Machine Learning Methods for Turbulence Closure Modelling

by

Ryley McConkey

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2024

© Ryley McConkey 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Amir Keshmiri Reader Dept. of Mechanical Engineering, University of Manchester
Supervisors:	Fue-Sang Lien Professor Dept. of Mechanical Engineering, University of Waterloo
	Eugene Yee Adjunct Professor Dept. of Mechanical Engineering, University of Waterloo
Internal Members:	Jean-Pierre Hickey Associate Professor Dept. of Mechanical Engineering, University of Waterloo
	Arash Arami Associate Professor Dept. of Mechanical Engineering, University of Waterloo
Internal-External Member:	Francis Poulin Professor Dept. of Applied Mathematics, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis contains verbatim content from the following publications:

Chapter 2: R. McConkey, E. Yee, and F. S. Lien, "A curated dataset for data-driven turbulence modelling", *Scientific Data* 8, 255 (2021), https://doi.org/10.1038/s41597-021-01034-2.

Chapter 3: R. McConkey, E. Yee, and F. S. Lien, "Deep structured neural networks for turbulence closure modeling", *Physics of Fluids* 34, 035110 (2022), https://doi.org/10.1063/5.0083074.

Chapter 4: R. McConkey, E. Yee, and F. S. Lien, "On the Generalizability of Machine-Learning-Assisted Anisotropy Mappings for Predictive Turbulence Modelling", *International Journal of Computational Fluid Dynamics* 36(7), 555-577 (2023), https://doi.org/10.1080/10618562.2022.2113520.

Chapter 5: R. McConkey, E. Yee, and F. S. Lien, "Realizability-Informed Machine Learning for Turbulence Anisotropy Mappings", *arXiv* (2024), https://arxiv.org/abs/5672858.

Chapter 6: R. McConkey, N.Kalia, E. Yee, and F. S. Lien, "Turbo-RANS: Straightforward and Efficient Bayesian Optimization of Turbulence Model Coefficients", *International Journal of Numerical Methods for Heat and Fluid Flow* (2024), https://doi.org/10.1108/HFF-12-2023-0726.

I solely authored all content in this thesis except for Section 6.3.3, which was co-authored with N. Kalia.

Abstract

In the aerospace, automotive, chemical, nuclear, hydroelectric, and wind industries, numerical simulations of turbulent flows are relied upon to design safe and efficient systems. However, the nature of the equations which give rise to turbulence makes them staggeringly expensive to solve numerically. Over the past decades, various techniques and modelling approaches have been proposed for the practical simulation of turbulent flows. The core modelling difficulty is the turbulence closure problem: additional unknowns appear when averaged quantities are used in the governing equations. Reynolds-averaged Navier-Stokes (RANS) is the modern "workhorse" in many industries. RANS makes many simplifications and approximations in order to enable a computationally practical technique. In particular, a major error source in most RANS approaches is the use of an eddy viscosity approximation for the turbulence closure problem.

While turbulence closure modelling for RANS has been an ongoing research area for decades, recent advancements in the field of deep learning have renewed interest in this area. The main value proposition of deep learning in turbulence closure modelling is the additional performance unlocked via the ability to infer complex functional relationships from data. Machine learning offers a promising method to augment intuition, heuristics, and simple physical arguments that have been used to traditionally construct turbulence closure models. Though RANS is considered the most popular industrial approach for simulating turbulent flow, countless examples remain where existing turbulence models are unable to capture industrially relevant physics. While the idea sounds simple, the details of how exactly to exploit machine learning for turbulence modelling is an ongoing research area, attracting substantial attention in the field.

This thesis presents two directions for augmenting turbulence closure models via machine learning. The first direction utilizes machine learning to train a highly expressive closure mapping which corrects the Reynolds stress anisotropy tensor, a major source of error for turbulence models. The corrected anisotropy tensor is injected back into the momentum equation, thereby producing corrected mean fields. The mapping is generated by training a machine learning model to predict high-fidelity closure terms from low-fidelity input features. While formulating an appropriate training procedure and model architecture for constructing this mapping receives significant attention in this thesis, another critical issue is injecting the model predictions back into a numerical simulation. Feeding the outputs from a machine learning model into a coupled set of partial differential equations is a nontrivial process, which requires attention to the stability and conditioning of the numerical solution.

This thesis consists of several published articles that address numerous issues in the broad areas of training, model architecture, and injection of data-driven anisotropy mappings. The central novelties in this area are:

- the creation of the first curated dataset for the purpose of training these anisotropy mappings,
- the proposal of two stable and well-conditioned injection frameworks,
- the formulation of several anisotropy mappings,
- the proposal of specialized neural network architectures for anisotropy mappings,
- a detailed investigation into the generalizability of data-driven anisotropy mappings,
- a new type of physics-informed loss function, termed "realizability-informed" learning, which embeds additional physics-based preferences into the learned anisotropy mapping.

While the majority of this thesis is focused on machine learning for generating anisotropy mappings, the second augmentation direction proposed is the calibration of turbulence model coefficients via Bayesian optimization. Traditional turbulence closure models contain several coefficients that can be used to tune their performance. Though tuning these coefficients can significantly enhance the performance of turbulence closure models, this tuning is not widely done. This thesis proposes a straightforward and automated procedure for tuning these coefficients, employing Bayesian optimization to efficiently locate their optimal values. Termed "turbo-RANS", the proposed calibration algorithm is demonstrated to efficiently tune coefficients within a standard turbulence closure model. A specialized objective function for the purpose of calibrating turbulence closure models is proposed. This objective function is data-flexible in that it can handle a mixture of dense, sparse, and integral parameter reference data from a variety of sources.

The recommended augmentation pathway depends on the type and availability of reference data. While machine learning augmented anisotropy mapping techniques are highly expressive, they require computationally expensive reference data. In the event that only sparse or integral parameter reference data is available, the proposed turbo-RANS algorithm can be used. Ultimately, the techniques proposed in this thesis provide a flexible set of options for leveraging data to enable accurate numerical simulations of industrially relevant turbulent flows.

Acknowledgements

It was in Dr. Alexandra Komrakova's class that I developed a passion for computational fluid dynamics. She is an inspirational teacher and an incredible mentor. I cannot thank her enough.

I greatly thank my first physics teacher, Powell Jones, for instilling in me the core principles of physics, a love of science, and a desire to pursue understanding over magic.

My supervisors, Dr. Fue-Sang Lien and Dr. Eugene Yee, have provided their expertise, ideas, and guidance during my thesis work, and for this I thank them tremendously. Dr. Alex Skillen and Dr. Alistair Revell hosted me as a visiting student at the University of Manchester, where I gained a wealth of knowledge about turbulence closure modelling. I thank them both, as well as their research group, for welcoming me and contributing to this thesis work. I also thank all members of my examining committee for their time invested in reading my thesis and providing suggestions.

Dr. Goncalo Pedro, Junting Chen, and Stephen Dajka at RWDI provided an excellent opportunity for me to apply these techniques to real-world engineering flows. I thank the three of them for their ideas and discussions during our collaboration.

Thank you to Mom, Dad, Camille, and Jenna for your love and support during my graduate studies.

Dedication

This thesis is dedicated to Jamie, because she made it possible.

Table of Contents

Examining Committee ii

Author's Declaration iii

Statement of Contributions iv

Abstract v

Acknowledgements vii

Dedication viii

List of Figures xv

List of Tables xxv

1 Introduction 1

2 A curated dataset for data-driven turbulence modelling 7

Abstract 7

2.1 Background & Summary 8

2.2 Methods 10

2.2.1 Selection of reference cases 10

2.2.2 Computational method . . . 11
2.2.3 Mesh . . . 13
2.2.4 Periodic hills . . . 14
2.2.5 Square duct . . . 15
2.2.6 Parametric bumps . . . 17
2.2.7 Converging-diverging channel . . . 21
2.2.8 Curved backward-facing step . . . 23
2.3 Data Records . . . 25
2.3.1 Labels . . . 29
2.3.2 Invariants of tensor bases . . . 29
2.3.3 Other input scalars . . . 30
2.4 Technical Validation . . . 30
2.5 Usage Notes . . . 35
2.6 Code availability . . . 38

3 Deep structured neural networks for turbulence closure modelling 39
Abstract . . . 39
3.1 Introduction . . . 40
3.2 Methodology . . . 42
3.2.1 Optimal eddy viscosity . . . 43
3.2.2 Non-linear part of the Reynolds stress tensor . . . 44
3.2.3 Conditioning . . . 49
3.2.4 Injection procedure . . . 50
3.2.5 The ϕ-f model . . . 53
3.2.6 Numerical methods . . . 54
3.3 Deep learning procedure for predicting the Reynolds stress tensor . . . 55
3.3.1 Dataset . . . 55
3.3.2 Model architecture . . . 55

3.3.3 Input feature selection . . . 57
3.3.4 Label calculation . . . 60
3.3.5 Pre-processing . . . 60
3.3.6 Training . . . 62
3.4 Results . . . 63
3.4.1 *A priori* assessment . . . 64
3.4.2 *A posteriori* assessment . . . 66
3.4.3 Extrapolation test case . . . 73
3.4.4 Interpreting the data-driven corrective closure model . . . 75
3.5 Conclusion . . . 77

4 On the generalizability of machine-learning-assisted anisotropy mappings for predictive turbulence modelling 81

Abstract . . . 81
4.1 Introduction . . . 82
4.2 Methodology . . . 84
4.2.1 Machine learning for RANS closure modelling . . . 85
4.2.2 Optimal tensor basis expansion for the Reynolds stress anisotropy . . . 87
4.2.3 Machine learning models . . . 90
4.2.4 Input feature selection . . . 91
4.2.5 Dataset and preprocessing . . . 94
4.2.6 Training, hyperparameter optimization, and evaluation . . . 96
4.2.7 Libraries and code . . . 99
4.3 Results and discussion . . . 100
4.3.1 Coefficients for optimal tensor basis expansion . . . 100
4.3.2 Predictions of optimal tensor basis coefficients . . . 108
4.4 Conclusions . . . 120

5 Realizability-informed machine learning for turbulence anisotropy mappings 123

Abstract . . . 123

5.1 Introduction . . . 124

5.2 Methodology . . . 126

5.2.1 Reynolds stress tensor decompositions used in data-driven closure modelling frameworks . . . 126

5.2.2 Conditioning analysis . . . 130

5.2.3 Realizability-informed training . . . 131

5.2.4 Neural network architecture . . . 138

5.2.5 Machine learning procedure . . . 143

5.3 Results . . . 147

5.3.1 Generalization tests . . . 147

5.3.2 Impact of realizability-informed training . . . 161

5.4 Conclusion . . . 164

6 Turbo-RANS: straightforward and efficient Bayesian optimization of turbulence model coefficients 166

Abstract . . . 166

6.1 Introduction . . . 167

6.2 Background and Methodology . . . 171

6.2.1 Bayesian optimization . . . 171

6.2.2 Gaussian process regression . . . 172

6.2.3 Utility functions . . . 173

6.2.4 Generalized Error and Default Coefficient Preference objective function 175

6.2.5 turbo-RANS framework . . . 181

6.2.6 Implementation . . . 184

6.3 Demonstrations . . . 185

6.3.1 Example 1: Flow over an airfoil . . . 185

6.3.2 Example 2: Periodic hills . . . 189
6.3.3 Example 3: Converging-diverging channel . . . 196
6.3.4 Discussion . . . 200
6.4 Hyperparameter optimization . . . 200
6.4.1 Utility function . . . 203
6.4.2 Matern kernel parameters . . . 207
6.4.3 Number of initial samples . . . 207
6.4.4 Recommendations . . . 208
6.5 Conclusion . . . 210

7 Conclusion **213**
7.1 Future work . . . 215

References **217**

APPENDICES **234**

A Neural network hyperparameter selection **235**

B Invariants of the basis tensors for 2D flows **237**

C Hyperparameter search space for generalizability investigation **242**

D Additional results for generalizability investigation **244**
D.1 Periodic hills test set . . . 245
D.1.1 PHLL4 training set . . . 245
D.1.2 BUMP4 training set . . . 246
D.1.3 DUCT15 training set . . . 248
D.1.4 FULL training set . . . 249
D.2 Parametric bump test set . . . 251

D.2.1 PHLL4 training set . . . 251
D.2.2 BUMP4 training set . . . 252
D.2.3 DUCT15 training set . . . 254
D.2.4 FULL training set . . . 255
D.3 Square duct test set . . . 257
D.3.1 PHLL4 training set . . . 257
D.3.2 BUMP4 training set . . . 258
D.3.3 DUCT15 training set . . . 260
D.3.4 FULL training set . . . 261
D.4 Converging-diverging channel test set . . . 263
D.4.1 PHLL4 training set . . . 263
D.4.2 BUMP4 training set . . . 264
D.4.3 DUCT15 training set . . . 266
D.4.4 FULL training set . . . 267
D.5 Curved backward-facing step test set . . . 269
D.5.1 PHLL4 training set . . . 269
D.5.2 BUMP4 training set . . . 270
D.5.3 DUCT15 training set . . . 272
D.5.4 FULL training set . . . 273

E Extended invariants of the basis tensors **275**

List of Figures

2.1 The geometry for the five periodic hills cases. 15

2.2 Structured hexahedral mesh used to discretize the $\alpha = 0.5$ periodic hills case. 16

2.3 The mesh and geometry for the square duct cases. 17

2.4 The geometry for the five parametric bump cases. 18

2.5 Structured hexahedral mesh used to discretize the $h = 42$ mm parametric bump case. 19

2.6 The mesh and geometry for the two converging-diverging channel cases. . . 21

2.7 The geometry and mesh for the curved backward-facing step case. 23

2.8 Profiles of U for three meshes of varying density for the $\alpha = 0.5$ periodic hills case. 31

2.9 Profiles of V for three meshes of varying density for the $\alpha = 0.5$ periodic hills case. 32

2.10 Profiles of U for three meshes of varying density for the $Re = 3,500$ square duct case. 33

2.11 Profiles of U for three meshes of varying density for the $h = 42$ mm parametric bump case. 34

2.12 Profiles of V for three meshes of varying density for the $h = 42$ mm parametric bump case. 34

2.13 Profiles of U for three meshes of varying density for the $Re = 20,580$ converging-diverging channel case. 35

2.14 Profiles of V for three meshes of varying density for the $Re = 20,580$ converging-diverging channel case. 36

2.15 Profiles of U for three meshes of varying density for the curved backward-facing step case. 36
2.16 Profiles of V for three meshes of varying density for the curved backward-facing step case. 37

3.1 Error distribution in each anisotropy component after invoking the linear eddy viscosity approximation $a = -2\nu_t S$. 46
3.2 Fit quality metrics after invoking the linear eddy viscosity approximation $a = -2\nu_{t\theta}^{\dagger} S_\theta$. 47
3.3 *A priori* contours of error in each component of a after invoking the linear eddy viscosity approximation $a = -2\nu_{t\theta}^{\dagger} S_\theta$. 47
3.4 *A posteriori* contours of error in the velocity vector components after injecting the optimal linear eddy viscosity into the RANS equations, without any non-linear terms. 48
3.5 *A posteriori* contours of error in the x-velocity component after injecting the Reynolds stress anisotropy tensor from DNS into the RANS equations. 50
3.6 *A posteriori* contours of error in the y-velocity component after injecting the Reynolds stress anisotropy tensor from DNS into the RANS equations. 51
3.7 "Qualitative" residual plot for the machine learning corrective framework. . 52
3.8 The geometry and flow variables used for the periodic hills cases. 56
3.9 Loss curves during training for the EVNN. 63
3.10 Loss curve during training for the apNN. 64
3.11 *A priori* prediction accuracy for the EVNN on the $\alpha = 1.2$ test case. . . . 65
3.12 *A priori* contours of $\nu_t^{\dagger}$ for the test case. 66
3.13 *A priori* prediction accuracy for the apNN on the $\alpha = 1.2$ test case. 67
3.14 *A priori* contours of each component of $a^{\perp}$ for the test case. 68
3.15 *A posteriori* contours of velocity magnitude after injecting the model predictions into the RANS equations. 69
3.16 *A posteriori* samples of the U velocity component along several lines throughout the flow field. 69
3.17 *A posteriori* samples of the V velocity component along several lines throughout the flow field. 70

3.18 Error distribution for the U and V velocity components after injecting the neural network model predictions for $\nu_t^\dagger$ and $a^\perp$ into the RANS equations. . 71
3.19 *A posteriori* contours of the anisotropy tensor magnitude. 71
3.20 *A posteriori* comparison of conditioning error in U to conditioning errors in a. 72
3.21 Wall shear stress for a portion of the bottom wall. 73
3.22 *A posteriori* samples of the U velocity component along several lines throughout the flow field for the extrapolation test case. 74
3.23 Mean absolute SHAP values for each feature over the entire test case. . . . 76
3.24 Relative feature importance at six locations in the flow. 78

4.1 The flows used in the training dataset. 95
4.2 Optimal values for $g^{(1)}$ for one selected case from each flow type. 101
4.3 Optimal values for $g^{(2)}$ for one selected case from each flow type. 102
4.4 Optimal values for $g^{(3)}$ for one selected case from each flow type. 104
4.5 Three-dimensional scatter plot of the optimal $g^{(n)}$ coefficients. 106
4.6 Violin plots showing the distributions of optimal $g^{(n)}$ coefficients. 107
4.7 The $g^{(1)}$ fields for the periodic hills test case. 111
4.8 The $g^{(1)}$ fields for the parametric bump test case. 111
4.9 The $g^{(1)}$ fields for the square duct test case. 112
4.10 Generalizability test on the periodic hills test case for the XGBoost model trained on the PHLL4 dataset. 114
4.11 Generalizability test on the parametric bump test case for the XGBoost model trained on the PHLL4 dataset. 115
4.12 Generalizability test on the periodic hills test case for the XGBoost model trained on the BUMP4 dataset. 116
4.13 Generalizability test on the parametric bump test case for the XGBoost model trained on the BUMP4 dataset. 117
4.14 Components of b calculated using the $g^{(n)}$ coefficients from Figure 4.10, and the tensors from the base k-ω SST model. 118

4.15 Components of b calculated using the $g^{(n)}$ coefficients from Figure 4.12, and the tensors from the base k-ω SST model. 119

5.1 Conditioning test comparing the errors in U_1 after injecting various decompositions of the DNS anisotropy tensor into the RANS equations. 132

5.2 Conditioning test comparing the errors in U_2 after injecting various decompositions of the DNS anisotropy tensor into the RANS equations. 133

5.3 Realizability penalty for the diagonal and off-diagonal components of b_{ij}. . 136

5.4 Model architecture and training configuration for the original TBNN proposed by Ling et al. [23]. 140

5.5 Model architecture and training configuration proposed in the present investigation. 141

5.6 Test-time injection configuration proposed in the present investigation. . . 142

5.7 Loss function vs epoch for the TBNN and KCNN models trained on each dataset. 148

5.8 Computational domain for the zero pressure gradient flat plate boundary layer case. 149

5.9 Streamwise velocity profile in the $Re_\theta = 3630$ boundary layer predicted by the k-ω SST model, compared to the reference DNS data from Schlatter and Örlü [125]. 150

5.10 Momentum thickness Reynolds number growth along the flat plate as predicted by the k-ω SST model, and the reference DNS data from Schlatter and Örlü [125]. 151

5.11 Evolution of a_{11}, a_{12}, a_{22}, and a_{33} in the $Re_\theta = 3630$ boundary layer, as predicted by the DNS data from Schlatter and Örlü [125], the TBNN/KCNN model, and the baseline k-ω SST model. 152

5.12 Computational domain and RANS mesh for the square duct flow. 153

5.13 Components of a_{ij} predicted by the DNS data from Pinelli et al. [45], the k-ω SST model, and the TBNN/KCNN model from the present investigation. 154

5.14 Contours of turbulent kinetic energy k predicted by the k-ω SST model, the KCNN in the present investigation, and the DNS data from Pinelli et al. [45]. 155

5.15 Velocity vectors predicted by the DNS data from Pinelli et al. [45] and injecting the TBNN/KCNN predictions into the RANS momentum equations. 156

5.16 Profiles of the in-plane velocity components U_2 and U_3, as predicted by the k-ω SST model, the injected TBNN/KCNN predictions, and the DNS data from Pinelli et al. [45]. 157

5.17 Computational domain and mesh for the periodic hills case, with $\alpha = 1.2$. . 158

5.18 Contours of non-zero a_{ij} components predicted by the DNS data from Xiao et al. [44], the k-ω SST model, and the TBNN/KCNN model predictions from the present investigation. 159

5.19 Contours of U_1 and U_1 predicted by the DNS data from Xiao et al. [44], the k-ω SST model, and the injected TBNN/KCNN model predictions from the present investigation. 160

5.20 Contours of error in U_1 and U_2 predicted by the k-ω SST model and the injected TBNN/KCNN model predictions, as compared to the DNS data from Xiao et al. [44]. 160

5.21 Projection of the predicted b_{ij} from the k-ω SST model, the DNS reference data, and the TBNN/KCNN with and without realizability-informed learning onto the barycentric triangle. 163

6.1 Objective function for the airfoil case. 179

6.2 Objective function for the periodic hills case. 180

6.3 The turbo-RANS framework summarized in an algorithm flowchart. 182

6.4 Computational setup for the airfoil example case. 187

6.5 Convergence history for the optimization process for the airfoil example case. 189

6.6 Computational setup for the periodic hills example case. 191

6.7 Convergence of the optimization process for the case of the periodic hills. . 193

6.8 The streamwise (U) component of velocity along various vertical sampling lines for the case of the periodic hills. 194

6.9 The vertical (V) component of velocity along various vertical sampling lines for the case of the periodic hills. 194

6.10 The turbulent kinetic energy k along various vertical sampling lines for the case of the periodic hills. 195
6.11 The wall shear stress along the bottom wall for the case of the periodic hills. 195
6.12 Computational domain for the converging-diverging channel. 196
6.13 Mesh for the converging-diverging channel. 196
6.14 Pressure coefficient along the bottom wall for the converging-diverging channel case. 199
6.15 Utility function hyperparameter search results for the airfoil case. 205
6.16 Utility function hyperparameter search results for the periodic hills case. . 206
6.17 Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence for the Matern kernel for different values of the hyperparameter ν. 208
6.18 Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence for various numbers n_s of initial sample points. 209

