symbolic functions. In the *a priori* setting, further training was required to compensate this loss. Whether the effect of this loss is detrimental to the quality of the correction field or convergence in the *a posteriori* setting can be studied.
- The effect of regularization can further be studied. Squared $\mathcal{L}^2$ norm was used to align it with the already calculated error in the velocity fields. However, a different regularization can be implemented to select a more appropriate penalty factor.
- The training was performed on the Periodic Hill and testing also done on Periodic Hills but characterized by different slopes. To further verify the generalization of the correction field, a vastly different testing case such as a backward facing step can be implemented and the CFD solution be observed.

- Since the current study dealt with incompressible flows and developing the *a posteriori* framework, velocity field was chosen as the target flow quantity. However, a different flow variable can also be used or added in the objective function such as viscosity or pressure fields for compressible flows. This can help develop a correction field that may significantly improve multiple flow quantities simultaneously and improve turbulence modeling.
- The current study did not involve implementing a reverting mechanism. If the CFD solution diverged due to a poor correction, it was not possible for the CFD solver to run again with the previous best model and have the optimizer use this information to update its parameters. This can be implemented further which can tackle the issue of stability in the *a posteriori* framework and maybe also allow for more improved results.

References

[1] Milad Taleby Ahvanooey et al. “A survey of genetic programming and its applications”. In: *KSII Transactions on Internet and Information Systems (TIIS)* 13.4 (2019), pp. 1765–1794.

[2] Amy E. Alving, Alexander J. Smits, and Jonathan H. Watmuff. “Turbulent boundary layer relaxation from convex curvature”. In: *Journal of Fluid Mechanics* 211 (1990), pp. 529–556. doi: 10.1017/S0022112090001689.

[3] J. D. Anderson. “Governing Equations of Fluid Dynamics”. In: *Computational Fluid Dynamics: An Introduction*. Ed. by John F. Wendt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 15–51. isbn: 978-3-662-11350-9. doi: 10.1007/978-3-662-11350-9_2. url: https://doi.org/10.1007/978-3-662-11350-9_2.

[4] Sunitha Basodi et al. “Gradient amplification: An efficient way to train deep neural networks”. In: *Big Data Mining and Analytics* 3.3 (2020), pp. 196–207.

[5] J.M. Benitez, J.L. Castro, and I. Requena. “Are artificial neural networks black boxes?” In: *IEEE Transactions on Neural Networks* 8.5 (1997), pp. 1156–1164. doi: 10.1109/72.623216.

[6] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. “A survey on optimization metaheuristics”. In: *Information sciences* 237 (2013), pp. 82–117.

[7] Mary Ann Branch, Thomas F Coleman, and Yuying Li. “A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems”. In: *SIAM Journal on Scientific Computing* 21.1 (1999), pp. 1–23.

[8] Michael Breuer et al. “Flow over periodic hills–numerical and experimental study in a wide range of Reynolds numbers”. In: *Computers & Fluids* 38.2 (2009), pp. 433–457.

[9] Robert H Bush et al. “Recommendations for future efforts in RANS modeling and simulation”. In: *AIAA scitech 2019 forum*. 2019, p. 0317.

[10] Richard H Byrd et al. “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[11] Sai Hung Cheung et al. “Bayesian uncertainty analysis with applications to turbulence modeling”. In: *Reliability Engineering & System Safety* 96.9 (2011), pp. 1137–1149.

[12] Kim Gail Clarke. *Bioprocess engineering: an introductory engineering and life science approach*. Elsevier, 2013.

[13] Fengzhi Dai et al. “A survey of genetic algorithm-based face recognition”. In: *Artificial Life and Robotics* 16 (2011), pp. 271–274.

[14] Rajeev Jaiman David Corson and Farzin Shakib. “Industrial application of RANS modelling: capabilities and needs”. In: *International Journal of Computational Fluid Dynamics* 23.4 (2009), pp. 337–347. doi: 10.1080/10618560902776810. eprint: https://doi.org/10.1080/10618560902776810. url: https://doi.org/10.1080/10618560902776810.

[15] Gianluca De Carlo, Andrea Mastropietro, and Aris Anagnostopoulos. “Kolmogorov-arnold graph neural networks”. In: *arXiv preprint arXiv:2406.18354* (2024).

[16] Sébastien Deck et al. “High-fidelity simulations of unsteady civil aircraft aerodynamics: stakes and perspectives. Application of zonal detached eddy simulation”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 372.2022 (2014), p. 20130325.

[17] Charles R Doering and Peter Constantin. “Variational bounds on energy dissipation in incompressible flows: shear flow”. In: *Physical Review E* 49.5 (1994), p. 4087.

[18] Eric Dow and Qiqi Wang. “Quantification of Structural Uncertainties in the k-w Turbulence Model”. In: *52nd AIAA/ASME/ASCE/AHS/ASC Structures, structural dynamics and materials conference 19th AIAA/ASME/AHS adaptive structures conference 13t*. 2011, p. 1762.

[19] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. “Turbulence modeling in the age of data”. In: *Annual review of fluid mechanics* 51.1 (2019), pp. 357–377.

[20] Karthikeyan Duraisamy, Ze J Zhang, and Anand Pratap Singh. “New approaches in turbulence and transition modeling using data-driven techniques”. In: *53rd AIAA Aerospace sciences meeting*. 2015, p. 1284.

[21] WN Edeling, Pasquale Cinnella, and Richard P Dwight. “Predictive RANS simulations via Bayesian model-scenario averaging”. In: *Journal of Computational Physics* 275 (2014), pp. 65–91.

[22] Michael Emory, Rene Pecnik, and Gianluca Iaccarino. “Modeling structural uncertainties in Reynolds-averaged computations of shock/boundary layer interactions”. In: *49th AIAA Aerospace sciences meeting including the new horizons forum and aerospace exposition*. 2011, p. 479.

[23] Björn Fabritius. *Application of genetic algorithms to problems in computational fluid dynamics*. University of Exeter (United Kingdom), 2014.

[24] Thomas B Gatski, Sutanu Sarkar, and Charles G Speziale. “Modeling the pressure-strain correlation of turbulence: An invariant dynamical systems approach”. In: (1990).

[25] Hao Hao et al. “A first look at kolmogorov-arnold networks in surrogate-assisted evolutionary algorithms”. In: *arXiv preprint arXiv:2405.16494* (2024).

[26] Trevor Hastie et al. “The elements of statistical learning: data mining, inference and prediction”. In: *The Mathematical Intelligencer* 27.2 (2005), pp. 83–85.

[27] Mark Hauschild and Martin Pelikan. “An introduction and survey of estimation of distribution algorithms”. In: *Swarm and evolutionary computation* 1.3 (2011), pp. 111–128.

[28] Ping He et al. “An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM”. In: *Computers & Fluids* 168 (2018), pp. 285–303.

[29] Ping He et al. “Dafoam: An open-source adjoint framework for multidisciplinary design optimization with openfoam”. In: *AIAA journal* 58.3 (2020), pp. 1304–1319.

[30] K.N. Hoefnagel. “Multi-Flow Generalization in Data-Driven Turbulence Modeling: An Exploratory Study”. Master Thesis. Delft: Delft University of Technology, 2023. url: `https://resolver.tudelft.nl/324d4b2d-bf58-40a0-b60d-4e2e0b992797`.

[31] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[32] Louis N Howard. “Bounds on flow quantities”. In: *Annual Review of Fluid Mechanics* 4.1 (1972), pp. 473–494.

[33] Tim Janus et al. “Optimization of a hydroformulation process in a thermomorphic solvent system using a commercial steady-state process simulator and a memetic algorithm”. In: *Computer Aided Chemical Engineering*. Vol. 46. Elsevier, 2019, pp. 469–474.

[34] W Peter Jones and Brian Edward Launder. “The prediction of laminarization with a two-equation model of turbulence”. In: *International journal of heat and mass transfer* 15.2 (1972), pp. 301–314.

[35] Mikael LA Kaandorp and Richard P Dwight. “Data-driven modelling of the Reynolds stress tensor using random forests with invariance”. In: *Computers & Fluids* 202 (2020), p. 104497.

[36] Marc C Kennedy and Anthony O’Hagan. “Bayesian calibration of computer models”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.3 (2001), pp. 425–464.

[37] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[38] John R Koza and James P Rice. “Automatic programming of robots using genetic programming”. In: *AAAI*. Vol. 92. 1992, pp. 194–207.

[39] William La Cava et al. “Contemporary symbolic regression methods and their relative performance”. In: *Advances in neural information processing systems* 2021.DB1 (2021), p. 1.

[40] M. Lacatus. “Improving Data-Driven RANS Turbulence Modelling For Separated Flow Scenarios In Support of Formula 1 Aerodynamic Development”. Master Thesis. Delft: Delft University of Technology, 2024. url: http://resolver.tudelft.nl/uuid:755d3b8c-78d4-4926-9b1a-7516ef8b00e0.

[41] B. E. Launder, G. J. Reece, and W. Rodi. “Progress in the development of a Reynolds-stress turbulence closure”. In: *Journal of Fluid Mechanics* 68.3 (1975), pp. 537–566. doi: 10.1017/S0022112075001814.

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

[43] Julia Ling et al. “Uncertainty Analysis and Data-Driven Model Advances for a Jet-in-Crossflow”. In: *Journal of Turbomachinery* 139.2 (Oct. 2016), p. 021008.

[44] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[45] Ziming Liu et al. “Kan: Kolmogorov-arnold networks”. In: *arXiv preprint arXiv:2404.19756* (2024).

[46] David J Livingstone, David T Manallack, and Igor V Tetko. “Data modelling with neural networks: Advantages and limitations”. In: *Journal of computer-aided molecular design* 11 (1997), pp. 135–142.

[47] Hammad Majeed, Abdul Wali, and Mirza Beg. “Optimizing genetic programming by exploiting semantic impact of sub trees”. In: *Swarm and Evolutionary Computation* 65 (2021), p. 100923.

[48] Florian Menter and Christopher Rumsey. “Assessment of two-equation turbulence models for transonic flows”. In: *Fluid Dynamics Conference*. 1994, p. 2343.

[49] Igor Molybog et al. “A theory on adam instability in large-scale machine learning”. In: *arXiv preprint arXiv:2304.09871* (2023).

[50] Jorge J Moré. “The Levenberg-Marquardt algorithm: implementation and theory”. In: *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*. Springer. 2006, pp. 105–116.

[51] FTM Nieuwstadt (†), Bendiks Jan Boersma, and Jerry Westerweel. *Turbulence: Introduction to Theory and Applications of Turbulent Flows*. English. Springer, 2016. isbn: 978-3-319-31597-3. doi: 10.1007/978-3-319-31599-7.

[52] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[53] Todd Oliver and Robert Moser. “Uncertainty quantification for RANS turbulence model predictions”. In: *APS division of fluid dynamics meeting abstracts*. Vol. 62. 2009, pp. LC–004.

[54] Randal S. Olson et al. “PMLB: a large benchmark suite for machine learning evaluation and comparison”. In: *BioData Mining* 10.36 (Dec. 2017), pp. 1–13. issn: 1756-0381. doi: 10.1186/s13040-017-0154-4. url: https://doi.org/10.1186/s13040-017-0154-4.

[55] Panos Y Papalambros and Douglass J Wilde. *Principles of optimal design: modeling and computation*. Cambridge university press, 2000.

[56] Wojciech Paszkowicz. “Genetic algorithms, a nature-inspired tool: survey of applications in materials science and related fields”. In: *Materials and Manufacturing Processes* 24.2 (2009), pp. 174–197.

[57] Yanhong Peng et al. “Predictive modeling of flexible EHD pumps using Kolmogorov–Arnold Networks”. In: *Biomimetic Intelligence and Robotics* 4.4 (2024), p. 100184.

[58] Tien Dat Pham and Won-Kee Hong. “Genetic algorithm using probabilistic-based natural selections and dynamic mutation ranges in optimizing precast beams”. In: *Computers & Structures* 258 (2022), p. 106681.

[59] Gloria Pietropolli et al. “On the hybridization of geometric semantic GP with gradient-based optimizers”. In: *Genetic Programming and Evolvable Machines* 24.2 (2023), p. 16.

[60] Gloria Pietropolli et al. “Parametrizing GP Trees for Better Symbolic Regression Performance through Gradient Descent.” In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 2023, pp. 619–622.

[61] Stephen Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[62] Stephen B Pope. “A more general effective-viscosity hypothesis”. In: *Journal of Fluid Mechanics* 72.2 (1975), pp. 331–340.

[63] Joseph D Romano et al. “PMLB v1.0: an open source dataset collection for benchmarking machine learning methods”. In: *arXiv preprint arXiv:2012.00058v2* (2021).

[64] B Ruck and B Makiola. “Flow separation over the step with inclined walls”. In: *Near-Wall Turbulent Flows* (1993), p. 999.

[65] Martin Schmelzer, Richard P Dwight, and Paola Cinnella. “Discovery of algebraic Reynolds-stress models using sparse symbolic regression”. In: *Flow, Turbulence and Combustion* 104 (2020), pp. 579–603.

[66] Christian Seis. “Scaling bounds on dissipation in turbulent flows”. In: *Journal of Fluid Mechanics* 777 (2015), pp. 591–603.

[67] Anand Pratap Singh and Karthik Duraisamy. “Using field inversion to quantify functional errors in turbulence closures”. In: *Physics of Fluids* 28.4 (2016).

[68] Léo Françoso Dal Piccol Sotto and Vinícius Veloso de Melo. “A probabilistic linear genetic programming with stochastic context-free grammar for solving symbolic regression problems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 1017–1024.

[69] Philippe Spalart and Steven Allmaras. “A one-equation turbulence model for aerodynamic flows”. In: *30th aerospace sciences meeting and exhibit*. 1992, p. 439.

[70] Philippe R Spalart. “Comments on the Feasibility of LES for Wings and on the Hybrid RANS/LES Approach”. In: *Proceedings of the First AFOSR International Conference on DNS/LES, 1997*. 1997, pp. 137–147.

[71] Philippe R. Spalart. “Philosophies and fallacies in turbulence modeling”. In: *Progress in Aerospace Sciences* 74 (2015), pp. 1–15. issn: 0376-0421. doi: https://doi.org/10.1016/j.paerosci.2014.12.004. url: https://www.sciencedirect.com/science/article/pii/S037604211400102X.

[72] PR Spalart and M Shur. “On the sensitization of turbulence models to rotation and curvature”. In: *Aerospace Science and Technology* 1.5 (1997), pp. 297–302.

[73] Alexander Topchy, William F Punch, et al. “Faster genetic programming based on local gradient search of numeric leaf values”. In: *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*. Vol. 155162. Morgan Kaufmann San Francisco, CA. 2001.

[74] Brendan Tracey, Karthik Duraisamy, and Juan Alonso. “Application of supervised learning to quantify uncertainties in turbulence and combustion modeling”. In: *51st AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*. 2013, p. 259.

[75] Brendan D Tracey, Karthikeyan Duraisamy, and Juan J Alonso. “A machine learning strategy to assist turbulence model development”. In: *53rd AIAA aerospace sciences meeting*. 2015, p. 1287.

[76] Pedro Stefanin Volpiani. “Are random forests better suited than neural networks to augment RANS turbulence models?” In: *International Journal of Heat and Fluid Flow* 107 (2024), p. 109348. issn: 0142-727X. doi: https://doi.org/10.1016/j.ijheatfluidflow.2024.109348. url: https://www.sciencedirect.com/science/article/pii/S0142727X24000730.

[77] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. “Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data”. In: *Physical Review Fluids* 2.3 (2017), p. 034603.

[78] Jianxun Wang et al. *Physics-informed machine learning for predictive turbulence modeling: Towards a complete framework*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[79] Q Wang and EA Dow. “Quantification of structural uncertainties in the k- ω turbulence model”. In: *Proceedings of the Summer Program*. 2010, p. 41.

[80] Z Warhaft. “An experimental study of the effect of uniform strain on thermal fluctuations in grid-generated turbulence”. In: *Journal of Fluid Mechanics* 99.3 (1980), pp. 545–573.

[81] Jack Weatheritt and Richard Sandberg. “A novel evolutionary algorithm applied to algebraic modifications of the RANS stress–strain relationship”. In: *Journal of Computational Physics* 325 (2016), pp. 22–37.

[82] Jack Weatheritt and Richard D Sandberg. “Use of Symbolic Regression for construction of Reynolds-stress damping functions for Hybrid RANS/LES”. In: *53rd AIAA Aerospace Sciences Meeting*. 2015, p. 0312.

[83] R. Weber, B.M. Visser, and F. Boysan. “Assessment of turbulence modeling for engineering prediction of swirling vortices in the near burner zone”. In: *International Journal of Heat and Fluid Flow* 11.3 (1990), pp. 225–235. issn: 0142-727X. doi: https://doi.org/10.1016/0142-727X(90)90041-9. url: https://www.sciencedirect.com/science/article/pii/0142727X90900419.

[84] David C Wilcox. “Multiscale model for turbulent flows”. In: *AIAA journal* 26.11 (1988), pp. 1311–1320.

[85] David C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, Canada, 2006.

[86] Jin-Long Wu et al. “Physics-informed machine learning for predictive turbulence modeling: A priori assessment of prediction confidence”. In: *arXiv preprint arXiv:1607.04563* (2016).

[87] Heng Xiao, Jian-Xun Wang, and Roger G Ghanem. “A random matrix approach for quantifying model-form uncertainties in turbulence modeling”. In: *Computer Methods in Applied Mechanics and Engineering* 313 (2017), pp. 941–965.

[88] Heng Xiao et al. “Quantifying and reducing model-form uncertainties in Reynolds-averaged Navier–Stokes simulations: A data-driven, physics-informed Bayesian approach”. In: *Journal of Computational Physics* 324 (2016), pp. 115–136.

[89] Kunpeng Xu, Lifei Chen, and Shengrui Wang. “Kolmogorov-arnold networks for time series: Bridging predictive power and interpretability”. In: *arXiv preprint arXiv:2406.02496* (2024).

[90] “2 - Field Modeling Approach”. In: *Computational Fluid Dynamics in Fire Engineering*. Ed. by Guan Heng Yeoh and Kwok Kit Yuen. Burlington: Butterworth-Heinemann, 2009, pp. 29–133. isbn: 978-0-7506-8589-4. doi: https://doi.org/10.1016/B978-0-7506-8589-4.00002-8. url: https://www.sciencedirect.com/science/article/pii/B9780750685894000028.

[91] Peng Zeng et al. “Differentiable genetic programming for high-dimensional symbolic regression”. In: *arXiv preprint arXiv:2304.08915* (2023).

[92] Ze Jia Zhang and Karthikeyan Duraisamy. “Machine learning methods for data-driven turbulence modeling”. In: *22nd AIAA computational fluid dynamics conference*. 2015, p. 2460.

[93] Yaomin Zhao et al. “RANS turbulence model development using CFD-driven machine learning”. In: *Journal of Computational Physics* 411 (2020), p. 109413.

A

Methodology

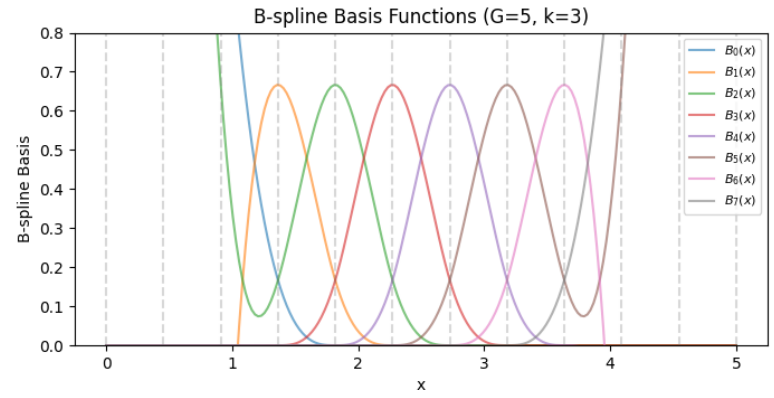
A.1. Tournament Selection Algorithm

Algorithm 9: Tournament Selection

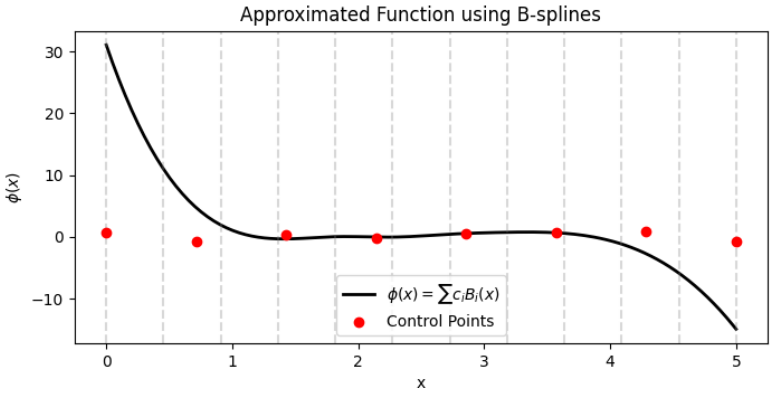
Input: Population `Pop`, tournament size `k`
Result: Best individual selected based on fitness

```
1 Tournament_Selection(population, tournament size) ;
2 Best ← null ;
3 for i = 1 to tournament size do
4 |   Individual ← population [random(1, population size)] ;
5 |   if Best = null or fitness (Individual) better than fitness (Best) then
6 |   |   Best ← Individual;
7 return Best;
```

A.2. Illustration of control points, B-splines and basis functions



(a) B spline basis functions



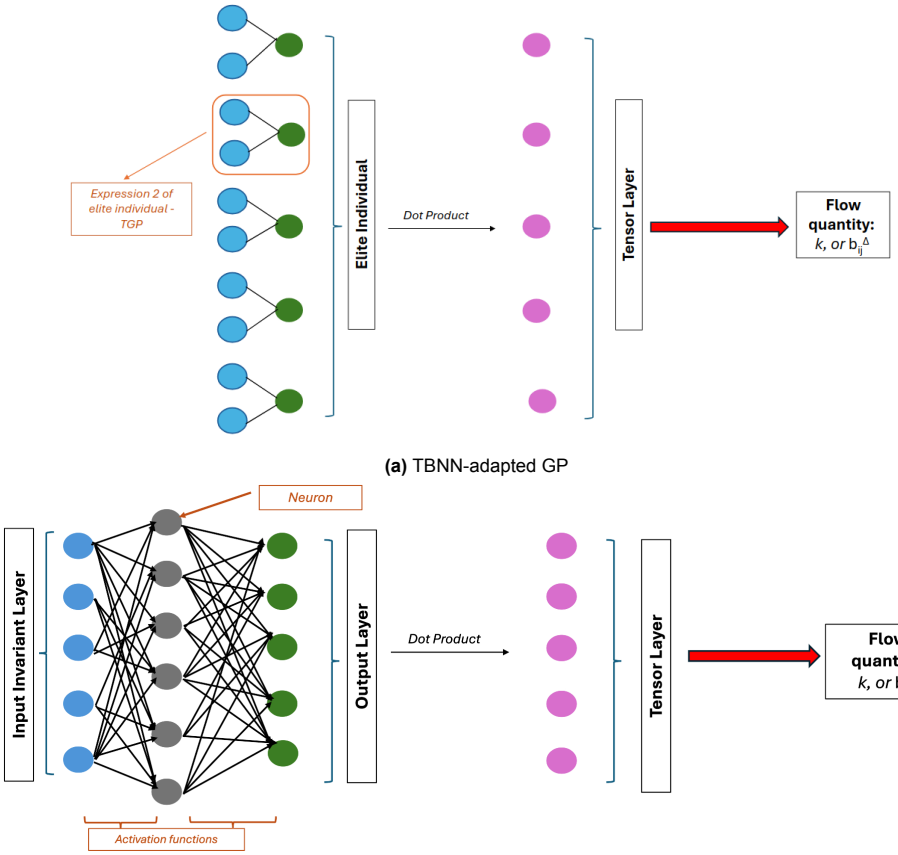
(b) One half approximation of the complete activation function

Figure A.1: B splines, and its parameters

The dashed vertical lines in Figure A.1 indicate the knots.

A.3. Adopted TBNN concept

The concept of adapting the TBNN structure to KANs and genetic programming is visually shown below, for better understanding.