D.1 Generalizability test on the periodic hills test case for the neural network model trained on the periodic hills PHLL4 set. 245
D.2 Generalizability test on the periodic hills test case for the random forest model trained on the periodic hills PHLL4 set. 245
D.3 Generalizability test on the periodic hills test case for the XGBoost model trained on the periodic hills PHLL4 set. 246
D.4 Generalizability test on the periodic hills test case for the neural network model trained on the parametric bump BUMP4 set. 246
D.5 Generalizability test on the periodic hills test case for the random forest model trained on the parametric bump BUMP4 set. 247
D.6 Generalizability test on the periodic hills test case for the XGBoost model trained on the parametric bump BUMP4 set. 247
D.7 Generalizability test on the periodic hills test case for the neural network model trained on the square duct DUCT15 set. 248
D.8 Generalizability test on the periodic hills test case for the random forest model trained on the square duct DUCT15 set. 248

D.9 Generalizability test on the periodic hills test case for the XGBoost model trained on the square duct DUCT15 set. 249
D.10 Generalizability test on the periodic hills test case for the neural network model trained on the FULL set. 249
D.11 Generalizability test on the periodic hills test case for the random forest model trained on the FULL set. 250
D.12 Generalizability test on the periodic hills test case for the XGBoost model trained on the FULL set. 250
D.13 Generalizability test on the parametric bumps test case for the neural network model trained on the periodic hills PHLL4 set. 251
D.14 Generalizability test on the parametric bumps test case for the random forest model trained on the periodic hills PHLL4 set. 251
D.15 Generalizability test on the parametric bumps test case for the XGBoost model trained on the periodic hills PHLL4 set. 252
D.16 Generalizability test on the parametric bumps test case for the neural network model trained on the parametric bump BUMP4 set. 252
D.17 Generalizability test on the parametric bumps test case for the random forest model trained on the parametric bump BUMP4 set. 253
D.18 Generalizability test on the parametric bumps test case for the XGBoost model trained on the parametric bump BUMP4 set. 253
D.19 Generalizability test on the parametric bumps test case for the neural network model trained on the square duct DUCT15 set. 254
D.20 Generalizability test on the parametric bumps test case for the random forest model trained on the square duct DUCT15 set. 254
D.21 Generalizability test on the parametric bumps test case for the XGBoost model trained on the square duct DUCT15 set. 255
D.22 Generalizability test on the parametric bumps test case for the neural network model trained on the FULL set. 255
D.23 Generalizability test on the parametric bumps test case for the random forest model trained on the FULL set. 256
D.24 Generalizability test on the parametric bumps test case for the XGBoost model trained on the FULL set. 256

D.25 Generalizability test on the square duct test case for the neural network model trained on the periodic hills PHLL4 set. 257
D.26 Generalizability test on the square duct test case for the random forest model trained on the periodic hills PHLL4 set. 257
D.27 Generalizability test on the square duct test case for the XGBoost model trained on the periodic hills PHLL4 set. 258
D.28 Generalizability test on the square duct test case for the neural network model trained on the parametric bump BUMP4 set. 258
D.29 Generalizability test on the square duct test case for the random forest model trained on the parametric bump BUMP4 set. 259
D.30 Generalizability test on the square duct test case for the XGBoost model trained on the parametric bump BUMP4 set. 259
D.31 Generalizability test on the square duct test case for the neural network model trained on the square duct DUCT15 set. 260
D.32 Generalizability test on the square duct test case for the random forest model trained on the square duct DUCT15 set. 260
D.33 Generalizability test on the square duct test case for the XGBoost model trained on the square duct DUCT15 set. 261
D.34 Generalizability test on the square duct test case for the neural network model trained on the FULL set. 261
D.35 Generalizability test on the square duct test case for the random forest model trained on the FULL set. 262
D.36 Generalizability test on the square duct test case for the XGBoost model trained on the FULL set. 262
D.37 Generalizability test on the converging-diverging channel test case for the neural network model trained on the periodic hills PHLL4 set. 263
D.38 Generalizability test on the converging-diverging channel test case for the random forest model trained on the periodic hills PHLL4 set. 263
D.39 Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the periodic hills PHLL4 set. 264
D.40 Generalizability test on the converging-diverging channel test case for the neural network model trained on the parametric bump BUMP4 set. 264

D.41 Generalizability test on the converging-diverging channel test case for the random forest model trained on the parametric bump BUMP4 set. 265

D.42 Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the parametric bump BUMP4 set. 265

D.43 Generalizability test on the converging-diverging channel test case for the neural network model trained on the square duct DUCT15 set. 266

D.44 Generalizability test on the converging-diverging channel test case for the random forest model trained on the square duct DUCT15 set. 266

D.45 Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the square duct DUCT15 set. 267

D.46 Generalizability test on the converging-diverging channel test case for the neural network model trained on the FULL set. 267

D.47 Generalizability test on the converging-diverging channel test case for the random forest model trained on the FULL set. 268

D.48 Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the FULL set. 268

D.49 Generalizability test on the curved backward-facing step test case for the neural network model trained on the periodic hills PHLL4 set. 269

D.50 Generalizability test on the curved backward-facing step test case for the random forest model trained on the periodic hills PHLL4 set. 269

D.51 Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the periodic hills PHLL4 set. 270

D.52 Generalizability test on the curved backward-facing step test case for the neural network model trained on the parametric bump BUMP4 set. 270

D.53 Generalizability test on the curved backward-facing step test case for the random forest model trained on the parametric bump BUMP4 set. 271

D.54 Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the parametric bump BUMP4 set. 271

D.55 Generalizability test on the curved backward-facing step test case for the neural network model trained on the square duct DUCT15 set. 272

D.56 Generalizability test on the curved backward-facing step test case for the random forest model trained on the square duct DUCT15 set. 272

D.57 Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the square duct DUCT15 set. 273

D.58 Generalizability test on the curved backward-facing step test case for the neural network model trained on the FULL set. 273

D.59 Generalizability test on the curved backward-facing step test case for the random forest model trained on the FULL set. 274

D.60 Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the FULL set. 274

List of Tables

2.1 Inputs and outputs for dataset generation. 10
2.2 Cases in the dataset. 10
2.3 Kinematic molecular viscosity used for each case. 11
2.4 Units for each variable requiring boundary conditions. 13
2.5 Meshes used for discretizing the domain. 14
2.6 Boundary conditions for the periodic hills and square duct cases. 17
2.7 Boundary conditions for the flat developing flow case, used to generate an inlet profile for the bump cases. 20
2.8 Boundary conditions for the bump cases. 20
2.9 Boundary conditions for the flat developing flow case, used to generate an inlet profile for the converging-diverging channel cases. 22
2.10 Boundary conditions for the converging-diverging channel cases. 22
2.11 Boundary conditions for the flat developing flow case, used to generate an inlet profile for the curved backward facing step cases. 24
2.12 Boundary conditions for curved backward facing step case. 24
2.14 Derived feature fields available in the dataset. 25
2.13 Base fields available in the dataset. 28
2.15 Derived label fields available in the dataset. 29

3.1 Input features for EVNN and apNN. 59

4.1 Types of machine learning methodologies for turbulence closure modelling used in the generalizability investigation. 91

4.2 Input feature set used for all machine learning models. 93
4.3 Training and test datasets. 97
4.4 Mean squared error (MSE) for various training and test datasets for each model type. 109
4.5 Wall-clock time taken for the hyperparameter tuning procedure on the FULL training dataset. 110

5.1 Datasets used for training, validation, and testing. 144
5.2 Hyperparameters selected for each model. 147
5.3 Comparison of mean squared error and number of non-realizable predictions when training with and without a realizability-informed loss function. . . . 161

6.1 Summary of the airfoil lift coefficient calibration case. 189
6.2 The values of MAPE before and after turbo-RANS calibration of the model coefficients in the k-ω SST turbulence model for flow over periodic hills. . . 192
6.3 Recommended range of values for the two GEKO coefficients C_{SEP} and C_{NW}. 197
6.4 Hyperparameter search space. 202
6.5 Recommended hyperparameter settings. 210

A.1 Selected EVNN hyperparameter combinations from the optimization procedure. The bolded hyperparameters were selected for use in testing. 236

B.1 Basis tensors in the minimal integrity basis for S, R, A_k, and A_{v2}. 237

C.1 Hyperparameter tuning search space and results for each model. 243

E.1 Non-zero invariants of the minimal integrity basis tensor formed by $\hat{S}_{ij}$, $\hat{R}_{ij}$, $\hat{C}_{ij}$, and $\hat{D}_{ij}$. 280

Chapter 1

Introduction

The study of turbulence has been a focus of many mathematicians, physicists, and engineers over the past few centuries. Understanding the nature of turbulence is considered one of the greatest unsolved problems in physics; it is a nonlinear, nonequilibrium, chaotic, and high degree of freedom phenomenon involving systems of non-linear partial differential equations. It is a ubiquitous natural phenomenon. Turbulence influences the movement of a bee flapping its wings, the evolution of a shark's skin to reduce drag, the movement of Jupiter's Great Red Spot, the magnetohydrodynamic flow of plasma on the sun, and the accretion disk of a black hole.

While fundamental research into the nature of turbulence is important, we also need to be able to predict its effects. We need to predict the path of hurricanes, the weather today and tomorrow, the changing wind direction over a forest fire, and the ocean currents. These predictive requirements demand practical techniques for use by scientists and engineers. We need practical methods which are able to predict how an aircraft wing stalls at high angles of attack, how two fluids mix in a reactor, how biofouling increases the drag on a boat's hull, and where to place wind turbines for optimal energy extraction.

Fortunately, we know the equations that describe turbulence for most of these flows. They are called the Navier-Stokes equations, and in Einstein notation they are:

$$\frac{\partial v_j}{\partial x_j} = 0 \ , \tag{1.1}$$

$$\frac{Dv_j}{Dt} = -\frac{1}{\nu}\frac{\partial p}{\partial x_j} + \nu \frac{\partial^2 v_j}{\partial x_i \partial x_i} \ , \tag{1.2}$$

where v_j is velocity, and p is pressure.

From these equations, turbulence is born. Unfortunately, knowing the equation is different than knowing the solution, and a general smooth solution for Equations 1.1–1.2 is not currently known. Further, we do not even know that such a solution exists — despite a $\$10^6$ bounty being placed on such a proof in the year 2000, the general existence of a smooth solution remains unproven. Nevertheless, our aircraft need to stay airborne, our cars need to be energy-efficient, our nuclear and chemical plants need to operate safely, and our wind turbines need to draw power from the air. We cannot wait on a general solution for these equations, if it even exists.

The availability of computers over the past century has made *approximate* solutions of Equations 1.1–1.2 possible via numerical methods. Our description of turbulence indicates that it is a multiscale phenomenon. For flow over an aircraft's wing, the "size" of the generated turbulence can start at a similar size to the wing itself, but these turbulent structures in the air break apart and cascade downward in size until they are at the millimeter scale or smaller. The large scale motions affect the small scale motions, and the reverse is also true. Therefore, accurately approximating Equations 1.1–1.2 requires resolving all of these scales. Directly solving these equations with appropriate resolution in both time and space is called direct numerical simulation (DNS), and it is *tremendously* expensive for flows of engineering relevance. As an example, Hosseini et al. [1] used DNS to simulate the flow over a small aircraft wing section. It cost $30(10)^6$ CPU core-hours. For reference, this calculation would take over 800 years on a typical 4-core laptop. This computational cost is staggering, especially when considering the turbulence simulated is only relevant for small aircraft such as drones. Directly simulating turbulence for a modern commercial airliner wing would be even more expensive.

While DNS is an available technique, we need the ability to create efficient and safe designs without waiting 800 years for a calculation to complete. The tremendous cost of DNS originates from the requirement to resolve all turbulent scales. If instead, we choose to *model* smaller scales, we can greatly reduce the resolution requirement. This technique is referred to as large eddy simulation (LES), since we simulate the large scales, and model the small scales. This technique is seen as "filtering" out small scale effects in Equations 1.1–1.2, and replacing them with a model [2]. However, LES is still prohibitively expensive in many cases. On a 4-core laptop, a simulation could easily take a year or more. However, with modern high performance computing (HPC) resources, hundreds or thousands of CPUs can be accessed for a calculation. These HPC resources mean that LES is currently accessible and used for some industrial flows, generally with simulations producing results in days or weeks.

A further reduction in cost can be made by selecting a Reynolds-averaged Navier-Stokes (RANS) approach. In RANS, turbulence is modelled as a single-scale phenomenon. This approach comes with a substantial reduction in computational cost, making turbulent flow predictions accessible for some flows on a 4-core laptop. For this reason, RANS is considered the "workhorse" for most practical predictions of turbulent flows. However, the use of RANS comes with a substantial amount of modelling. Massive industrial interest in RANS has resulted in hundreds of turbulence models being proposed [3, 4], each with their own advantages and disadvantages. Although development is still ongoing today, most academic interest in new RANS models occurred in the years 1970–2005, after which an "asymptotic" relationship between model complexity and accuracy was reached. There are several limitations behind the RANS approach that ultimately limit these techniques. As model complexity grows, the generalizability tends to decrease due to the no free lunch theorem. Additionally, complex models often exhibit numerical instability, preventing their widespread use.

It should be noted that there are other simulation techniques other than DNS, LES, and RANS. For example, hybrid techniques such as detached eddy simulation (DES) exist, which use RANS for some parts of the flow, and LES for the other parts of the flow. Sub-techniques exist within these categories, such as Reynolds stress transport modelling (RSTM), which is within the realm of RANS. The aforementioned techniques are usually implemented within a finite volume discretization method (FVM), but an even broader range of techniques are currently available for simulating turbulent flows. Simulation of turbulent flows can be completed using the mesoscale lattice Boltzmann method (LBM) and smoothed particle hydrodynamics (SPH), which are outside the scope of this thesis. This thesis focuses on the "workhorse" technique of eddy viscosity modelling within RANS, which remains the most popular technique for simulating turbulent flows [5, 6].

The rise of deep learning in the 2010's brought renewed interest to developing RANS techniques, with a new tool able to learn functional relationships from data. Whereas previous RANS modelling efforts approached the limits of what can be accomplished by constructing model equations via a mix of intuition and physics-based arguments, deep learning offers a way to unlock an additional level of performance by learning highly complex relationships that often cannot be reasoned from simple physics-based arguments. Though various algorithms for training perceptrons and neural networks were proposed before 2000, deep learning became possible and widespread in the 2010's due to various technological breakthroughs such as advancements in gradient descent algorithms to address gradient vanishing. Additionally, widespread availability of massive datasets such as ImageNet [7] empowered deeper and deeper models to be trained, with benchmarks in a variety of fields being repeatedly shattered. Widespread interest and continued develop-

ment of new techniques has led to prominent large language models such as the generative pretrained transformer (GPT) model series [8]. The rising tide of deep learning has floated boats in a variety of scientific fields, such molecular biology [9], materials science [10], and medicine [11].

Within the realm of fluid mechanics, deep learning has also drawn extensive interest [12]. Deep learning offers a readily available solution to many open problems in fluid mechanics. Substantial efforts have been made in the area of physics informed neural networks (PINNs) [13, 14], a technique which is able to use neural networks to approximate physical systems. While superresolution is a major area of research in the broader machine learning context [15], specialized superresolution algorithms for fluids are a promising research area with applications in experimental and computational fluid dynamics. [16]. In the area of computational turbulence modelling, data-driven approaches to various sub-problems have been proposed [17], such as field inversion machine learning [18, 19], and accelerating computations [20]. In the area of RANS, substantial efforts have been invested training and discovering algebraic Reynolds stress models [17, 21, 22, 23].

This thesis focuses on the area of using machine learning to improve RANS simulations. There are two main research pathways in this thesis: machine learning anisotropy mappings for augmenting RANS turbulence models (Chapters 2–5), and Bayesian optimization of turbulence model coefficients (Chapter 6). These two pathways are distinct—though they address the same problem of increasing the accuracy of a RANS simulation using data, their implementation and use are different. Machine learning anisotropy mappings require a rich dataset containing dense high-fidelity data and RANS data for the same flow. In exchange for this expensive requirement, a highly complex anisotropy mapping (e.g., a non-linear eddy viscosity mapping) can be generated to significantly increase the accuracy for flows similar to the training dataset. On the other hand, calibrating global turbulence model coefficients requires significantly less data—it can be done based on a single reference integral parameter, such as a target lift coefficient. However, the calibrated simulation retains the same fundamental limitations as the original turbulence model, which in most cases contain linear eddy viscosity approximations. The decision to pursue either of these pathways depends on data availability, and if a highly sensitized anisotropy mapping is desired, or just a calibrated standard turbulence model.

This thesis consists of several standalone Chapters (Ch. 2–6). These standalone chapters constitute a series of journal articles, all having been previously published in peer-reviewed journals at the time of this thesis except Chapter 5. Therefore, these chapters are self-contained in that they first discuss and position the investigation relative to available literature, present their own methodology and results, and contain their own conclusions. Key findings from each chapter, outlook on the overall field, and suggestions for future

work are presented in Chapter 7. The overarching story that connects each chapter is summarized in the following paragraphs.

Chapter 2 begins by generating a dataset suitable for training a data-driven anisotropy mapping. Prior to publication of this chapter, various fragmented data sources existed from resources such as the John Hopkins database [24], the NASA turbulence modelling resource [25], and the ERCOFTAC database [26]. The primary novelty of Chapter 2 is the collocation of both RANS and high-fidelity data for the same flow, in an easy-to-use format. Providing both the RANS and high-fidelity data in a curated format means that future modellers do not need to run their own RANS simulations. Further, this open-source dataset makes standardization and benchmarking more accessible—two algorithms in two separate investigations can be easily compared on the same data, without each investigation needing to generate their own. Inspiration for this dataset was drawn from the broader machine learning community, which has leveraged large open-source datasets for benchmarking and model development.

After generating an appropriate dataset, Chapter 3 proposes a new augmented closure framework. The central novelty of this chapter is to propose a well-conditioned and stable augmented closure framework, primarily through the use of a positive eddy viscosity. Whereas previous frameworks did not constrain the eddy viscosity to be positive, the framework in Chapter 3 includes this stability-enhancing constraint. The results in Chapter 3 further demonstrate the merits of the proposed framework in terms of conditioning, stability, and generalizability. Additional novelties here include the generation of a new input feature set using the second tensor invariant, an inductive bias created through the combination of two machine learning models, and an interpretability analysis via SHapley Additive exPlanation (SHAP) values. The dataset used in Chapter 3 comes from the work in Chapter 2. The closure framework proposed in Chapter 3 was used in several other investigations throughout my research, such as augmented turbulence modelling for three-dimensional flow over an array of cubes at the University of Manchester [27], and RANS modelling of flows in an urban environment with a focus on pedestrian wind comfort in collaboration with Rowan, Williams, Davies, and Irwin (RWDI) Consulting.

An important question that arose during the training and evaluation of the deep learning models in Chapter 3 was the generalizability of anisotropy mappings. Chapter 4 contains a thorough investigation into the generalizability question. First, a model-agnostic anisotropy mapping is proposed, based on predicting optimal coefficients in Pope's tensor basis expansion for the anisotropy tensor [28]. This model-agnostic closure relationship is novel in that constraints are applied to the coefficients in order to ensure a positive eddy viscosity. Three machine learning regression model types are trained and tested on a variety of flows, to systematically test the ability of these models to generalize to new

flows. Specifically, we compare the generalizability and training costs of random forests, gradient boosting (XGBoost [29]), and deep neural networks. The results of this investigation are of interest to the broader community, since they indicate a somewhat pessimistic view regarding the ability of data-driven anisotropy mappings to generalize to new flows. Nevertheless, we show that these models generalize well to unseen configurations of the flows in the training dataset.

In Chapter 5, we address further issues in the area of data-driven anisotropy mappings. A new loss function is proposed in this chapter, which aims to embed a preference for realizable predictions into the model. Another novelty of this chapter is the closure framework used. It is the first framework to unite eddy viscosity-based approaches, and tensor basis neural network (TBNN) approaches, while retaining the advantages of both approaches. The merits of the proposed "realizability-informed" loss function and the proposed closure framework are demonstrated in this chapter. This chapter concludes the data-driven anisotropy mapping work in this thesis.

Chapter 6 presents an alternative framework for using data to improve the accuracy of a RANS turbulence model. This chapter presents an optimization framework, termed "turbo-RANS" (tuning parameters via Bayesian optimization-RANS), that can efficiently calibrate expensive simulations. Bayesian optimization is leveraged here because it can efficiently optimize expensive-to-evaluate objective functions. The main objective of turbo-RANS is to serve as a black box tuner for calibration coefficients in turbulence models. Though these coefficients exist in all turbulence models, they are almost never tuned by users because of a lack of available tools. Chapter 6 presents a framework for applying this useful optimization technique to RANS simulations, centering around a specialized objective function that can accommodate a mix of data types. Additionally, a thorough investigation into Bayesian optimization hyperparameters is completed in Chapter 6, with clear recommendations being provided to the end user. The turbo-RANS framework is entirely open-source, and code has been made available on Github for use by others [30]. While the turbo-RANS framework is more data-flexible than the anisotropy mapping frameworks, it is much less expressive in terms of the closure relationship. In turbo-RANS, the original turbulence model's (often linear) eddy viscosity relationship is retained and calibrated.

Chapter 2

A curated dataset for data-driven turbulence modelling

Associated reference:

R. McConkey, E. Yee, and F. S. Lien, "A curated dataset for data-driven turbulence modelling", *Scientific Data* 8, 255 (2021), https://doi.org/10.1038/s41597-021-01034-2.

Abstract

The recent surge in machine learning augmented turbulence modelling is a promising approach for addressing the limitations of Reynolds-averaged Navier-Stokes (RANS) models. This work presents the development of the first open-source dataset, curated and structured for immediate use in machine learning augmented corrective turbulence closure modelling. The dataset features a variety of RANS simulations with matching direct numerical simulation (DNS) and large-eddy simulation (LES) data. Four turbulence models are selected to form the initial dataset: k-ε, k-ε-ϕ_t-f, k-ω, and k-ω SST. The dataset consists of 29 cases per turbulence model, for several parametrically sweeping reference DNS/LES cases: periodic hills, square duct, parametric bumps, converging-diverging channel, and a curved backward-facing step. At each of the 895,640 points, various RANS features with DNS/LES labels are available. The feature set includes quantities used in current state-of-the-art models, and additional fields which enable the generation of new feature sets. The dataset reduces effort required to train, test, and benchmark new corrective RANS models. The dataset is available at https://doi.org/10.34740/kaggle/dsv/2044393.

2.1 Background & Summary

Numerical simulations in weather forecasting, wind and hydroelectric energy, aerospace vehicle design, automotive design, turbomachinery, nuclear plant design, and many other applications all rely on closure models to accelerate simulations while modelling the complex physical phenomenon of turbulence. While higher resolution techniques such as large-eddy simulation (LES) and direct numerical simulation (DNS) are becoming more widespread, the computational demands compared to current capabilities make these techniques unaffordable for many industrial simulations. For this reason, Reynolds-averaged Navier-Stokes (RANS) simulations are expected to remain the dominant tool for predicting flows of practical relevance to engineering and industrial problems over the next few decades [6]. However, flows with strong adverse pressure gradients [31], separation [32], streamline curvature [28], and reacting chemistry are often poorly predicted by RANS approaches. Developing methods to improve the accuracy of RANS simulations will help bridge this critical capability gap between RANS and LES[5].

Several recent investigations have demonstrated the potential of applying machine learning to the development of corrective turbulence closure models for RANS. Ling et al. [23] constructed a tensor basis neural network (TBNN), which predicts the anisotropy tensor using five invariant scalars derived from the mean strain and rotation rate tensors. The TBNN turbulence closure model developed by Ling et al. [23] is effectively a fifth-order eddy viscosity model, with locally varying coefficients predicted via deep learning. The ability to express such a locally-tuned, high-order relationship between the strain rate and anisotropy tensors is a powerful method to improve the accuracy of RANS simulations. Wu et al. [33] developed a random-forests-based model, which directly predicts the Reynolds stress anisotropy. Kaandorp [34] and Kaandorp and Dwight [35] proposed a tensor basis random forest (TBRF) model, which is the random forests analogue to the TBNN proposed by Ling et al. [23]. While the different models by Ling et al. [23], Wu et al. [33], Kaandorp and Dwight [35], Zhu and Dinh [36], Zhang et al. [37], Fang et al. [38], and Song et al. [39] all show promise, the results cannot be directly compared—each investigation used a different set of input features and labels, with different numerical settings chosen for feature generation. For this reason, Duraisamy [22] recently highlighted the need for a benchmark dataset for machine-learnt closure models.

The approach used by Ling et al. [23], Wu et al. [33], Kaandorp and Dwight [35], Zhang et al. [37] and others is referred to as corrective or open loop augmented closure modelling. In this open loop framework, the machine learning model is used to generate a one-time mapping between the fields from a converged RANS solution, and fields from DNS. A contrasting approach is the closed loop framework [21, 40], where the training process

involves conducting RANS simulations in an iterative manner, to repeatedly update the feature set until the model predictions and turbulence closure coefficients converge. The present work aims to present a dataset useful primarily for corrective augmented closure modelling, where the machine learning model is queried once to predict the Reynolds stress. However, this dataset can also be used as an initial set of fields for closed loop closure modelling, and the provided OpenFOAM case files are convenient for performing iterative simulations in a closed loop framework.

To generate a set of input features, the current requirement is for every investigator to generate a set of RANS simulations that match the DNS/LES reference cases. This requirement has several drawbacks. As the number of included datasets grows, the effort required grows. The development of the ImageNet dataset spurred rapid growth of the computer vision field, which would not have been possible otherwise. From an effort point of view, the availability of a curated dataset dramatically increases the time spent developing the models themselves, rather than setting up many RANS simulations to gather input features.

Another major drawback of the current approach arises from the issue of reproducibility in the field of computational fluid dynamics (CFD). Often, CFD studies are difficult to reproduce, due to a large number of input conditions [41]. Each investigation will use different meshes, numerical schemes, turbulence models, and other selections which affect the solution. The field of machine learning has also been plagued with reproducibility challenges, even with the widespread use of benchmark datasets [42]. While machine-learnt turbulence models are a promising approach, the development of these models could be significantly impeded by mixing two fields where reproducibility is a challenge. A well-documented, widely available dataset solves at least one aspect of the reproducibility issue, in that all models can at least be trained in the same environment, using the same input features and labels.

Motivated by the lack of a sufficient dataset, the present work aims to develop a set of RANS simulations of highly resolved reference cases in order to generate a curated dataset[43]. In this work, the numerical methods for the RANS simulations are presented, along with the selection and calculation of the input features for machine learning models. In doing so, the present work aims to present a large computational dataset, curated and logically structured for immediate use in developing next-generation turbulence closure models for RANS using data-driven machine learning. Table 2.1 summarizes the inputs and outputs of the present work.

Table 2.1: Inputs and outputs of the present study.

Inputs		Outputs
Present work	Previous work	A set of **features** and **labels** for developing models
Numerical settings (e.g. schemes, grids) for generating coarsely-resolved flow fields	Highly-resolved flow fields, suitable for use as "truth" values in machine learning	which map the coarse variables to highly-resolved variables **Features**: Coarsely-resolved flow fields with curated machine learning input features **Labels**: Highly-resolved flow fields, mapped onto the coarse grid, with curated machine learning labels

Table 2.2: Cases in the dataset. Re_L is the Reynolds number based on the characteristic length and velocity scales shown in Figures 2.1 to 2.7.

Flow case	Ref.	Re_L	Dim.	Num. cases	Parameter
Periodic hills	[44]	$5,600$	2D	5	Steepness
Square duct	[45]	1,100–3,500	3D	16	Re
Parametric bump	[46]	13,260–27,850	2D	5	Bump height
Converging-diverging channel	[47, 48]	12,600–20,580	2D	2	Re
Curved backward-facing step	[49]	13,700	2D	1	-

2.2 Methods

2.2.1 Selection of reference cases

An important aspect of dataset selection for data-driven turbulence modelling is sweeping of a parameter space. A deep insight followed by a deeper understanding of the fluid phenomena can be obtained by providing information on how the geometry and/or the Reynolds number changes the flow behaviour. In contrast, single-point measurements are only valuable in approximating a universal mapping between inputs and outputs. The majority of the datasets used here involve sweeping through some parameter space. Table 2.2 summarizes the cases used in the dataset.

Table 2.3: Kinematic molecular viscosity used for each case.

Flow case	ν (m^2/s)
Periodic hills	$5(10)^{-6}$
Square duct	$0.241/Re_H$
Parametric bump	$2.529(10)^{-5}$
Converging-diverging channel	$1/Re_H$
Curved backward-facing step	$7.3(10)^{-5}$

2.2.2 Computational method

The flow is assumed to be incompressible, viscous, steady, and turbulent for all cases. Under these conditions, the fluid properties are specified by the kinematic molecular viscosity ν. Table 2.3 summarizes the viscosity used for each case.

The open-source library OpenFOAM v2006 [50] was used to generate the dataset. The ability to replicate CFD is greatly improved by supplying the mesh and settings files [41]. The dataset includes the OpenFOAM case files, including the meshes used for all the cases, and the full details of the settings used. Supplying the OpenFOAM files also reduces the effort required for *a posteriori* testing. This practice is following Xiao et al. [44], who included the OpenFOAM files with their dataset. While this section highlights the basic numerical settings used, the reader is referred to the dataset for the complete OpenFOAM settings.

Numerical schemes

A standardized set of numerical schemes was used for all cases. The numerical schemes represent commonly used RANS schemes, which represent a good trade-off between stability and accuracy. For discretizing the convective terms in the momentum equations, a second-order upwind scheme was used. For discretizing convective terms in the turbulence transport equations, a first-order upwind scheme was used. For the diffusive terms, a second-order central difference scheme was used. Since all the flow cases are steady, the transient terms were set to zero.

The simpleFoam solver was used to solve the equations iteratively. The semi-implicit method for pressure-linked equations-consistent (SIMPLEC) algorithm was used to accelerate convergence. For some cases, additional non-orthogonality correcting loops were applied to the pressure equation. The generalized geometric algebraic multigrid (GAMG)

solver was used for the pressure equation, and the preconditioned bi-conjugate gradient (PBiCGStab) solver was used for all other equations.

Iterative residual convergence below 10^{-6} was generally achieved, with most simulations converging below 10^{-8}. The residual plots for each simulation are provided along with the dataset. The exceptions to this tight residual convergence criteria are the U_y U_z, and p fields for the square duct cases. The linear eddy viscosity model is unable to accurately predict the secondary vortices resulting from non-zero U_y and U_z components in the square duct case, and therefore minimal convergence is seen in these residuals as the in-plane velocity fields remain close to the initial condition of zero. The pressure field for the square duct case does not converge below 10^{-6} due to the presence of a forcing term which maintains the bulk velocity, resulting in uniform streamwise zero pressure equal to the initial condition of zero.

Turbulence modelling

The two most common families of turbulence closure models, k-ε and k-ω, include many sub-models. Previous investigations on machine-learnt models for predicting the anisotropy tensor have augmented the standard k-ε model[23], the Launder-Sharma low Reynolds number k-ε model [34], and the k-ω model [33, 35]. Four representative turbulence models were selected for the dataset: namely, the standard k-ε [51], k-ε-ϕ_t-f [52], k-ω [31], and the k-ω shear stress transport (SST) [53] turbulence closure models. In this work, ϕ_t is used to denote the anisotropy measure $\overline{v'^2}/k$ to align with the variable naming in OpenFOAM. Here, $\overline{v'^2}$ denotes the wall-normal Reynolds stress. The default coefficients were used for all turbulence models [50].

The k-ε-ϕ_t-f model is a more sophisticated model than the k-ε and k-ω models, through the inclusion of an additional transport equation for the anisotropy measure $\phi_t \equiv \overline{v'^2}/k$, and an elliptic equation for f. f is a scalar which predicts TKE redistribution from the streamwise to the wall-normal Reynolds stress. This model is an improved version of the original $\overline{v'^2}$-f model proposed by Durbin [54], and the improved "code-friendly" version developed by Lien and Kalitzin [55]. The additional quantities enable the creation of new input features not available in the previous two-equation investigations. Both additional scalars satisfy all desired invariance properties, including Galilean invariance.

For all turbulence models, the mesh was sufficient for a low Reynolds number wall treatment. Low Reynolds number wall boundary conditions are provided for k, ε, and ω in OpenFOAM [56]. A fixed-value $k = 0$ boundary condition was applied at no-slip walls. At no-slip walls, the following low Reynolds number fixed value boundary condition was

Table 2.4: Units for each variable requiring boundary conditions.

Description	Field	Units
Velocity	$\vec{U}$	m/s
Kinematic pressure	p	m^2/s^2
Turbulent kinetic energy	k	m^2/s^2
TKE dissipation rate	ε	m^2/s^3
TKE specific dissipation rate	ω	s^{-1}
Anisotropy measure	ϕ_t	-
TKE redistribution scalar	f	s^{-1}

applied for ε:

$$\varepsilon = \varepsilon_{vis} = 2wk\frac{\nu}{y^2} \ , \tag{2.1}$$

where w are the cell corner weights[50]. For ω the following fixed value boundary condition was applied at no-slip walls:

$$\omega = \frac{6\nu}{\beta_1 y^2} \ , \tag{2.2}$$

where $\beta_1 = 0.075$.

Domain and boundary conditions

The domain and boundary conditions for all cases were selected to match the DNS or LES reference simulations. There are two main types of boundary conditions used in the dataset: fixed-free, and streamwise cyclic. While the periodic hills and square duct cases utilize a streamwise cyclic boundary condition, the bump, converging-diverging channel, and curved-backward facing step cases employ a fully-developed inlet velocity profile, and a zero-gradient outlet. The simulations here involve four different turbulence models, each with different fields. The units used for each variable are given in Table 2.4.

2.2.3 Mesh

OpenFOAM's utilities were used to generate the meshes. The mesh generation method varied from case to case, as some cases have changing geometries. Table 2.5 summarizes the meshes used. All meshes met the low Reynolds number wall treatment criterion of

Table 2.5: Meshes used for discretizing the domain.

Case	Dim.	Mesh type	N	Generation method
Periodic hills	2D	Structured hexahedral	14,751	Provided by Xiao et al. [44]
Square duct	3D	Structured hexahedral	691,300	`blockMesh`[50]
Parametric bump	2D	Structured hexahedral	72,100	`blockMesh`[50]
Converging-diverging channel	2D	Structured hexahedral	183,750	`blockMesh`[50]
Curved backward-facing step	2D	Unstructured hexahedral dominant	37,082	`snappyHexMesh`[50]

$y^+ \approx 1$ or below. Here, $y^+ \equiv u_\tau y_w/\nu$ is the normalized wall-normal distance, where y_w is the wall-normal distance, and u_τ is the wall friction velocity. In all cases, the mesh was either hexahedral or hexahedral-dominant. A high-quality mesh is important for generating input features for machine learning, in that some terms are sensitive to the mesh quality. For example, the basis tensor $\hat{\mathcal{T}}_{10}$ in a general representation of the Reynolds stress tensor proposed by Pope [28] is fifth order in terms of the velocity gradient tensor. In developing the feature set here, we found that to keep these terms stable, the number of tetrahedral cells in the domain must be minimized. However, many industrial meshes contain tetrahedral cells, and are of poorer quality than the structured meshes generated here. While CFD results are normally sensitive to the mesh used, machine learning models are especially sensitive to the mesh quality. Poorer meshes result in increased noise and more outliers in the input feature set.

2.2.4 Periodic hills

Flow over periodic hills with cyclic boundary conditions is a common benchmark problem for turbulence modelling. The periodic hills case features separation, an important phenomenon for RANS models to accurately capture due to the prominence of strongly separated flows in many industrial settings. To provide a parameterized dataset for data-driven turbulence modelling, Xiao et al. [44] performed DNS of flow over a series of periodic hills. This dataset consists of five cases, characterized by the steepness ratio α. The values of α selected are $\alpha = 0.5, 0.8, 1.0, 1.2$, and 1.5, which results in a range of separated flows.

The geometry for the five periodic hills cases is shown in Figure 2.1. The Reynolds number based on bulk velocity and crest height for all cases is fixed at $Re = 5,600$.

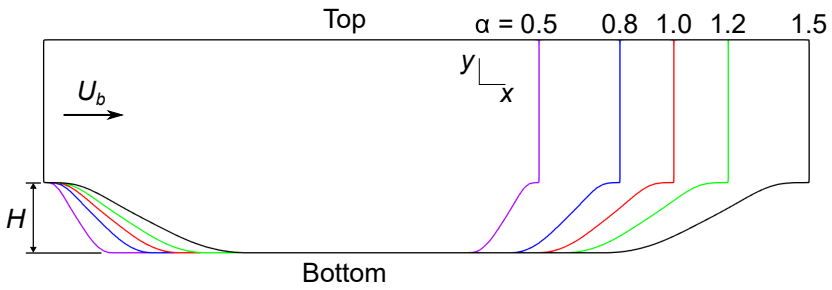


Figure 2.1: The geometry for the five periodic hills cases. Further detail is given in Xiao et al. [44]. The Reynolds number for this case is calculated based on the hill height H and mean bulk velocity U_b. These parameters are fixed for all cases, so Re_H remains fixed at 5,600.

The periodic hills case is a two-dimensional (2D) flow, with the domain geometry characterized in terms of the hill height H, as shown in Figure 2.1. The domain height is fixed at $3.04H$, and the domain width changes from $7.07H$ to $10.9H$, as the parameter α changes. The boundary conditions for the periodic hills case are streamwise cyclic for all flow variables. Both the top and bottom boundaries are treated as no-slip walls. To maintain a constant bulk velocity in the flow, a mean pressure gradient source term is added to the momentum equation. Therefore, the pressure field for cases with cyclic boundary conditions should be interpreted as the deviation from the mean pressure field.

The mesh for the steepest periodic hills case ($\alpha = 0.5$) is shown in Figure 2.2. The RANS meshes for all periodic hills cases were provided by Xiao et al. [44]. The periodic hills mesh is a structured mesh, with cells concentrated near the boundary layer. While the geometry changes by varying the hill steepness and domain length, the number of cells for all cases is the same.

2.2.5 Square duct

The DNS dataset for flow in a square duct by Pinelli et al. [45] has been widely used in data-driven turbulence modelling. This dataset consists of 16 cases, all with the same

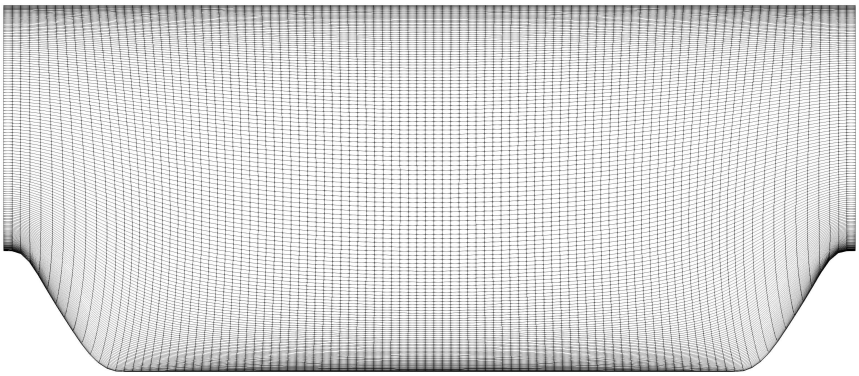


Figure 2.2: Structured hexahedral mesh used to discretize the $\alpha = 0.5$ periodic hills case.

fixed geometry shown in Figure 2.3. The Reynolds number based on the duct half-width varies between 1,100 and 3,500. The flow in a square duct is a challenging test case for eddy viscosity models. Linear eddy viscosity models are unable to predict the secondary corner vortices which form in the duct. These structures are Prandtl's secondary motion of the second kind [45]. The dataset contains the mean velocities and Reynolds stresses in a cross-section of the duct. The inclusion of this dataset allows the machine-learnt model to incorporate the Reynolds number dependence of these challenging secondary motions, from the transitional to the fully turbulent regimes. Additionally, it is the only three-dimensional (3D) flow in the dataset, for which the Reynolds shear stresses $\overline{u'w'}$ and $\overline{v'w'}$ are nonzero.

The geometry for the square duct is shown in Figure 2.3. The dimensions for this 3D case are given in terms of the duct half-width H. The duct is a $2H \times 2H \times 5H$ box. Wall boundary conditions were applied for the top, bottom and sides of the duct. The streamwise cyclic boundary conditions for the square duct case are summarized in Table 2.6. A mean pressure gradient source term was added to the momentum equation, to maintain a constant bulk velocity.

The mesh for the square duct case is shown in Figure 2.3. This mesh is also structured. Cells are concentrated near the boundary layer. The mesh for all square duct cases is identical. The $y^+ \leq 1$ criterion was verified for the highest Reynolds number flow case. The mesh is 3D, with the dataset for machine learning being generated using a cross-section of the mesh.

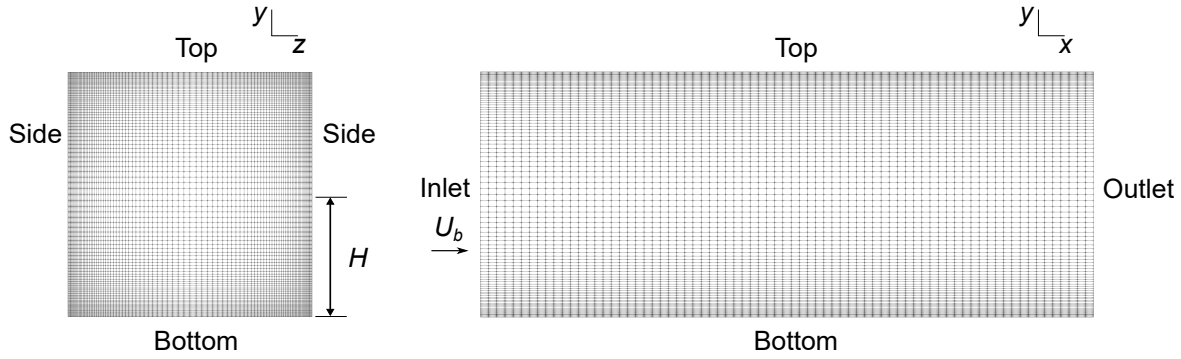


Figure 2.3: The mesh and geometry for the square duct cases. The cases vary by changing the Reynolds number from 1,100 to 3,500, which is calculated based on the duct half-width H and mean bulk velocity U_b.

Table 2.6: Boundary conditions for the periodic hills and square duct cases.

	Inlet	Outlet	Walls
$\vec{U}$	Cyclic	Cyclic	$\vec{U} = 0$
p	Cyclic	Cyclic	Zero-gradient
k	Cyclic	Cyclic	$k = 0$
ε	Cyclic	Cyclic	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	Cyclic	Cyclic	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Cyclic	Cyclic	$\phi_t = 0$
f	Cyclic	Cyclic	$f = 0$

2.2.6 Parametric bumps

The LES dataset for flow over a family of bumps by Matai and Durbin [46] has been recently made available for data-driven closures. The bump is a circular arc, with convex fillets on either end. The dataset is characterized by the bump height h, which is the highest point of the circular arc as shown in Figure 2.4. The Reynolds number based on momentum thickness and inlet free stream velocity U_∞ is fixed at $Re_\theta = 2,500$, while the Reynolds number based on bump height and U_∞ varies from $Re_h \approx 13,250$ to $27,850$. At $h = 20$ mm, the flow remains attached along the bump, while increasing the height further results in slight separation at $h = 26$ mm. For the highest bump corresponding to $h = 42$ mm, a small separated region forms behind the bump. While the periodic hills dataset features massively separated flows, the bump cases incorporate a smaller degree of separation. Matai and Durbin found that the mild separation causes a high turbulent

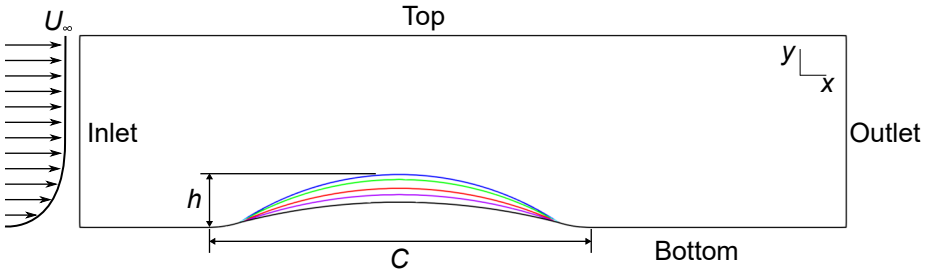


Figure 2.4: The geometry for the five parametric bump cases. The bump length C is fixed at 305 mm, and the bump height varies as $h = 20$, 26, 31, 38, and 42 mm. Further detail is given in Matai and Durbin [46]. The Reynolds number based on maximum inlet velocity and step height varies from $Re_h = 13,260$ to $Re_h = 27,850$, with the momentum thickness Reynolds number fixed at $Re_\theta = 2,500$.

kinetic energy (TKE) zone to depart from the bump ahead of the separated region, which is not the case for massively separated flows. Matai and Durbin attributed this region to the adverse pressure gradient generating a mean shear profile. Another important effect captured in the parametric bump case is strong disequilibrium. The parametric bump dataset is highly valuable for training machine-learnt closure models due to the high Reynolds number, parametrically sweeping geometry, physics unique to mildly separated flows, and strong disequilibrium.