(a) TBNN-adapted GP

(b) TBNN-adapted KANs

Figure A.2: Algorithm architecture

A.4. Apriori CFD setups

Periodic Hill

Table A.1: Boundary Conditions for PH CFD case

Variable	bottomWall	topWall	inlet	outlet
U	`noSlip`	`noSlip`	`cyclic`	`cyclic`
p	`zeroGradient`	`zeroGradient`	`cyclic`	`cyclic`
omega	`omegaWallFunction`	`omegaWallFunction`	`cyclic`	`cyclic`
k	`fixedValue` $k = 10^{-15}$	`fixedValue` $k = 10^{-15}$	`cyclic`	`cyclic`
nut	`nutLowReWallFunction`	`nutLowReWallFunction`	`cyclic`	`cyclic`

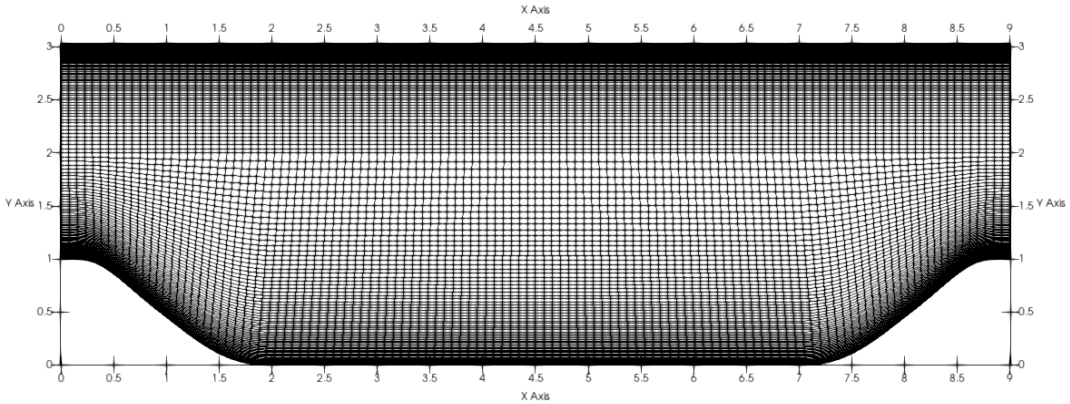


Figure A.3: Periodic Hill - frozen setup

Square Duct

Table A.2: Boundary Conditions for SD CFD case

Variable	bottomWall	topWall	inlet	outlet
U (velocity)	`noSlip`	`noSlip`	`cyclic`	`cyclic`
p (pressure)	`zeroGradient`	`zeroGradient`	`cyclic`	`cyclic`
omega	`omegaWallFunction` $\omega = 10$	`omegaWallFunction` $\omega = 10$	`cyclic`	`cyclic`
k	`fixedValue` $k = 10^{-15}$	`fixedValue` $k = 10^{-15}$	`cyclic`	`cyclic`
nut	`nutLowReWallFunction` $\nu_t = 0$	`nutLowReWallFunction` $\nu_t = 0$	`cyclic`	`cyclic`

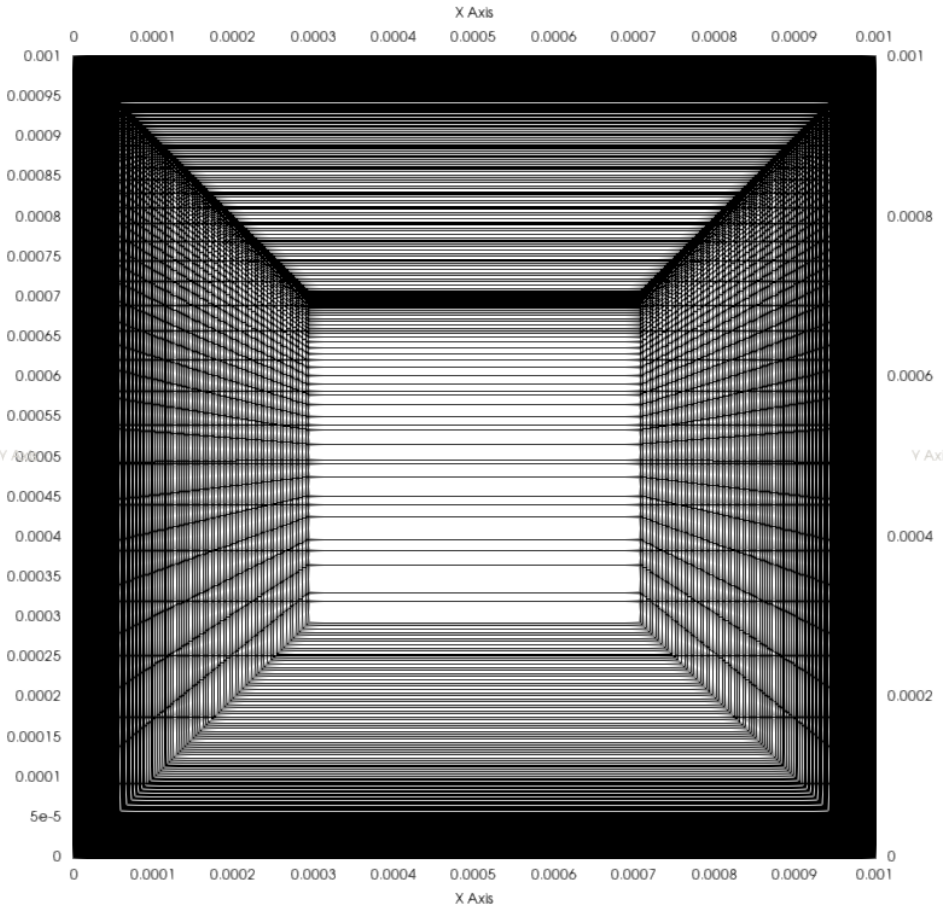


Figure A.4: Square Duct - frozen setup

A.5. Apriori CFD Modeling

Augmented k-ω SST model

Turbulence Kinetic Energy Equation

$$\rho\frac{\partial k}{\partial t}+\rho\frac{\partial(U_j k)}{\partial x_j}=(\hat{P}_k+\sigma R)-\beta^*\rho\omega k+\frac{\partial}{\partial x_j}\left[(\mu+\sigma_k\mu_t)\frac{\partial k}{\partial x_j}\right] \tag{A.1}$$

Specific Dissipation Rate Equation

$$\rho\frac{\partial \omega}{\partial t}+\rho\frac{\partial(U_j \omega)}{\partial x_j}=\frac{\gamma}{\nu_t}(\hat{P}_k+\sigma R)-\beta\rho\omega^2+\frac{\partial}{\partial x_j}\left[(\mu+\sigma_\omega\mu_t)\frac{\partial \omega}{\partial x_j}\right]+2\rho(1-F_1)\frac{\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \tag{A.2}$$

The production term in the above equations and the Reynolds-stress tensor are augmented by b_{ij}^{Δ}:

$$\hat{P}_k=\min\left(-2k\left(b_{ij,\text{Boussinesq}}+\sigma b_{ij}^{\Delta}\right)\frac{\partial U_i}{\partial x_j},\ 10\beta^*\omega k\right) \tag{A.3}$$

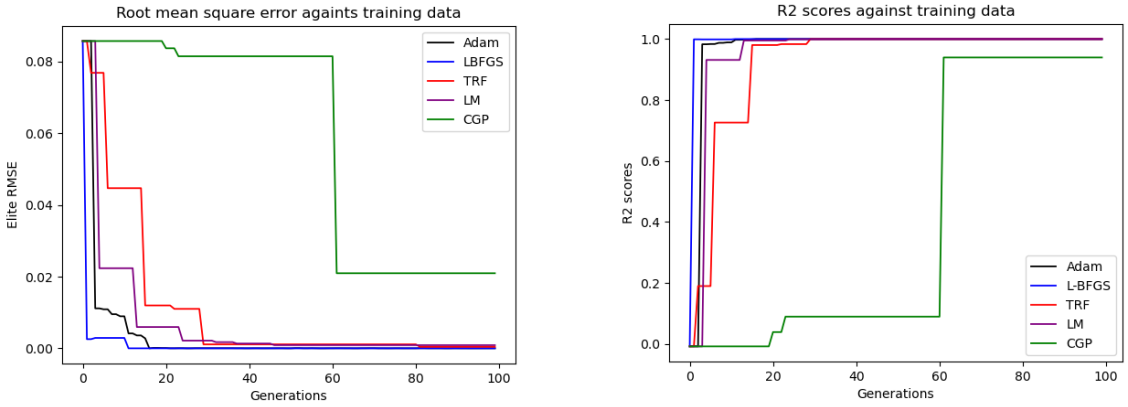
$$\tau_{ij}=2k\left(b_{ij,\text{Boussinesq}}+\sigma b_{ij}^{\Delta}+\frac{1}{3}\delta_{ij}\right) \tag{A.4}$$

B

Preliminary Results

B.1. Terminal Optimization - RMSE & Training R^2

B.1.1. $Function\ 1 - f_1(x_1, x_2)$

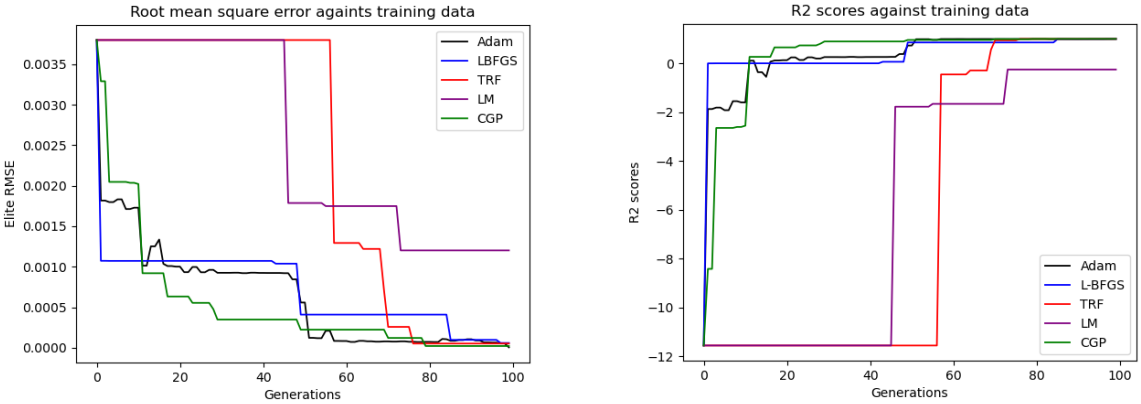


(a) RMSE of elite individual

(b) R^2 of elite individual

Figure B.1: Performance metrics (RMSE and R^2) for the elite individual of $f_1(x_1, x_2)$.

B.1.2. $Function\ 2 - f_2(x_1, x_2)$

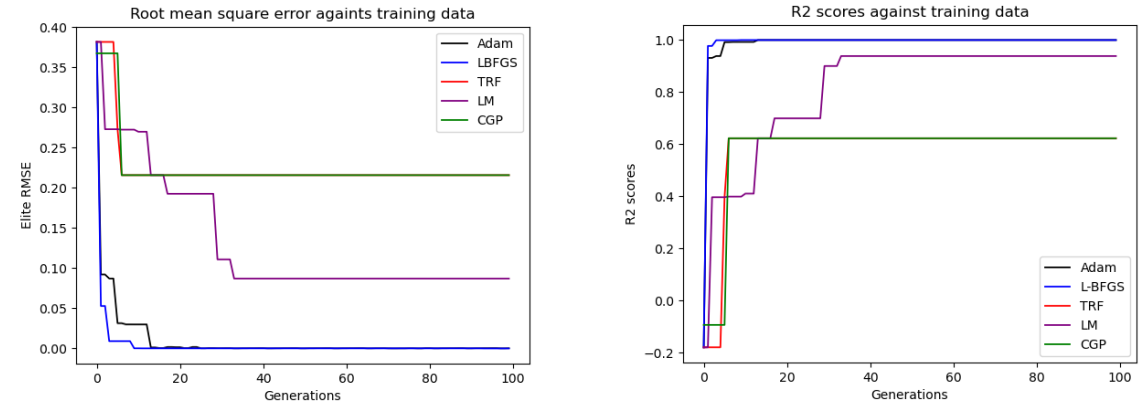


(a) RMSE of elite individual

(b) R^2 of elite individual

Figure B.2: Performance metrics (RMSE and R^2) for the elite individual of $f_2(x_1, x_2)$.

B.1.3. $Function\ 3 - f_3(x_1, x_2)$



(a) RMSE of elite individual

(b) R^2 of elite individual

Figure B.3: Performance metrics (RMSE and R^2) for the elite individual of $f_3(x_1, x_2)$.

B.2. Operator Optimization - Training R^2

B.2.1. $Function\ 1 - f_1(x_1, x_2)$

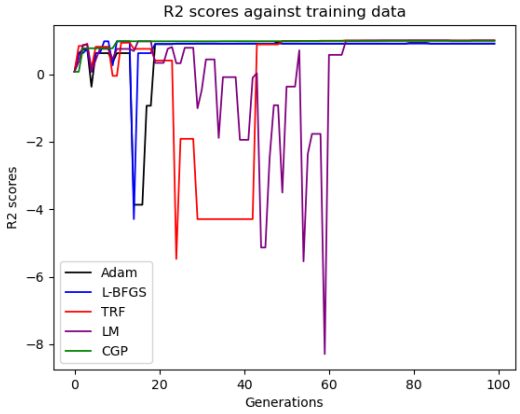


Figure B.4: R^2 of elite individual

B.2.2. $Function\ 2 - f_2(x_1, x_2)$

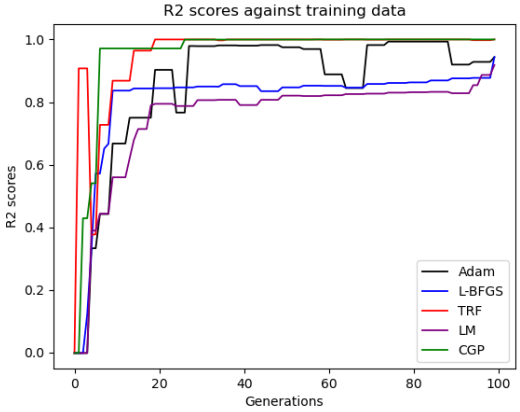


Figure B.5: R^2 of elite individual

B.2.3. $Function\ 3 - f_3(x_1, x_2)$

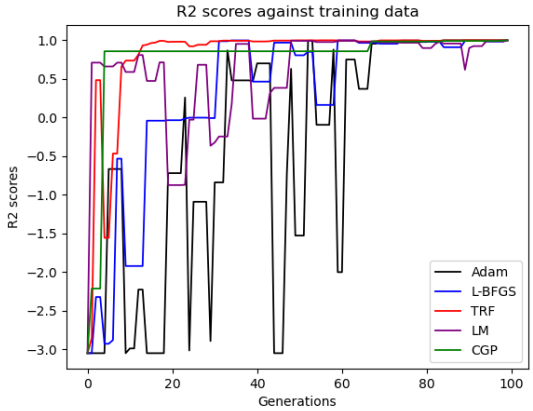


Figure B.6: R^2 of elite individual

B.3. Benchmark Testing - Phase I

B.3.1. Dataset 1: Median RMSE, Training R^2, Average Function Count & CPU Time

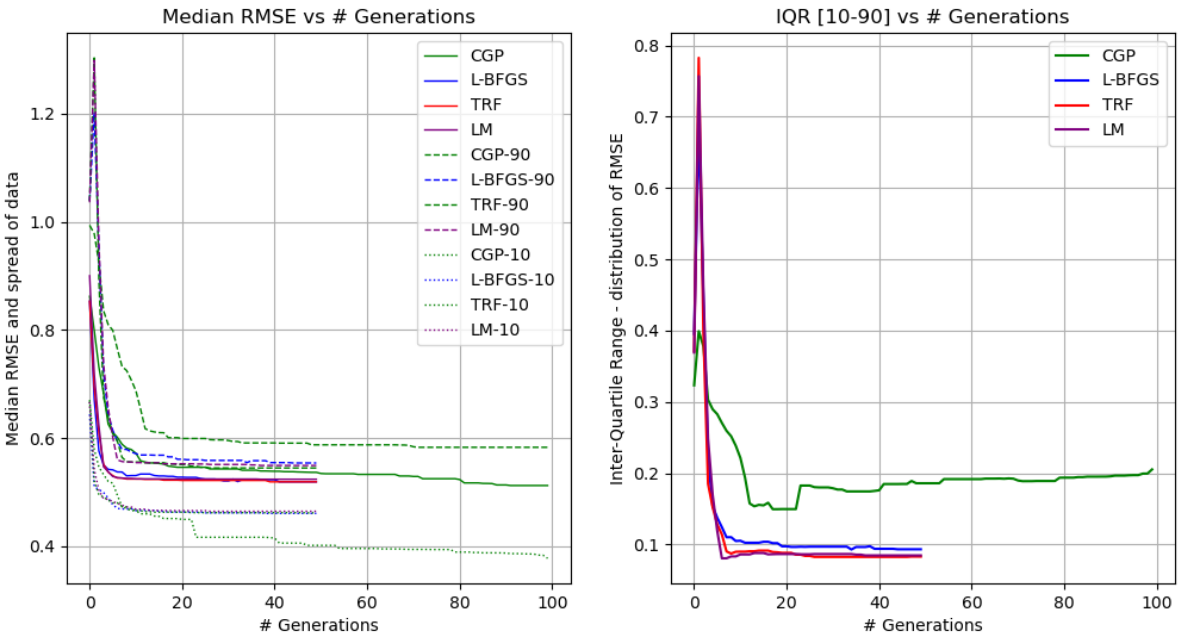


Figure B.7: RMSE: Dataset-1

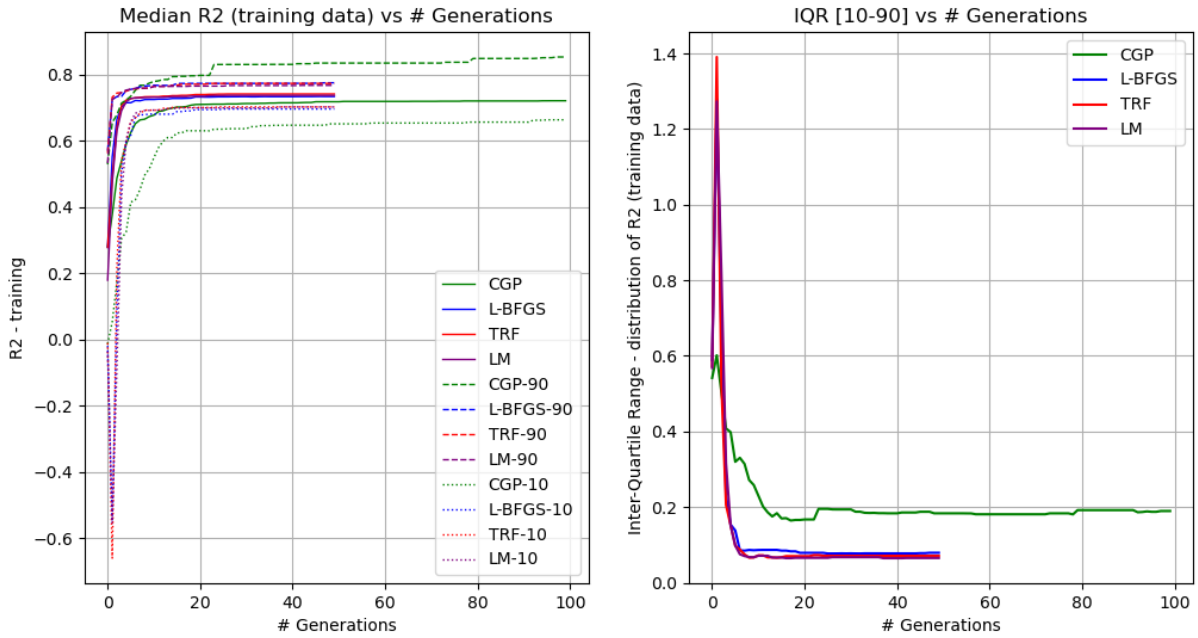


Figure B.8: Training R^2: Dataset-1

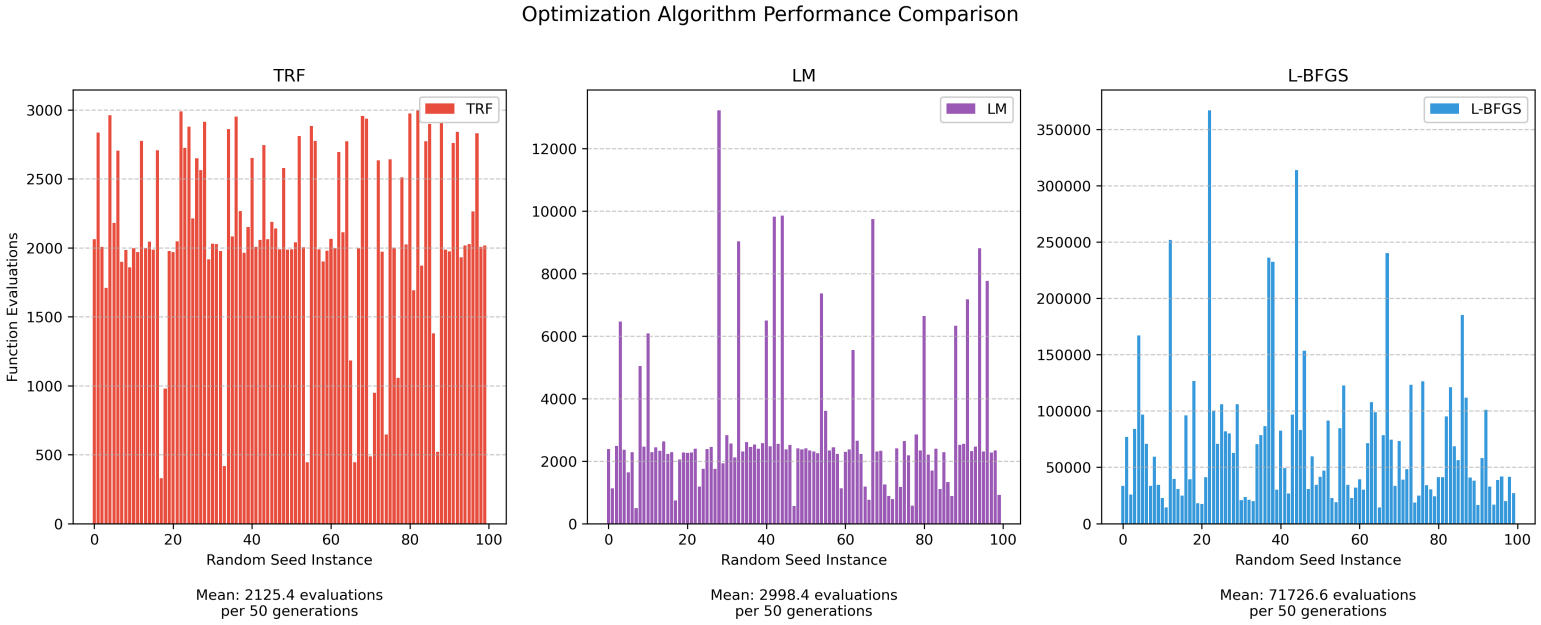


Figure B.9: Total function evaluations: Dataset-1

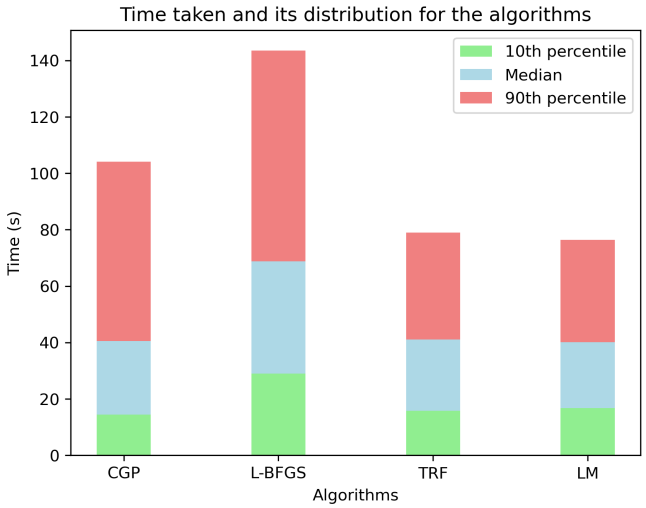


Figure B.10: Median computation time: Dataset-1

The above figure shows us the computational time, in seconds, that was required by each of the algorithms. Although this only serves as a pseudo-representation to the effectiveness of the algorithms, it does give us some insights. L-BFGS took up the most computational time which can be explained by the exponentially increased number of function evaluations. Most of the TRF, and LM iterations were in fact cheaper than the conventional method.

B.3.2. Dataset 2: Median RMSE, Training R^2, Average Function Count & CPU Time

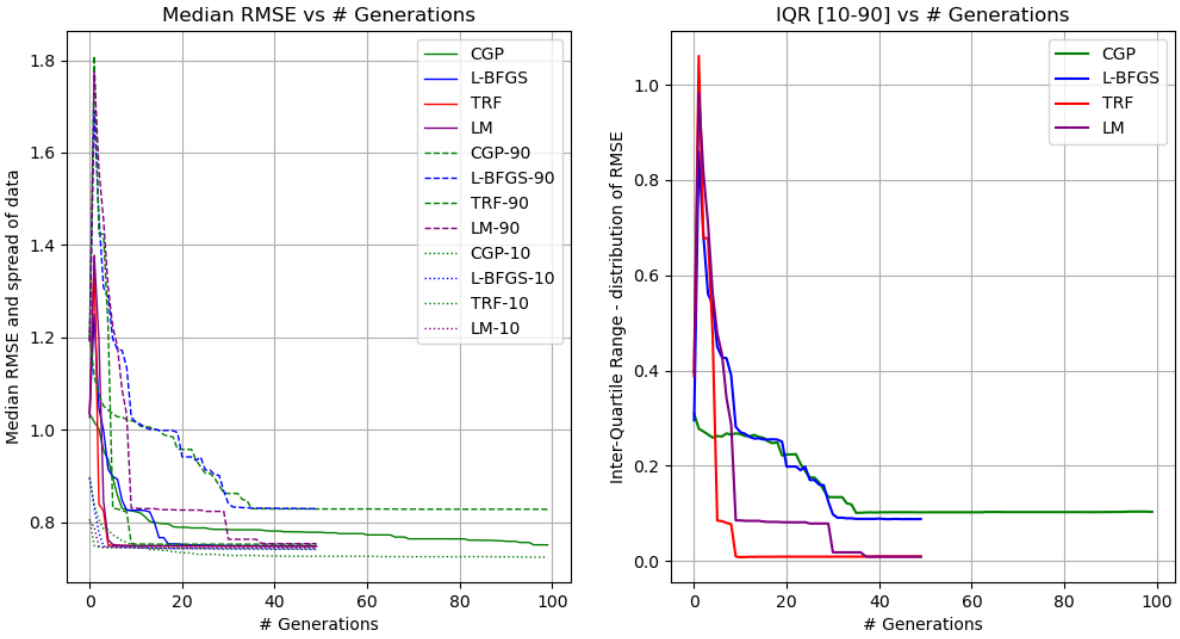


Figure B.11: RMSE: Dataset-2

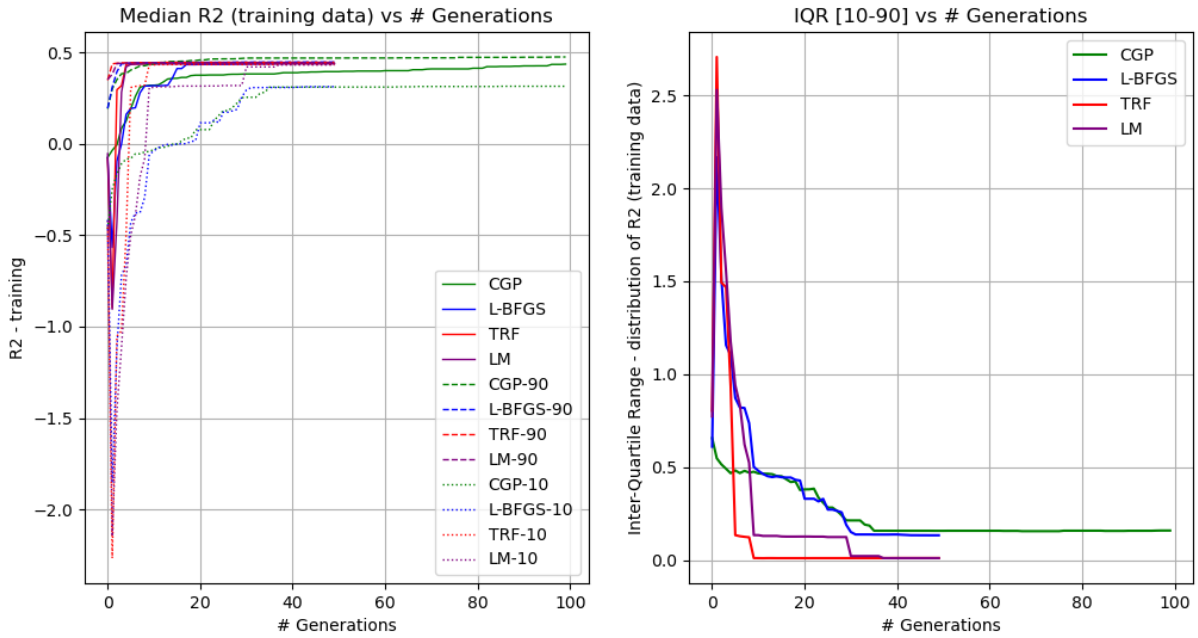


Figure B.12: Training R^2: Dataset-2

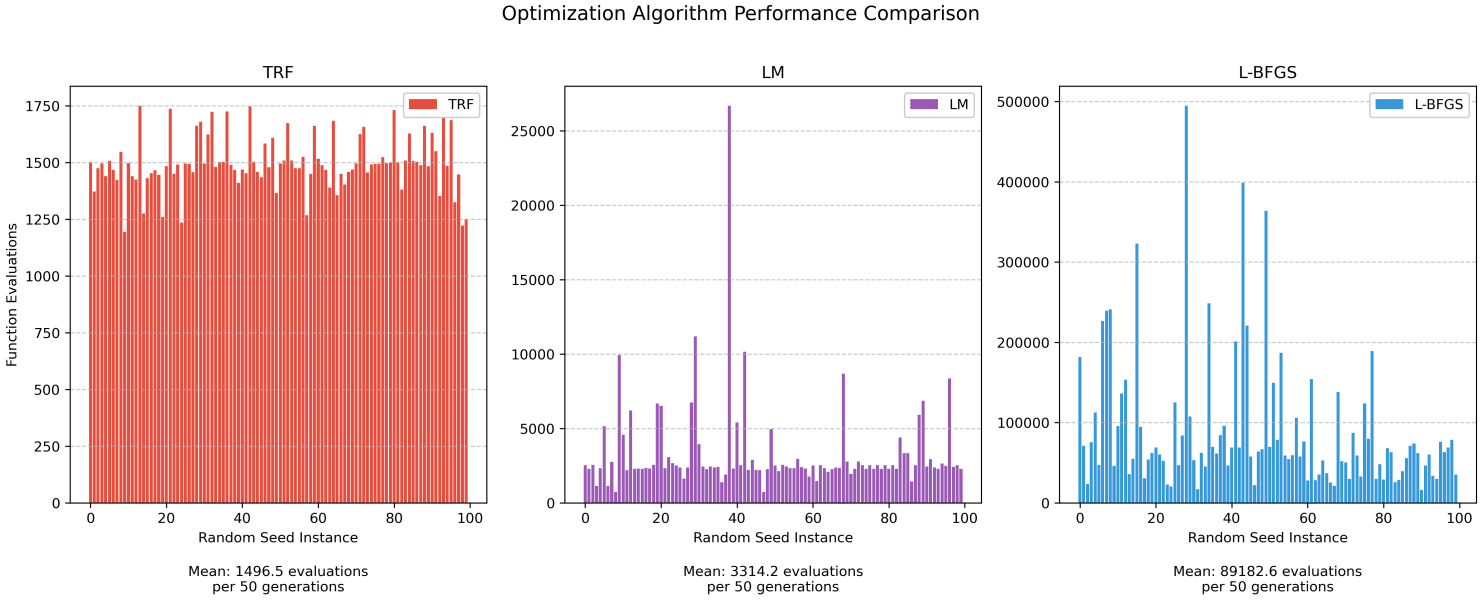


Figure B.13: Total function evaluations: Dataset-2

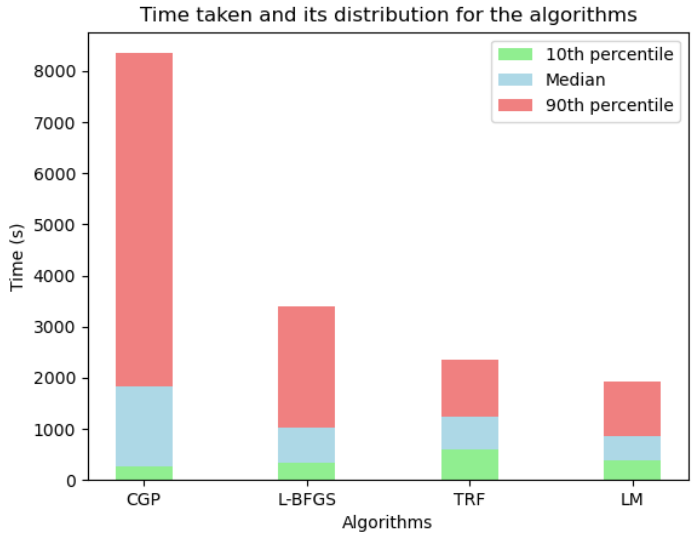


Figure B.14: Median computation time: Dataset-2

In terms of CPU time taken by the algorithms, the gradient based methods take impressively less time than the conventional methods. This can be explained by 2 reasons. Primarily, the said algorithms are run for 50 generations, while the conventional is made to run for 100 generations, hence having more individuals to evaluate. In addition to this, larger and/or complex expressions require more computational time to be evaluated and having a fitness value assigned to them. From the analysis, it was clear that CGP, more than often, had the lengthiest expressions with multiple sub-trees and nodes, which would eventually require more CPU time to be evaluated. Therefore, this explains the large 90^{th} distribution region, which would be affected by this.

B.3.3. Dataset 3: Median RMSE, Training R^2, Average Function Count & CPU Time

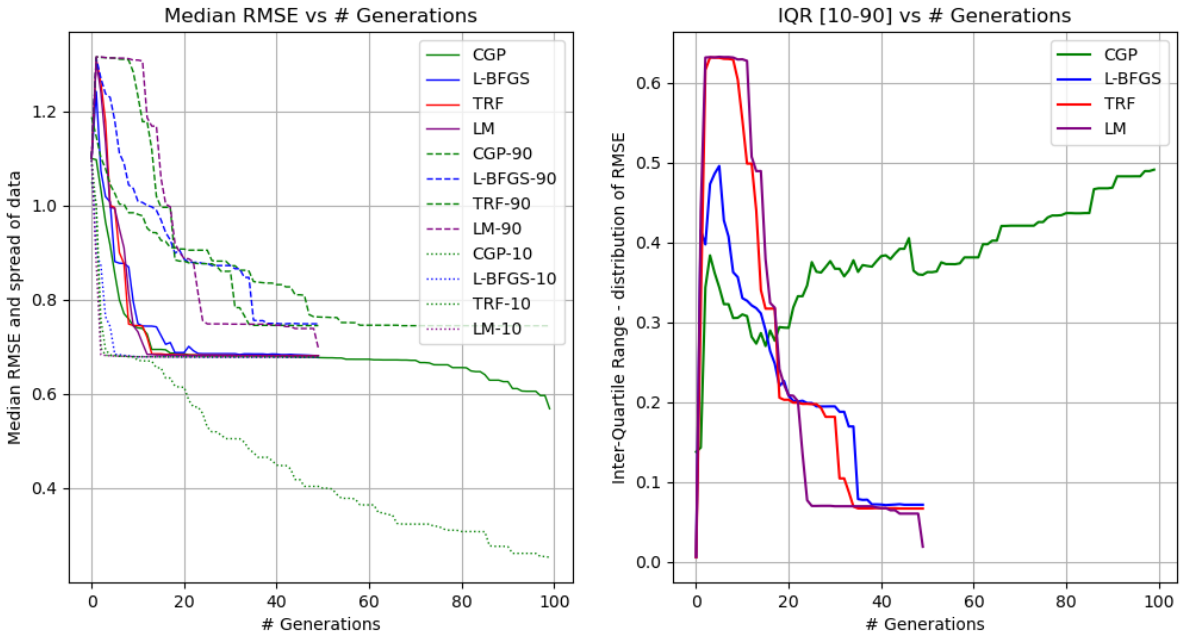


Figure B.15: RMSE: Dataset-3

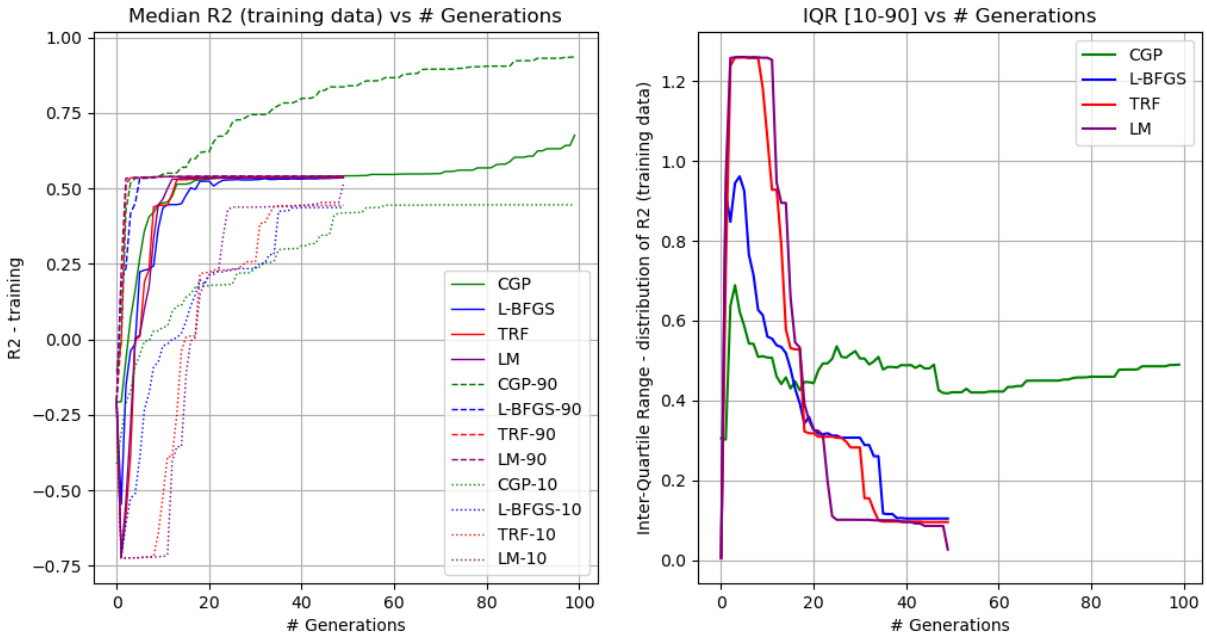


Figure B.16: Training R^2: Dataset-3

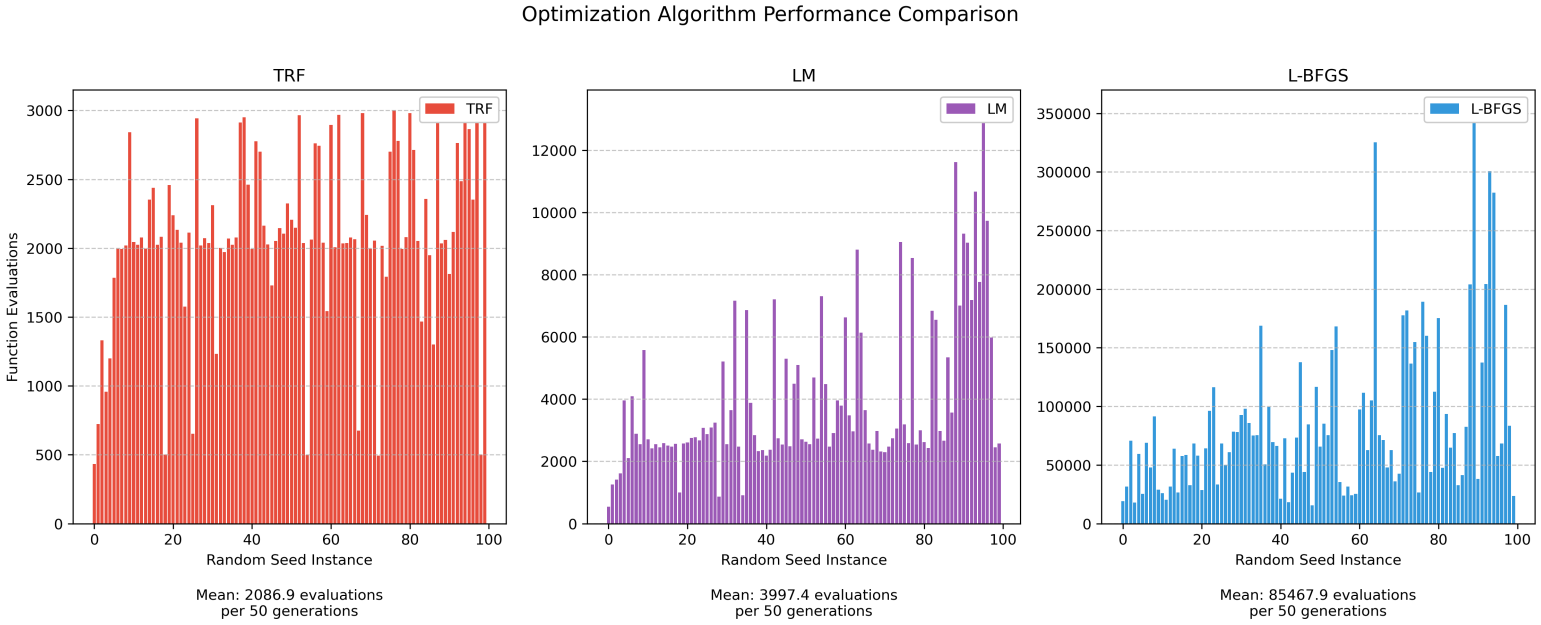


Figure B.17: Total function evaluations: Dataset-3

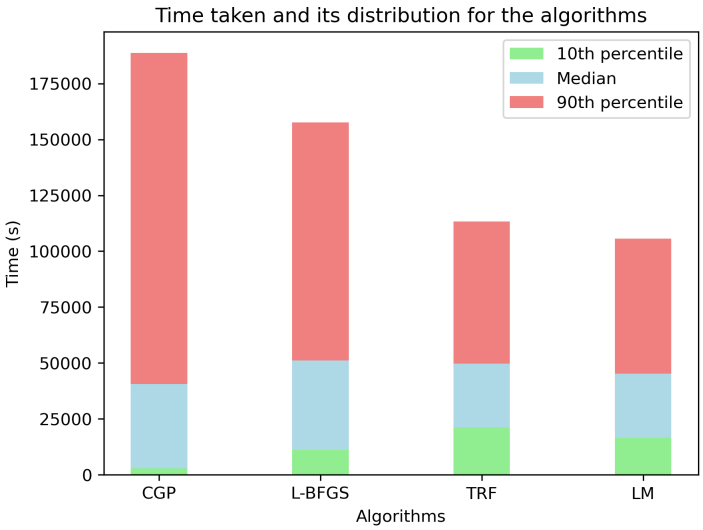


Figure B.18: Median computation time: Dataset-3

Identical to our previous insights, the conventional method incurs the highest computational time, followed by the optimized algorithms. The reasons remain the same as explained earlier: the reduced number of generations and the tendency to converge to shorter, simpler expressions allowing for faster evaluation, thereby contributing to lower CPU times for the gradient based algorithms and allowing it to finish a random seed iteration quickly.