For the parametric bumps, the DNS and LES simulations utilized a fully-developed inlet flow generated by a "feeder" simulation. To generate the RANS inlet condition, a similar approach to the DNS and LES was taken: a flat version of the domain was simulated with fixed-free boundary conditions to allow the flow to fully develop before entering the domain of interest. Equations for isotropic turbulence are commonly used to estimate the RANS boundary conditions for fixed turbulence inlets. For the feeder simulations, the following equations were used to estimate turbulence quantities at the inlet:

$$k = \frac{3}{2}(UI)^2 \ , \tag{2.3}$$

$$\varepsilon = C_\mu^{3/4} \frac{k^{3/2}}{L_t} \ , \tag{2.4}$$

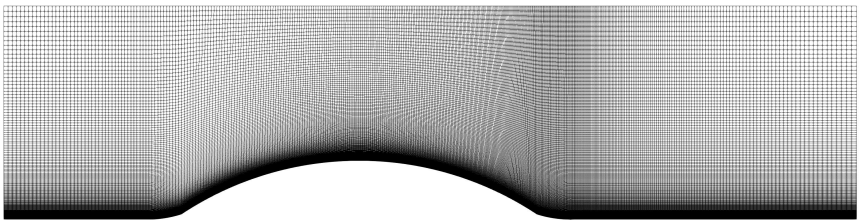


Figure 2.5: Structured hexahedral mesh used to discretize the $h = 42$ mm parametric bump case.

$$\omega = \frac{\varepsilon}{0.09k} \ , \tag{2.5}$$

where I is the turbulent intensity, L_t is the turbulence length scale and C_μ is a turbulence closure coefficient.

The parametric bumps case is unique in this dataset in that the top boundary is zero-gradient, compared to the walls used in the other cases. The inlet free-stream velocity U_∞ for the LES reference simulation was 16.77 m/s. To recreate these conditions, the inlet boundary conditions for the flat cases were adjusted to produce $U_\infty = 16.77$ m/s. It should be noted that this is an approximation of the LES inlet condition, because the four different turbulence models all produce different U_∞. For the dataset, the mean velocity used for all turbulence models was the same (Table 2.7), so that the boundary conditions are comparable between turbulence models. The boundary conditions for generating a fully-developed inlet profile for the bump case are summarized in Table 2.7. After generating a fully-developed profile, the U, k, ε, and ω fields were used as fixed-value inlet conditions for the bump cases. The boundary conditions for the bump cases are summarized in Table 2.8. The domain size for the parametric bump set is fixed at $2C \times 0.5C$.

The parametric bump mesh is shown in Figure 2.5, and the converging-diverging channel mesh is shown in Figure 2.6. Both cases use a structured mesh over an obstruction in the flow. Cells are concentrated in the wake region, and the boundary layer. For the parametric bump, the changing geometry was created by adjusting the bump profile in the structured mesh generator, which resulted in the same number of cells for all cases. The mesh shown in Figure 2.5 is for the highest bump. For the converging-diverging channel, the mesh density for both Reynolds numbers is identical, with the $Re = 20,580$ having an extended domain, and therefore more cells.

Table 2.7: Boundary conditions for the flat developing flow case, used to generate an inlet profile for the bump cases.

	Inlet	Outlet	Top	Bottom
$\vec{U}$	$\vec{U} = (16.683,\ 0,\ 0)$	Zero-gradient	Zero-gradient	$\vec{U} = 0$
p	Zero-gradient	$p = 0$	Zero-gradient	Zero-gradient
k	$k = 0.16699 (I = 2\%)$	Zero-gradient	Zero-gradient	$k = 0$
ε	$\varepsilon = 0.266974$ ($L_t = 42$ mm)	Zero-gradient	Zero-gradient	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	$\omega = 17.764$	Zero-gradient	Zero-gradient	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Zero-gradient	Zero-gradient	Zero-gradient	$\phi_t = 0$
f	Zero-gradient	Zero-gradient	Zero-gradient	$f = 0$

Table 2.8: Boundary conditions for the bump cases.

	Inlet	Outlet	Top	Bottom
$\vec{U}$	Fully-developed, $U_\infty = 16.77$ m/s	Zero-gradient	Zero-gradient	$\vec{U} = 0$
p	Zero-gradient	$p = 0$	Zero-gradient	Zero-gradient
k	Fully-developed	Zero-gradient	Zero-gradient	$k = 0$
ε	Fully-developed	Zero-gradient	Zero-gradient	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	Fully-developed	Zero-gradient	Zero-gradient	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Zero-gradient	Zero-gradient	Zero-gradient	$\phi_t = 0$
f	Zero-gradient	Zero-gradient	Zero-gradient	$f = 0$

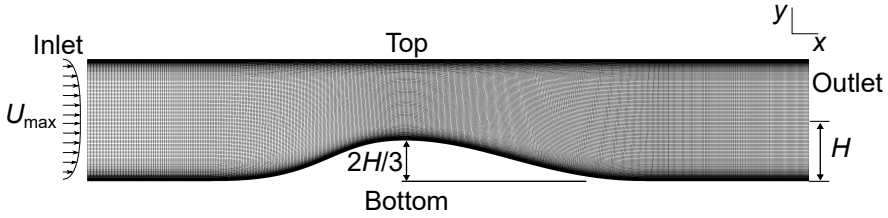


Figure 2.6: The mesh and geometry for the two converging-diverging channel cases, corresponding to Reynolds numbers of $Re_H = 12,600$ and 20,580. The Reynolds number for these two cases is based on the channel half height H and the maximum inlet velocity $U_{\max}$. The $Re_H = 12,600$ converging-diverging channel case uses a smaller domain than the $Re_H = 20,580$ case, but with an identical mesh.

2.2.7 Converging-diverging channel

Two datasets are available for flow over an identical converging-diverging geometry at $Re_H = 12,600$ and $Re_H = 20,580$, shown in Figure 2.6. The Reynolds number for this case is based on the maximum inlet velocity and the channel half-height H. The lower Reynolds number dataset comes from the DNS by Laval and Marquielle [47], and Marquillie et al. [57]. The higher Reynolds number dataset was generated by Schiavo et al. [48] using LES. The bump height is approximately $2H/3$. A fully developed internal channel flow enters the domain and impinges on the abrupt upstream side of the bump. The flow accelerates as the channel converges, then decelerates over the gradual downstream side of the bump. At $Re_H = 12,600$, a thin separation bubble forms along the downstream slope. Along the flat upper wall, the flow remains attached but on the cusp of separation. At $Re_H = 20,580$, the separation bubble grows. The cases contain valuable information about the Reynolds number effect on separation, reattachment, and development of a turbulent boundary layer under an adverse pressure gradient. The long domain downstream of the bump for $Re_H = 20,580$ effectively provides an additional set of LES information for developing plane channel flow.

A similar procedure for the bump case was completed to generate inlet conditions for the converging-diverging channel. However, for the converging-diverging channel, the top boundary is a wall. The boundary conditions for the converging-diverging channel case were adjusted to produce a maximum velocity of $U_{\max} = 1.0$ m/s, to match the reference simulations. The boundary conditions for the flat, developing flow case is shown in Table 2.9, and the boundary conditions for the cases in the data set are shown in Table 2.10. The domain size for the $Re_H = 12,600$ converging-diverging channel is $12.6H \times 2H$, while for $Re_H = 20,580$ the domain is enlarged to $25.3H \times 2H$ by extending the outlet length.

Table 2.9: Boundary conditions for the flat developing flow case, used to generate an inlet profile for the converging-diverging channel cases.

	Inlet	Outlet	Top	Bottom
$\vec{U}$	$\vec{U} = (0.845, 0, 0)$	Zero-gradient	$\vec{U} = 0$	$\vec{U} = 0$
p	Zero-gradient	$p = 0$	Zero-gradient	Zero-gradient
k	$k = 4.28421(10)^{-4}(I = 2\%)$	Zero-gradient	$k = 0$	$k = 0$
ε	$\varepsilon = 1.0408(10)^{-5}$ $(L_t = 0.07H_{\text{chan}})$	Zero-gradient	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	$\omega = 0.26993$	Zero-gradient	$\omega = 6\nu/(\beta_1 y^2)$	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Zero-gradient	Zero-gradient	$\phi_t = 0$	$\phi_t = 0$
f	Zero-gradient	Zero-gradient	$f = 0$	$f = 0$

Table 2.10: Boundary conditions for the converging-diverging channel cases.

	Inlet	Outlet	Top	Bottom
$\vec{U}$	Fully-developed, $U_{\max} = 1.0$ m/s	Zero-gradient	$\vec{U} = 0$	$\vec{U} = 0$
p	Zero-gradient	$p = 0$	Zero-gradient	Zero-gradient
k	Fully-developed	Zero-gradient	$k = 0$	$k = 0$
ε	Fully-developed	Zero-gradient	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	Fully-developed	Zero-gradient	$\omega = 6\nu/(\beta_1 y^2)$	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Zero-gradient	Zero-gradient	$\phi_t = 0$	$\phi_t = 0$
f	Zero-gradient	Zero-gradient	$f = 0$	$f = 0$

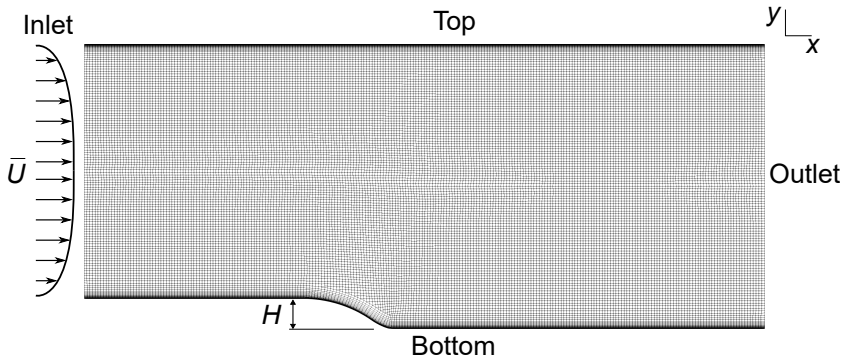


Figure 2.7: The geometry and mesh for the curved backward-facing step case. The Reynolds number $Re_H = 13,700$ is based on the mean inlet velocity $\overline{U}$ and the step height H.

2.2.8 Curved backward-facing step

The curved backward-facing step case simulated by Bentaleb et al. [49] using LES was also included in the dataset. The geometry for this case is shown in Figure 2.7. While this is the only case that does not feature parametric variation, it contains an additional set of data on separation and reattachment. While other cases in the dataset feature separation after an acceleration of the flow, the curved backward-facing step case features separation of a fully developed turbulent boundary layer. This phenomenon is difficult for RANS models to predict, and therefore the LES results were included in the dataset. While the original work by Bentaleb et al. [49] defined a Reynolds number based on the maximum inlet velocity, we found that the large channel height meant that the mean velocity for all turbulence models was within 10% of the maximum velocity, so to approximate the reference case, defining the Reynolds number for the dataset based on the mean inlet velocity was sufficient. The top and bottom boundaries are walls. An identical procedure to the converging-diverging channel case was used to develop the inlet boundary condition, with $U_{\max} = 1.0$ m/s. The curved backward-facing step domain is $22.7H \times 9.48H$. The boundary conditions for the flat, developing flow case is shown in Table 2.11, and the boundary conditions for the cases in the data set are shown in Table 2.12.

The only unstructured mesh in the dataset is the curved backward-facing step, shown in Figure 2.7. While it was feasible to generate a structured mesh for this case, an unstructured mesh was generated to include some more typical industrial cells into the dataset. Specifically, near the backward-facing step, the mesh transitions out of the inflation layer using some triangular cells.

Table 2.11: Boundary conditions for the flat developing flow case, used to generate an inlet profile for the curved backward facing step cases.

	Inlet	Outlet	Top	Bottom
$\vec{U}$	$\vec{U} = (1.0,\ 0,\ 0)$	Zero-gradient	$\vec{U} = 0$	$\vec{U} = 0$
p	Zero-gradient	$p = 0$	Zero-gradient	Zero-gradient
k	$k = 6.00(10)^{-4} (I = 2\%)$	Zero-gradient	$k = 0$	$k = 0$
ε	$\varepsilon = 2.415(10)^{-6}\ (L_t = H)$	Zero-gradient	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	$\omega = 4.472(10)^{-2}$	Zero-gradient	$\omega = 6\nu/(\beta_1 y^2)$	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Zero-gradient	Zero-gradient	$\phi_t = 0$	$\phi_t = 0$
f	Zero-gradient	Zero-gradient	$f = 0$	$f = 0$

Table 2.12: Boundary conditions for curved backward facing step case.

	Inlet	Outlet	Top	Bottom
$\vec{U}$	Fully-developed, $\overline{U} = 1.0$ m/s	Zero-gradient	$\vec{U} = 0$	$\vec{U} = 0$
p	Zero-gradient	$p = 0$	Zero-gradient	Zero-gradient
k	Fully-developed	Zero-gradient	$k = 0$	$k = 0$
ε	Fully-developed	Zero-gradient	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$	$\varepsilon = \varepsilon_{vis} = 2wk\nu/y^2$
ω	Fully-developed	Zero-gradient	$\omega = 6\nu/(\beta_1 y^2)$	$\omega = 6\nu/(\beta_1 y^2)$
ϕ_t	Zero-gradient	Zero-gradient	$\phi_t = 0$	$\phi_t = 0$
f	Zero-gradient	Zero-gradient	$f = 0$	$f = 0$

2.3 Data Records

The dataset [43] is hosted on Kaggle, a common platform for machine learning. A total of 29 simulations (Table 2.2) per turbulence model were completed to match the reference data. The DNS or LES reference data were interpolated onto the RANS grid, using linear interpolation. Any points which required extrapolation of the reference data were dropped, and the interpolated reference data were checked for realizability using the criteria from Banerjee et al. [58]. After interpolation and data quality checks, 895,640 points of RANS data paired with corresponding DNS or LES data are available for each turbulence model. Each data point represents a cell of a RANS simulation in the dataset, with the corresponding DNS/LES quantity interpolated onto the RANS cell. At each data point, the full set of base, derived, and labels fields described in the following sections are provided. A separate file is provided for each field, for each RANS simulation. An explanation of the file naming convention is provided with the dataset.

To maximize the usefulness of the dataset, a comprehensive set of input features and labels was generated. The dataset is organized into two types of data: base variables, and derived quantities provided for convenience. The base variables contain the bare minimum fields that need to be provided to construct the rest of the fields, which are the RANS fields and grid points. The available base fields in the dataset are summarized in Table 2.13, and the derived fields are summarized in Tables 2.14 and 2.15.

Table 2.14: Derived feature fields available in the dataset. For the definition of ∇U, i is the row index, and j is the column index. All fields are derived based on cell center quantities for the collocated grid arrangement in OpenFOAM, which means that trace(∇U) may not be zero. The divergence-free velocity field imposed by the continuity equation is enforced at the cell faces, and Rhie-Chow interpolation[59] is used to handle pressure-velocity coupling on the collocated grid.

Quantity	Units	Symbol	Field name	Expression
Mean velocity gradient tensor	s^{-1}	∇U	`gradU`	$\frac{\partial U_i}{\partial x_j}$
Mean strain rate tensor	s^{-1}	S	`S`	$\frac{1}{2}\left(\nabla U + \nabla U^T\right)$

(continued on next page)

Table 2.14 (continued)

Quantity	Units	Symbol	Field name	Expression
Mean rotation rate tensor	s^{-1}	R	`R`	$\frac{1}{2}\left(\nabla U - \nabla U^T\right)$
Non-dimensional strain rate tensor	-	$\hat{S}$	`Shat`	$T_t S$
Non-dimensional rotation rate tensor	-	$\hat{R}$	`Rhat`	$T_t R$
TKE gradient vector	m/s^2	∇k	`gradk`	$\frac{\partial k}{\partial x_j}$
Pressure gradient vector	m/s^2	∇p	`gradp`	$\frac{\partial p}{\partial x_j}$
Antisymmetric tensor associated with ∇k	m/s^2	A_k	`Ak`	$\begin{bmatrix} 0 & -\partial_z k & \partial_y k \\ \partial_z k & 0 & -\partial_x k \\ -\partial_y k & \partial_x k & 0 \end{bmatrix}$
Antisymmetric tensor associated with ∇p	m/s^2	A_p	`Ap`	See A_k, replacing k with p
Non-dimensional A_k	-	$\hat{A}_k$	`Akhat`	$\frac{\sqrt{k} A_k}{\varepsilon}$
Non-dimensional A_p	-	$\hat{A}_p$	`Aphat`	$\frac{A_p}{\lvert \mathrm{D}U/\mathrm{D}t \rvert}$
Turbulent time scale	s	T_t	`T_t`	k/ε
Kolmogorov time scale	s	T_k	`T_k`	$\sqrt{\frac{\nu}{\varepsilon}}$
Pope's 10 basis tensors	-	$\hat{\mathcal{T}}_n$	`Tensors`	See Pope [28]

(continued on next page)

Table 2.14 (continued)

Quantity	Units	Symbol	Field name	Expression
Pope's 5 invariants of S and R	-	λ_i	`Lambda`	See Pope [28]
47 invariants of $\{\hat{S}, \hat{R}, \hat{A}_k, \hat{S}_p\}$ as used by Wu et al. [33]	-	$\mathcal{I}$	`I`	See Wu et al. [33]
Ratio of excess rotation to strain rate	-	-	`q[:,0]`	$\frac{\|\hat{R}\|^2 - \|\hat{S}\|^2}{2\|\hat{S}\|^2}$
Wall-distance based Reynolds number	-	-	`q[:,1]`	$\min\left(\frac{\sqrt{k}y_w}{50\nu}, 2\right)$
Ratio of turbulent time scale to mean strain time scale	-	-	`q[:,2]`	$\frac{k}{\varepsilon}\|S\|$
Ratio of total Reynolds stress to TKE	-	-	`q[:,3]`	$\frac{\|\overline{u_i' u_j'}\|}{k}$
Wall distance	m	y_w	`wallDistance`	-
Material derivative of velocity field (equal to convective derivative)	m/s^2	DU/Dt	`DUDt`	$U \cdot \nabla U$

The more useful portion of this dataset is the set of pre-constructed machine learning input features. The selection of input features is a critical area of ongoing research in machine-learnt turbulence models. The typical practice in machine learning Reynolds stress modelling is to derive a set of invariants from a tensor basis, combined with other invariant scalars. This was the approach used in [23, 33, 34, 44, 35] and others. While the input feature set varies, an effort has been made to provide sufficient fields in the dataset to conveniently reproduce past feature sets, and develop new ones. For example, all of the input features and labels used by Ling et al. [23] are directly provided: the five invariants of the mean strain and rotation rate tensor, the ten basis tensors described in Pope [28], and the anisotropy tensor labels.

Table 2.13: Base fields available in the dataset.

Quantity	Units	Symbol	Fieldname
Features from RANS			
x-coordinate	m	x	`x`
y-coordinate	m	y	`y`
z-coordinate	m	z	`z`
x velocity component	m/s	U_x	`Ux`
y velocity component	m/s	U_y	`Uy`
z velocity component	m/s	U_z	`Uz`
Kinematic pressure	m^2/s^2	p	`p`
Turbulent kinetic energy	m^2/s^2	k	`k`
TKE dissipation rate	m^2/s^3	ε	`epsilon`
TKE specific dissipation rate	s^{-1}	ω	`omega`
Anisotropy measure	-	ϕ_t	`phit`
TKE redistribution scalar	s^{-1}	f	`f`
Labels from DNS/LES			
x mean velocity component	m/s	$\overline{u}$	`um`
y mean velocity component	m/s	$\overline{v}$	`vm`
z mean velocity component	m/s	$\overline{w}$	`wm`
x Reynolds normal stress	m^2/s^2	$\overline{u'u'}$	`uu`
xy Reynolds shear stress	m^2/s^2	$\overline{u'v'}$	`uv`
xz Reynolds shear stress	m^2/s^2	$\overline{u'w'}$	`uw`
y Reynolds normal stress	m^2/s^2	$\overline{v'v'}$	`vv`
yz Reynolds shear stress	m^2/s^2	$\overline{v'w'}$	`vw`
z Reynolds normal stress	m^2/s^2	$\overline{w'w'}$	`ww`

Table 2.15: Derived label fields available in the dataset.

Quantity	Units	Symbol	Field name	Expression
Labels from DNS/LES				
Reynolds stress tensor	m^2/s^2	τ	`tau`	$\begin{bmatrix} \overline{u'^2} & \overline{u'v'} & \overline{u'w'} \\ \overline{u'v'} & \overline{v'^2} & \overline{v'w'} \\ \overline{u'w'} & \overline{v'w'} & \overline{w'^2} \end{bmatrix}$
Turbulent kinetic energy	m^2/s^2	k	`k`	$\frac{1}{2}\text{trace}(\tau)$
Non-dimensional anisotropy tensor	-	b	`b`	$\frac{\tau}{2k} - \frac{1}{3}I$

2.3.1 Labels

This dataset is suited for models that predict the Reynolds stress tensor, an equivalent problem to predicting the anisotropy tensor. The provided label set includes the individual Reynolds stress components (the base labels), and other fields that are sometimes more convenient to use. The Reynolds stress tensor, TKE, and anisotropy tenor are provided as ready-to-use labels.