B.4. Benchmark Testing - phase II

B.4.1. Dataset 1: Median RMSE, Training R^2 & Average Function Count

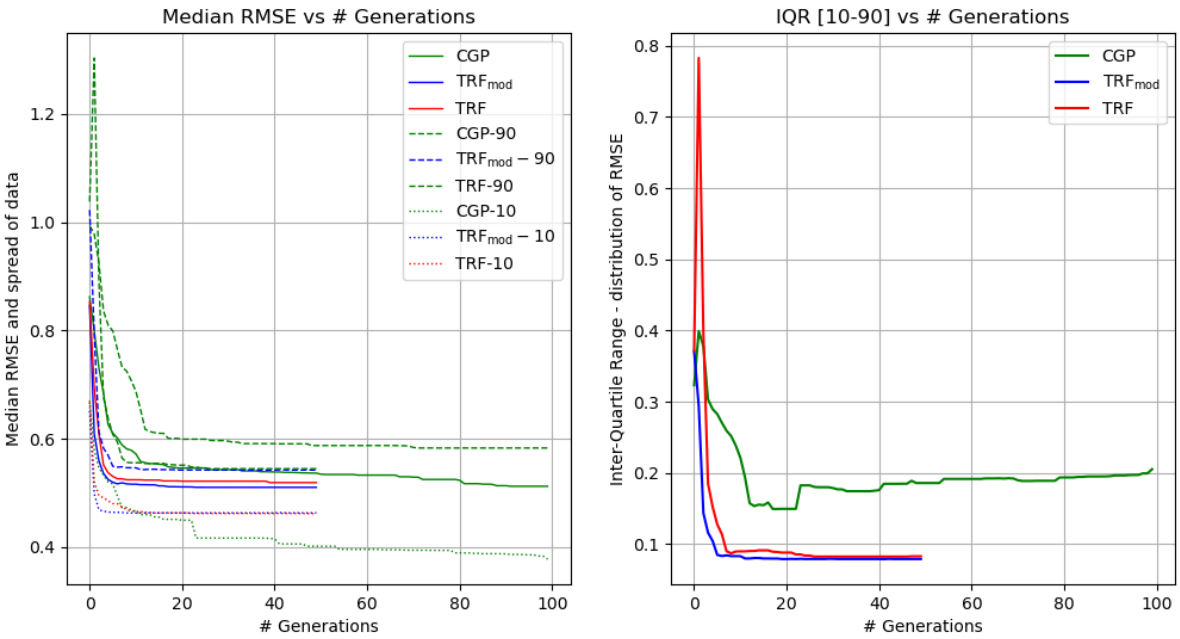


Figure B.19: Median RMSE: Dataset-1

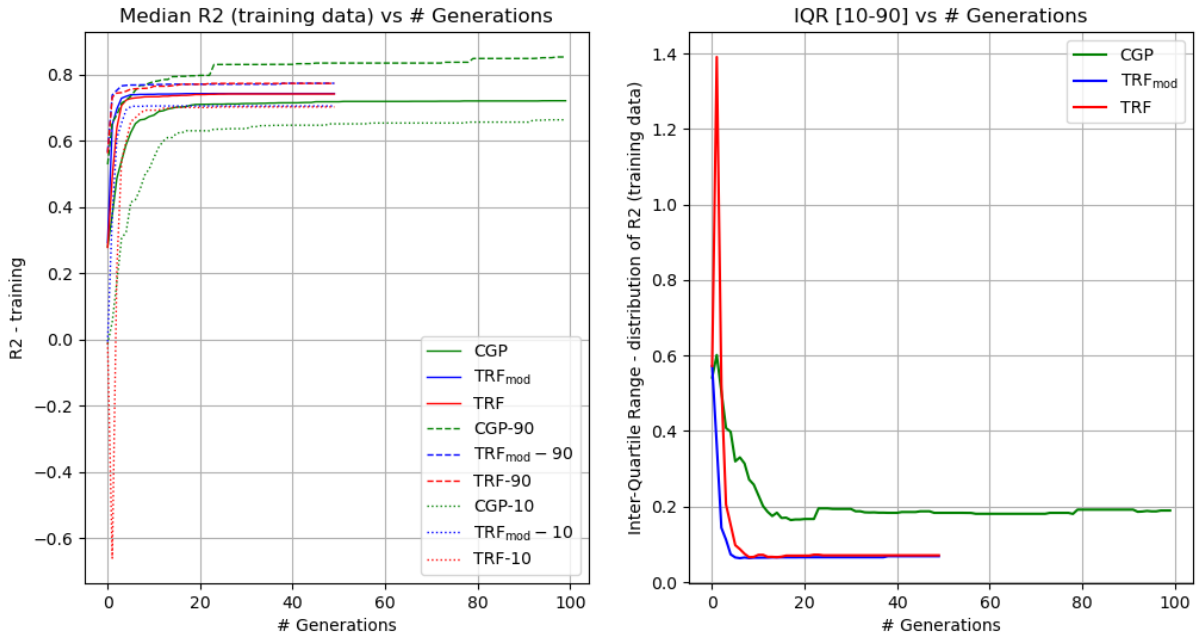


Figure B.20: Median Training R^2: Dataset-1

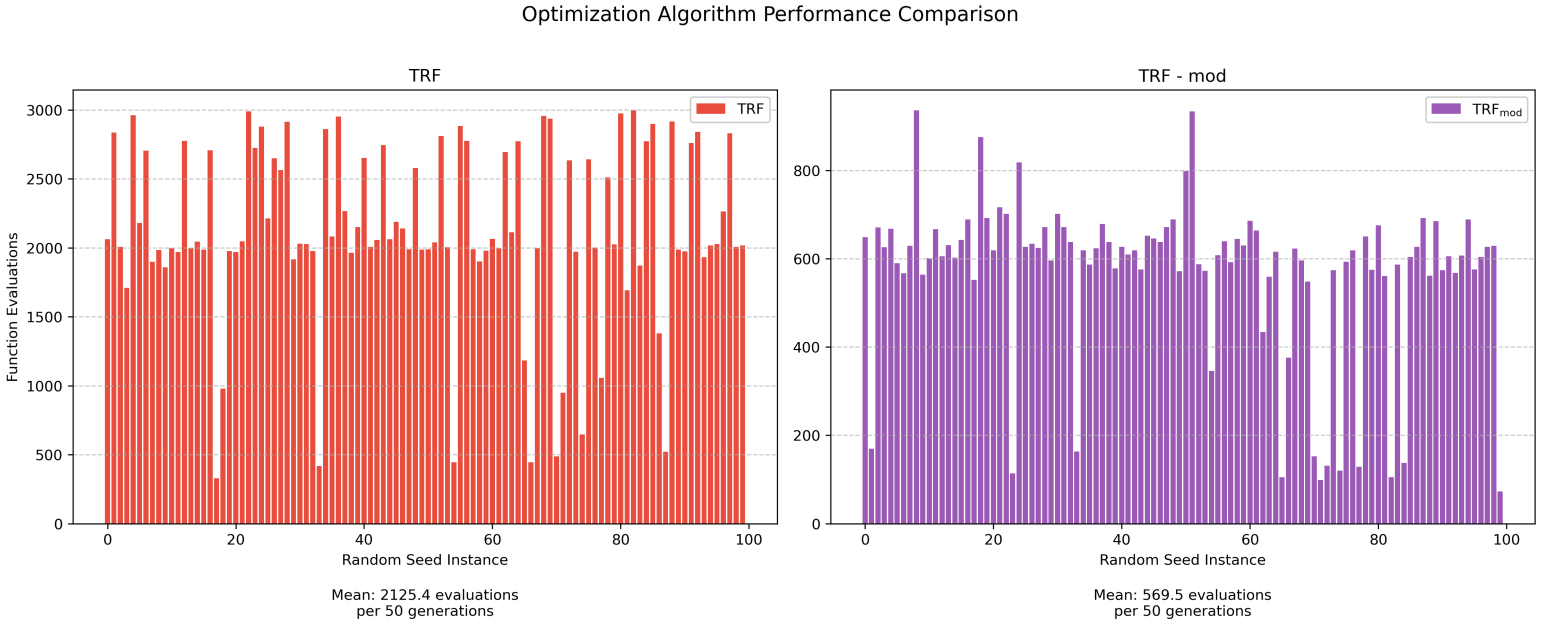


Figure B.21: Total function evaluations: Dataset-1

B.4.2. Dataset 2: Median RMSE, Training R^2 & Average Function Count

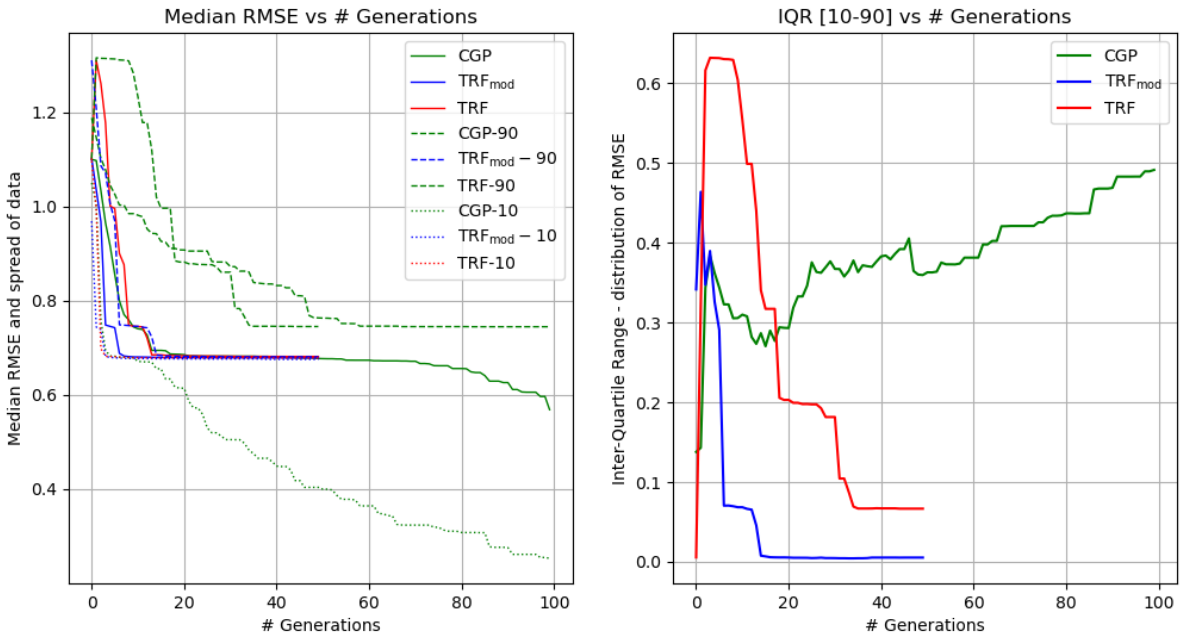


Figure B.22: Median RMSE: Dataset-2

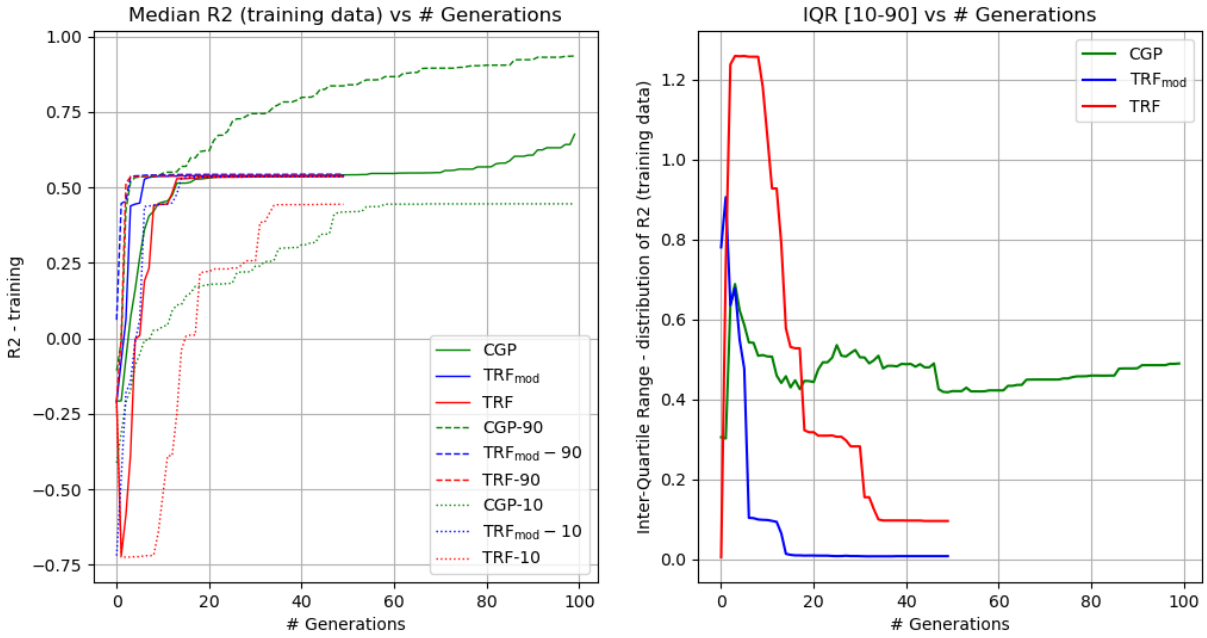


Figure B.23: Median Training R^2: Dataset-2

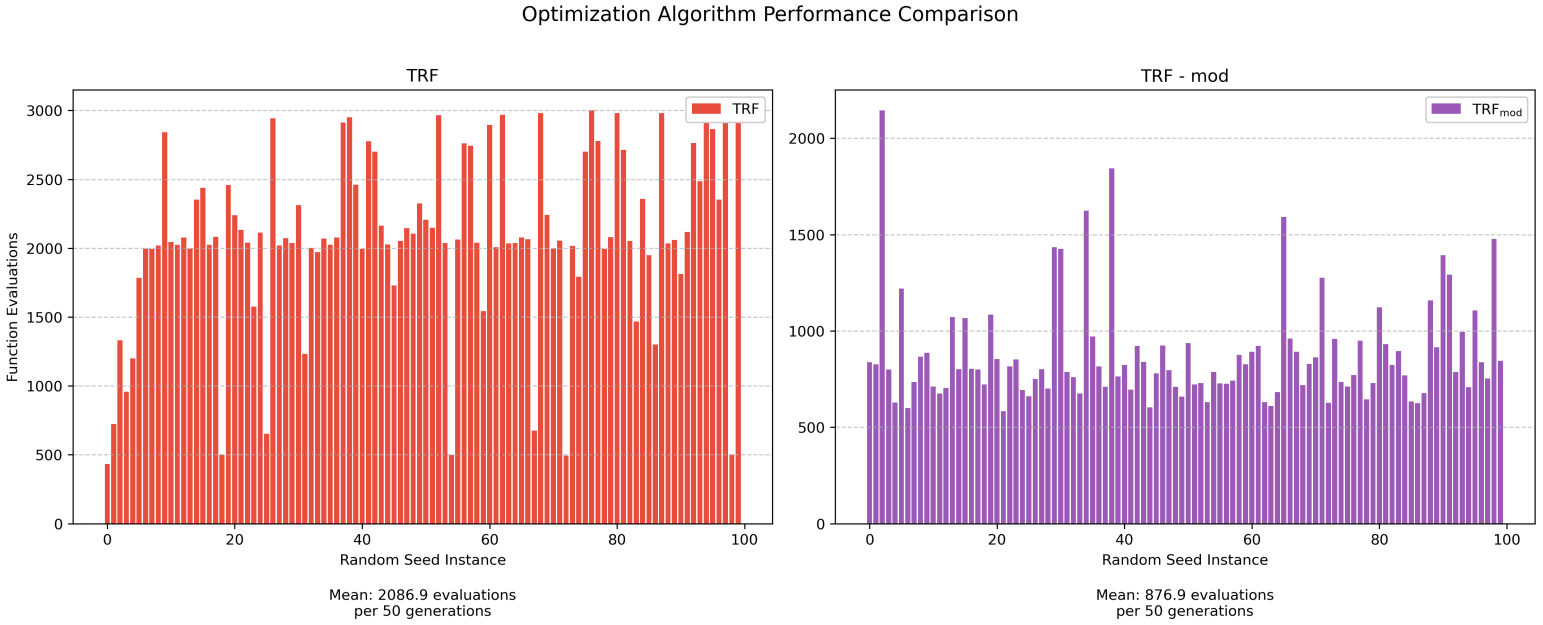


Figure B.24: Total function evaluations: Dataset-2

B.5. Preliminary Testing of Kolmogorov Arnold Networks

B.5.1. Optimizer Parameters

Some important parameters that were used during this preliminary testing of KANs with such datasets are tabulated below.

Parameter	Settings
`grid`	5
`grid_eps`	1
`optimizer`	$LBFGS$
`max_iter`	20
`epochs`	7
`batch_size`	500

Table B.1: Parameter settings for preliminary testing of KANs

B.5.2. Symbolic Formula of Function 1

`[2, 5, 1] KAN network`

The final symbolic formula is

$$
\begin{aligned}
f(x_1, x_2) = {} & 26.9805\,(0.9732\sin(0.1097x_2 + 8.052) - 1)^2 \\
& + 0.7703\left(-0.3738(-x_2 - 0.1632)^2 + \sin(1.7715x_1 - 3.1889) - 0.1454\right)^2 \\
& - 0.3241\sin\left(21.165\sin(0.1658x_1 + 7.9741) - 32.9382\sin(0.1602x_2 - 4.6184) + 5.684\right) \\
& - 0.4899\sin\left(0.9236\sin(1.3389x_1 - 0.4286) - 1.5045\sin(0.5587x_2 - 4.0216) + 11.2972\right) \\
& - 6.4014\sin\left(0.1235\sin(2.6429x_1 - 2.4094) - 3.9255\sin(0.1287x_2 + 4.5107) + 3.6472\right) \\
& + 6.2865
\end{aligned}
$$

`[2, 5, 5 ,1] KAN network`

The final symbolic formula is

$$
\begin{aligned}
f(x_1, x_2) = 49.6885\Bigg(& 0.0899\,(0.1357\sin(1.4846x_1 + 6.0228) - 0.8703\sin(0.4x_2 - 4.64) + 1)^2 \\
& + 0.0358\sin(1.1694x_1 + 5.7952) - 0.0652\sin(0.772x_2 - 4.4224) \\
& + 0.0129\sin\left(0.2274(-0.9625x_2 - 1)^2 - 0.622\sin(1.2639x_1 - 9.7794) + 2.6953\right) \\
& + \sin\left(0.691\sin(0.5242x_1 + 5.1474) - 0.0258\sin(1.9986x_2 + 1.2052) + 2.1343\right) \\
& - 0.1012\sin\left(-1.4627\sin(0.9252x_1 + 5.552) + 0.6652\sin(1.164x_2 + 1.6066) + 0.0549\right) - 0.8745\Bigg)^2 \\
& - 0.7336\sin\Big(0.5949\sin(0.9252x_1 + 5.552) - 0.2705\sin(1.164x_2 + 1.6066) \\
& \quad + 1.7365\sin(2.0149\sin(0.5242x_1 + 5.1474) - 0.0751\sin(1.9986x_2 + 1.2052) + 0.797) - 1.395\Big) \\
& - 0.3155\sin\Big(0.4811\sin(0.1208(-0.9625x_2 - 1)^2 - 0.3304\sin(1.2639x_1 - 9.7794) + 9.7487) \\
& \quad - 3.3261\sin(2.1509\sin(0.5242x_1 + 5.1474) - 0.0802\sin(1.9986x_2 + 1.2052) + 2.1056) \\
& \quad - 0.84\sin(-1.7569\sin(0.9252x_1 + 5.552) + 0.799\sin(1.164x_2 + 1.6066) + 7.919) + 7.5524\Big) \\
& + 0.668\sin\Big(-0.4721\sin(1.1694x_1 + 5.7952) + 0.8586\sin(0.772x_2 - 4.4224) \\
& \quad + 0.8031\sin(4.1842\sin(0.5242x_1 + 5.1474) - 0.156\sin(1.9986x_2 + 1.2052) + 11.8769) \\
& \quad - 1.1154\sin(1.8125\sin(0.9252x_1 + 5.552) - 0.8243\sin(1.164x_2 + 1.6066) + 6.3279) + 8.5172\Big) \\
& - 1.229\sin\Big(0.3077\sin(-0.3216(-0.9625x_2 - 1)^2 + 0.8797\sin(1.2639x_1 - 9.7794) + 8.7574) \\
& \quad - 5.927\sin(1.2754\sin(0.5242x_1 + 5.1474) - 0.0476\sin(1.9986x_2 + 1.2052) + 0.1779) \\
& \quad + 0.6347\sin(2.95\sin(0.9252x_1 + 5.552) - 1.3416\sin(1.164x_2 + 1.6066) + 11.0924) \\
& \quad + 1.1114\sin(0.71\sin(1.1694x_1 + 5.7952) - 1.2912\sin(0.772x_2 - 4.4224) + 10.8675) \\
& \quad + 1.938\sin(0.6429\sin(1.4846x_1 + 6.0228) - 4.1242\sin(0.4x_2 - 4.64) + 3.5257) - 5.3774\Big) \\
& + 0.8504
\end{aligned}
$$

It is not at all user-friendly to understand the equation and its interpretability. However, this is not surprising. As each activation function is being approximated by the library, and we can observe that not many connections are pruned, hence, the number of approximations to be made increases drastically. This translates to an overall complex equation and comparable behavior was also reported when KANs were applied to Knot Theory, wherein the resulting symbolic expression doubled in its length when just one neuron was added in a hidden layer. Similar effect was observed when a new hidden layer was added.

B.5.3. Symbolic Formula of Function 2

`[2, 5, 1] KAN network`

The final symbolic formula is

$$
\begin{aligned}
f(x_1, x_2) = \; & -2.148 \sin(0.2702 x_2 + 1.3568) \\
& + 3.0112 \sin(-0.4222 x_1 + 0.4331 x_2 + 1.3744) \\
& + 1.5009 \sin(0.6948 x_1 + 17.0494 \sin(0.0906 x_2 - 7.7259) + 15.7934) \\
& - 8.3982 \sin(0.1834 x_2 - 0.8191 \sin(0.116 x_1 + 2.0) + 2.1223) \\
& + 0.2783 \sin(3.5614 \sin(0.6192 x_1 - 0.9952) + 38.5033 \sin(0.0962 x_2 - 1.4746) + 42.7439) \\
& + 8.5873
\end{aligned}
$$

Despite converging to a very a good model, the symbolic equation remains rather complex to understand and lengthy as well. A way around this issue is to give a more shorter library with only simple functions, an idea explored further ahead.

`[2, 5, 5, 1] KAN network`

The symbolic formula is

$$
\begin{aligned}
& f(x_1, x_2) = \\
& \quad 0.2514 \cdot \sin\Big(1.9542 \cdot (-\sin(0.2601 x_1 - 4.5027) + 0.9084 \cdot \sin(0.5943 x_2 + 5.1777) + 0.9444)^2 \\
& \quad + 2.0183 \cdot \sin(0.5609 x_1 + 2.3803) + 0.5348 \cdot \sin(0.2859 x_2 - 2.0241) \\
& \quad - 1.1482 \cdot \sin(0.1319 x_2 - 1.779 \cdot \sin(0.6029 x_1 - 3.9735) + 7.894) + 3.5044 \Big) \\
& - 0.759 \cdot \sin\Big(0.2768 \cdot (-0.5541 x_2 - 0.2833 \cdot \sin(2.9578 x_1 + 6.8118) - 1)^2 \\
& \quad - 1.7121 \cdot (0.1085 x_2 - \sin(0.1559 x_1 + 1.7361) + 0.6554)^2 \\
& \quad - 0.1061 \cdot \sin(-0.5073 x_2 + 6.8422 \cdot \sin(0.6029 x_1 - 3.9735) + 0.5224) \\
& \quad + 0.5307 \cdot \sin(0.3618 \cdot \sin(0.2601 x_1 - 4.5027) - 0.3286 \cdot \sin(0.5943 x_2 + 5.1777) + 8.4657) \\
& \quad + 1.974 \cdot \sin(1.7995 \cdot \sin(0.5609 x_1 + 2.3803) + 0.4769 \cdot \sin(0.2859 x_2 - 2.0241) + 3.5167) + 7.2767 \Big) \\
& + 0.5887 \cdot \sin\Big(3.9345 \cdot \sin(0.5609 x_1 + 2.3803) + 1.0426 \cdot \sin(0.2859 x_2 - 2.0241) \\
& \quad + 6.3952 \cdot \sin(0.2045 x_2 + 0.1046 \cdot \sin(2.9578 x_1 + 6.8118) + 8.319) \\
& \quad - 0.4524 \cdot \sin(1.7621 x_2 - 16.2394 \cdot \sin(0.1559 x_1 + 1.7361) + 21.5225) \\
& \quad + 0.2535 \cdot \sin(3.3934 \cdot \sin(0.2601 x_1 - 4.5027) - 3.0827 \cdot \sin(0.5943 x_2 + 5.1777) + 4.0266) - 13.0778 \Big) \\
& - 3.8156 \cdot \sin\Big(0.2884 \cdot \sin(-0.2611 x_2 + 3.5213 \cdot \sin(0.6029 x_1 - 3.9735) + 7.1809) \\
& \quad - 5.1186 \cdot \sin(0.1305 x_2 - 1.2025 \cdot \sin(0.1559 x_1 + 1.7361) + 2.7964) \\
& \quad + 1.6922 \cdot \sin(0.2066 x_2 + 0.1056 \cdot \sin(2.9578 x_1 + 6.8118) - 4.8701) \\
& \quad - 0.4383 \cdot \sin(-1.8254 \cdot \sin(0.2601 x_1 - 4.5027) + 1.6583 \cdot \sin(0.5943 x_2 + 5.1777) + 6.7094) \\
& \quad - 0.2997 \cdot \sin(8.3457 \cdot \sin(0.5609 x_1 + 2.3803) + 2.2115 \cdot \sin(0.2859 x_2 - 2.0241) - 10.9034) + 11.7175 \Big) \\
& + 2.4627
\end{aligned}
$$

As expected, with the increased depth of the network, the final equation also consequentially becomes more and more intricate and challenging to comprehend.

B.5.4. Symbolic Formula of Function 3

`[4, 3, 3, 1] KAN network`

The resulting symbolic equation is

$$
\begin{aligned}
&f(x_1, x_2, x_3, x_4) = \\
&\quad - 0.4545 \cdot \sin\Big(2.1 \cdot \cos\Big(0.0658x_2 + 0.102x_3 - 0.1088x_4 + 0.0314 \cdot \cos(1.4134x_1 - 5.5828) + 5.9437\Big) - 3.5832\Big) \\
&+ 1.4964 \cdot \sin\Big(0.1074x_2 + 0.1667x_3 - 0.1777x_4 - 1.1263 \cdot \sin\Big(- 0.2164x_2 + 0.2776x_3 - 0.3475x_4 \\
&\qquad + 0.1899 \cdot \cos(1.4005x_1 + 6.7965) + 4.6985\Big) + 0.0513 \cdot \cos(1.4134x_1 - 5.5828) + 1.5488\Big) \\
&+ 1.0843 \cdot \cos\Big(0.0973x_2 + 0.151x_3 - 0.161x_4 - 3.3593 \cdot \sin\Big(- 0.1116x_2 + 0.1431x_3 - 0.1792x_4 \\
&\qquad + 0.0979 \cdot \cos(1.4005x_1 + 6.7965) + 4.2424\Big) + 0.0464 \cdot \cos(1.4134x_1 - 5.5828) - 1.5842\Big) + 6.3465
\end{aligned}
$$

`[4, 3, 1] KAN network`

The resulting symbolic formula is

$$f(x_1, x_2, x_3, x_4) = 0.5583x_2 - 0.5121x_3 + 0.5666x_4 + 0.5615 \cdot \cos(1.2388x_1 - 9.3118) - 1.0542$$

The inherent simplistic nature of the resulting equation itself makes the choice of a shorter network as more favorable.

`Pruned Inputs - [4, 3, 1] KAN network`

The resulting symbolic formula is

$$f(x_1, x_2, x_3, x_4) = 0.5027x_1 - 0.4828x_2 + 0.5176x_3 - 0.223$$

The complexity of the equation is further reduced with just 3 inputs and no trigonometric functions as well.

B.5.5. Function 4

For the given function, a `[2, 5, 1]` KAN network is applied. The same library as subsection 4.2.3 is implemented. Note that this is the dimensionless formula of the gravitational potential energy, defined by the following.

$$F = Gm_1m_2(\frac{1}{r_2} - \frac{1}{r_1}) \tag{B.1}$$

The results are presented below.

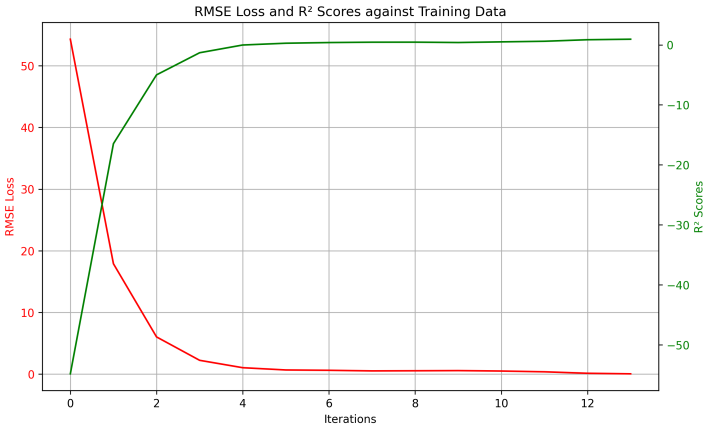


Figure B.25: RMSE and R^2 - Function 4: `[2, 5, 1]`

Despite early struggles, the resulting model is able to converge to outstanding metrics in less than 10 iterations. This once again highlights the impressive capability of KANs to very quickly learn the optimized parameters, and model the functional space.

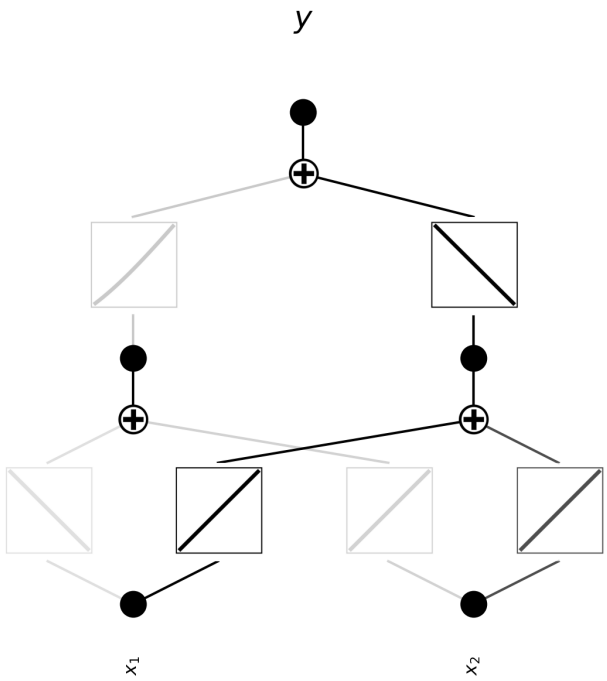


Figure B.26: Pruned network - Function 4: `[2, 5, 1]`

KANs continue to exhibit remarkable effectiveness of pruning for shorter networks. It can be sen that a `[2, 5, 1]` network is now reduced to just a `[2, 2, 1]` architecture, reducing the number of parameters and the computational cost by a large margin. This further speeds up the training process, and makes the model itself more easily understandable and interpretable.

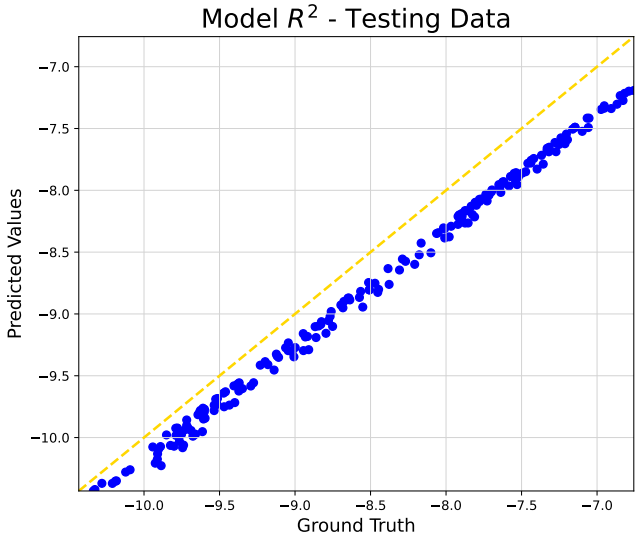


Figure B.27: Scatter plot - Function 4: `[2, 5, 1]`

Table B.2: Model Performance: R^2 Scores

Dataset	R^2 **Score**
Training	0.968
Testing	0.970

Strong correlations are evident from the above plot. The model is able to represent the functional space in a superior manner, with excellent scores.

The obtained symbolic formula is

$$f(x_1, x_2) = -0.9032x_1 - 0.1025x_2 - 2.71 \cdot \sin(-0.0558x_1 + 0.0785x_2 + 8.0287) + 2.7203$$

We observe that the resulting equation is relatively concise and not particularly difficult to evaluate. Although it appears more computationally intensive than the underlying function, it is important to note that it achieves high correlation and significantly low RMSE values using only three functions—none of which include a division operator, despite the original function being primarily composed of such operations.

C

A priori CFD additional Results

C.1. Genetic Programming with gradient descent for PH - ϵ

C.1.1. Median Plots

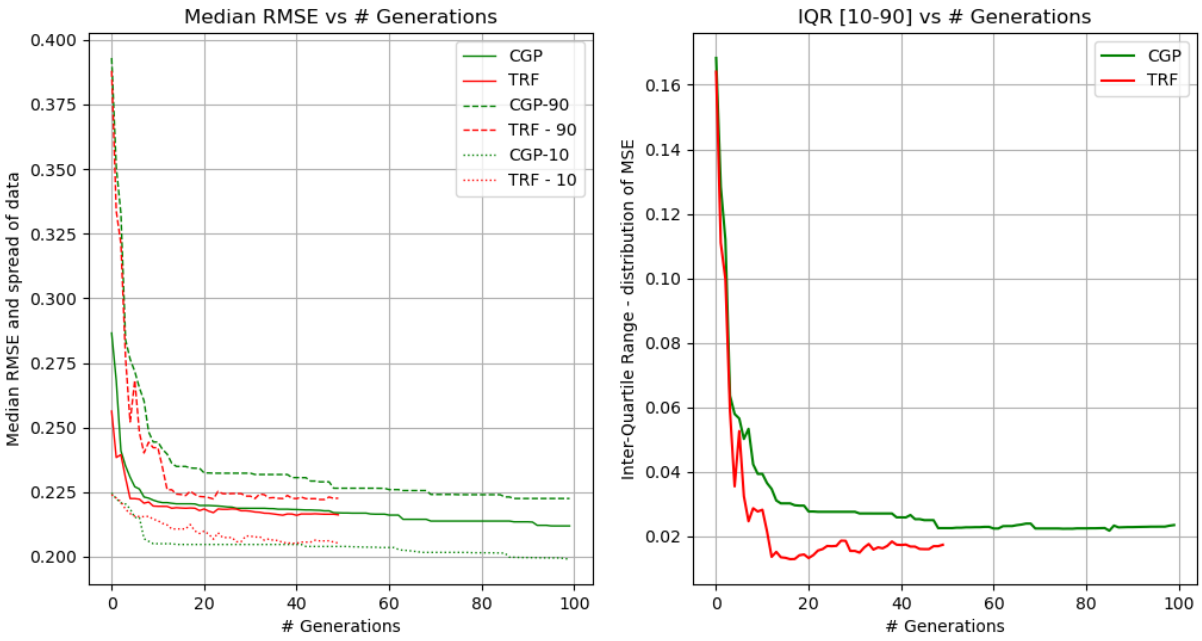


Figure C.1: Median RMSE: $k_{deficit}$ for PH based on ϵ

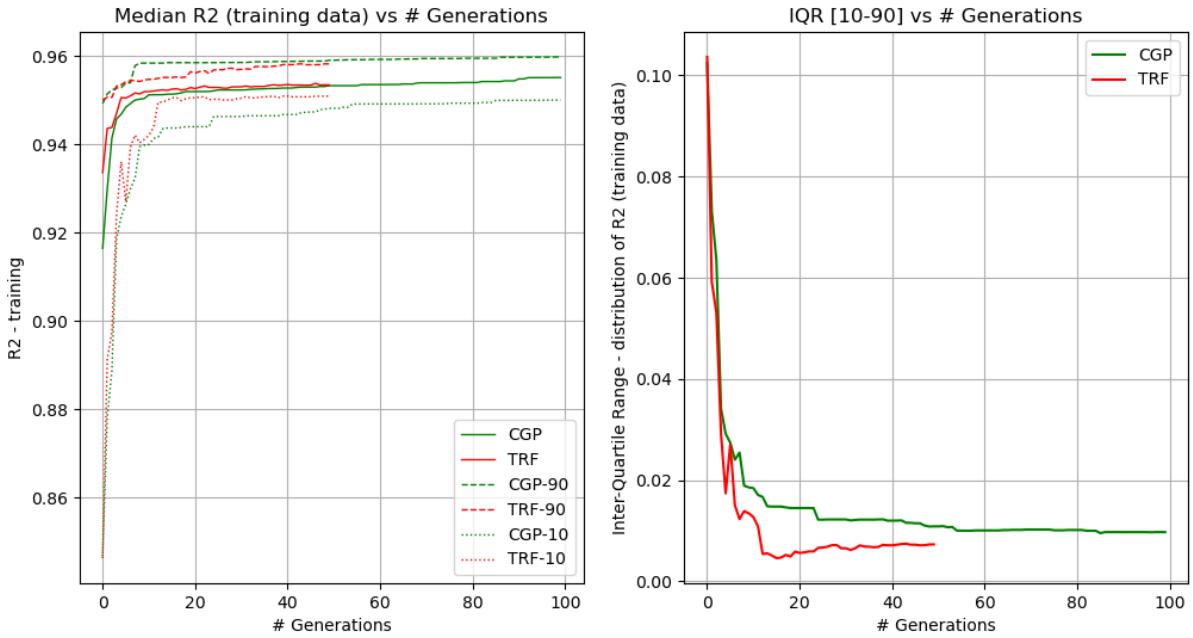


Figure C.2: Median training R^2: $k_{deficit}$ for PH based on ϵ

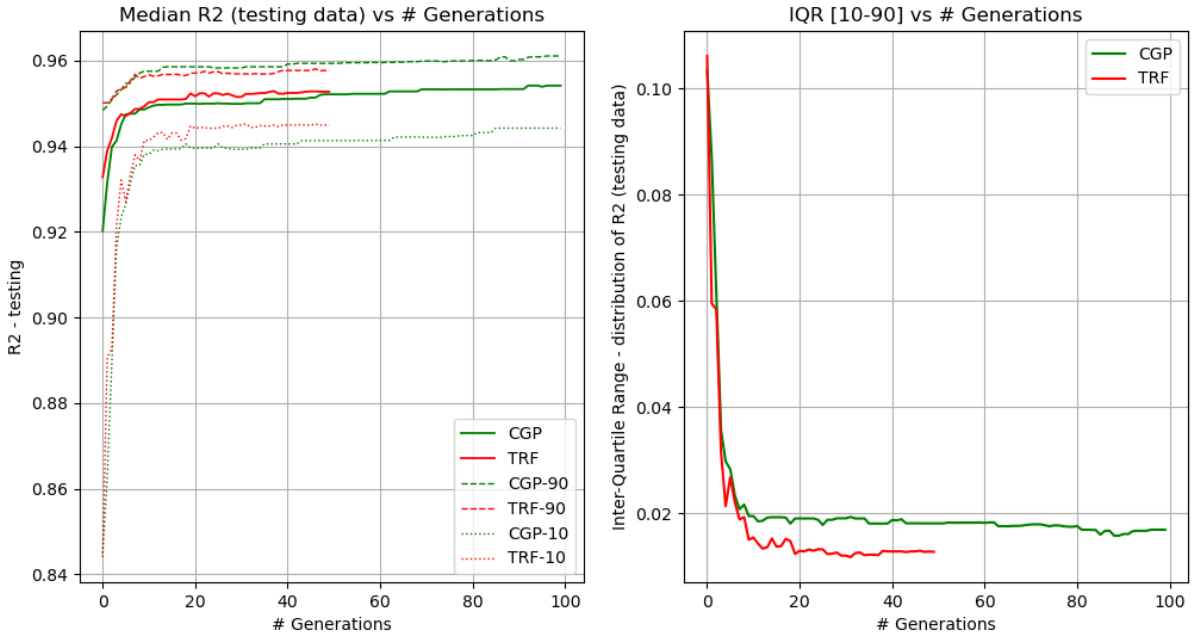


Figure C.3: Median Testing R^2: $k_{deficit}$ for PH based on ϵ

The above results clearly indicate that the combination of genetic programming with gradient optimizer is able to achieve superior results. Although the R^2, and $RMSE$ scores are only slightly better, the solution space is much more confined when compared to the classical method. This is exactly what the current research aimed to accomplish - gradient optimization able to exploit the solution space, allowing for consistent convergence to high quality individuals with little variation in the converged solutions.