2.3.2 Invariants of tensor bases

The invariants are derived from a set of basis tensors, which form a basis for the space spanned by a set of feature tensors. First, the feature tensors need to be selected. The selection of the feature tensors determines what flow variable gradients are incorporated into the model. Previous investigations have selected the set of feature tensors as $\{\hat{S}, \hat{R}\}$ [23], $\{\hat{S}, \hat{R}, \nabla k\}$ [34, 35], and $\{\hat{S}, \hat{R}, \nabla k, \nabla p\}$ [33]. If the feature tensors were directly employed as input features, the model would not be invariant because these inputs change with the coordinate system. Therefore, the procedure presented by Spencer and Rivlin [60] is commonly employed to generate a tensor basis for the feature set. After constructing the tensor basis, the invariants of the tensor basis are taken—in other words, the traces of the basis tensors are used as input features. This procedure guarantees that the model has the same invariance properties as the trace of the basis tensors.

The dataset includes several quantities which are convenient in generating tensor bases.

Along with the velocity gradient tensor ∇U, the strain rate and rotation rate tensors S, R are provided. While the strain and rotation rate tensors are provided without normalization, a set of pre-normalized strain and rotation rate tensors $\hat{S}, \hat{R}$ are provided, with the normalizations shown in Table 2.14. A similar set of features for the kinematic pressure and TKE gradients are provided. The gradients themselves, a vector quantity, and the associated antisymmetric tensors for both the un-normalized and normalized forms are provided.

The provided dataset is sufficient to form the most comprehensive tensor bases used to date, which is the 47 tensor basis used by Wu et al. [33]. However, it is the traces of these 47 tensors which are of interest. These 47 invariant traces are included in the dataset to be directly used as input features to a machine learning model. Also included is the set of 5 invariants (λ_i), which arise from using the strain and rotation rate as the feature tensors, as in Ling et al. [23].

2.3.3 Other input scalars

After gathering the set of tensor basis invariants, an additional set of scalars is added. Care must be taken that these scalars are invariant to not corrupt the invariance of the constructed tensor basis invariants. While many scalars have been proposed, many of them are not Galilean invariant, which is a property desired in machine-learnt turbulence models. Therefore, four Galilean invariant scalars used by Kaandorp and Dwight [35] are included as ready-to-use features in the dataset. While this set of input scalars is not comprehensive, the dataset includes sufficient fields to conveniently generate other scalar quantities.

2.4 Technical Validation

The RANS results are sensitive to the mesh used. While the mesh must be compatible with the selected wall treatment, it must also be sufficiently fine to reduce discretization errors. To demonstrate that the selected meshes do not affect the result, a mesh independence study was completed for each of the five flow cases. The most demanding case was selected for each flow type: the steepest periodic hills case, the highest Reynolds number square duct, the highest bump, the highest Reynolds number converging-diverging channel, and the curved backward-facing step. Mesh independence was demonstrated using the k-ε turbulence model. The mesh study was conducted by examining the change in the velocity fields between varying mesh sizes.

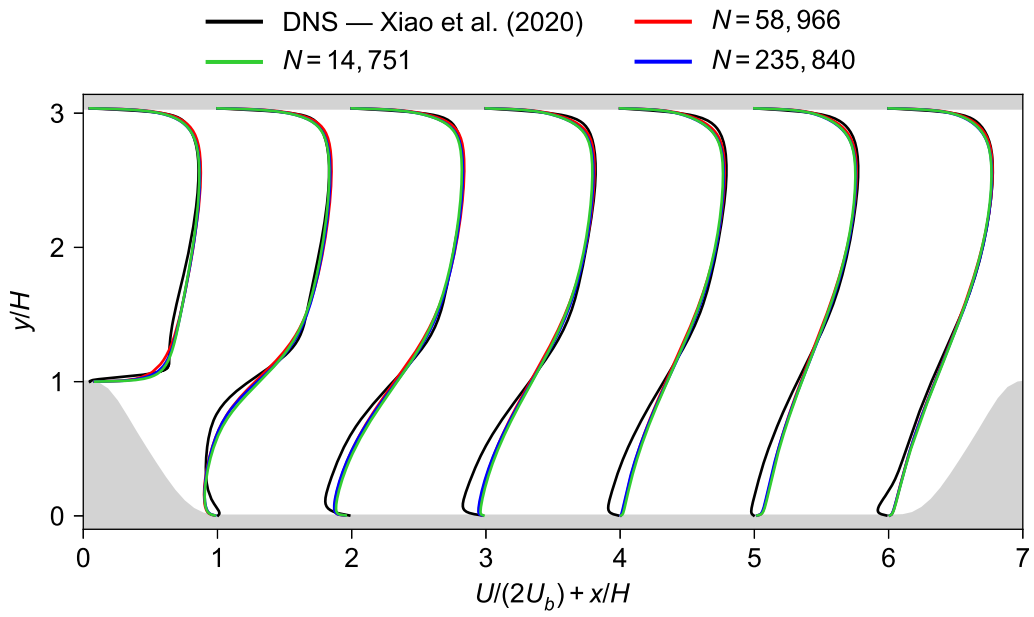


Figure 2.8: Profiles of U for three meshes of varying density for the $\alpha = 0.5$ periodic hills case.

Figures 2.8 and 2.9 show the results of the mesh convergence study for the periodic hills case. The meshes provided by Xiao et al. [44] were refined two times, each by a factor of 2 in the x and y directions. A small group of cells could not be refined while maintaining reasonable quality, which is why the meshes shown in Figures 2.8 and 2.9 do not exactly contain $N, 4N$, and $16N$ cells. The results for the periodic hills case demonstrate good mesh convergence for the grid with the smallest number of cells used in the study. There is almost no change for the U velocity for grids whose number of cells is greater than $N = 14,751$. The V profiles near the inlet boundary shown changes between the mesh sizes. For this case, the mesh convergence is non-monotonic, but the differences of the V profiles between the various meshes used are small. Therefore, the $N = 14,751$ mesh is sufficiently converged.

One of the main considerations for the square duct mesh is sufficient resolution in the $y-z$ plane to extract machine learning features. The reference data by Pinelli et al. [45] are provided as a set of statistics in the $y-z$ plane. Even though Figure 2.10 shows that the solution is mesh-converged at $N = 87,552$, the resolution in the $y-z$ plane is too coarse. The $N = 87,552$ mesh results in 2,304 dataset points per case, while the $N = 691,200$

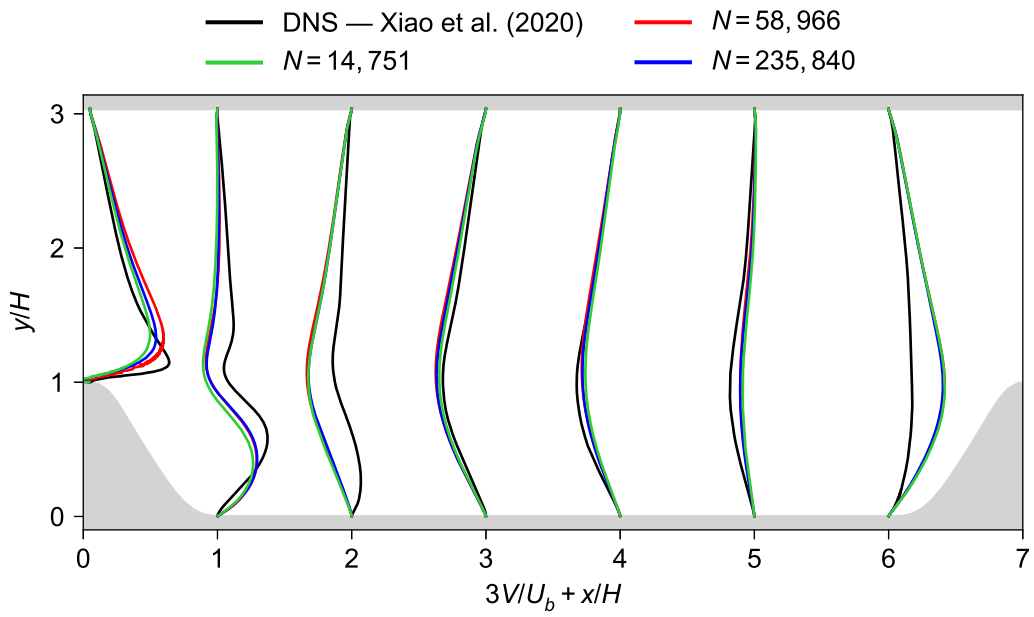


Figure 2.9: Profiles of V for three meshes of varying density for the $\alpha = 0.5$ periodic hills case.

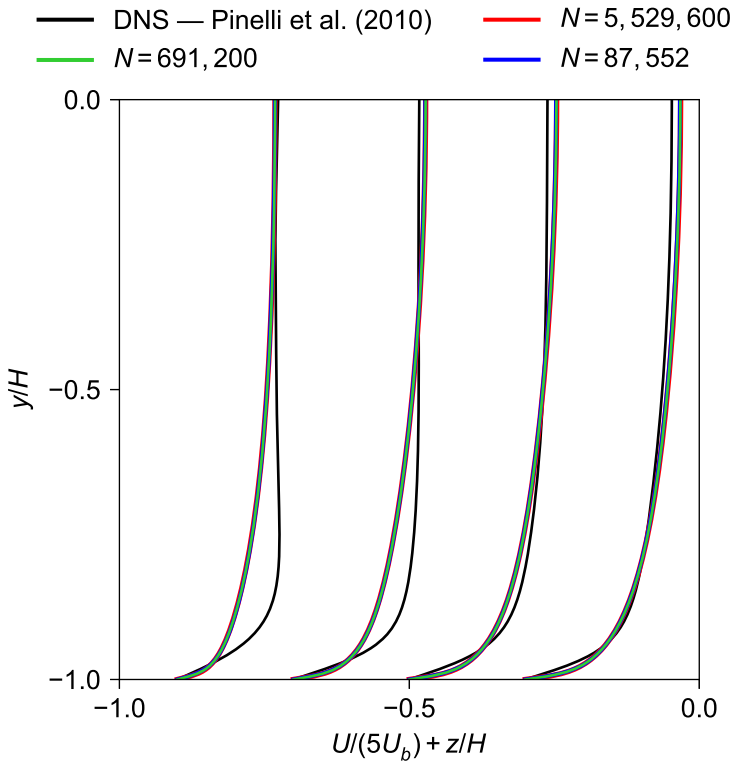


Figure 2.10: Profiles of U for three meshes of varying density for the $Re = 3,500$ square duct case.

mesh results in 9,216 points per case. Therefore, the $N = 691,200$ mesh is selected for generating the dataset, because the solution is mesh independent, and there are sufficient cells in the $y - z$ plane to generate features for machine learning.

The parametric bump is the highest Reynolds number flow in the dataset ($Re_H \approx 27,850$) and, as a consequence, it requires a dense mesh. Solution convergence at the coarsest mesh with $N = 72,100$ cells was demonstrated by increasing the number of cells in the structured mesh generator by a factor of two, and then four, and comparing the velocity profiles for the corresponding $N, 4N$, and $16N$ cases. Figures 2.11 and 2.12 show the comparisons made. For the U velocity profile, there are small differences in the wake of the bump, and in the far-field above the bump. The V velocity field reflects these small far-field differences above the bump. However, the differences are comparatively small, and the mesh demonstrates good convergence to generate the dataset.

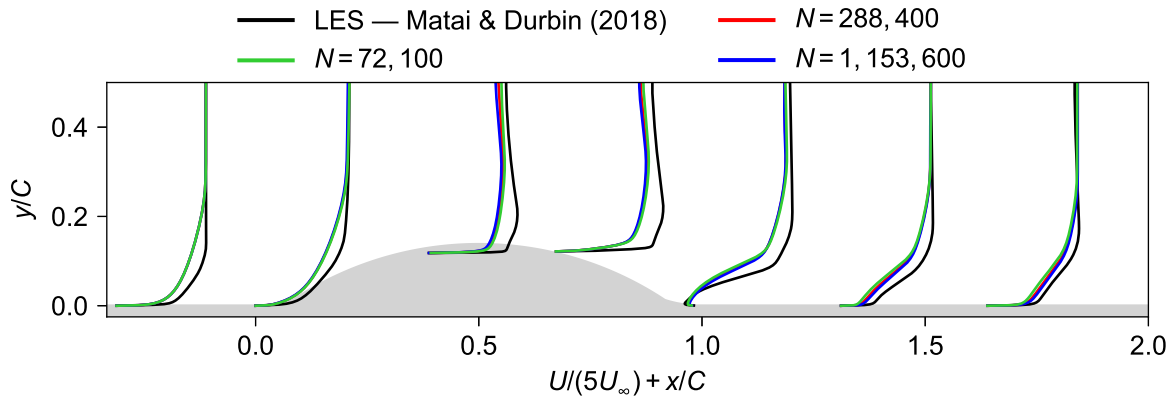


Figure 2.11: Profiles of U for three meshes of varying density for the $h = 42$ mm parametric bump case.

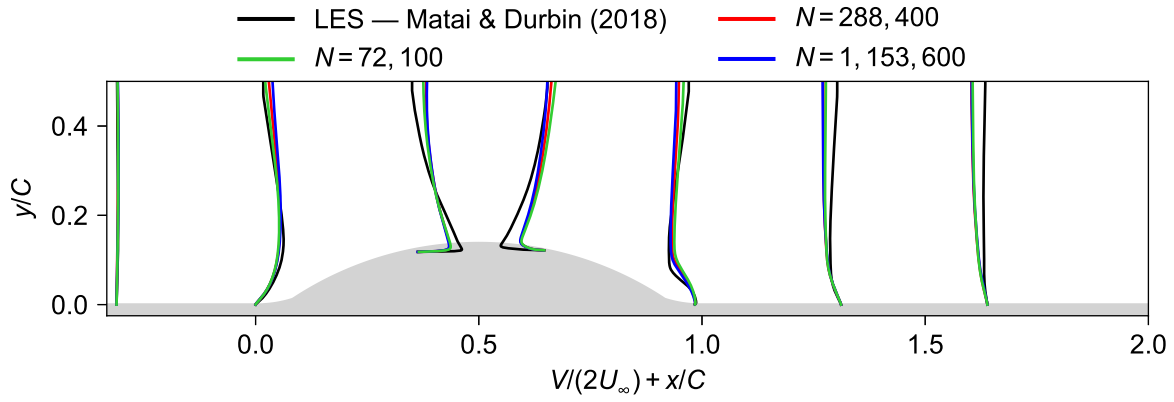


Figure 2.12: Profiles of V for three meshes of varying density for the $h = 42$ mm parametric bump case.

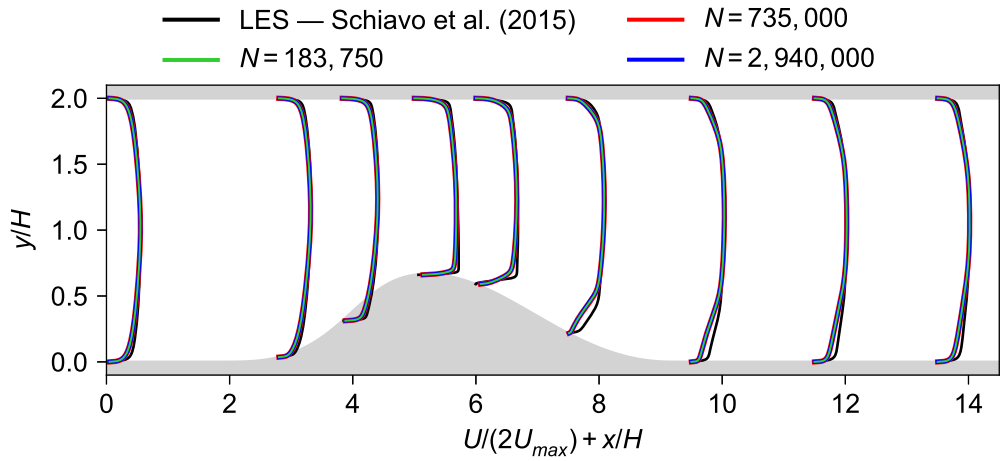


Figure 2.13: Profiles of U for three meshes of varying density for the $Re = 20,580$ converging-diverging channel case.

Mesh convergence for the converging-diverging channel case was demonstrated similarly to the bump case. The number of cells in the structured mesh generator was increased by a factor of two, and then four. Figures 2.13 and 2.14 show that there are almost no differences between the solutions as the mesh is refined, even by a factor of 16. Therefore, the mesh for the converging-diverging channel case is sufficiently converged at $N = 183,750$.

Demonstrating mesh convergence for the curved backward-facing step case was completed similarly to the periodic hills case, by refining the mesh twice in each direction. Some cells could not be refined while maintaining reasonable mesh quality, which is the reason that the meshes in Figures 2.15 and 2.16 do not exactly have N, $4N$, and $16N$ cells. The solution has excellent mesh convergence at $N = 37,082$, in both the U and V velocity fields.

2.5 Usage Notes

The dataset structure consists of a folder for each turbulence model, with an additional folder for the DNS/LES labels [43]. The RANS features for each case are provided using a consistent naming scheme. This structure allows the data to be accessed and processed in a coherent manner for immediate use in open-source machine learning frameworks such as TensorFlow and PyTorch. An example notebook of how to use the data to develop a

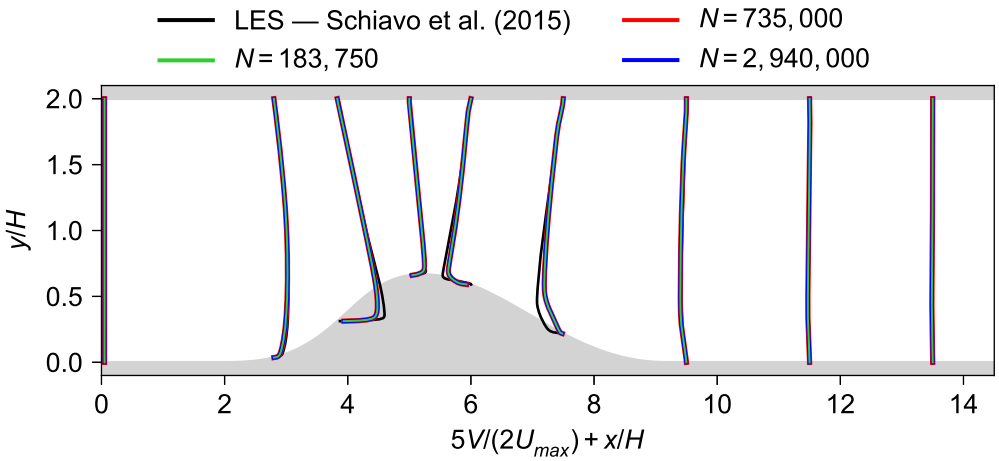


Figure 2.14: Profiles of V for three meshes of varying density for the $Re = 20,580$ converging-diverging channel case.

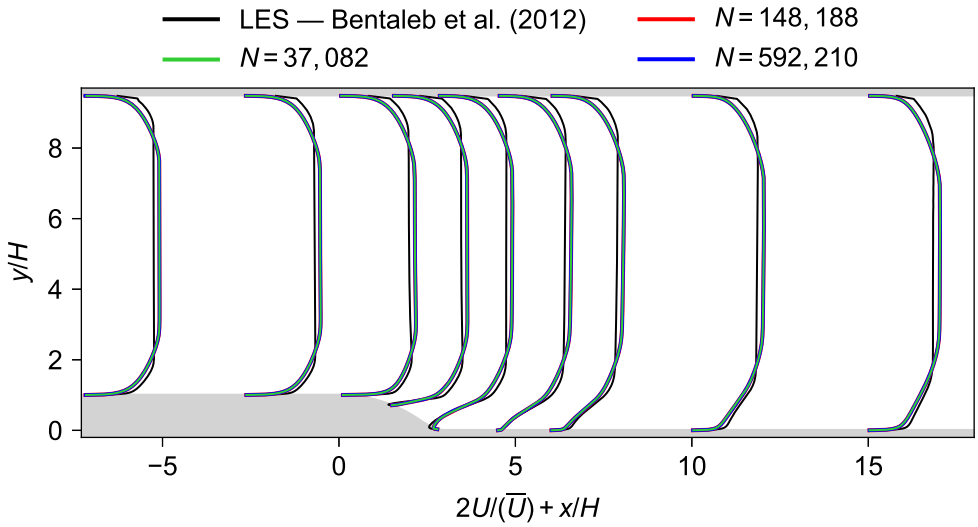


Figure 2.15: Profiles of U for three meshes of varying density for the curved backward-facing step case.

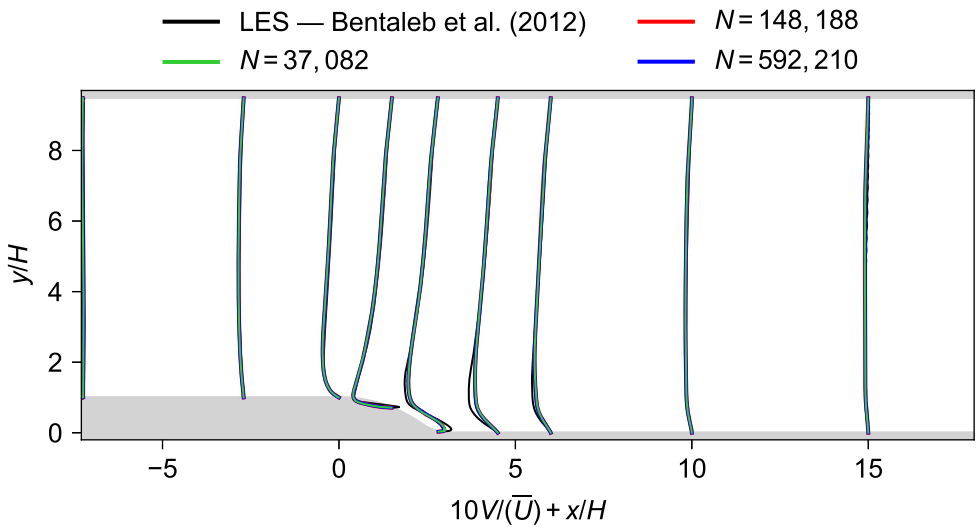


Figure 2.16: Profiles of V for three meshes of varying density for the curved backward-facing step case.

simple machine learning model for the Reynolds stress anisotropy tensor are provided on the dataset page. Another example notebook is provided that demonstrates the field formats, using the square duct case as an example. The dataset will be updated as more DNS/LES reference datasets become available, or if there is demand to include additional RANS turbulence models. The curated dataset is most suitable for direct use in corrective (open loop) RANS turbulence modelling using machine learning. While the dataset presented here is not targeted for iterative (closed loop) machine learning-based RANS turbulence modelling (Schmelzer et al. [21], Taghizadeh et al. [40]), it nevertheless can be used to provide the initial set of fields as well as to facilitate the implementation of the iterative approach for a particular RANS closure model (at least for the four turbulence closure models included in the dataset).