This is inferred from the percentile and the IQR plots. It can be clearly observed that the values of the latter are significantly lower for the proposed algorithm. The difference mainly arises from the methods' capability to not only have its top 10% converge to at least the same scores as those of the conventional method, but also have the bottom 10% of the individuals be significantly better than that of classical genetic programming. Thus, the solution space is more refined, and of better quality overall.

This gives a strong indication that the proposed algorithm is very effective, and able to achieve its objectives.

C.1.2. Logarithmic Plots

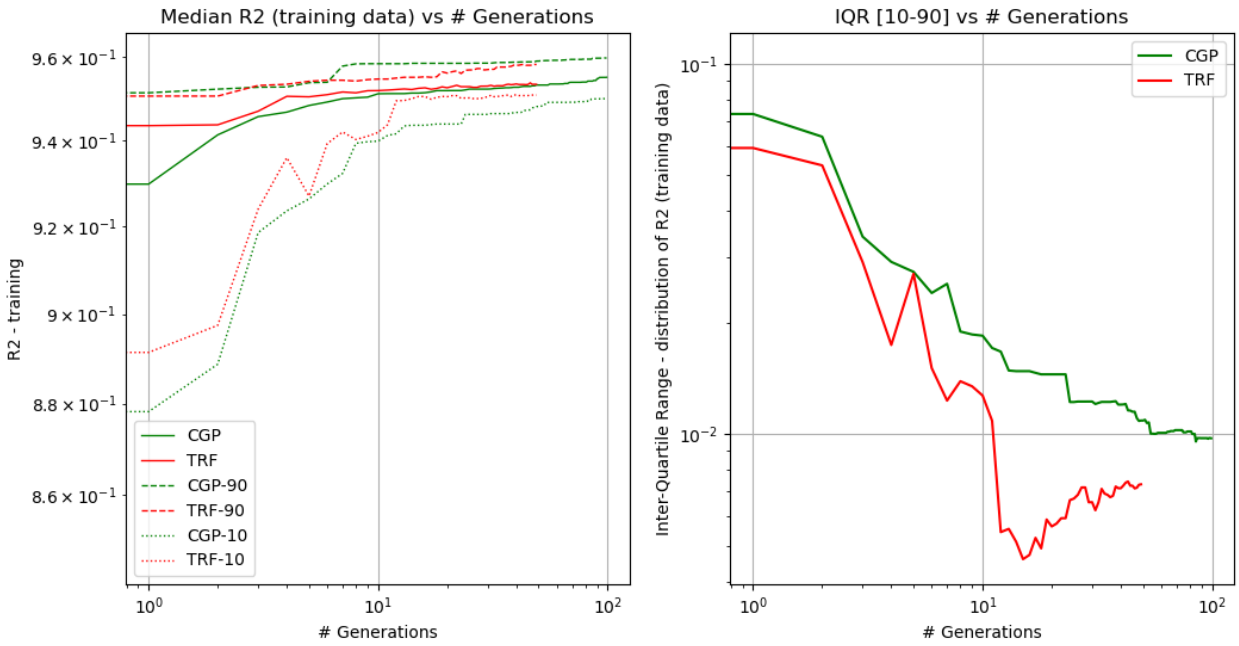


Figure C.4: Log Median training R^2: $k_{deficit}$ for PH based on ϵ

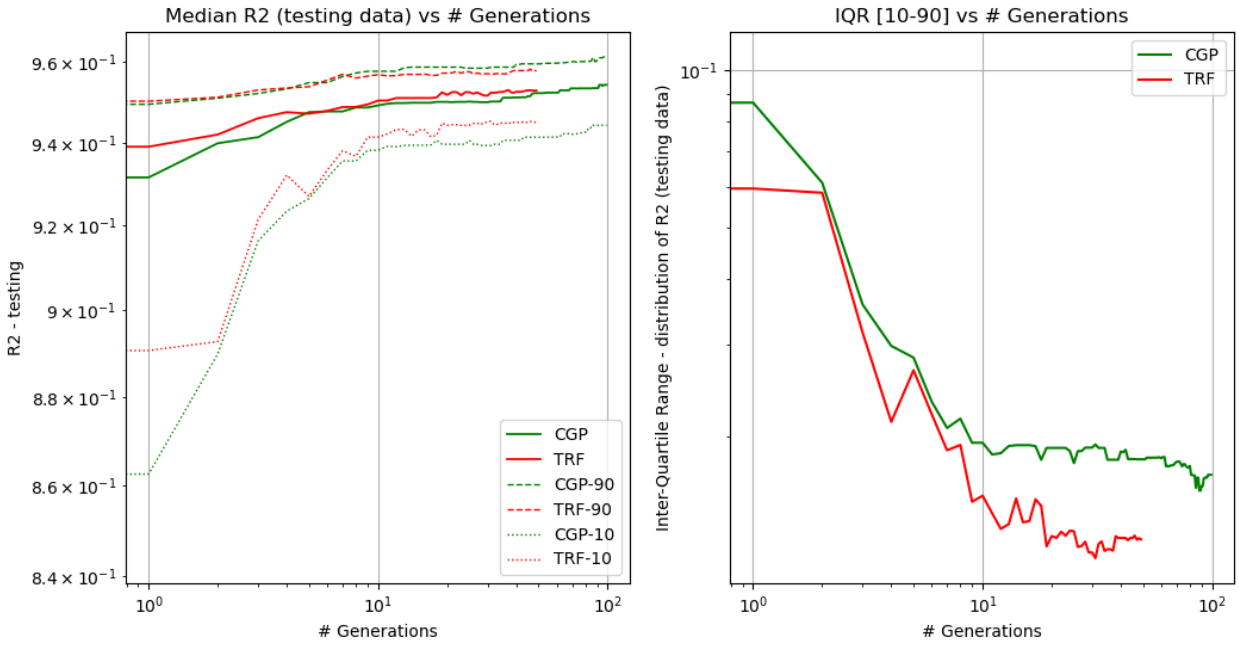


Figure C.5: Log Median Testing R^2: $k_{deficit}$ for PH based on ϵ

C.2. Genetic Programming with gradient descent and combined optimization - ϵ

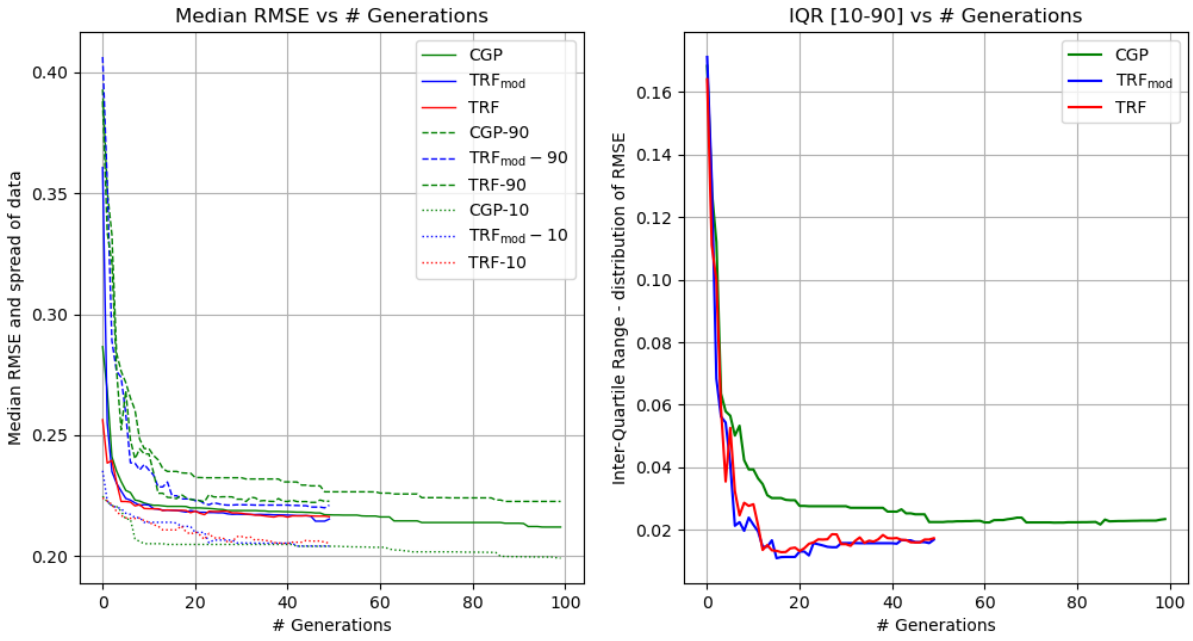


Figure C.6: Median RMSE: $k_{deficit}$ for PH based on ϵ - combined optimization

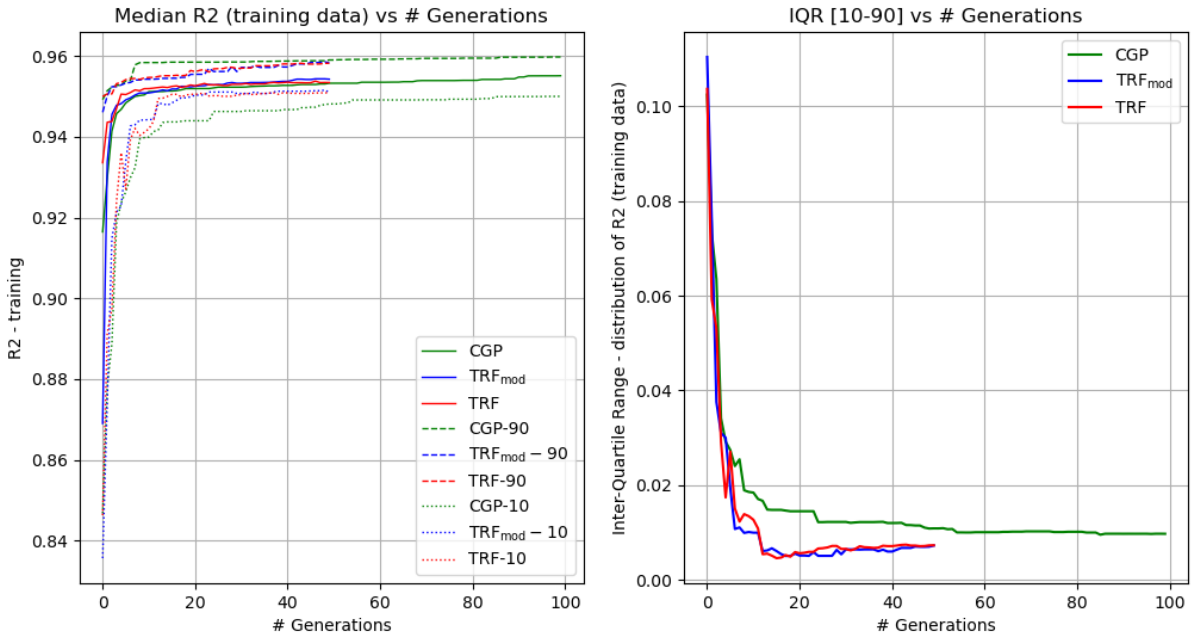


Figure C.7: Median training R^2: $k_{deficit}$ for PH based on ϵ - combined optimization

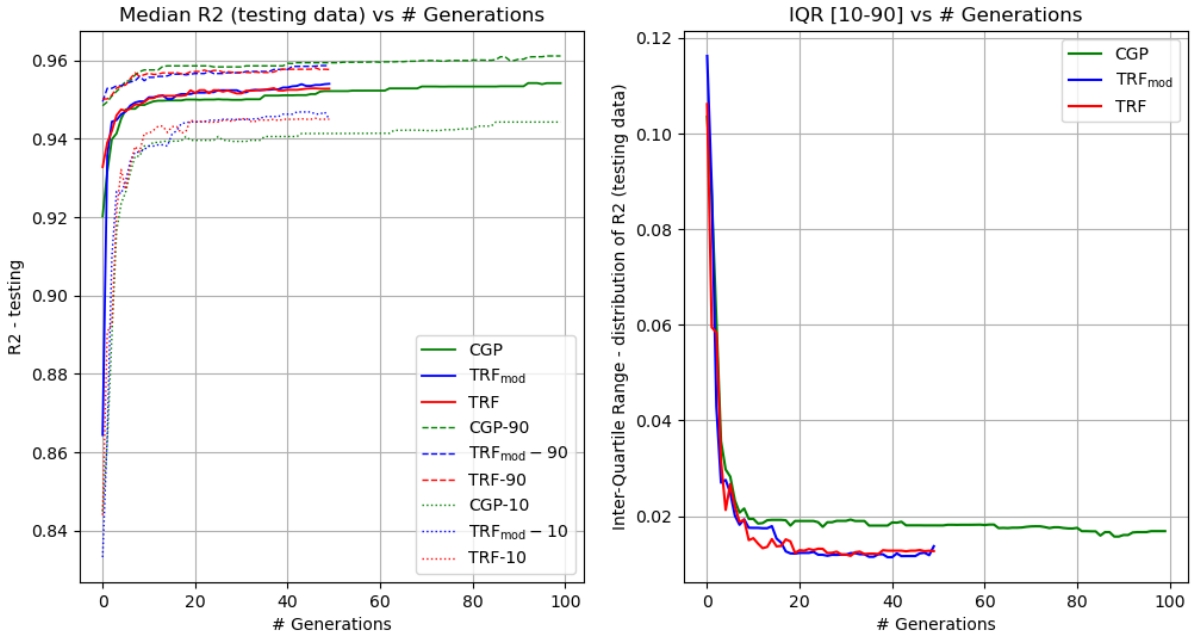


Figure C.8: Median Testing R^2: $k_{deficit}$ for PH based on ϵ - combined optimization

It is clear from the above median, percentile, and IQR scores that combining operator and terminal optimization does not downgrade the performance of the algorithm. It is able to achieve a nearly identical behavior with the previously implemented optimizer. To further gain insights into the rate and order of convergence, the plots are also made on the logarithmic scale, and presented below. Note that given the similarity in the performances of these algorithms over the metrics, as seen from above, only a few

plots are made. This is done to avoid analyzing any repetitive patterns, and be able to come to firm conclusions efficiently.

C.2.1. Analysis of quantities with respect to function calls

To observe how quantities change as the objective function is evaluated repeatedly, the following plots are presented, and a brief analysis made, before concluding this section.

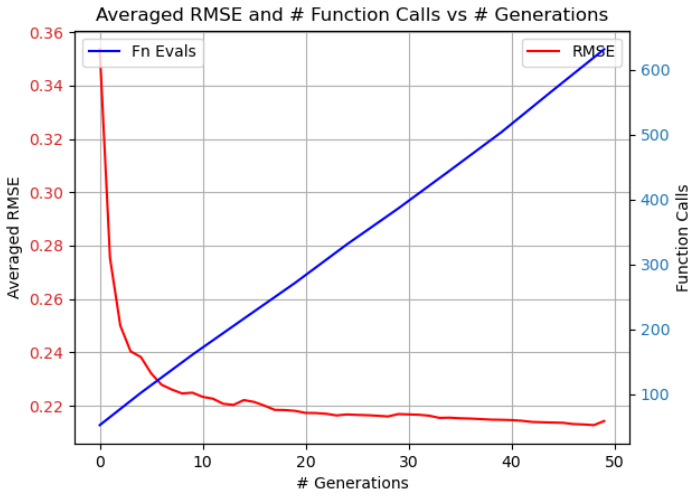


Figure C.9: RMSE and Function Calls vs Generations: $k_{deficit}$ for PH based on ϵ - combined optimization

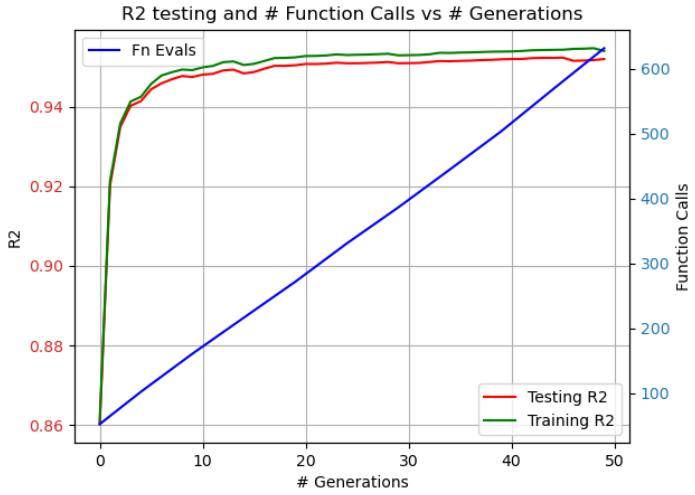


Figure C.10: R^2 and Function Calls vs Generations: $k_{deficit}$ for PH based on ϵ - combined optimization

The above figures give useful insights into how early the algorithm can achieve convergence with the function calls. An interesting observation is the near-linear line for the function calls. The optimization cycles occurred 10 times over the course of 50 generations, with each cycle requiring nearly 60 function calls to optimize the selected individuals. It can be observed that with just $\approx$ 300-400 function calls made by the optimizer, the algorithm is able to achieve convergence, which occurs quite before the 50 generations mark. Hence, to potentially make it more efficient, the function calls could be limited, or the algorithm made to run for fewer than 50 generations. Both the alternatives could further have a positive impact on the overall performance of the algorithms. However, as far as the current study is concerned, and the scope of the existing research, these alternatives will not be studied or implemented.

C.3. KANs for PH - ϵ

The symbolic formula is

$$
\begin{aligned}
f(x) = {} & 0.0051\,x_1 - 0.8563\,\sin\Big(1.3341\,x_3 - 0.049\,\cos(2.3102\,x_1 - 6.773) \\
& + 1.6238\,\cos(0.5261\,x_2 + 9.7931) + 0.5231\,\cos(3.1081\,x_4 - 5.2153) \\
& - 19.743\,\cos(0.1714\,x_5 + 9.236) - 17.1287\Big) \\
& + 0.1271\,\cos(0.6517\,x_1 - 2.3859) - 0.0564\,\cos(0.7934\,x_5 - 8.9887) \\
& - 0.3153\,\cos(0.5787\,x_1 - 0.9008\,x_3 + 0.1994\,x_5 + 8.8271) \\
& + 0.1237\,\cos(0.9566\,x_1 + 1.9999\,x_3 + 1.4906\,x_5 - 4.6646) + 0.6823
\end{aligned}
$$

Note that in the equation, x_1, x_2, x_3, x_4, x_5 refer to the pruned input space defined by $[q_Q, q_T, q^{(B)}_{\tau/k}, q_\gamma, q_{Re}]$, respectively.

C.4. Mutual Information Values

The thresholds set for invariants and scalars for different CFD datasets is tabulated below.

Table C.1: Mutual Information - PH & SD cases

Target Variable	Invariants	Tensors
$k_{deficit}$	0.5	0.9
b^{Δ}_{ij}	0.6	1.0

C.5. Completed Function Set for SD

Table C.2: Complete Functional Space for Correction Terms - GP

Set Type	Variables	Target Application
Non-dimensional Set	$\mathrm{S2_s}$, $\mathrm{W2_s}$, $\mathrm{S3_s}$, $\mathrm{W2S_s}$, $\mathrm{W2S2_s}$ $\mathrm{W2SWS2_s}$ Ak2, $\mathrm{Ak2S_s}$, $\mathrm{Ak2S2_s}$, $\mathrm{Ak2SAkS2_s}$, $\mathrm{WAk_s}$, $\mathrm{WAkS_s}$, $\mathrm{WAkS2_s}$, $\mathrm{W2AkS_s}$, $\mathrm{Ak2WS_s}$, $\mathrm{W2AkS2_s}$, $\mathrm{W2SAkS2_s}$, $\mathrm{Ak2WS2_s}$, $\mathrm{Ak2SWS2_s}$ q_Q, q_T, $q_{\mathrm{CP\_B}}$, $q_{\mathrm{tauk\_B}}$, q_γ, q_{Re}	Both Corrections
Dimensional Set	$\mathrm{G_{1s}}$, $\mathrm{G_{2s}}$, $\mathrm{G_{3s}}$, $\mathrm{G_{4s}}$, $\mathrm{G_{5s}}$, $\mathrm{G_{6s}}$, $\mathrm{G_{7s}}$, $\mathrm{G_{8s}}$, $\mathrm{G_{9s}}$, $\mathrm{G_{10s}}$, ϵ	$\mathrm{k_{deficit}}$
	$\mathrm{T_{1s}}$, $\mathrm{T_{2s}}$, $\mathrm{T_{3s}}$, $\mathrm{T_{4s}}$, $\mathrm{T_{5s}}$, $\mathrm{T_{6s}}$, $\mathrm{T_{7s}}$, $\mathrm{T_{8s}}$, $\mathrm{T_{9s}}$, $\mathrm{T_{10s}}$	$\mathrm{b^{\Delta}_{ij}}$

C.6. Mathematical Representation of Individuals and Overall Correlations

C.6.1. $k_{deficit}$ PH

Conventional Genetic Programming

$$\text{Tree 1:} \quad q_{\text{Re}} \times q_\gamma \tag{C.1}$$

$$\text{Tree 2:} \quad (0.7795 + \text{W2\_s}) - [(\text{W2\_s} \times \text{W2\_s}) \times (\text{W2\_s} \times 0.7795)] \tag{C.2}$$

$$\text{Tree 3:} \quad 1.6707 \times q_T \tag{C.3}$$

Genetic Programming with gradient descent

$$\text{Tree 1:} \quad \mathbf{0.6764} \cdot \Big[\mathbf{-1.2834} \cdot \Big[\mathbf{1.214} \cdot (1.5874 - 1.3365) - 1.0781\Big] - \mathbf{0.518} \cdot (1.4376 - 0.9952)\Big] \tag{C.4}$$

$$\text{Tree 2:} \quad \mathbf{0.7888} \cdot [4.9509 - 0.8309] \tag{C.5}$$

$$\text{Tree 3:} \quad \mathbf{1.4873} \cdot \Big[\mathbf{0.1485} \cdot \Big[\mathbf{0.0563} \cdot (6.2873 - 1.7301) \times (-1.1039)\Big] - q_\gamma\Big] \tag{C.6}$$

KANs

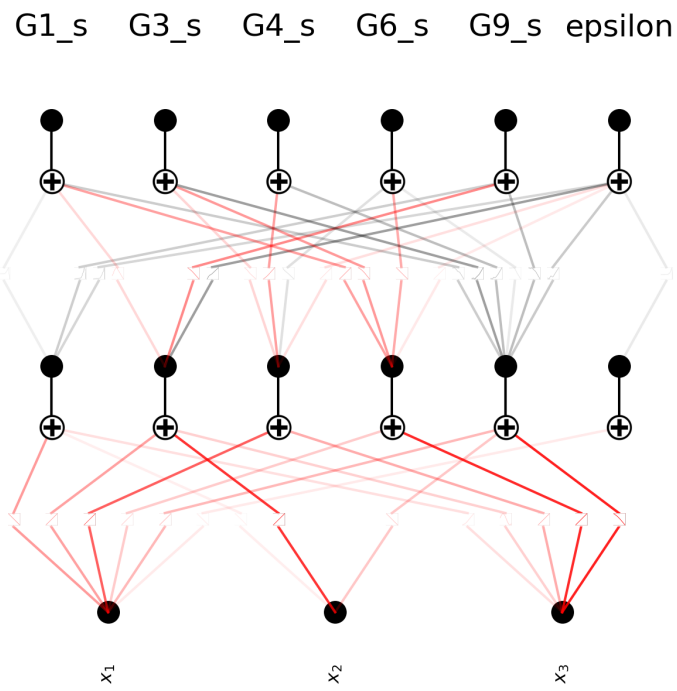


Figure C.11: Symbolically Active converged model

$$\begin{aligned}\text{Output 1:}\quad & 0.126q_{\text{Re}} + 0.0050q_T - 0.0882\cos\left(0.0644q_T + 2.1855q_{\text{tauk}_B}\right.\\ & -0.1882\sin(2.5281q_{\text{Re}} + 3.4246) - 2.0347)\\ & - 0.673\tanh\left(0.0143q_T - 0.1337q_{\text{tauk}_B}\right.\\ & +0.6163\cos(0.5922q_{\text{Re}} - 5.7974) - 1.3128)\\ & + 0.515\tanh\left(0.1226q_T + 1.1137\sin(0.2507q_{\text{tauk}_B} - 0.9939)\right.\\ & +0.1935\sin(1.0714q_{\text{Re}} + 1.7538) + 1.9154) - 1.19 \end{aligned} \tag{C.7}$$

$$\begin{aligned}\text{Output 2:}\quad & 0.173\cos(0.0270q_T + 0.0519q_{\text{Re}} + 6.7761)\\ & - 0.219\cos(0.0125q_T + 0.3150q_{\text{Re}} + 9.7753)\\ & + 0.498\tanh\left(0.0285q_T - 0.2672q_{\text{tauk}_B}\right.\\ & +1.2315\cos(0.5922q_{\text{Re}} - 5.7974) - 2.5920) + 0.0851 \end{aligned} \tag{C.8}$$

$$\begin{aligned}\text{Output 3:}\quad & - 1.6\cos(0.0130q_T + 0.0250q_{\text{Re}} - 5.6969)\\ & + 4.1\tanh\left(0.0065q_T - 0.0605q_{\text{tauk}_B}\right.\\ & +0.2787\cos(0.5922q_{\text{Re}} - 5.7974) - 2.0270) + 5.24 \end{aligned} \tag{C.9}$$

$$\begin{aligned}\text{Output 4:}\quad & - 0.0049q_T - 0.122q_{\text{Re}}\\ & - 1.46\tanh(0.0131q_T + 0.0252q_{\text{Re}} + 0.6081)\\ & - 0.538\tanh\left(0.0439q_T - 0.4112q_{\text{tauk}_B}\right.\\ & +1.8949\cos(0.5922q_{\text{Re}} - 5.7974) - 4.0724) + 0.595 \end{aligned} \tag{C.10}$$

$$\begin{aligned}\text{Output 5:}\quad & 2.9\cos\left(0.0043q_T + 0.1464q_{\text{tauk}_B}\right.\\ & -0.0126\sin(2.5281q_{\text{Re}} + 3.4246) + 0.1658)\\ & - 0.508\tanh\left(0.0295q_T - 0.2768q_{\text{tauk}_B}\right.\\ & +1.2756\cos(0.5922q_{\text{Re}} - 5.7974) - 2.7654)\\ & + 0.0382\tanh\left(0.5136q_T + 4.6639\sin(0.2507q_{\text{tauk}_B} - 0.9939)\right.\\ & +0.8105\sin(1.0714q_{\text{Re}} + 1.7538) + 1.8057) - 3.24 \end{aligned} \tag{C.11}$$

$$\begin{aligned}\text{Output 6:}\quad & 0.0887q_{\text{Re}} + 0.02q_T + 0.117\tanh(1.0060q_T - 1.8718)\\ & + 0.778\tanh\left(0.0527q_T + 1.7886q_{\text{tauk}_B}\right.\\ & -0.1540\sin(2.5281q_{\text{Re}} + 3.4246) - 2.5846)\\ & + 0.107\tanh\left(0.1893q_T - 1.7746q_{\text{tauk}_B}\right.\\ & +8.1783\cos(0.5922q_{\text{Re}} - 5.7974) - 1.6613)\\ & + 0.151\tanh\left(1.0205q_T + 9.2678\sin(0.2507q_{\text{tauk}_B} - 0.9939)\right.\\ & +1.6106\sin(1.0714q_{\text{Re}} + 1.7538) + 5.1327) + 0.663 \end{aligned} \tag{C.12}$$

C.6.2. b_{ij}^{Δ} PH

Conventional Genetic Programming

$$\text{Tree 1:}\quad [\text{Ak2} \times \text{Ak2}] \tag{C.13}$$

$$\text{Tree 2:}\quad [q_{\tau k_B} \times q_{\gamma}] \tag{C.14}$$

$$\text{Tree 3:}\quad [\text{W2S\_s} \times \text{W2S\_s}] \tag{C.15}$$

$$\text{Tree 4:}\quad [\text{Ak2S2\_s} \times \text{Ak2S2\_s}] \tag{C.16}$$

$$\text{Tree 5:}\quad [-1.4313 + 0.7795] \tag{C.17}$$

$$\text{Tree 6:}\quad [q_{\gamma} \times q_{\gamma} + \text{S2\_s} \times \text{S2\_s} \times (-1.4313) \times q_{\gamma}] \tag{C.18}$$

Genetic Programming with gradient descent

Tree 1: **0.8929** · [Ak2WS2_s $\times$ 1.4885] (C.19)

Tree 2: **0.8045** · [0.679 $\times$ 0.58] (C.20)

Tree 3: **0.9444** · [Ak2WS2_s $-$ **0.9205** · (**1.3573** · (Ak2WS2_s $\times$ 4.7565) $\times$ **0.8756** · ($-$1.353 $\times$ 4.0546))] (C.21)

Tree 4: **0.8801** · [Ak2S2_s $\times$ (**0.7958** · (5.3408 $\times$ 4.5559))] (C.22)

Tree 5: $-$ 1.3945 (C.23)

Tree 6: **1.2121** · [**1.1161** · (1.5942 + 1.1183) $\times$ (**-0.29** · (**0.9239** · (**-0.1326** · (1.4409 + S3_s) + q_γ) + q_γ))] (C.24)

KANs

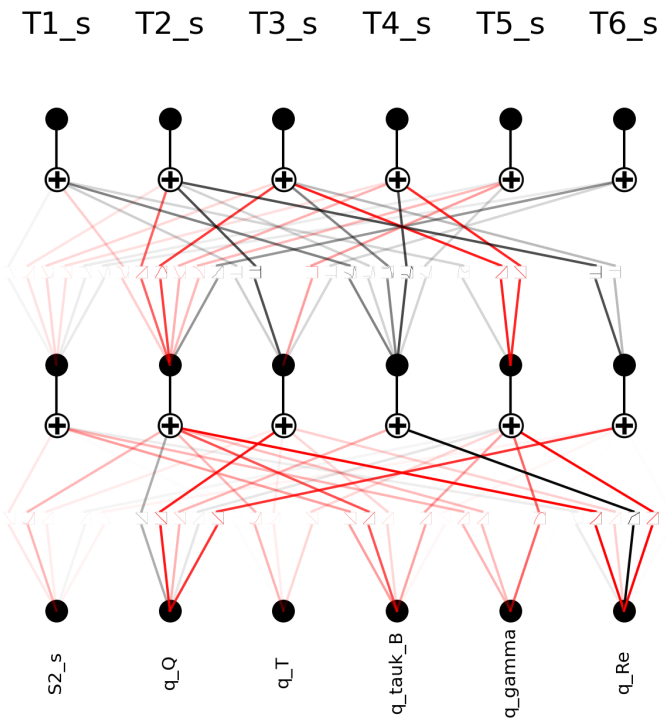


Figure C.12: Symbolically Active converged model

Out 1: $$\begin{aligned}&-0.216\sin(0.686q_{\text{tauk}_B}+0.525\sin(1.27q_T+4.37)+0.421\sin(1.37q_\gamma-6.79)\\&+3.81\cos(0.407S_{2s}+3.05)-174\cos(0.043q_{\text{Re}}-6.20)+935\tanh(0.004q_Q-2.8)+1112)\\&-0.119\tanh(0.104q_Q-52.1\tanh(0.576q_{\text{Re}}+1.2)+44.1)\\&-0.043\tanh(1.80S_{2s}-2.30q_Q+3.94q_{\text{tauk}_B}+1.67q_{\text{Re}}+1.50\cos(1.92q_T-3.35)-10.7)\\&+0.002\tanh(-4.31q_{\text{tauk}_B}+2.61\sin(3.11q_\gamma+6.02)+0.531\cos(2.66S_{2s}+6.96)\\&+0.033\tanh(9.0q_T-8.36)-0.213\tanh(5.0q_{\text{Re}}-2.85)+11.8)\\&-0.032\tanh(-1.12q_T+1.85q_{\text{tauk}_B}+5.23q_{\text{Re}}-3.98\sin(4.02q_\gamma-9.02)\\&+0.109\tanh(2.47S_{2s}-1.03)+38.0\tanh(0.012q_Q-3.0)+29.2)+0.241\end{aligned}\tag{C.25}$$

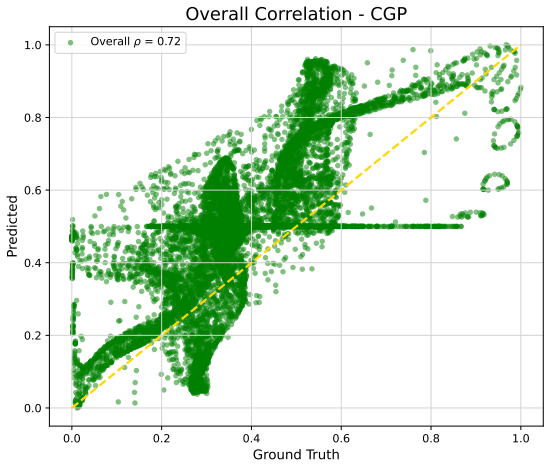
Out 2: $$\begin{aligned}&0.505q_{\text{tauk}_B}+0.387\sin(1.27q_T+4.37)+0.31\sin(1.37q_\gamma-6.79)\\&+3.44\sin(0.24q_{\text{tauk}_B}-0.145\sin(3.11q_\gamma+6.02)-0.030\cos(2.66S_{2s}+6.96)\\&-0.002\tanh(9.0q_T-8.36)+0.012\tanh(5.0q_{\text{Re}}-2.85)+4.05)\\&+2.81\cos(0.407S_{2s}+3.05)-128\cos(0.043q_{\text{Re}}-6.20)+688\tanh(0.004q_Q-2.8)\\&+0.361\tanh(0.068q_Q-34.0\tanh(0.576q_{\text{Re}}+1.2)+33.1)\\&-1.04\tanh(0.478S_{2s}-0.609q_Q+1.04q_{\text{tauk}_B}+0.444q_{\text{Re}}+0.397\cos(1.92q_T-3.35)-0.929)\\&-0.861\tanh(1.71q_Q+0.896q_{\text{tauk}_B}-1.13\sin(1.89S_{2s}+0.978)-0.349\sin(0.849q_\gamma-7.17)\\&+0.059\tanh(5.61q_{\text{Re}}-9.25)-2.05)+816\end{aligned}\tag{C.26}$$

Out 3: $$\begin{aligned}&-0.089\sin(2.12q_{\text{tauk}_B}-1.28\sin(3.11q_\gamma+6.02)-0.261\cos(2.66S_{2s}+6.96)\\&-0.016\tanh(9.0q_T-8.36)+0.105\tanh(5.0q_{\text{Re}}-2.85)+4.61)\\&+2.15\cos(-0.057q_T+0.095q_{\text{tauk}_B}+0.268q_{\text{Re}}-0.204\sin(4.02q_\gamma-9.02)\\&+0.006\tanh(2.47S_{2s}-1.03)+1.95\tanh(0.012q_Q-3.0)+5.85)\\&-0.981\cos(0.618q_{\text{tauk}_B}+0.473\sin(1.27q_T+4.37)+0.379\sin(1.37q_\gamma-6.79)\\&+3.44\cos(0.407S_{2s}+3.05)-157\cos(0.043q_{\text{Re}}-6.20)+842\tanh(0.004q_Q-2.8)+992)\\&+0.133\tanh(0.106q_Q-53.2\tanh(0.576q_{\text{Re}}+1.2)+45.0)-0.013\end{aligned}\tag{C.27}$$

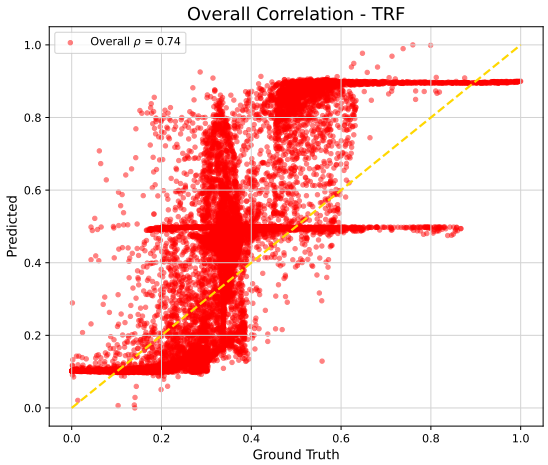
Out 4: $$\begin{aligned}&0.119q_T-0.197q_{\text{tauk}_B}-0.556q_{\text{Re}}+0.423\sin(4.02q_\gamma-9.02)\\&-0.535\sin(0.438q_{\text{tauk}_B}+0.335\sin(1.27q_T+4.37)+0.268\sin(1.37q_\gamma-6.79)\\&+2.43\cos(0.407S_{2s}+3.05)-111\cos(0.043q_{\text{Re}}-6.20)+596\tanh(0.004q_Q-2.8)+706)\\&+0.63\cos(-0.442q_{\text{tauk}_B}+0.267\sin(3.11q_\gamma+6.02)+0.054\cos(2.66S_{2s}+6.96)\\&+0.003\tanh(9.0q_T-8.36)-0.022\tanh(5.0q_{\text{Re}}-2.85)+7.47)\\&-0.012\tanh(2.47S_{2s}-1.03)-4.04\tanh(0.012q_Q-3.0)\\&-0.263\tanh(0.080q_Q-40.1\tanh(0.576q_{\text{Re}}+1.2)+34.3)-3.38\end{aligned}\tag{C.28}$$

Out 5: $$\begin{aligned}&0.571\cos(0.151q_{\text{tauk}_B}+0.116\sin(1.27q_T+4.37)+0.093\sin(1.37q_\gamma-6.79)\\&+0.838\cos(0.407S_{2s}+3.05)-38.3\cos(0.043q_{\text{Re}}-6.20)+205\tanh(0.004q_Q-2.8)+243)\\&-0.075\tanh(0.021q_Q-10.4\tanh(0.576q_{\text{Re}}+1.2)+9.98)\\&-0.22\tanh(0.237S_{2s}-0.302q_Q+0.518q_{\text{tauk}_B}+0.220q_{\text{Re}}+0.197\cos(1.92q_T-3.35)-1.20)\\&+0.295\tanh(0.690q_{\text{tauk}_B}-0.418\sin(3.11q_\gamma+6.02)-0.085\cos(2.66S_{2s}+6.96)\\&-0.005\tanh(9.0q_T-8.36)+0.034\tanh(5.0q_{\text{Re}}-2.85)+0.589)-0.903\end{aligned}\tag{C.29}$$

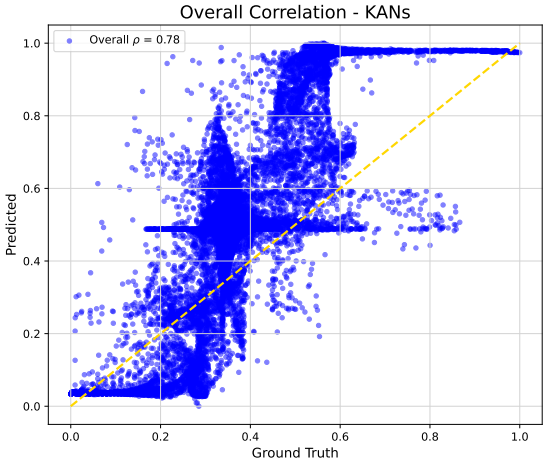
Out 6: $$\begin{aligned}&0.012\tanh(1.74S_{2s}-2.21q_Q+3.80q_{\text{tauk}_B}+1.61q_{\text{Re}}+1.44\cos(1.92q_T-3.35)-10.7)\\&-0.006\tanh(-3.09q_{\text{tauk}_B}+1.87\sin(3.11q_\gamma+6.02)+0.381\cos(2.66S_{2s}+6.96)\\&+0.023\tanh(9.0q_T-8.36)-0.153\tanh(5.0q_{\text{Re}}-2.85)+7.70)\\&+0.034\tanh(1.57q_{\text{tauk}_B}+1.21\sin(1.27q_T+4.37)+0.966\sin(1.37q_\gamma-6.79)\\&+8.75\cos(0.407S_{2s}+3.05)-400\cos(0.043q_{\text{Re}}-6.20)+2145\tanh(0.004q_Q-2.8)+2534)+0.27\end{aligned}\tag{C.30}$$



(a) CGP Scatter Plot



(b) TRF Scatter Plot



(c) KANs Scatter Plot

Figure C.13: Comparison of CGP, TRF, and KANs Scatter Plots - overall b_{ij}^{Δ} PH

C.6.3. $k_{deficit}$ SD

Conventional Genetic Programming

$$\text{Tree 1:} \quad \text{Ak2} \times 1.6707 \tag{C.31}$$

$$\text{Tree 2:} \quad \text{Ak2WS2\_s} + [\text{Ak2WS2\_s} + (\text{WAk\_s} - (\text{W2\_s} \times \text{W2S2\_s}))] \tag{C.32}$$

$$\text{Tree 3:} \quad \Big[\Big[[[(q_{\text{Re}} - \text{WAk\_s}) - ((\text{S2\_s} + 1.6707) \times \text{WAk\_s})] - \text{Ak2}\Big] - \text{Ak2}\Big] - (q_Q - \text{W2S\_s})\Big] - (q_T \times q_Q)\Big] \times \text{S2\_s} \tag{C.33}$$

Genetic Programming with gradient descent

$$\text{Tree 1:}\quad \mathbf{0.872}\cdot\Big[\mathbf{0.9076}\cdot(0.7048+0.7708) - \mathbf{0.9775}\cdot\Big[\mathbf{0.8197}\cdot(0.415+1.0156)+0.0482\Big]\Big] \tag{C.34}$$

$$\text{Tree 2:}\quad \mathbf{0.7546}\cdot\Big[\mathbf{0.8822}\cdot\Big[\mathbf{-2.6908}\cdot\big[0.828+\mathbf{0.9686}\cdot(\text{Ak2S2\_s}\times 0.7455)\big]\times 0.7278\Big]\times \text{Ak2S2\_s}\Big] \tag{C.35}$$

$$\text{Tree 3:}\quad \mathbf{0.9883}\cdot\Big[0.7669\times\mathbf{1.0481}\cdot\big[\mathbf{1.0469}\cdot(0.9549\times 0.7286)+0.7286\big]\Big] \tag{C.36}$$

KANs

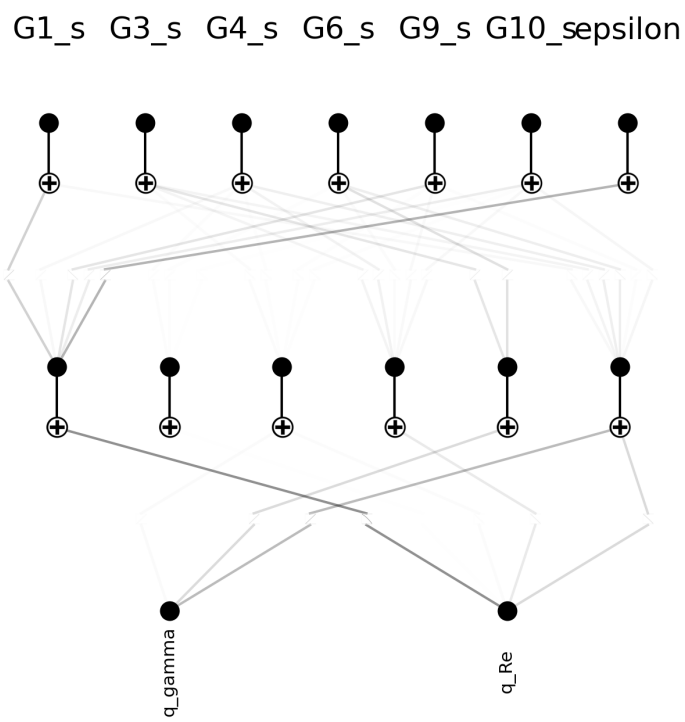


Figure C.14: Symbolically Active converged model

Output 1: $$\begin{aligned} &-0.0404\tanh\left(0.1217\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0643\right)\\ &-0.0118\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &+0.0508 \end{aligned} \tag{C.37}$$

Output 2: $$\begin{aligned} &-0.0127\tanh\left(0.0586\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0463\right)\\ &-0.0233\tanh\left(0.0362\tanh(10.0q_{\gamma}-5.0)-0.0263\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0246\right)\\ &+0.015\tanh\left(0.1095\tanh(10.0q_{\gamma}-5.0)-0.00121\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0530\right)\\ &+0.0117\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &+0.00233 \end{aligned} \tag{C.38}$$

Output 3: $$\begin{aligned} &0.0172\tanh\left(0.00889\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0815\right)\\ &-0.0118\tanh\left(0.0586\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0463\right)\\ &-0.00329\tanh\left(0.1217\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0643\right)\\ &-0.0118\tanh\left(0.0362\tanh(10.0q_{\gamma}-5.0)-0.0263\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0246\right)\\ &+0.0142\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &+0.0483 \end{aligned} \tag{C.39}$$

Output 4: $$\begin{aligned} &0.018\tanh\left(0.00889\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0815\right)\\ &+0.0047\tanh\left(0.0586\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0463\right)\\ &+0.018\tanh\left(0.1095\tanh(10.0q_{\gamma}-5.0)-0.00121\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0530\right)\\ &-0.0185\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &+0.0992 \end{aligned} \tag{C.40}$$

Output 5: $$\begin{aligned} &-0.0068\tanh\left(0.0586\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0463\right)\\ &+0.014\tanh\left(0.1217\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0643\right)\\ &-0.0139\tanh\left(0.0362\tanh(10.0q_{\gamma}-5.0)-0.0263\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0246\right)\\ &+0.00282\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &-0.0454 \end{aligned} \tag{C.41}$$

Output 6: $$\begin{aligned} &-0.0126\tanh\left(0.00889\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0815\right)\\ &-0.0103\tanh\left(0.0586\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0463\right)\\ &-0.0122\tanh\left(0.1217\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0643\right)\\ &-0.0199\tanh\left(0.0362\tanh(10.0q_{\gamma}-5.0)-0.0263\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0246\right)\\ &-0.0127\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &-0.0126 \end{aligned} \tag{C.42}$$

Output 7: $$\begin{aligned} &-0.118\tanh\left(0.1217\tanh(10.0q_{\mathsf{Re}}+5.0)-0.0643\right)\\ &-0.000347\tanh\left(0.1306\tanh(10.0q_{\gamma}-5.0)-0.0703\tanh(10.0q_{\mathsf{Re}}+5.0)+0.0383\right)\\ &+0.989 \end{aligned} \tag{C.43}$$

C.6.4. $b_{ij}^{\triangle}$ SD

Conventional Genetic Programming

Tree 1: $$[Ak2SAkS2\_s + Ak2SAkS2\_s] \tag{C.44}$$

Tree 2: $$[0.7795 + W2SAkS2\_s] \tag{C.45}$$

Tree 3: $$Ak2SAkS2\_s \tag{C.46}$$

Tree 4: $$-1.4313 \tag{C.47}$$

Tree 5: $$4.9403 \times W2\_s \tag{C.48}$$

Tree 6: $$\begin{aligned} &\Big[\Big[\Big[\Big[\Big[W2AkS\_s - WAk\_s\Big] - WAk\_s\Big] - WAk\_s\Big] - WAk\_s\Big] - WAk\_s\Big] \times WAk\_s\\ &-(WAk\_s \times WAk\_s) \end{aligned} \tag{C.49}$$

Genetic Programming with gradient descent

Tree 1: $\mathbf{1.0918} \cdot [\mathbf{0.3265} \cdot (0.7744 + 1.6102) + W2AkS2\_s]$ (C.50)

Tree 2: $\mathbf{0.8144} \cdot [-0.0828 - 0.8234]$ (C.51)

Tree 3: $\mathbf{0.9193} \cdot [1.765 - 0.0855]$ (C.52)

Tree 4: $-\mathbf{3.934} \cdot [\mathbf{1.0418} \cdot (-0.0839 \times -0.094) + \mathbf{0.9975} \cdot (0.6061 - 0.8062)]$ (C.53)

Tree 5: $\mathbf{0.9772} \cdot (0.7676 \times -1.5092)$ (C.54)

Tree 6: $\mathbf{0.9707} \cdot [\mathbf{0.8787} \cdot (-1.4786 - 0.4529) - Ak2WS2\_s]$ (C.55)