The dataset includes ready-to-use quantities, OpenFOAM files, and residual plots for all simulations. The ready-to-use input features are provided in the folders named by each turbulence model. The ready-to-use labels are provided in the `labels` folder. The `openfoam` folder provides the base quantities in OpenFOAM format, which is convenient for testing the corrective model. The `residuals` folder contains residual plots for all simulations.

There are approximately 1,000 fields per turbulence model, provided as `numpy` arrays.

The first index for all fields in the dataset is the data point index, equivalent to the cell index. The remaining indices in the array depends on the nature of the field. For example, all tensors are given with shape $(N, 3, 3)$, where N is the data point index. The ten basis tensors used in a general representation of the anisotropy tensor proposed by Pope [28] are given as an array with shape $(N, 10, 3, 3)$. Relatively few pre-processing steps have been performed on the dataset—no normalization or outlier elimination has been performed. The only deletions arise from a small subset (less than 50 points) of non-realizable LES label values, and any points requiring extrapolation of the reference data. Therefore, it is recommended that after a specific input feature set is formed using the provided fields, the input features should be standardized as is typical in machine learning. The RANS results also contain some outliers that may need to be dropped. For example, Kaandorp [34] dropped datapoints outside of $\mu \pm 5\sigma$, where μ is the mean, and σ is the standard deviation.

2.6 Code availability

Both the code used for generating this dataset and input files for the OpenFOAM simulations are available on the Kaggle page for this dataset [43]. The software used was OpenFOAM v2006, with all scripts written in Python 3.

Chapter 3

Deep structured neural networks for turbulence closure modelling

Associated reference:

R. McConkey, E. Yee, and F. S. Lien, "Deep structured neural networks for turbulence closure modeling", *Physics of Fluids* 34, 035110 (2022), https://doi.org/10.1063/5.0083074.

Abstract

Despite well-known limitations of Reynolds-averaged Navier-Stokes (RANS) simulations, this methodology remains the most widely used tool for predicting many turbulent flows, due to computational efficiency. Machine learning is a promising approach to improve the accuracy of RANS simulations. One major area of improvement is using machine learning models to represent the complex relationship between the mean flow field gradients and the Reynolds stress tensor. In the present work, modifications to improve the stability of previous optimal eddy viscosity approaches for RANS simulations are presented and evaluated. The optimal eddy viscosity is reformulated with a non-negativity constraint, which promotes numerical stability. We demonstrate that the new formulation of the optimal eddy viscosity improves the conditioning of the RANS equations for a periodic hills test case. To demonstrate the suitability of this proportional/orthogonal tensor decomposition for use in a physics-informed data-driven turbulence closure, we use two neural networks (structured on this specific tensor decomposition which is incorporated as an inductive bias into the network design) to predict the newly reformulated linear and non-linear parts of

the Reynolds stress tensor. Injecting these network model predictions for the Reynolds stresses into a RANS simulation improves predictions of the velocity field, even when compared to a sophisticated (state of the art) physics-based turbulence closure model. Finally, we apply SHAP (SHapley Additive exPlanations) values to obtain insights from the learned representation for the inner workings of the neural network used to predict the optimal eddy viscosity from the input feature data.

3.1 Introduction

In eddy viscosity modelling, the specific relationship between the Reynolds stress tensor and the mean velocity gradient tensor is the primary subject of interest. Many relationships have been proposed, from simple mixing length models to the sophisticated elliptic blending k-ε model. Despite the well-known limitations of the eddy viscosity models for turbulent flows, most industrial simulations still rely on these types of turbulence closures within the Reynolds-averaged Navier-Stokes (RANS) framework because of their computational simplicity and efficiency [5]. While the inherent assumption of a local dependence in eddy viscosity modelling is nearly impossible to avoid, improving the functional relationship between the Reynolds stress tensor and the mean velocity gradient tensor remains critically important for accurate industrial simulations [6].

Traditionally, turbulence closure models are formulated using heuristic and physics-based arguments. Common examples include the analogy drawn between turbulent stresses and viscous stresses, or the model transport equations for turbulent kinetic energy (TKE) k and a length-scale determining variable such as ε (TKE dissipation rate). However, a new approach in turbulence modelling is the physics-*informed* data-driven approach [17], which leverages the capabilities of machine learning. Rather than simplifying the underlying physics to close the RANS equations, the functional relationship between the mean flow gradients and Reynolds stresses is determined from high-fidelity datasets. Machine learning is well-suited to "discover" this (complex) functional relationship using a data-driven approach.

Within the framework of the application of machine learning for data-driven closure modelling, a wide range of methods have been presented and nearly all of them have demonstrated promising results for simple flows [17, 12, 61, 62]. Several machine learning and injection frameworks have been proposed [22, 36, 63, 64], including open-loop [65] and iterative frameworks [66]. These methods have been demonstrated to improve RANS results in canonical flows [67, 68, 69, 70], flow over airfoils [71, 72], combustion [73], and transonic flows [74]. Several investigations [34, 35, 39, 37] have used machine learning to

determine the coefficients of a tensor basis expansion for the anisotropy tensor, based on the seminal work of Ling *et al.* [23]. Other frameworks include the optimal eddy viscosity approach proposed by Wu *et al.* [33], or the Reynolds force vector approach advocated by Cruz *et al.* [75]. However, the issue of ill-conditioning was recently shown by Brener *et al.* [76] to affect a number of these modelling frameworks. When the RANS equations are ill-conditioned, small errors in the Reynolds stress tensor can be amplified, resulting in large errors in the prediction of the mean velocity field [77]. This error amplification is a major issue for physics-informed data-driven closures, as some amount of model error in predicting the Reynolds stress tensor is expected. Brener *et al.* [76] showed that the optimal eddy viscosity approach proposed by Wu *et al.* [33] is an effective strategy to address the ill-conditioning problem.

While previous investigations have presented optimal eddy viscosity-based frameworks that improve the conditioning of the RANS equations, these formulations provide no guarantee of a non-negative eddy viscosity. For practical purposes, the eddy viscosity should always be non-negative—a negative eddy viscosity prediction has the potential to destabilize convergence in an iterative RANS solver. A new formulation of the optimal eddy viscosity is proposed in the present work, which guarantees a non-negative optimal eddy viscosity. The conditioning of the new formulation is analyzed, and the suitability for use in a physics-informed data-driven turbulence closure model is assessed by using a neural network to predict this optimal eddy viscosity. An additional neural network is used to predict the remaining (non-linear) portion of the Reynolds stress tensor. The prediction of the velocity field is greatly improved after injecting the neural network-predicted Reynolds stress tensor into the mean momentum equations.

The present work is the first to propose a stability constraint on the optimal eddy viscosity and to implement this constraint into a neural network. Along with proposing a neural network architecture for predicting the optimal eddy viscosity, the present work also aims to demonstrate the ability of deep learning augmentation to improve the results of a sophisticated physics-based turbulence closure model. While previous studies have improved results from the simpler k-ε [23] and k-ω models [35], the model augmented in the present work is the k-ε-ϕ_t-f model [78], which is an improved reformulation of the $v2$-f model [54]. This model contains three turbulent transport equations for k, ε, and ϕ_t, along with an elliptic equation for the damping variable f.

The present work is structured as follows: in Section 3.2, we motivate and discuss the proposed optimal eddy viscosity formulation, and injection process. The choice of the network topology and the associated hyperparameter tuning for the customization of deep neural networks for the task of turbulence closure modelling is then described in Section 3.3. Section 3.4 presents the model predictions for an unseen test case. An *a*

priori analysis of the model predictions is given in Section 3.4.1. After injecting the model predictions into a RANS solver, the converged mean fields are analyzed in an *a posteriori* sense in Section 3.4.2. Having demonstrated the accuracy improvements gained through machine learning augmentation, Section 3.4.4 presents an interpretation and discussion of the data-driven closure. Conclusions and recommendations for further study are given in Section 3.5.

3.2 Methodology

The Reynolds-averaged continuity and momentum equations for an incompressible, isothermal, steady-state mean velocity field $\vec{U} = (U, V, W)$ and pressure field p is given by

$$\nabla \cdot \vec{U} = 0 \ , \tag{3.1}$$

$$\nabla \cdot (\vec{U}\vec{U}) = -\nabla p + \nu \nabla^2 \vec{U} - \nabla \cdot \tau \ , \tag{3.2}$$

where ν is the molecular kinematic viscosity of the fluid and $\tau \equiv \overline{\vec{u}'\vec{u}'}$ is the Reynolds stress tensor. Here, mean values are denoted using an overbar and $\vec{u}' \equiv (\vec{u} - \vec{U})$ is the fluctuating velocity ($\vec{u}$ is the total instantaneous turbulent velocity).

These equations are hereinafter referred to as the RANS equations. The unclosed term (Reynolds stress tensor), τ, is the subject of turbulence closure modelling. The common modelling approach is to approximate the anisotropic part of τ analogously to the viscous stress term, so

$$a \equiv \tau - \frac{1}{3}\mathrm{tr}(\tau)I \approx -2\nu_t S \ , \tag{3.3}$$

where a is the anisotropy tensor and I is the identity tensor. Equation (3.3) is known as the eddy viscosity approximation, where ν_t is the eddy viscosity, and S is the mean strain-rate tensor (symmetric part of the mean velocity gradient tensor) defined by

$$S \equiv \tfrac{1}{2}\left(\nabla\vec{U} + \nabla\vec{U}^{\mathrm{T}}\right) \ ,$$

where the superscript T denotes matrix transposition.

This approximation for τ permits the eddy viscosity to be treated implicitly, and added to the molecular viscosity ν in Eq. (3.2). Numerically, this has a stabilizing effect and therefore this approach is widely used in RANS modelling. Additionally, the term $\nabla \cdot \frac{1}{3}\mathrm{tr}(\tau)I$ (which is equal to $\nabla \cdot \frac{2}{3}kI$) is typically absorbed into the isotropic term ∇p. The

resulting isotropic term becomes the gradient of the "modified pressure" given by $p' = p + \frac{2}{3}k$.

While this approximation for τ is common in traditional turbulence modelling, the approach in data-driven turbulence modelling has not been uniform. Liu *et al.* [66] and Brener *et al.* [76] used the eddy viscosity approximation as follows:

$$\tau \approx -2\nu_t S. \tag{3.4}$$

However, a deficiency with the approximation given by Eq. (3.4) is that τ (whose non-zero trace is $2k$) can never be completely aligned with S (whose trace is necessarily zero owing to the incompressibility of the flow). Therefore, when formulating an "optimal eddy viscosity approach" (Section 3.2.1), it is more appropriate to use the more conventional approximation given in Eq. (3.3).

3.2.1 Optimal eddy viscosity

Within the commonly used eddy viscosity approach, closing the RANS equations is achieved using a single constitutive coefficient ν_t. However, calculating ν_t in a way that minimizes the error in the mean flow field is a complex task, and has led to the development of hundreds of RANS closure models. In the present work, we propose to use a neural network to directly estimate ν_t, effectively eliminating the need for any additional turbulent scalar transport equations. This approach was also applied by Wu *et al.* [33] and Liu *et al.* [66].

The optimal eddy viscosity ν_t^* is obtained by minimizing the error in the approximation of the anisotropy tensor as being directly proportional to the mean strain-rate tensor. More specifically, the optimal eddy viscosity is obtained as the solution of the following least-squares approximation problem:

$$\nu_t^* = \arg\min_{\nu_t} \|a - (-2\nu_t S)\| \ , \tag{3.5}$$

where ν_t^* is the optimal eddy viscosity in a least-squares sense ($\|\cdot\|$ denotes the Euclidean norm). Equation (3.5) has the following closed-form analytical solution:

$$\nu_t^* = -\frac{1}{2}\frac{a:S}{S:S} \ , \tag{3.6}$$

where the colon operator denotes double contraction. More specifically, for two Cartesian tensors A and B of rank two, their double contraction is given by $A : B \equiv A_{ij}B_{ij} = \mathrm{tr}(AB^T) = \mathrm{tr}(BA^T)$ (Einstein summation convention implied on repeated indices), where

AB denotes the matrix product of A and B. For the *special* case where both A and B are symmetric tensors (as in the current case), $A : B = \mathrm{tr}(AB) = \mathrm{tr}(BA)$.

While optimal, the formulation in Eq. (3.6) has the disadvantage that ν_t^* can become negative (especially when the eigenframes of a and S are poorly aligned). When treated implicitly in Eq. (3.2), a negative eddy viscosity can result in a combined negative effective viscosity, greatly destabilizing the numerical solution. Therefore, we propose the following formulation for the estimation of the optimal eddy viscosity:

$$\nu_t^\dagger = \arg\,\min_{\nu_t \geq 0} \| a - (-2\nu_t S) \| \;, \tag{3.7}$$

where $\nu_t^\dagger$ is the optimal eddy viscosity obtained using a non-negative least-squares approximation. The formulation in Eq. (3.7) guarantees that the injected eddy viscosity will not destabilize the iterative solution, at least in terms of reducing the diagonal dominance of the (assembled) coefficient matrix resulting from the discretization of the mean momentum transport equation. This formulation also allows an non-negativity constraint to be enforced on the output of a predictive model, further guaranteeing that any erroneous predictions at testing time do not destabilize the iterative solution.

In summary, the present optimal eddy viscosity formulation ($\nu_t^\dagger$) is the non-negative least-squares fit for $a \approx -2\nu_t S$. This formulation harmonizes the conventional eddy viscosity approximation used in turbulence modelling, which is in terms of the anisotropy tensor, and promotes iterative stability by guaranteeing a non-negative effective viscosity.

3.2.2 Non-linear part of the Reynolds stress tensor

The formulation in Eq. (3.7) optimizes the *linear* eddy viscosity approximation for a. An important question that arises is: how accurate is this linear approximation itself?

To investigate this question, we consider a case of turbulent flow over periodic hills, simulated using both RANS [79] and direct numerical simulation (DNS) [44]. Specifically, the $\alpha = 1.2$ case is considered, where α is the slope steepness factor. Figure 3.1 shows the distributions of the error in a after applying a linear eddy viscosity approximation for this canonical flow. Since this flow is two-dimensional, there are only three independent components of a: namely, a_{xx}, a_{xy}, and a_{yy} (owing to the fact that a is traceless). In the present work, the subscript θ is used for quantities obtained from a high-resolution DNS simulation, and the subscript ψ indicates a quantity obtained from a RANS closure model. Even though the linear eddy viscosity approximation has been optimized on high-fidelity data for $a = -2\nu_{t\theta}^* S_\theta$ and $a = -2\nu_{t\theta}^\dagger S_\theta$, errors on the order of 100% are present in all

components of a. In particular, the diagonal components of a are severely under predicted, even with an optimized eddy viscosity. Compared to the base RANS simulation, an optimal eddy viscosity approach applied to high-fidelity DNS data does not result in substantial accuracy gain.

Figure 3.2 shows two fit quality metrics for the approximation $a \approx -2\nu_t^\dagger S$. In Fig. 3.2(a), the local R^2 value for the non-negative least squares fit $a_{ij\theta} \approx -2\nu_{t\theta}^\dagger S_{ij\theta}$ is shown. While the R^2 values are higher near the bulk flow above the separated region, the R^2 values are generally poor near the walls, in the separated region, and during reattachment. Figure 3.2(b) shows the eddy viscosity fit quality metric proposed by Thompson *et al.* [80] which is expressed as

$$R_i = 1 - \frac{2}{\pi}\cos^{-1}\left(\sqrt{\frac{\mathrm{tr}(4\nu_{t\theta}^{\dagger 2}S_\theta^2)}{\mathrm{tr}(a_\theta^2)}}\right) \quad . \tag{3.8}$$

This second fit quality metric generally agrees with the R^2 value in the bulk flow, in supporting a reasonable quality of the linear eddy viscosity approximation. However, for the separation over the left hill, and the acceleration over the right hill, higher values of the R_i metric suggest a better quality in the optimal linear eddy viscosity fit. Nevertheless, visualizing both of these quality metrics demonstrates that there is substantial room for improvement in terms of representing a using only a linear eddy viscosity approximation for this separated flow.

The errors in each component of a arising from the linear eddy viscosity approximation are shown in Fig. 3.3. The linear eddy viscosity approximation fails significantly for all components of a during separation of the flow, a well-known deficiency [3]. Noticeable errors also exists in a_{yy} during reattachment of the flow along the bottom wall. Though Thompson *et al.*'s R_i value suggests a good approximation for a as the flow accelerates over the right hill, the largest errors in the shear component a_{xy} occur here.

While the linear eddy viscosity approximation, even after optimization, was found to be deficient in approximating a, it was of interest to see whether these errors would affect the accuracy of the mean velocity field. Figure 3.4 shows the errors in the mean field components after injecting the optimal eddy viscosity implicitly into the RANS equation, and allowing the solution to converge around the fixed $\nu_{t\theta}^\dagger$. Figure 3.4(a) shows that significant errors occur during separation and reattachment along the bottom wall in the streamwise velocity component U. These results agree with the known issues regarding misalignment of stress and strain, which have received attention in RANS modelling [3].

Even though the solution in Fig. 3.4 was produced by injecting a high-fidelity optimal eddy viscosity into the RANS equations, there are significant errors in the mean flow field.

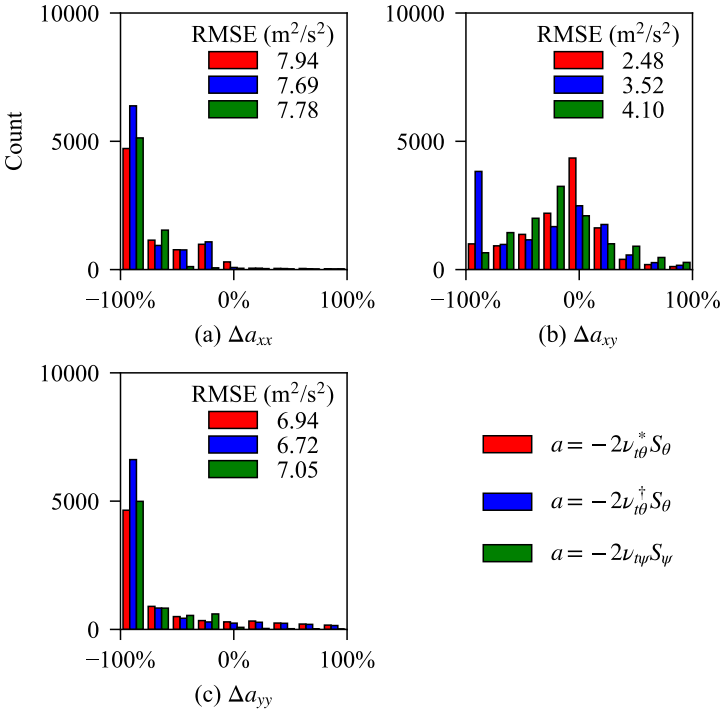


Figure 3.1: Error distribution in each anisotropy component after invoking the linear eddy viscosity approximation $a = -2\nu_t S$: (a) relative error in a_{xx}; (b) relative error in a_{xy}; and, (c) relative error in a_{yy}. Relative error is calculated by $\Delta a_{ij} = (a_{ij} - a_{ij\theta})/a_{ij\theta}$. $\nu^*_{t\theta}$ is the optimal eddy viscosity calculated using Eq. (3.6), and $\nu^\dagger_{t\theta}$ is the optimal eddy viscosity calculated using Eq. (3.7). The subscript θ indicates a quantity from DNS, and the subscript ψ indicates a quantity from a RANS simulation using the ϕ-f model (Section 3.2.5). The RMSE for each component of a is shown in each plot.

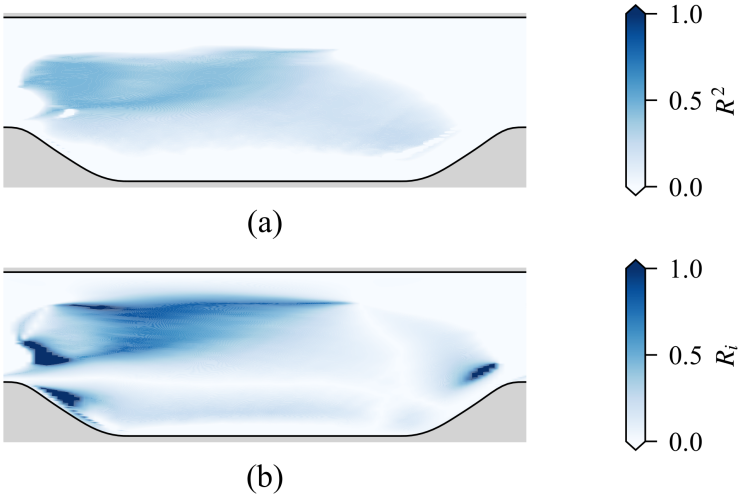


Figure 3.2: Fit quality metrics after invoking the linear eddy viscosity approximation $a = -2\nu_{t\theta}^{\dagger} S_\theta$: (a) R^2 value and (b) eddy viscosity model fit factor R_i [Eq. (3.8)].

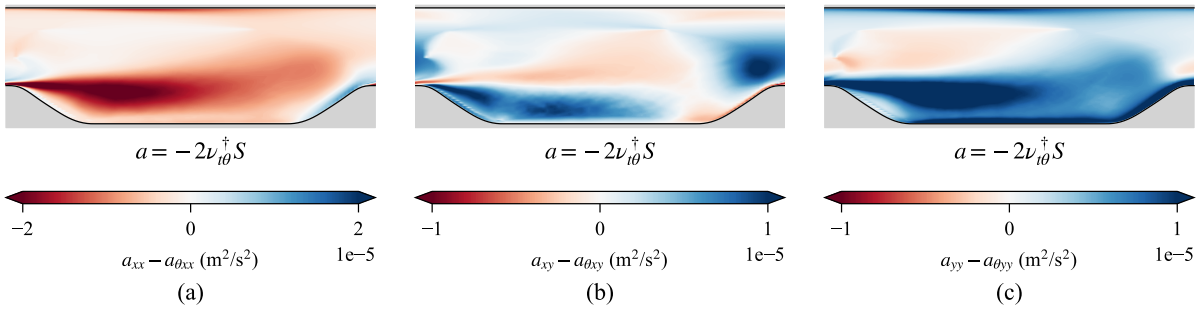


Figure 3.3: *A priori* contours of error in each component of a after invoking the linear eddy viscosity approximation $a = -2\nu_{t\theta}^{\dagger} S_\theta$: (a) error in a_{xx}; (b) error in a_{xy}; and, (c) error in a_{yy}.