KANs

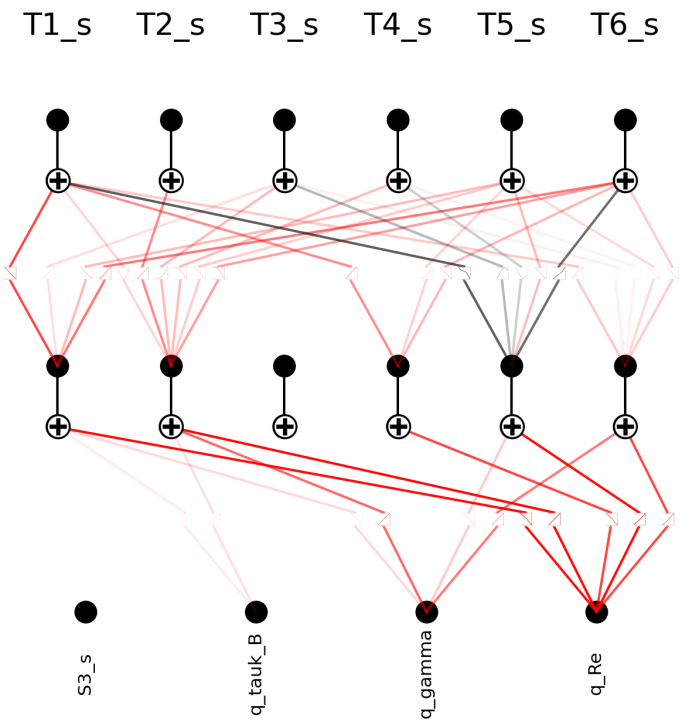


Figure C.15: Symbolically Active converged model

Output 1:
$$\begin{aligned}
&-0.0828\sin(2.126\cos(0.655q_{\mathsf{Re}}+0.392)-3.743)\\
&-0.285\sin(1.134q_{\gamma}+0.471\sin(0.730q_{\mathsf{Re}}+5.239)-2.274)\\
&-1.73\sin(0.331q_{\mathsf{tauk}_B}+0.289q_{\mathsf{Re}}-0.214\sin(2.441q_{\gamma}+9.187)-5.617)\\
&+0.183\cos(0.752q_{\mathsf{tauk}_B}+0.546\cos(1.543q_{\gamma}-5.979)\\
&+1.727\cos(0.701q_{\mathsf{Re}}+6.645)-1.028)\\
&-0.389\tanh(-0.152\sin(2.392q_{\gamma}-0.808)+37.91\sin(0.0597q_{\mathsf{Re}}-1.391)+35.79)\\
&+1.22
\end{aligned} \tag{C.56}$$

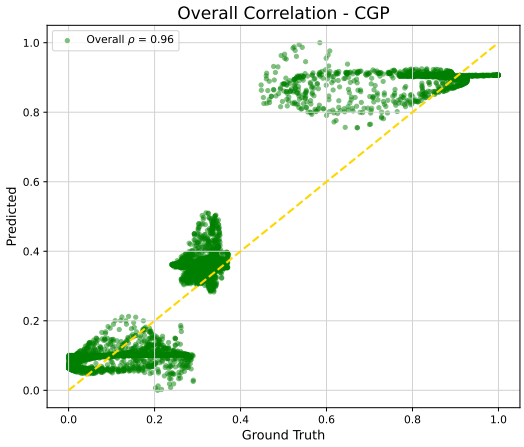
Output 2:
$$\begin{aligned}
&0.715\cos(0.566q_{\mathsf{tauk}_B}+0.494q_{\mathsf{Re}}-0.366\sin(2.441q_{\gamma}+9.187)+9.457)\\
&+0.594
\end{aligned} \tag{C.57}$$

Output 3:
$$\begin{aligned}
&0.0386q_{\mathsf{tauk}_B}+0.028\cos(1.543q_{\gamma}-5.979)+0.0886\cos(0.701q_{\mathsf{Re}}+6.645)\\
&+0.388\cos(0.917q_{\gamma}+0.381\sin(0.730q_{\mathsf{Re}}+5.239)-0.432)\\
&-0.348\cos(0.786q_{\mathsf{tauk}_B}+0.686q_{\mathsf{Re}}-0.509\sin(2.441q_{\gamma}+9.187)-1.614)\\
&+0.122\tanh(-0.850\sin(2.392q_{\gamma}-0.808)+211.5\sin(0.0597q_{\mathsf{Re}}-1.391)+206.7)\\
&+0.0856
\end{aligned} \tag{C.58}$$

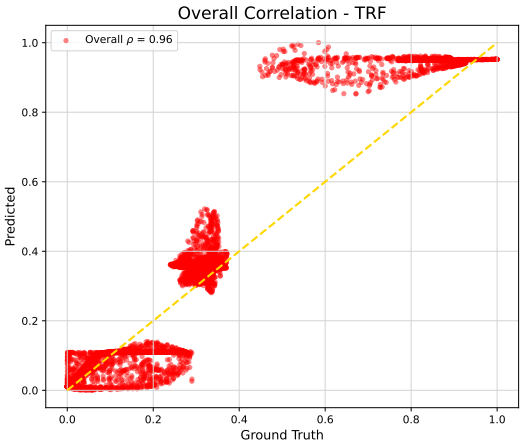
Output 4:
$$\begin{aligned}
&-1.39\sin(0.599q_{\gamma}+0.249\sin(0.730q_{\mathsf{Re}}+5.239)-4.950)\\
&+0.703\cos(0.576q_{\mathsf{tauk}_B}+0.503q_{\mathsf{Re}}-0.373\sin(2.441q_{\gamma}+9.187)-1.027)\\
&-0.111\tanh(-0.976\sin(2.392q_{\gamma}-0.808)+243.0\sin(0.0597q_{\mathsf{Re}}-1.391)+237.6)\\
&+0.337
\end{aligned} \tag{C.59}$$

Output 5:
$$\begin{aligned}
&-0.0323q_{\mathsf{tauk}_B}+0.0107\sin(2.392q_{\gamma}-0.808)-2.67\sin(0.0597q_{\mathsf{Re}}-1.391)\\
&-0.0235\cos(1.543q_{\gamma}-5.979)-0.0742\cos(0.701q_{\mathsf{Re}}+6.645)\\
&-1.12\cos(0.248\cos(0.655q_{\mathsf{Re}}+0.392)+0.0807)\\
&+0.0771\cos(1.524q_{\gamma}+0.634\sin(0.730q_{\mathsf{Re}}+5.239)+1.304)\\
&+0.159\cos(0.361q_{\mathsf{tauk}_B}+0.315q_{\mathsf{Re}}-0.234\sin(2.441q_{\gamma}+9.187)-0.177)\\
&-1.67
\end{aligned} \tag{C.60}$$

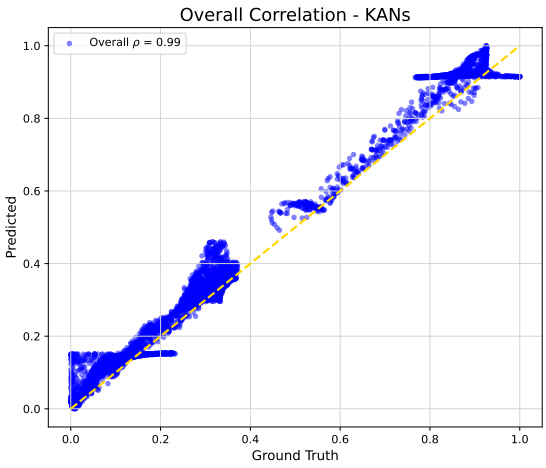
Output 6:
$$\begin{aligned}
&-0.439\sin(0.550q_{\gamma}+0.229\sin(0.730q_{\mathsf{Re}}+5.239)-5.426)\\
&-0.919\sin(0.326q_{\mathsf{tauk}_B}+0.284q_{\mathsf{Re}}-0.211\sin(2.441q_{\gamma}+9.187)-2.226)\\
&-0.107\sin(0.714q_{\mathsf{tauk}_B}+0.518\cos(1.543q_{\gamma}-5.979)\\
&+1.639\cos(0.701q_{\mathsf{Re}}+6.645)-11.88)\\
&-0.0784\cos(0.655q_{\mathsf{Re}}+0.392)\\
&+3.03\tanh(-0.0666\sin(2.392q_{\gamma}-0.808)+16.58\sin(0.0597q_{\mathsf{Re}}-1.391)+14.31)\\
&+2.36
\end{aligned} \tag{C.61}$$



(a) CGP Scatter Plot



(b) TRF Scatter Plot



(c) KANs Scatter Plot

Figure C.16: Comparison of CGP, TRF, and KANs Scatter Plots - Overall b_{ij}^{Δ} SD

D

A posteriori CFD additional Data

D.1. Initial *a posteriori*

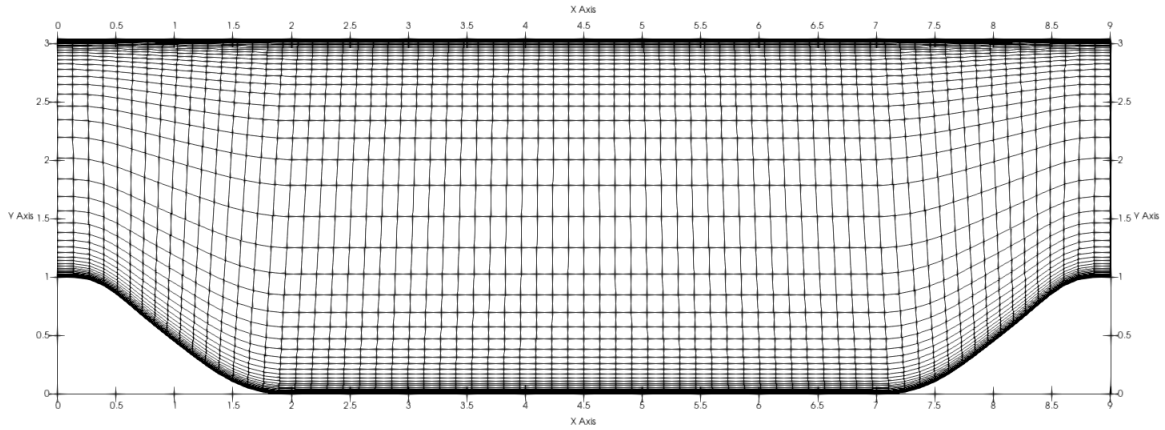


Figure D.1: 3500 mesh Periodic Hill - initial testing and training case

The results from the initial $a\ posteriori$ testing are presented below

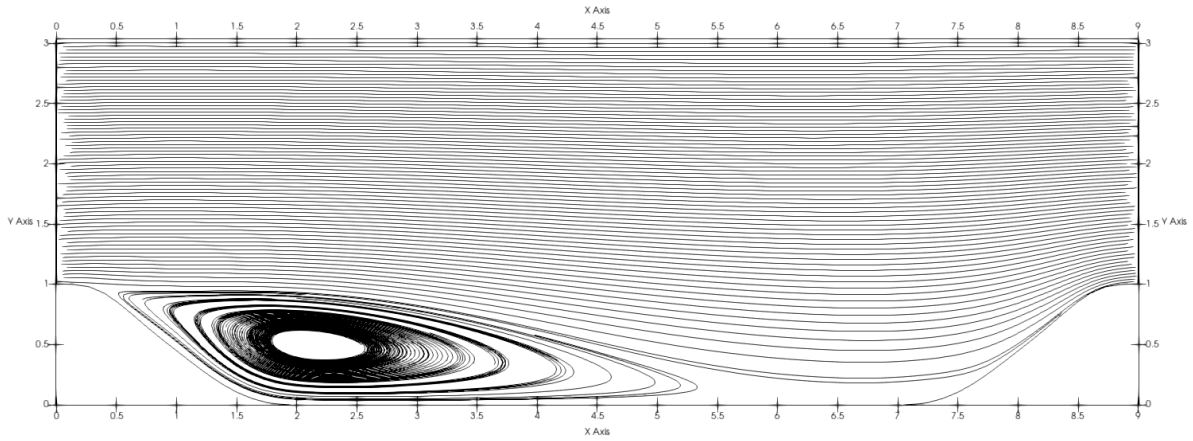


Figure D.2: $Baseline$ Spalart-Allmaras streamlines

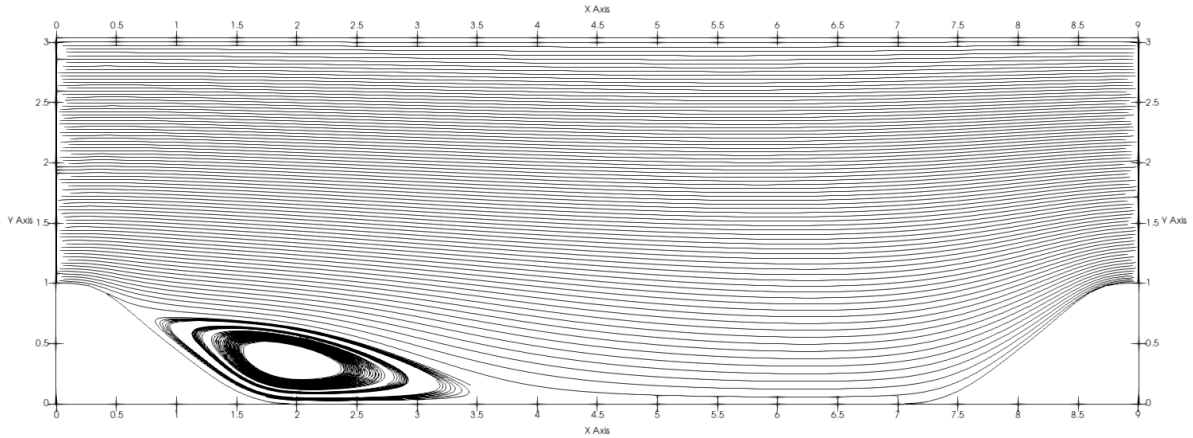


Figure D.3: $A\ posteriori$ Spalart Allmaras streamlines

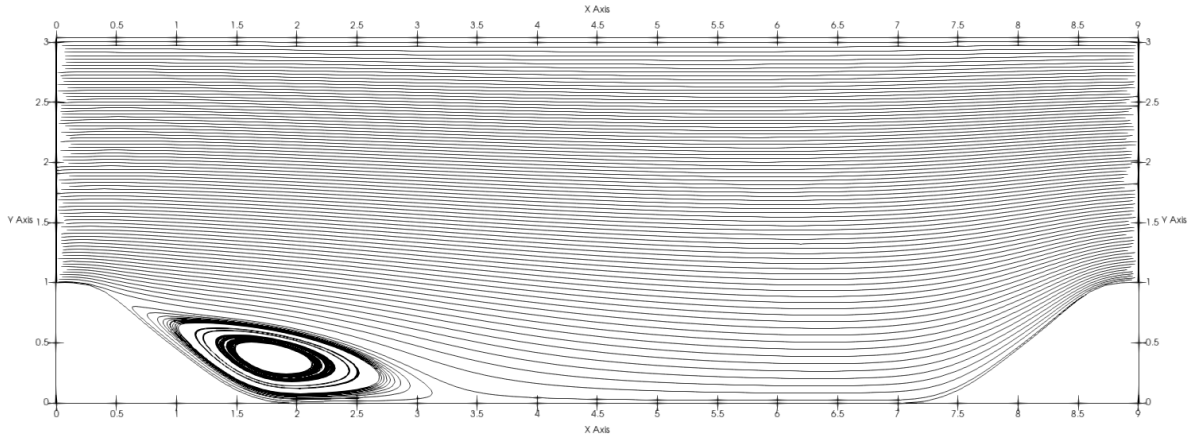


Figure D.4: $Reference\ k-\varepsilon$ velocity streamlines

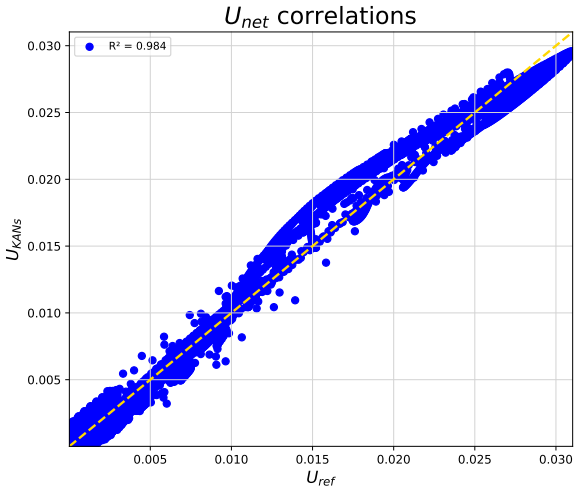


Figure D.5: Velocity Correlations

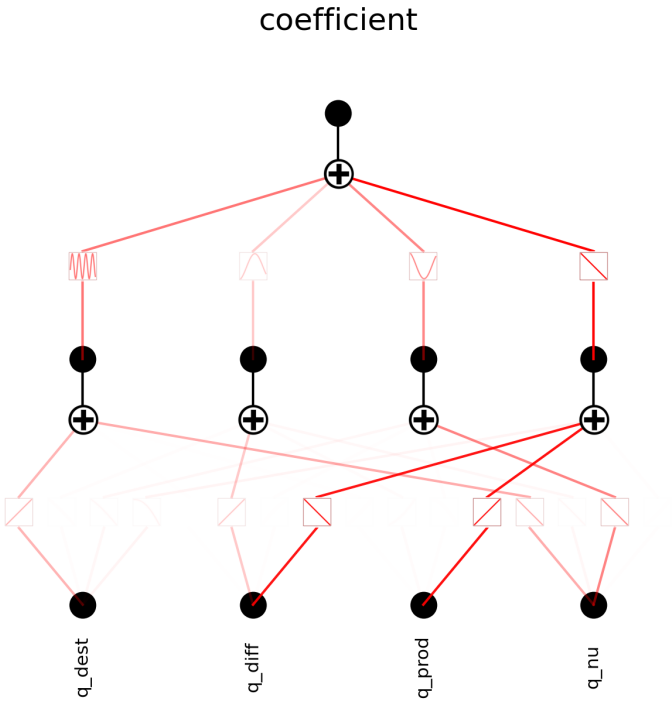


Figure D.6: Final KANs model

The obtained symbolic formula is

$$
\begin{aligned}
f(q_{\text{dest}}, q_{\text{diff}}, q_{\text{prod}}, q_\nu) = {} & 29.265 \cdot q_{\text{diff}} - 22.932 \cdot q_{\text{prod}} \\
& - 0.058 \sin(0.703 \cdot q_\nu - 1.051) \\
& - 1.551 \sin\Big(- 22.214 \cdot q_{\text{dest}} + 10.320 \cdot q_\nu \\
& \qquad + 0.367 \sin(1.654 \cdot q_{\text{diff}} - 4.591) \\
& \qquad + 0.296 \cos(1.614 \cdot q_{\text{prod}} + 6.990) + 0.233\Big) \\
& - 0.781 \sin\Big(10.095 \cdot q_{\text{diff}} + 0.536 \sin(0.777 \cdot q_{\text{dest}} + 2.206) \\
& \qquad + 2.224 \cos(0.704 \cdot q_{\text{prod}} - 9.251) \\
& \qquad + 12.573 \cos(0.069 \cdot q_\nu + 6.728) - 0.811\Big) \\
& + 1.710 \sin\Big(3.805 \cdot q_\nu + 0.367 \sin(0.860 \cdot q_{\text{diff}} + 3.996) \\
& \qquad + 0.154 \sin(1.958 \cdot q_{\text{prod}} - 7.829) \\
& \qquad + 0.340 \cos(1.724 \cdot q_{\text{dest}} - 8.776) + 3.043\Big) \\
& - 0.622 \cos(1.853 \cdot q_{\text{dest}} - 0.049) + 6.400
\end{aligned}
\tag{D.1}
$$

D.2. Final *a posteriori* - training and testing

D.2.1. Training Case

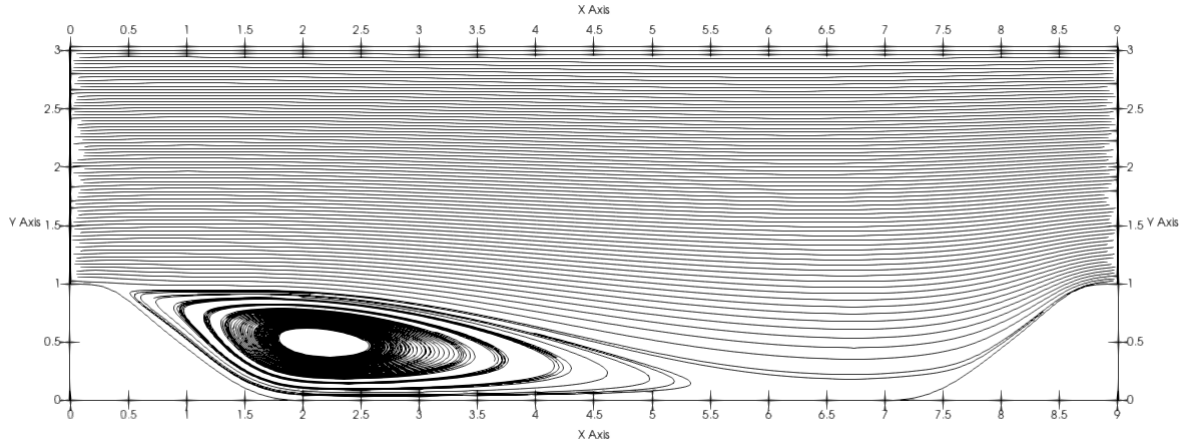


Figure D.7: $Baseline$ Spalart-Allmaras streamlines - Training with λ

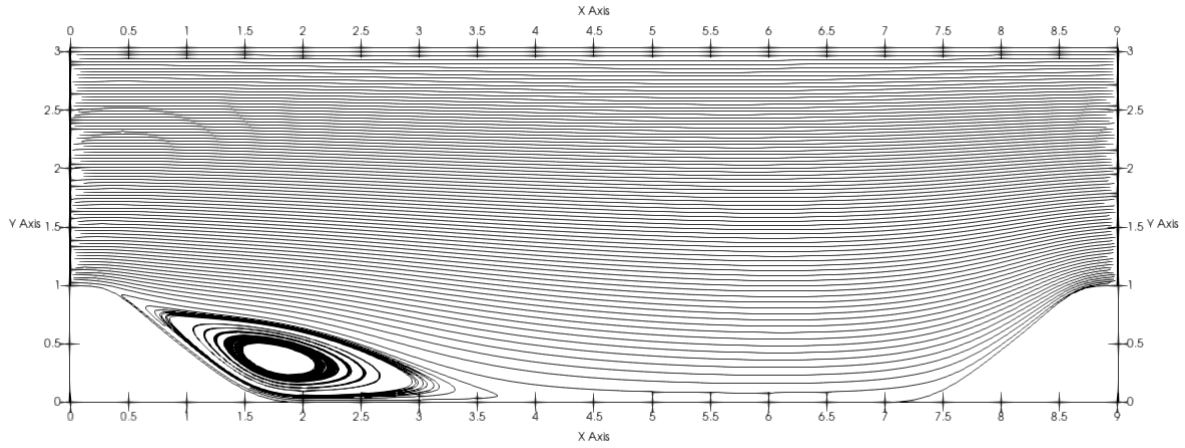


Figure D.8: $A\ posteriori$ Spalart Allmaras streamlines - Training with λ

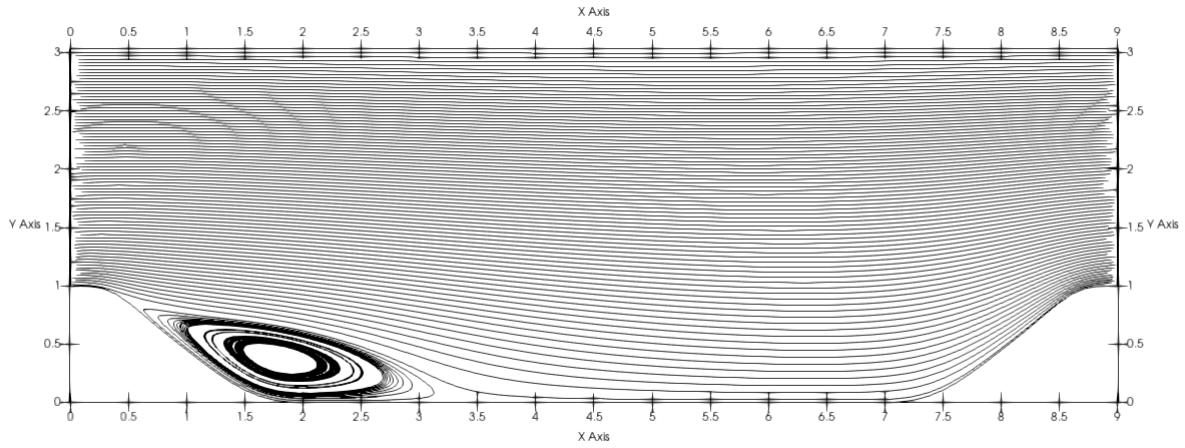


Figure D.9: $Reference\ k-\varepsilon$ velocity streamlines - Training with λ

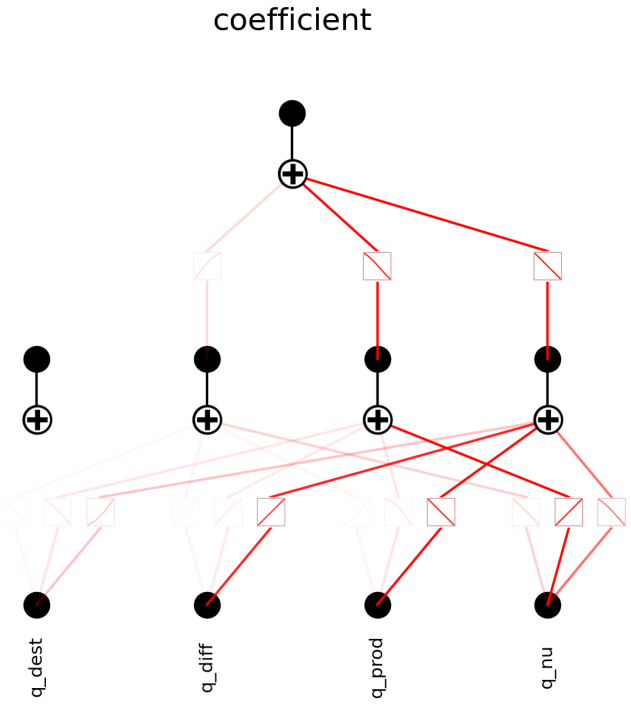


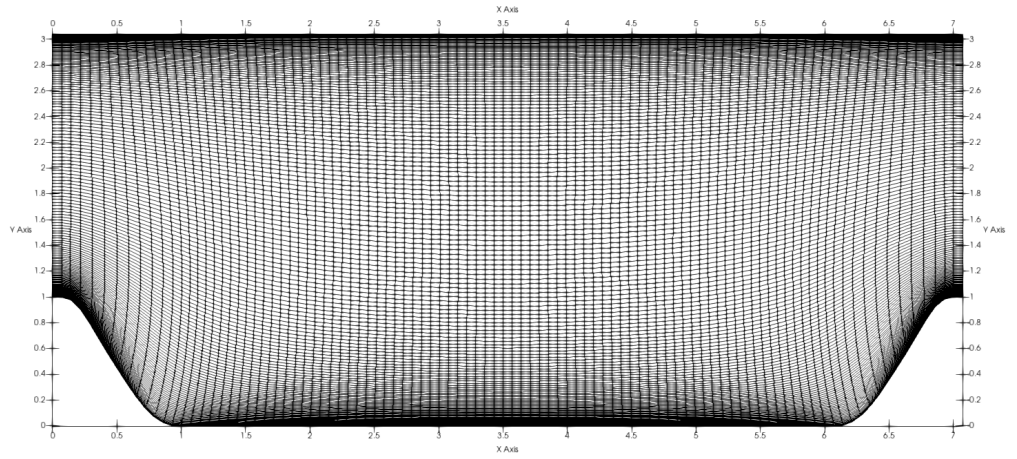
Figure D.10: Final KANs model - Training with λ