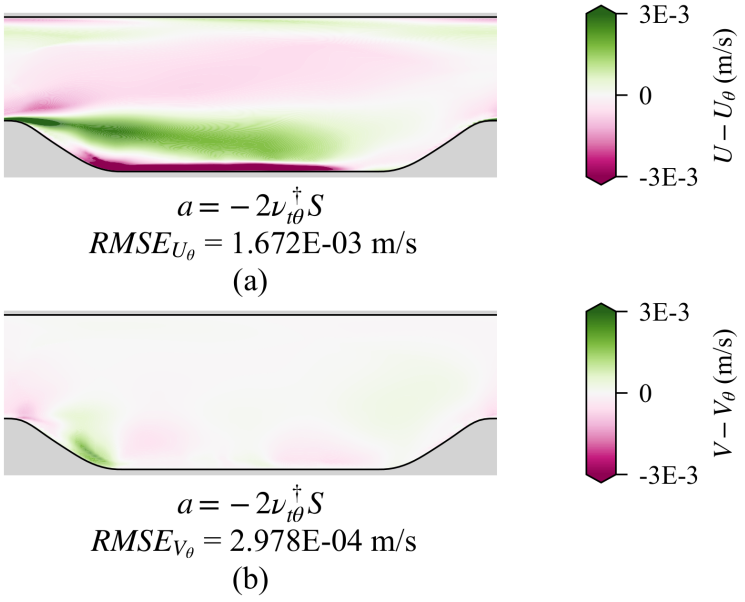


Figure 3.4: *A posteriori* contours of error in the velocity vector components after injecting the optimal linear eddy viscosity into the RANS equations, without any non-linear terms: (a) error in the x-component and (b) error in the y-component.

Furthermore, the analysis in Section 3.2.3 shows that conditioning errors with this optimal eddy viscosity formulation are minimal. If the target of the machine learning model is only the optimal eddy viscosity, mean field accuracy improvements are not guaranteed. In fact, the error fields shown in Fig. 3.4 represent the "upper limit" that could be achieved using a linear-component-only data-driven closure based on the optimal eddy viscosity as formulated in the present work.

Given the deficiencies of the linear eddy viscosity approximation, we propose an additional non-linear term in the anisotropy representation; namely, $a^{\perp}$ which is obtained from the following proportional/orthogonal tensor decomposition of a :

$$a = -2\nu_t^{\dagger} S + a^{\perp}. \tag{3.9}$$

In Eq. (3.9), a has been decomposed into a linear component ($-2\nu_t^{\dagger} S$) and a non-linear component ($a^{\perp}$). After calculating the optimal eddy viscosity using Eq. (3.7), the non-linear portion of the anisotropy tensor can be calculated using $a^{\perp} = a - (-2\nu_t^{\dagger} S)$.

3.2.3 Conditioning

A subject that is of immense importance for data-driven turbulence closure modelling is the issue of conditioning within the RANS equations. When the RANS equations are ill-conditioned, small errors in the closure term can be amplified, resulting in large errors in the mean flow field. This issue affects data-driven models because an error in the model prediction of the closure term is almost always expected. Even with an error-free closure term, an accurate converged mean field cannot be guaranteed [76].

To demonstrate the importance of the conditioning problem, a conditioning analysis similar to that performed by Brener *et al.* [76] was conducted. Figures 3.5 and 3.6 show the errors in the mean velocity field components for two injection experiments. The flow considered here is a turbulent flow at $Re_H = 5600$ over periodic hills[44]. In both Figs 3.5(a) and (b), the RANS equations have been closed using a highly accurate a from a DNS simulation undertaken by Xiao *et al.* [44]. In Figures 3.5(a) and 3.6(a), a_θ has been injected as an explicit source term into the RANS equations. In Figures 3.5(b) and 3.6(b), the decomposition proposed in the present work has been injected into the RANS equations in accordance to the following scheme: $\nu_{t\theta}^{\dagger}$ is treated implicitly and $a_\theta^{\perp}$ is treated as an explicit source term in the discretized mean momentum equations.

Despite both systems being closed by the same quantity arising from a high-fidelity simulation, the solution in Figs 3.5(a) and 3.6(a) has a root mean square error (RMSE)

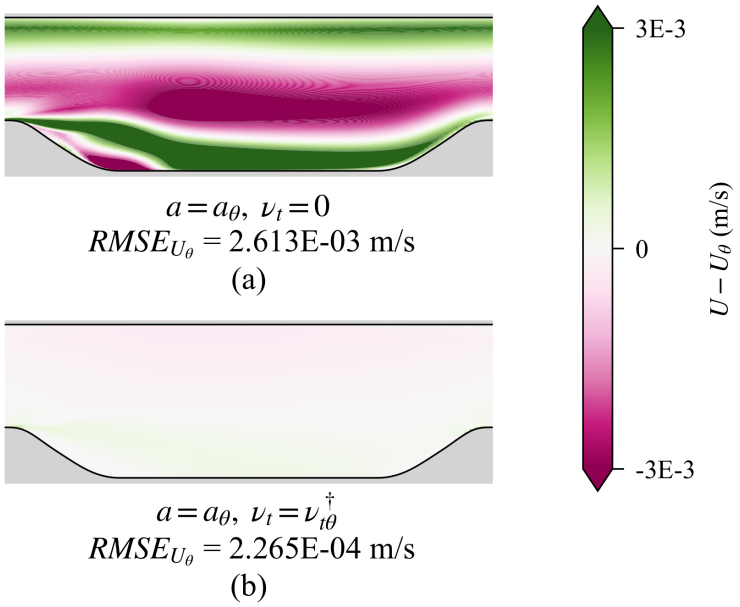


Figure 3.5: *A posteriori* contours of error in the x-velocity component after injecting the Reynolds stress anisotropy tensor from DNS into the RANS equations in two different ways: (a) fully explicit injection of a_θ and (b) the injection framework used in the present work.

that is an order of magnitude larger than the solution in Figs 3.5(b) and 3.6(b). It should be noted that additional differences between the two solutions may arise from the need to use first-order schemes to stabilize the solution for the fully explicit propagation.

The low error observed after injecting the full anisotropy representation [Eq. (3.9)] from a DNS simulation into the RANS equation highlights the merits of the present approach. As discussed in Section 3.3, since the quantities $\nu_{t\theta}^{\dagger}$ and $a_\theta^{\perp}$ are the training labels, injecting these quantities into the RANS equations provides the "upper limit" that the data-driven model could achieve. For this reason, we recommend that data-driven closure frameworks be evaluated in terms of the conditioning errors that arise even after a perfect model prediction ($\tilde{a} = a_\theta$).

3.2.4 Injection procedure

We use an open-loop, data-driven framework referred to by Ho and West[81] as a "one-time correction" model. To this purpose, our framework involves making a fixed correction to

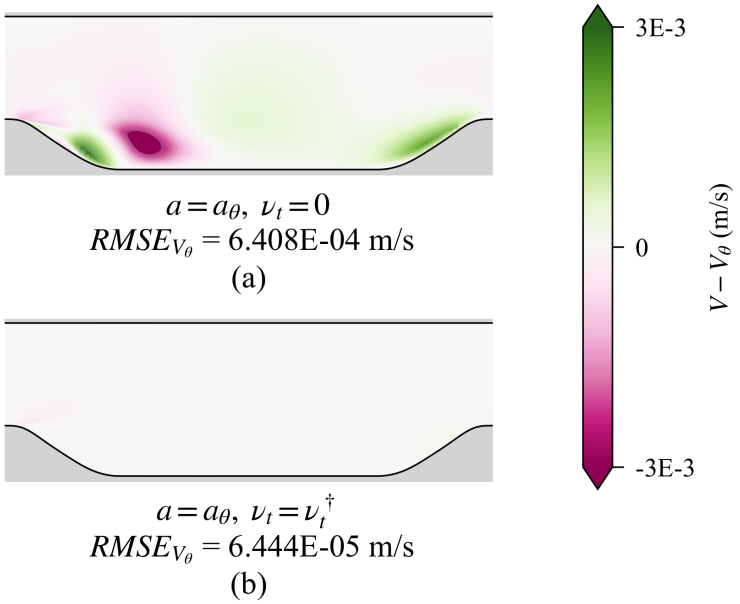


Figure 3.6: *A posteriori* contours of error in the y-velocity component after injecting the Reynolds stress anisotropy tensor from DNS into the RANS equations in two different ways: (a) fully explicit injection of a_θ and (b) the injection framework used in the present work.

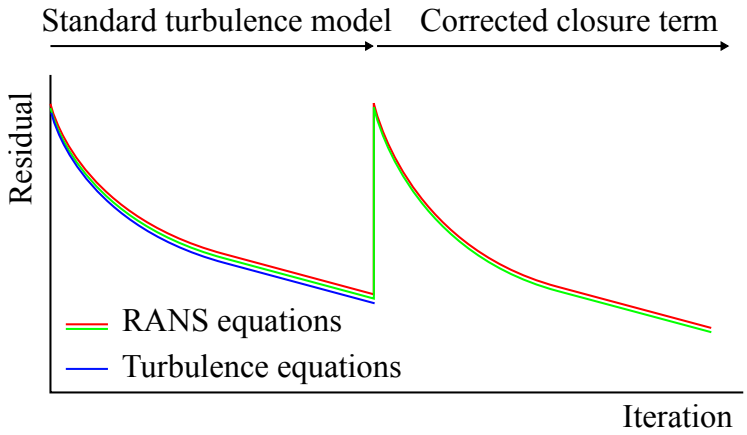


Figure 3.7: "Qualitative" residual plot for the machine learning corrective framework used in the present study. After injection, no further updates to the turbulence equations (e.g., the k and ε transport equations) are required.

the closure term, and then allowing the mean field to converge around this fixed correction. A qualitative description of the injection process in terms of the commonly used residual plot is shown in Figure 3.7. The correction is estimated using a converged RANS simulation obtained from a base turbulence model as an input. This framework is in contrast to an iterative framework, where the closure term correction is updated repeatedly as the velocity field evolves. For example, the iterative framework used by Liu *et al.* [66] calls the machine learning model during each iteration. While open-loop frameworks are limited to steady flows, the design and use of iterative correction frameworks remains an open question. For example, the converged solution from an iterative framework can depend on the initial flow field [81].

Motivated by the conditioning analysis of Brener *et al.* [76], and our own analysis shown in Figs 3.5 and 3.6, we make two closure corrections in the present work: namely, one using $\nu_t^\dagger$ and another using $a^\perp$. We correct the eddy viscosity to be the optimal eddy viscosity, formulated in Eq. (3.7), and treat it implicitly in the momentum equation through an effective viscosity. Having established the importance of including the non-linear part of the anisotropy tensor (Section 3.2.2), we also include the divergence of $a^\perp$ as a source term in the momentum equation.

After the correction, the RANS equations are solved iteratively until the mean fields converge around the fixed correction terms. The goal is that this converged solution will be more accurate than the solution produced by the base turbulence model. No update is required for the turbulent transport equations, as the eddy viscosity is directly corrected

in our framework.

3.2.5 The ϕ-f model

In the present work, we use the ϕ-f model implemented in OpenFOAM v2006 as the base RANS model [52, 50]. The ϕ-f model is a reformulated version of the v^2-f model, where ϕ is the ratio v^2/k, and f is a damping scalar. By adding a carefully constructed transport equation for the streamline-normal Reynolds stress component (v^2), this model accounts for the wall-blocking effects on the Reynolds stresses. The construction of the v^2 equation includes the effects of pressure strain on the streamline-normal Reynolds stress, a key contributor to wall-blocking effects. These wall-blocking effects are observed as unequal scaling (anisotropy) of the Reynolds stress components, which are not captured in a two-equation model. Often, the use of wall damping functions is required in two-equation models to correctly capture near-wall scaling. Laurence *et al.*'s [52] ϕ-f model is a reformulated version of a v^2-f model variant due to Lien and Kalitzin [55], which is, in turn, an improved version of Durbin's [54] original v^2-f model. Reformulating the v^2 and f equation in terms of ϕ results in a more robust model. For example, this model has reduced stiffness in terms of the near-wall damping singularity [52]. The ϕ-f model estimates the eddy viscosity as

$$\nu_t = C_\mu \phi k T \ , \tag{3.10}$$

where C_μ is a model coefficient, and T is the turbulent time scale.

Details on the OpenFOAM implementation of this model are given in the OpenFOAM documentation [50]. The original ϕ-f model equations can be summarized as follows:

$$\frac{\partial k}{\partial t} + U_i \frac{\partial k}{\partial x_i} = P - \varepsilon + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right] \ ; \tag{3.11}$$

$$\frac{\partial \varepsilon}{\partial t} + U_i \frac{\partial \varepsilon}{\partial x_i} = \frac{C_{\varepsilon_1} P}{T} - \frac{C_{\varepsilon_2}\varepsilon}{T} + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right] \ ; \tag{3.12}$$

$$\frac{\partial \phi}{\partial t} + U_i \frac{\partial \phi}{\partial x_i} = f - P\frac{\phi}{k} + \frac{2\nu_t}{k\sigma_k}\frac{\partial \phi}{\partial x_j}\frac{\partial k}{\partial x_j} + \frac{\partial}{\partial x_j}\left[\left(\frac{\nu_t}{\sigma_k}\right)\frac{\partial \phi}{\partial x_j}\right] \ ; \tag{3.13}$$

and,

$$L^2\frac{\partial^2 f}{\partial x_j} - f = \frac{1}{T}(C_{f_1} \quad -1)\left[\phi - \frac{2}{3}\right] \\ -C_{f_2}\frac{P}{k} - 2\frac{\nu}{k}\frac{\partial\phi}{\partial x_j}\frac{\partial k}{\partial x_j} - \nu\frac{\partial^2\phi}{\partial x_j} \ . \tag{3.14}$$

Equations (3.11), (3.12), and (3.13) are model transport equations for k, ε, and ϕ, respectively. Written in index notation, $U_i = \vec{U}$. Equation (3.14) is an elliptic relaxation equation for f, which is a scalar predicting near-wall damping effects.

The turbulent time scale T and length scale L are given, respectively, by

$$T = \max\left(\frac{k}{\varepsilon}, C_T\sqrt{\frac{\nu}{\varepsilon}}\right) \ , \tag{3.15}$$

$$L = C_L\max\left(\frac{k^{3/2}}{\varepsilon}, C_\eta\left(\frac{\nu^3}{\varepsilon}\right)^{1/4}\right) \ . \tag{3.16}$$

Finally, the model constants assume the values summarized as follows: $C_\mu = 0.22$; $C_{\varepsilon_1} = 1.4(1.0 + 0.05\sqrt{1.0/\phi})$; $C_{\varepsilon_2} = 1.9$; $C_T = 6.0$; $C_L = 0.25$; $C_{f_1} = 1.4$; $C_{f_2} = 0.3$; $C_\eta = 110.0$; $\sigma_k = 1.0$; and, $\sigma_\varepsilon = 1.3$.

3.2.6 Numerical methods

All simulations in the present work use OpenFOAM v2006. For the base RANS simulation, the PIMPLE algorithm was used to achieve a converged solution. Then, a modified PIMPLE solver, which accepts the corrected $\nu_t^\dagger$ and $a^\perp$, was used to inject these corrections into the RANS equations and iteratively solve for the mean fields. While all flows in the present work reach a steady state solution, the unsteady algorithm was used to stabilize the iterative solution.

For discretizing the RANS equations, a second-order upwind scheme was used for the convective terms, and a second-order central difference scheme was used for the diffusion terms. For the convective terms in the base turbulence model transport equations, a first-order scheme was used.

A wall-resolved mesh ($y^+ \leq 1$) was used for all simulations. Further details on the mesh, domain, and boundary conditions are provided in McConkey *et al.*'s [79] description of the machine learning dataset. Cyclic boundary conditions were used for all flow variables at the inlet and outlet. At the top and bottom walls, the boundary conditions were fixed-zero for velocity, and zero-gradient for pressure.

3.3 Deep learning procedure for predicting the Reynolds stress tensor

3.3.1 Dataset

To test the proposed decomposition and injection framework, a series of flow over periodic hills was chosen. This flow features separation over the left hill, reattachment along the bottom wall, and acceleration over the right hill. The dataset presented for data-driven turbulence modelling by McConkey *et al.* [79], includes these cases, based on the DNS simulations of Xiao *et al.* [44]. The periodic hills portion of the dataset consists of five cases, each with 14,751 data points, for a total of 73,755 data points. At each data point, the complete set of RANS input features and DNS fields are provided, so that the model can learn a mapping from the RANS features to the DNS labels. The RANS fields come from a converged solution using the k-ε-ϕ_t-f model [52] in OpenFOAM v2006. The five cases are generated by parametrically varying the hill steepness α and the overall length of the geometry. The five cases correspond to $\alpha = 0.5$, 0.8, 1.0, 1.2, and 1.5, as shown in Fig. 3.8.

3.3.2 Model architecture

The model consists of two independent neural networks: namely, EVNN, and apNN. The objective of EVNN is to predict the optimal eddy viscosity $\nu_t^\dagger$, whereas that of apNN is to predict the non-linear part of the anisotropy tensor, $a^\perp$. Both models are feed-forward, fully connected neural networks. The EVNN has one output neuron, and the apNN has three output neurons (one for each independent component of $a^\perp$).

Because $\nu_t^\dagger$ is non-negative, the stability of the EVNN predictions during training and injection can benefit by enforcing a non-negative prediction. For the EVNN, the activation function used for the output neuron was a simple exponential function $f(x) = e^x$, where x is the input value to the layer. This activation function guarantees a non-negative eddy viscosity prediction by the EVNN, regardless of the input activation layer (viz., the choice of the exponential function as the activation function induces an inductive bias of the network output). This stability constraint is possible due to the formulation of $\nu_t^\dagger$, which guarantees a non-negative training label. For the apNN, the three output neurons utilize a linear activation function. For the hidden layers, the scaled exponential linear unit (SELU) activation function was used. Combined with the LeCun initialization of

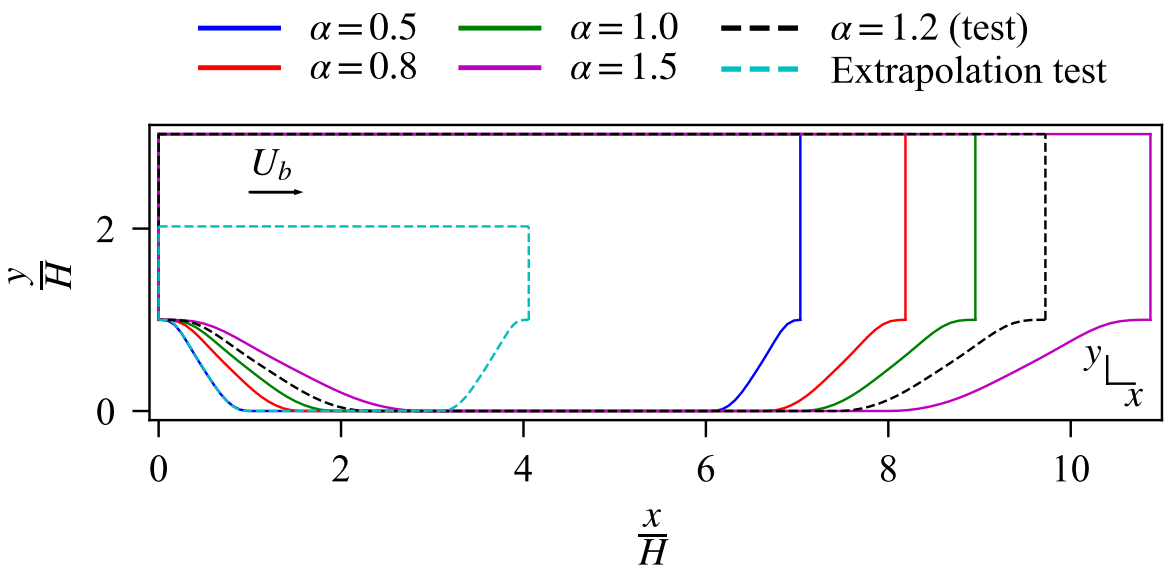


Figure 3.8: The geometry and flow variables used for the periodic hills cases in the present work. The extrapolation test case is taken from the extended periodic hills dataset [44], and is shorter in both height and flat section length than any of the training cases.

the network weights, a SELU-activated, fully connected, feed-forward neural network is self-normalizing [82], eliminating the need for batch normalization layers [83].

The neural networks were implemented using the high-level Keras application programming interface (API) in TensorFlow [84]. The hyperparameters used for both networks were identical: namely, both networks have $L = 14$ hidden layers, with $n = 30$ neurons in each layer, resulting in a network architecture with an depth-to-width aspect ratio of $L/n \approx 0.467$. The selection of the hyperparameters (e.g., number of neurons in each layer, number of hidden layers, L2 regularization parameter) used in the neural networks was obtained using a combination of a random and a grid search as described in greater detail in Appendix A.

The EVNN has 13,141 trainable parameters (weights and biases) with a single output, and the apNN has 13,203 trainable parameters with three outputs. For traditional turbulence modelling, tuning this number of parameters is a nearly impossible task. However, for deep learning, tuning these parameters is relatively "straightforward"—the number of learnable parameters is often orders of magnitude greater than those used in the EVNN and the apNN. Nevertheless, despite the large number of model parameters used in these types of overparameterized neural networks (where the number of model parameters far exceed the number of training data), there is mounting empirical (indeed, practical) evidence that these networks can be successfully trained to provide excellent predictions—perhaps supporting the notion that it is not the number of model parameters, but the representation learning provided by the subtle correlations in the intralayer and interlayer interactions of the neural units in the network that ultimately determines the predictive skill and generalizability of the model in deep learning.