The obtained symbolic formula is

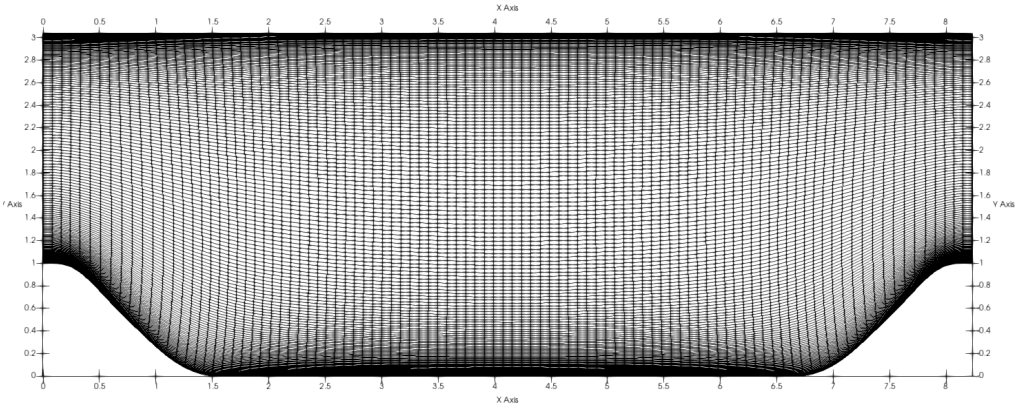
$$
\begin{aligned}
f(q_{\text{dest}}, q_{\text{diff}}, q_{\text{prod}}, q_\nu) = \; & -2.220\, q_{\text{diff}} + 2.753\, q_{\text{prod}} \\
& + 0.425\, \sin(0.846\, q_\nu - 0.996) \\
& - 0.001\, \sin(-1.375\, q_{\text{dest}} - 0.177\, q_\nu + 0.131\, \sin(1.652\, q_{\text{diff}} - 4.597) \\
& \qquad + 0.559\, \cos(1.634\, q_{\text{prod}} + 7.012) + 7.969) \\
& + 0.120\, \sin(0.266\, q_{\text{diff}} + 0.656\, \sin(0.781\, q_{\text{dest}} + 2.204) \\
& \qquad + 2.095\, \cos(0.721\, q_{\text{prod}} - 9.220) + 12.834\, \cos(0.169\, q_\nu + 6.673) - 9.856) \\
& + 1.091\, \sin(-1.162\, q_\nu + 0.298\, \sin(0.857\, q_{\text{diff}} + 3.966) \\
& \qquad + 0.139\, \sin(1.971\, q_{\text{prod}} - 7.801) + 0.073\, \cos(1.689\, q_{\text{dest}} - 8.789) + 1.372) \\
& + 0.190\, \cos(2.040\, q_{\text{dest}} + 0.222) + 0.490
\end{aligned}
\tag{D.2}
$$

D.2.2. Testing Case

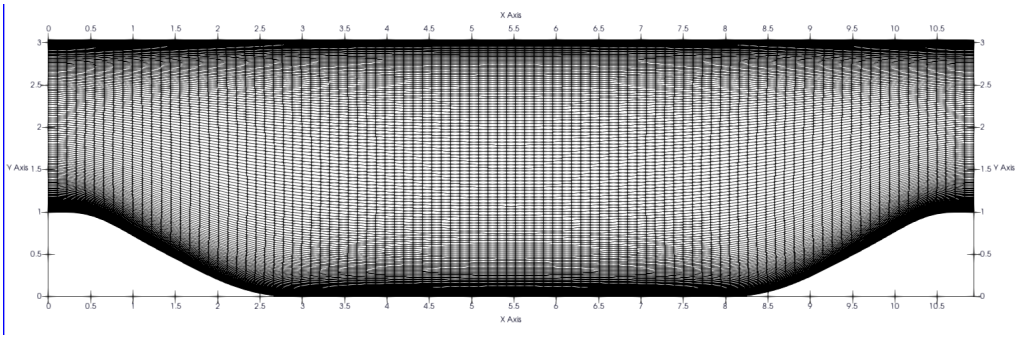
The mesh and its statistics are presented below



(a) $\alpha = 0.5$



(b) $\alpha = 0.8$



(c) $\alpha = 1.5$

Figure D.11: Testing geometries and mesh

Table D.1: y^+ statistics for $a\ posteriori$ testing across different mesh sizes (α)

Statistic	$\alpha = 0.5$	$\alpha = 0.8$	$\alpha = 1.5$
Minimum	0.02	0.050	0.042
Maximum	0.895	0.772	1.01
Mean	0.531	0.447	0.555

The distribution of invariants for the meshes is presented below

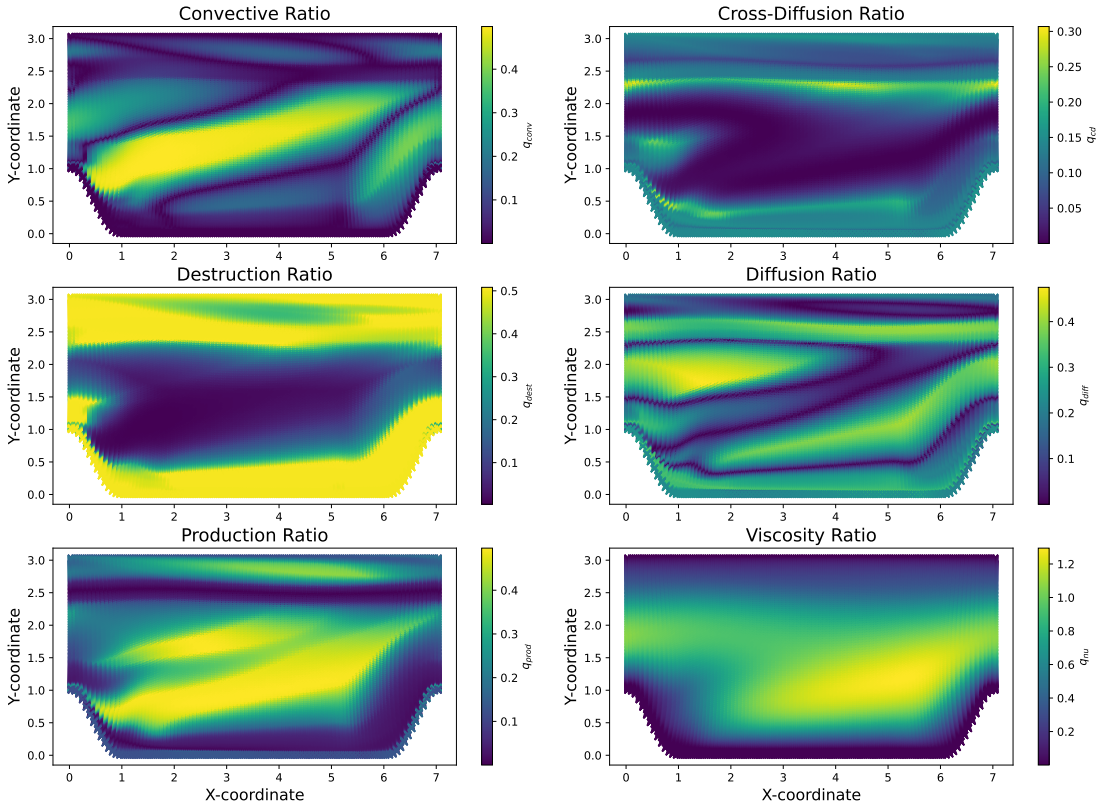


Figure D.12: Flow invariants for $\alpha = 0.5$.

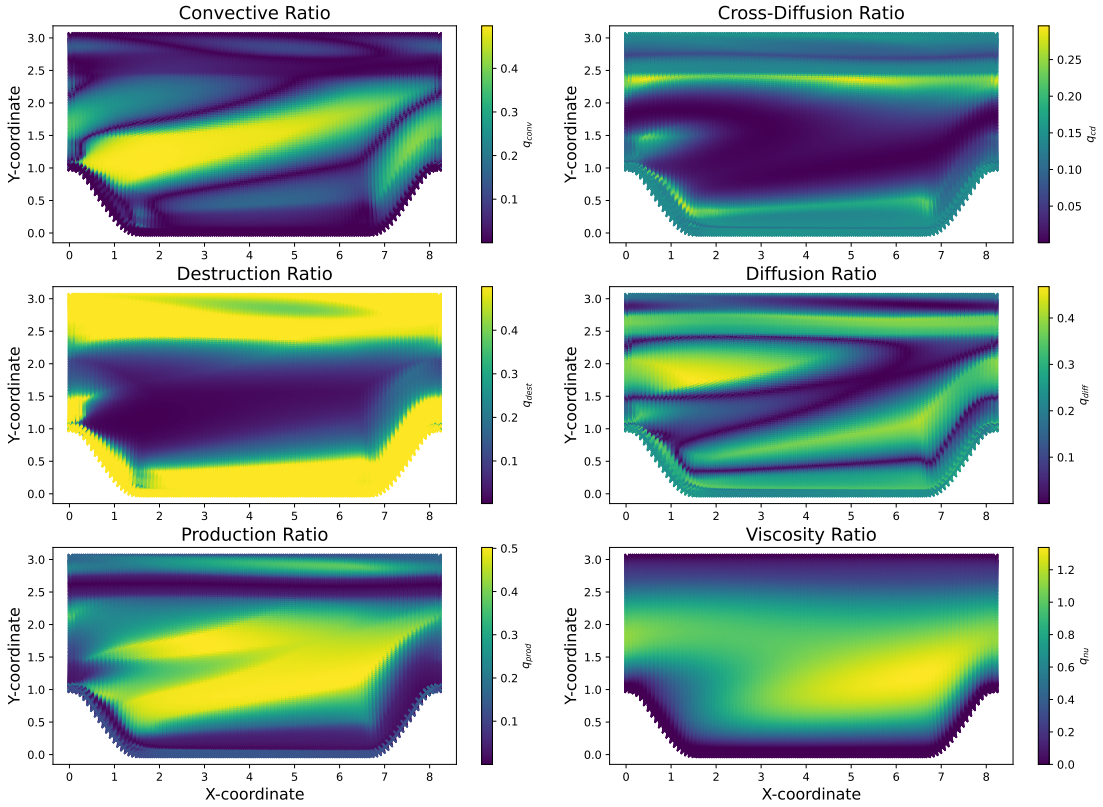


Figure D.13: Flow invariants for $\alpha = 0.8$.

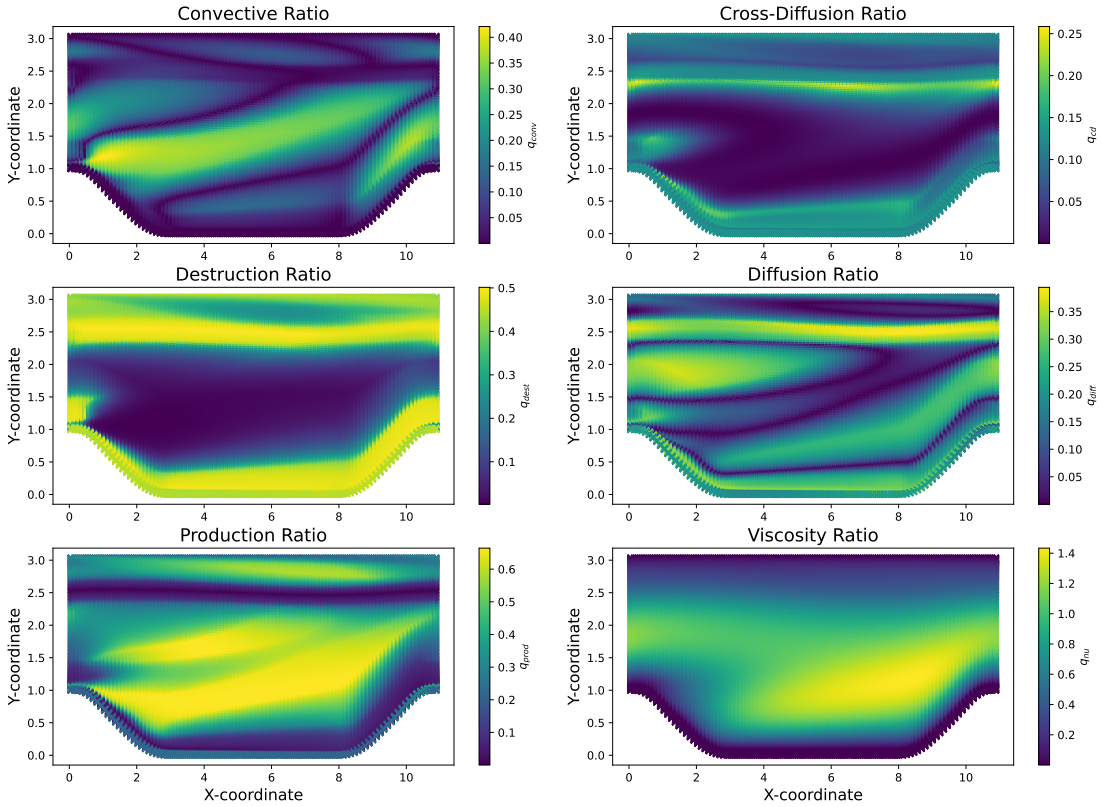
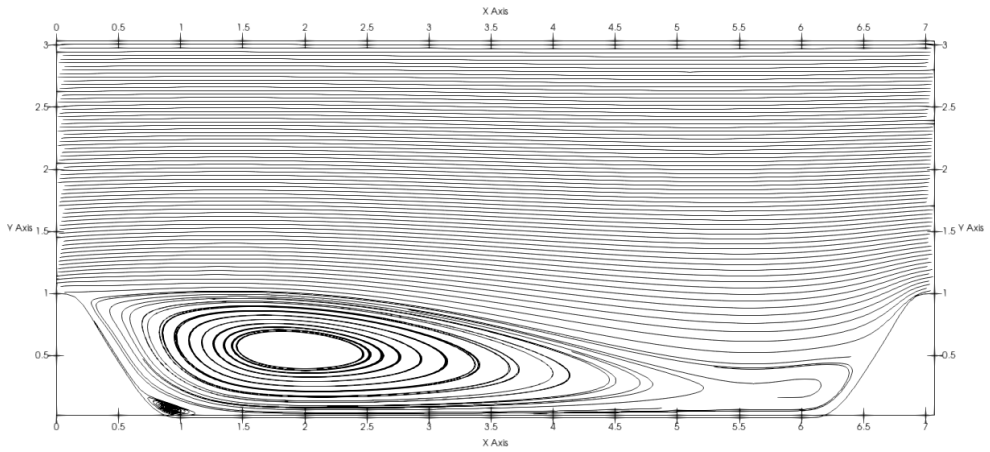


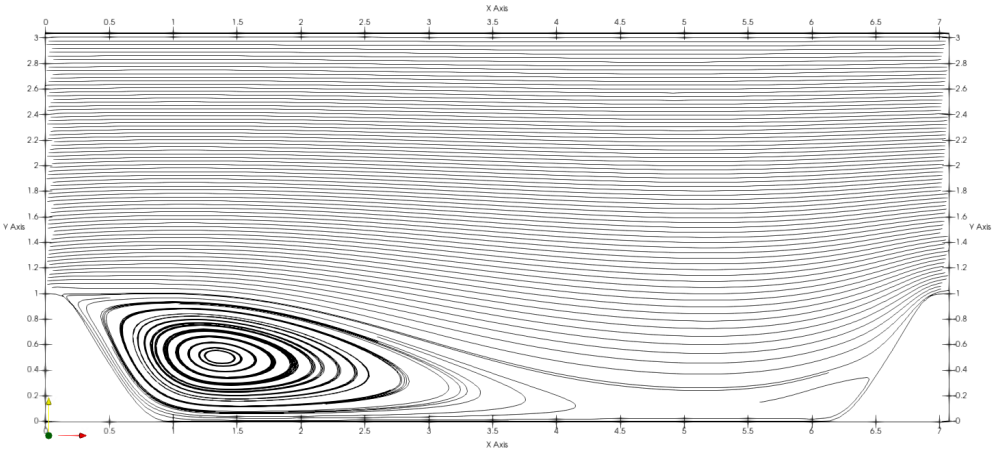
Figure D.14: Flow invariants for $\alpha = 1.5$.

The velocity streamlines for the testing cases are presented below

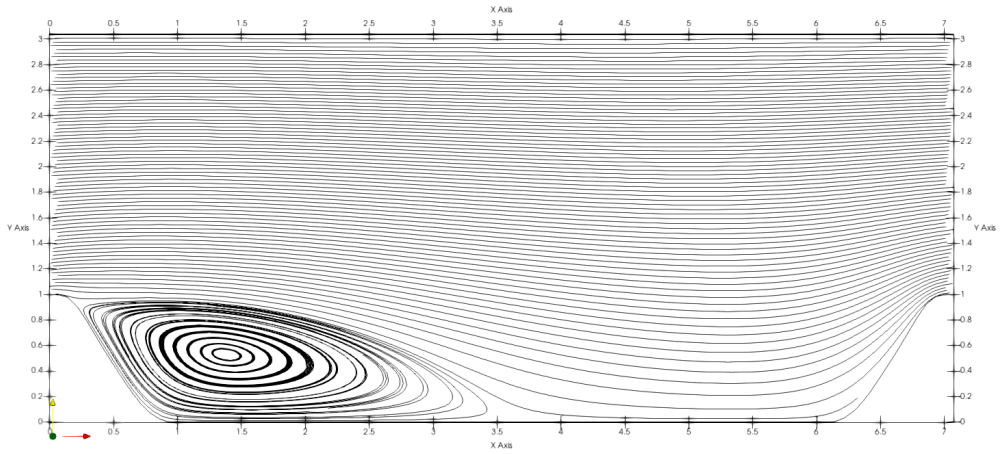
$\alpha = 0.5$



(a) $Baseline$ Spalart-Allmaras streamlines



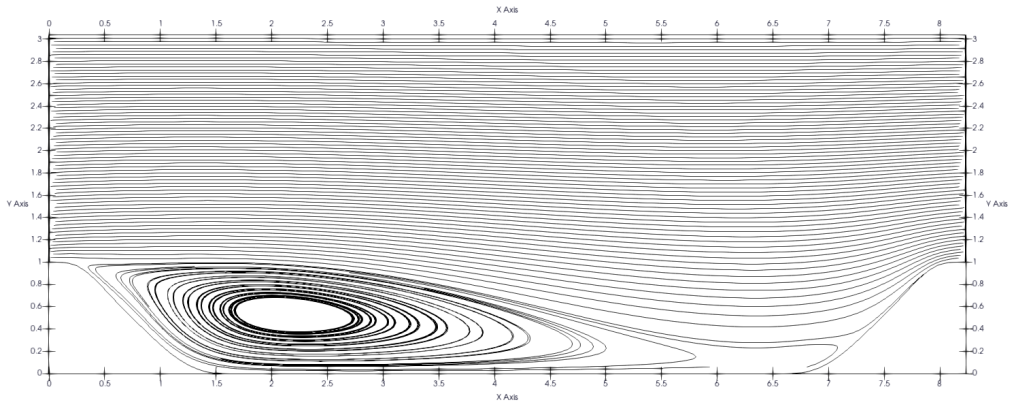
(b) $A\ posteriori$ Spalart Allmaras streamlines



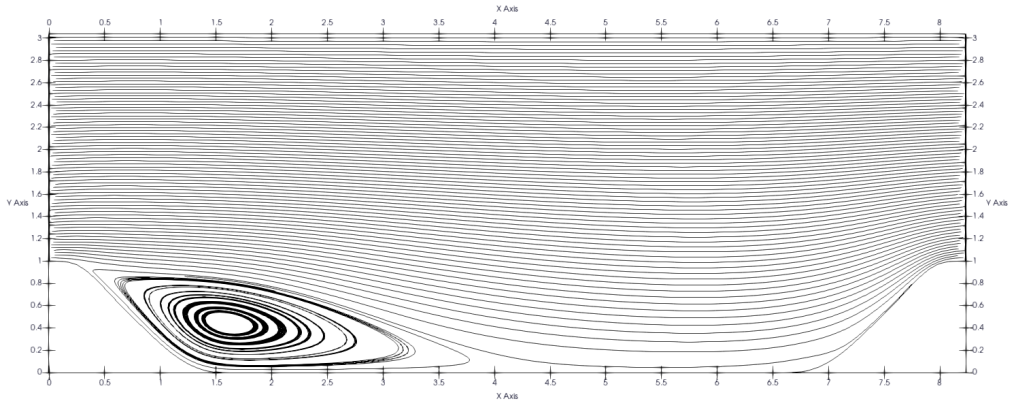
(c) $Reference\ k-\varepsilon$ streamlines

Figure D.15: Velocity streamlines - $\alpha = 0.5$

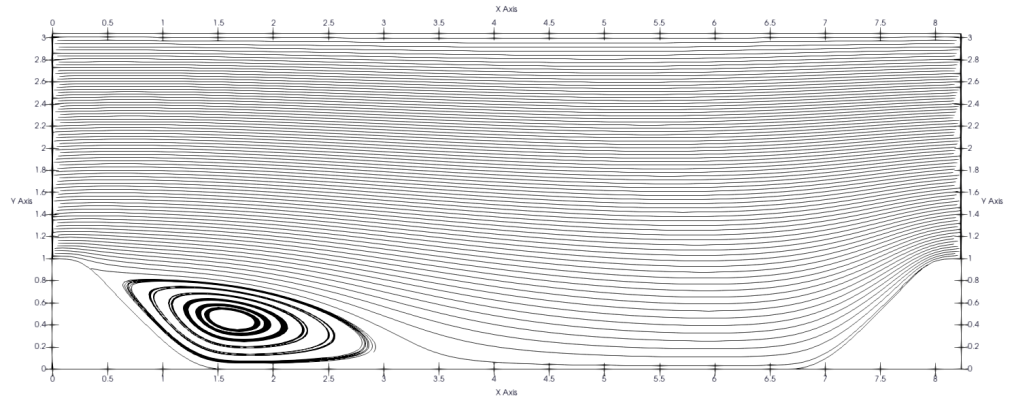
$\alpha = 0.8$



(a) $Baseline$ Spalart-Allmaras streamlines



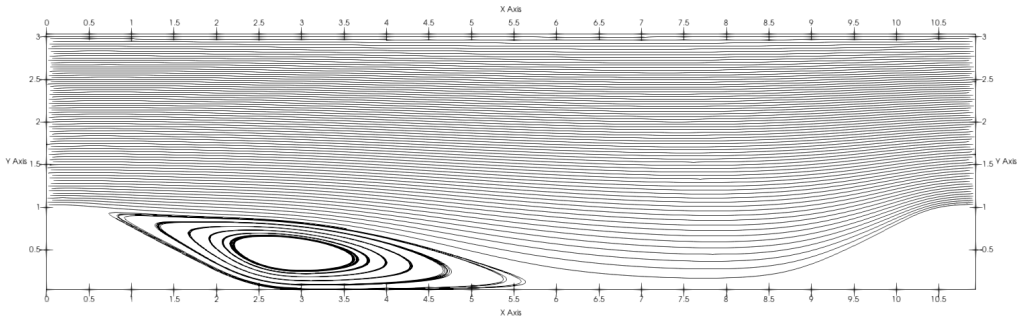
(b) $A\ posteriori$ Spalart Allmaras streamlines



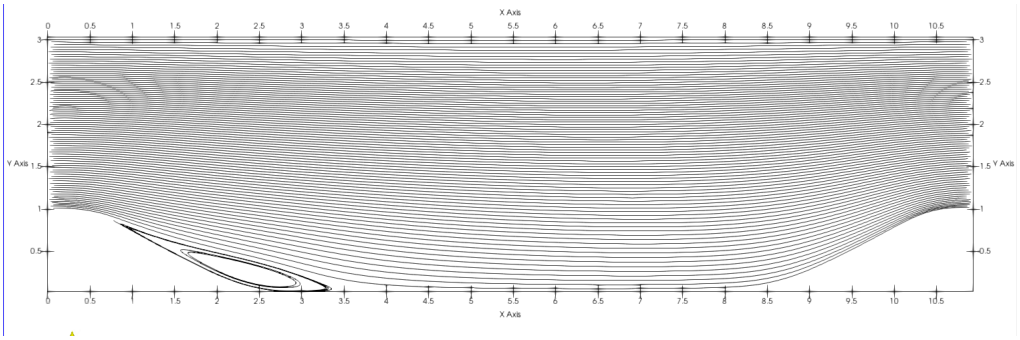
(c) $Reference\ k-\varepsilon$ streamlines

Figure D.16: Velocity streamlines - $\alpha = 0.8$

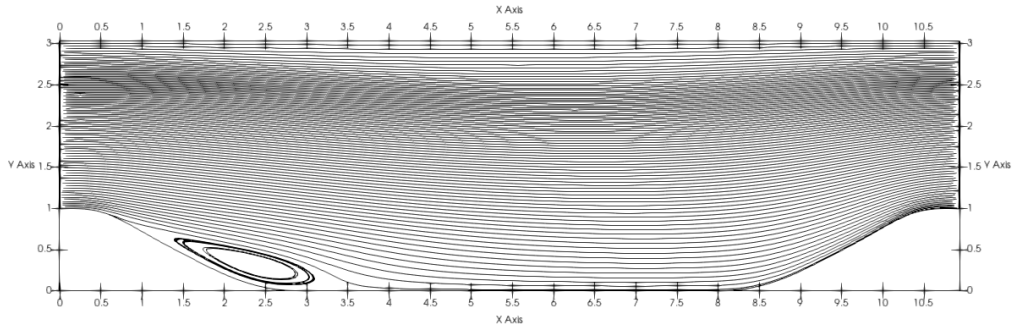
$\alpha = 1.5$



(a) $Baseline$ Spalart-Allmaras streamlines



(b) $A\ posteriori$ Spalart Allmaras streamlines



(c) $Reference\ k-\varepsilon$ streamlines

Figure D.17: Velocity Streamlines - $\alpha = 1.5$