3.3.3 Input feature selection

A wide range of input features have been used in previous data-driven closure frameworks. The input features for a data-driven closure are generally required to possess the same invariance properties as the underlying Navier-Stokes equations: reflection, rotation, and Galilean transformation invariance. Methods used to generate the input feature set vary from a purely heuristic selection, to a systematic generation of an integrity basis [33]. In the present work, we use a combination of these two methods. Three heuristic scalars are selected as input features given by

$$\begin{aligned} q_1 &\equiv \min(2.0, \sqrt{k} y_\perp/(50\nu)) \ , \\ q_2 &\equiv k/\varepsilon \|S\| \ , \\ q_3 &\equiv \|\tau\|/k \ . \end{aligned} \tag{3.17}$$

Here, $y_\perp$ is the wall-normal distance. q_1 is the wall distance based Reynolds number, q_2 is the ratio of the mean strain timescale to a turbulent time scale, and q_3 is the ratio of the total Reynolds stress magnitude to the diagonal Reynolds stresses. These Galilean invariant scalars were used previously by Kaandorp and Dwight [35].

In addition to these scalars, we modify the procedure presented by Wu *et al.* [33] to significantly augment the input feature set. In Wu *et al.*'s procedure, four gradient tensors were selected: S, R,∇k, and ∇p. The TKE and pressure gradient vectors were converted to second-order tensors, by casting their components into an anti-symmetric matrix. Then, the method outlined by Spencer and Rivlin [60] was used to generate an integrity basis for the space spanned by these four second-order tensors. This integrity basis consists of 47 tensors. Finally, a scalar input feature was extract from each tensor, by taking the first invariant (viz., the trace) of each integrity basis tensor.

While this procedure extracts a large number of input features from the flow, a number of issues arise that have received limited attention in the literature. Firstly, the terms related to ∇p in the integrity basis tend to be numerically unstable in our experience, possibly due to the normalization used by Wu *et al.* [33]. In their frameworks, Wang *et al.* [62] and Kaandorp and Dwight [35] omitted these terms, reducing the number of invariants down to 16. Secondly, many of these invariants are zero for two-dimensional flows. These zero invariants arise from either a direct result of the z-direction gradients being zero, or from the incompressibility of the flow. Lastly, this procedure only uses the first invariant I_1 (or, trace) of each integrity basis tensor—more information can be extracted from this basis by including further invariants in the input feature set. We have presented an analysis of Wu *et al.*'s [33] integrity basis in Appendix B, showing expressions for each of the basis tensor invariants for two-dimensional flows.

To circumvent the issue of unstable terms related to ∇p being unusable as input features, we replace ∇p in Wu *et al.*'s basis with ∇v^2. This replacement is possible since our base turbulence model is the ϕ-f model (Section 3.2.5). We found that terms including ∇v^2 were much more stable than the terms including ∇p, and therefore more features could be included. It is noted that ∇v^2 provides additional information compared to ∇k, because v^2 measures the degree of wall-blocking effects and anisotropy in the flow. Additionally, we augment the 47 original first invariants by also taking the second invariant I_2 of a tensor A ($I_2 = \frac{1}{2}[(\mathrm{tr}(A))^2 - \mathrm{tr}(A^2)]$). For a three-dimensional flow, this effectively doubles the number of basis tensor invariants that can be used as input features. However, as shown in Appendix B, many of these invariants are zero for two-dimensional flows. Nevertheless, after eliminating the zero-valued invariants, a set of 29 invariants remained as suitable input features. Table 3.1 summarizes the input features used.

Table 3.1: Input features for EVNN and apNN.

Number	Input feature	Expression	Transformation
1	$I_1(B_1)$	$I_1(S^2)$	$\log(\lvert x\rvert + 1)$
2	$I_1(B_3)$	$I_1(R^2)$	$\log(\lvert x\rvert + 1)$
3	$I_1(B_4)$	$I_1(A_{v2}^2)$	$\sqrt[3]{x}$
4	$I_1(B_5)$	$I_1(A_k^2)$	$\log(\lvert x\rvert + 1)$
5	$I_1(B_7)$	$I_1(R^2S^2)$	$\log(\lvert x\rvert + 1)$
6	$I_1(B_9)$	$I_1(A_{v2}^2S)$	$\sqrt[3]{x}$
7	$I_1(B_{10})$	$I_1(A_{v2}^2S^2)$	$\log(\lvert x\rvert + 1)$
8	$I_1(B_{12})$	$I_1(A_k^2S)$	$\sqrt[3]{x}$
9	$I_1(B_{13})$	$I_1(A_k^2S^2)$	$\log(\lvert x\rvert + 1)$
10	$I_1(B_{16})$	$I_1(A_{v2}A_k)$	$\log(\lvert x\rvert + 1)$
11	$I_1(B_{21})$	$I_1(A_{v2}RS)$	$\log(\lvert x\rvert + 1)$
12	$I_1(B_{25})$	$I_1(A_{v2}^2SRS^2)$	$\log(\lvert x\rvert + 1)$
13	$I_1(B_{29})$	$I_1(A_k^2RS)$	$\log(\lvert x\rvert + 1)$
14	$I_1(B_{33})$	$I_1(A_kSRS^2)$	$\log(\lvert x\rvert + 1)$
15	$I_1(B_{34})$	$I_1(A_{v2}A_kS)$	$\sqrt[3]{x}$
16	$I_1(B_{35})$	$I_1(A_{v2}A_kS^2)$	$\log(\lvert x\rvert + 1)$
17	$I_1(B_{42})$	$I_1(RA_{v2}A_k)$	$\sqrt[3]{x}$
18	$I_1(B_{43})$	$I_1(RA_{v2}A_kS)$	$\sqrt[3]{x}$
19	$I_1(B_{45})$	$I_1(RA_{v2}A_kS^2)$	$\sqrt[3]{x}$
20	$I_1(B_{46})$	$I_1(RA_kA_{v2}S^2)$	$\sqrt[3]{x}$
21	$I_2(B_1)$	$I_2(S^2)$	$\log(\lvert x\rvert + 1)$
22	$I_2(B_2)$	$I_2(S^3)$	$\log(\lvert x\rvert + 1)$
23	$I_2(B_3)$	$I_2(R^2)$	$\log(\lvert x\rvert + 1)$
24	$I_2(B_4)$	$I_2(A_{v2}^2)$	$\log(\lvert x\rvert + 1)$
25	$I_2(B_5)$	$I_2(A_k^2)$	$\log(\lvert x\rvert + 1)$
26	$I_2(B_6)$	$I_2(R^2S)$	$\log(\lvert x\rvert + 1)$
27	$I_2(B_7)$	$I_2(R^2S^2)$	$\log(\lvert x\rvert + 1)$
28	$I_2(B_8)$	$I_2(R^2SRS^2)$	$\log(\lvert x\rvert + 1)$
29	$I_2(B_{16})$	$I_2(A_{v2}A_k)$	$\log(\lvert x\rvert + 1)$
30	q_1	$\min(2.0, \sqrt{k}y_\perp/(50\nu))$	—
31	q_2	$k/\varepsilon\lVert S\rVert$	—
32	q_3	$\lVert\tau\rVert/k$	—
33	ϕ	ϕ	—

This input feature set represents one of the richest feature sets used in a data-driven turbulence closure model. While many studies relax the condition of Galilean invariance (most commonly, by including the turbulence intensity), all of our input features are Galilean invariant. Section 3.4.4 analyzes the relative importance of each of these features to the model predictions.

3.3.4 Label calculation

Each neural network has a different set of training labels, based on the desired output. For the EVNN, the goal is to predict the optimal eddy viscosity $\nu_t^\dagger$. As shown in Section 3.2.3, injecting the optimal eddy viscosity from DNS produced a well-conditioned closure. Therefore, the label field for the EVNN consists of the $\nu_{t\theta}^\dagger$ field. To calculate this field, scikit-learn [85] was used to perform the non-negative least-squares (NNLS) regression fit for $a_\theta = -2\nu_t S_\theta$ at each cell in the dataset.

The labels for the apNN consist of the three independent components of the non-linear part of the Reynolds stress tensor. Therefore, the three component labels were extracted from the tensor $a_\theta^\perp = a_\theta + 2\nu_{t\theta}^\dagger S_\theta$, where $\nu_{t\theta}^\dagger$ are the labels for the EVNN. At injection time, $a_{zz}^\perp$ can be calculated as $a_{zz}^\perp = -a_{xx}^\perp - a_{yy}^\perp$, because $a^\perp$ is a traceless tensor.

3.3.5 Pre-processing

After calculating the input feature and label sets, the five periodic hills cases were divided into a training set, a validation set, and a testing set. The training process consists of updating the model weights and biases using an optimization algorithm. The model makes predictions using the training set features and evaluates the predictions using the training set labels. The validation and testing sets are used similarly, but at different times in the process. The validation set is used to evaluate the predictive performance of the neural network throughout the training process in order to assess when the model has converged. The validation set is not used to update the weights and biases of the network. Instead, it helps identify overfitting during the training process. If the training process continues for too long, the model will begin to memorize noise and other undesired details in the training set. Repeated evaluation of the neural network predictive performance using the validation set will allow one to assess the generalization performance of the model during training. The testing set is used to evaluate the final, trained model on an entirely new set of data—it can be used to determine the model (or generalization) error and to evaluate the generalization capability of the fully-trained neural network model. More specifically, the

generalization error is perhaps the primary quantitative measure of the success of the fully-trained neural network model providing, as such, the standard for how well the network is really approximating the underlying function for the Reynolds stresses.

The $\alpha = 1.2$ case was selected as an interpolation test set. In terms of the parameter α, this represents an interpolation test case because the training and validation set contain α values both above and below 1.2. Furthermore, a case of $\alpha = 0.5$ with a domain height of $y_{\max} = 2.024H$ and a flat length between the periodic hills of $4.071H$, extracted from an extended periodic hills dataset provided recently by Xiao et al. [44], was selected as an extrapolation test case (cf. Fig. 3.8). This is a challenging test case owing to the fact that the geometry of this case is significantly different than that of the training set.

The training/validation set consists of 59,004 data points, for the remaining four cases. At this stage, the typical procedure is to shuffle and split all of the remaining data into a training and validation set. However, for training a data-driven turbulence closure neural network model, we propose a different method to better estimate the generalization performance during training. Our recommendation is that the validation set should also consist of distinct cases from the training set. For example, the validation set could be all data points for the $\alpha = 1.0$ test case, and the training data would then consist of the $\alpha = 0.5$, 0.8, and 1.5 cases.

It is important to stress that the choice of validation set greatly influences the training process. This issue is normally solved by using k-fold cross-validation, where the training data is split into k folds. Each of the folds is then used as a validation set, while the model is trained on the remaining folds. We apply 4-fold cross-validation, where the model is trained four times, each using a remaining case ($\alpha = 0.5$, 0.8, 1.0, 1.5) as the validation set. The model which exhibits the best generalization performance (defined as the difference between the training and validation error) is then selected for testing.

The input features and labels are often scaled and transformed in deep learning in order to eliminate scale differences between input features and to ensure well behaved weights and biases during training [83]. To this purpose, a transformation was selected for each of the input features and labels to ensure a more uniform distribution (histogram) of their values before they are used in training. These transformations are given in Table 3.1. The selection of input feature transformations is highly heuristic, and depends on the data used for each study. However, we have generally applied a logarithm transform ($\hat{x} = \log(|x| + 1)$) to data which is entirely positively- or negatively-valued, and a cube-root transform ($\hat{x} = x^{1/3}$) to data which contains both positive and negative values. For the majority of data containing RANS mean fields, the histograms tend to be highly skewed due to domain-specific factors. For example, the behavior of almost all fields in the near-wall region is completely different

than the bulk flow region. It was found that transforming the components of $a^{\perp}$ was not necessary for sufficient model performance. After transforming the features (Table 3.1), the values were scaled to lie in the unit interval [0,1] using scikit-learn's MinMaxScaler [85]. The $\nu^{\dagger}_{t\theta}$ labels were transformed using $\hat{x} = \log(|x| + 1)$, then scaled using the same method as the features. No transformation was necessary for the $a^{\perp}_{\theta}$ labels. Only scaling was applied to the $a^{\perp}_{\theta}$ label set.

3.3.6 Training

For the training process, the Nadam optimizer in Keras was selected [84]. This optimizer combines the adaptive moment estimation (Adam) optimizer with Nesterov momentum to accelerate training. The learning rate used was 1×10^{-4}. For the EVNN, the loss function was the mean-squared error, with L2 regularization used. The L2 regularization weight λ was 1×10^{-5}. L2 regularization adds an additional penalty term to the loss function to constrain the norm squared of the optimized model parameters (or, weights)—the neural network seeks the best compromise between minimizing the errors in the prediction of the output and constraining the magnitude of the neural network weights to be as small as possible (viz., shrinking the weights towards zero). In practice, this regularization is required for some deep learning scenarios. We did not apply any L2 regularization to the apNN loss function, as the generalization performance was found to be sufficient without any additional regularization term to constrain the weights.

The training process was completed for the four cross-validation folds, as described in Section 3.3.5. After completing the training procedure for each of these folds, the model which provided the best generalization performance was selected. The training loss curves for the selected EVNN and apNN are shown in Figs 3.9 and 3.10, respectively. While the apNN loss curve remains relatively stable, the loss curve for the EVNN contained instabilities throughout the training process. Typically, these instabilities are a symptom of the learning rate being too large, but the learning rate was not found to be responsible for the occurrence of these jumps. This indicates that there may be significant discontinuities in the total loss surface consisting of the mean-squared error for prediction of $\nu^{\dagger}_{t}$ and the L2 regularization (penalty) term involving the weights. After approximately 1000 epochs, the mean-square error (MSE) part of the EVNN loss function remains relatively stable, indicating that no further accuracy is being gained in predicting $\nu^{\dagger}_{t}$. The remaining drop in the overall loss function comes from the training procedure optimizing the weights in the EVNN. It was found that the optimal number of epochs at which to stop the EVNN training was 2000, which allows an additional 1000 epochs for the optimizer to tune the network weights.

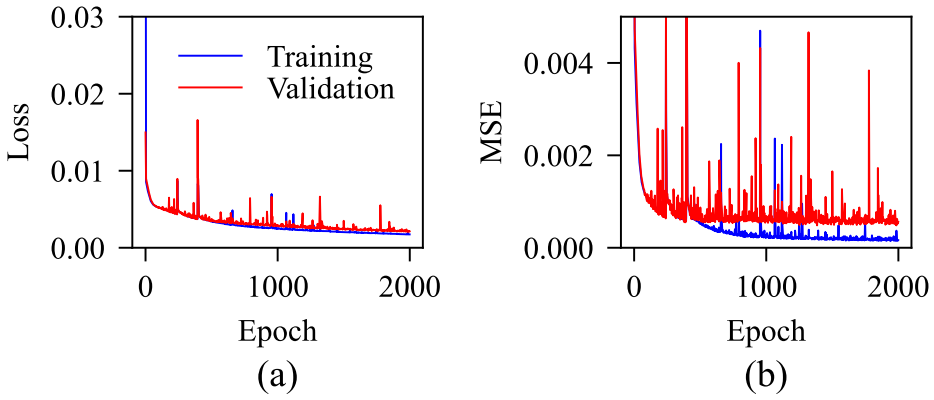


Figure 3.9: Loss curves during training for the EVNN: (a) the total loss function consisting of the mean squared error (MSE) and the L2 regularization (penalty term) of the weights and (b) the MSE portion of the total loss function.

The loss curve for the apNN (cf. Fig. 3.10) is simply the MSE of the prediction, because no regularization term for the weights was included in the training process for this neural network. After about 500 epochs, the training loss is smaller than the validation loss, and the training loss continues to decrease monotonically after this point. By assessing the *a priori* and *a posteriori* performance for various selections of the number of epochs to use as the training endpoint (early stopping), it was found that a training endpoint of 1500 epochs yielded the best results for the apNN.

3.4 Results

The final step in the deep learning procedure includes a test on a hold-out case—known also as the test case that is set aside and used to evaluate the neural network model only after training is complete. For the periodic hills training set, the $\alpha = 1.2$ case was held out of the training. Testing the network on a hold-out case provides an assessment of the network's ability to generalize by predicting a previously unseen case. The results for the hold-out test case are discussed in the following sections. A tilde is used to denote a quantity predicted by the neural networks (e.g., $\tilde{\nu}_t^\dagger$ is a prediction of the optimal eddy viscosity provided by the neural network).

For the open-loop framework used here, there are two stages of evaluation: namely, *a priori* and *a posteriori*. In Section 3.4.1, the model predictions are evaluated before

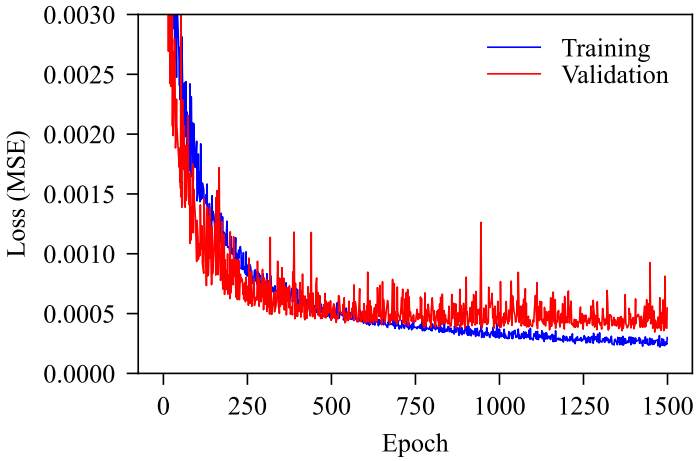


Figure 3.10: Loss curve during training for the apNN. The loss function for the apNN consists of the mean squared error (MSE) only (viz., no regularization term was added to the loss function).

injecting into the modified PIMPLE solver (*a priori* assessment). In Section 3.4.2, the resulting mean flow fields and combined anisotropy tensor (a) are evaluated after the iterative solution has converged around the fixed predictions (*a posteriori* assessment). The framework used here treats the linear component of a implicitly, so the combined prediction $\tilde{a}$ is not available in an *a priori* sense—it must be evaluated *a posteriori*.

3.4.1 *A priori* assessment

Prior to injecting the model predictions into the RANS equations, the predictions on the $\alpha = 1.2$ test case were analyzed for accuracy. The objective of the EVNN is to accurately predict $\nu_t^\dagger$. The mean-squared error is calculated by (for N data points)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\tilde{\nu}_{ti}^\dagger - \nu_{t\theta i}^\dagger)^2 \ . \tag{3.18}$$

For the $\alpha = 1.2$ test case, the MSE in the prediction of $\nu_t^\dagger$ is 1.89×10^{-7} m^4 s^{-2}. Figure 3.11 compares $\tilde{\nu}_t^\dagger$ to $\nu_{t\theta}^\dagger$. Figure 3.11(a) shows that while the predictions generally fall along the (ideal) line $\tilde{\nu}_t^\dagger = \nu_{t\theta}^\dagger$ at smaller values of $\nu_t^\dagger$, some scatter in the predictions is observed at larger values. The error distribution for the prediction of $\nu_t^\dagger$ [Fig. 3.11(b)] is roughly

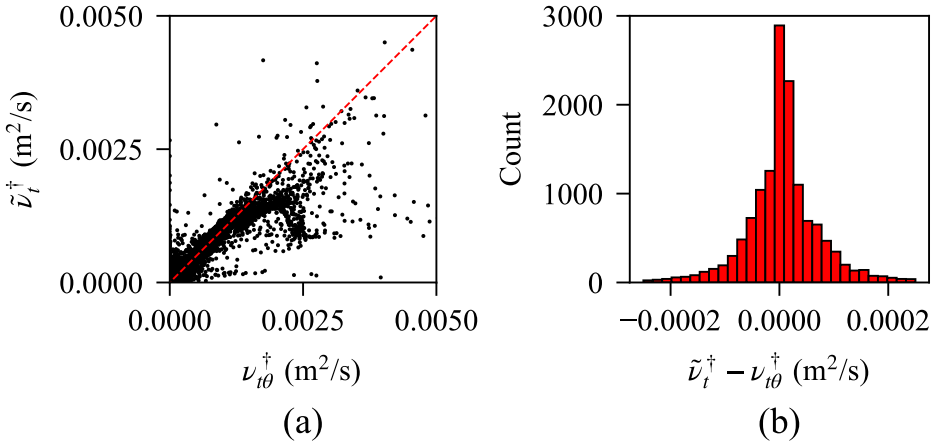
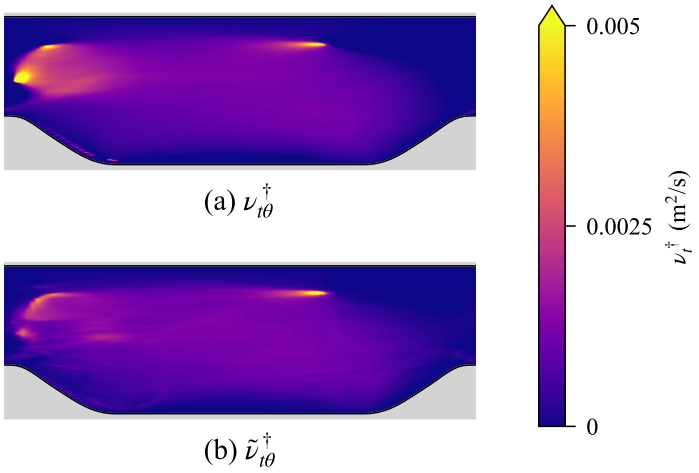


Figure 3.11: *A priori* prediction accuracy for the EVNN on the $\alpha = 1.2$ test case: (a) plot of the predicted value (ordinate) vs the ground-truth value obtained from DNS (abscissa) and (b) distribution of the $\nu_t^\dagger$ prediction errors.

symmetric, indicating that there is not a significant over or under prediction tendency for the EVNN. The large prediction errors in Fig. 3.11 demonstrate the importance of model conditioning—these errors will always be present at model testing time.

Figure 3.12 shows the spatial contours of $\tilde{\nu}_t^\dagger$ and $\nu_{t\theta}^\dagger$. This figure shows that the EVNN is able to capture trends in the spatial variation of $\nu_t^\dagger$ well. The large values for $\nu_t^\dagger$ in the center of the domain are predicted well by the EVNN. In general, the neural network predictions match the ground truth (DNS data), with some minor discrepancies in the high $\nu_t^\dagger$ region above the first crest. While Fig. 3.11 presents a pessimistic view of the ability of the EVNN to accurately predict $\nu_t^\dagger$, the contours in Fig. 3.12 provide a more optimistic view.

Figures 3.13 and 3.14 show that the apNN predicts the non-linear part of the anisotropy tensor well. The MSEs associated with the prediction of each component of $a^\perp$ are $\text{MSE}_{a^\perp_{xx}} = 2.31 \times 10^{-12}$, $\text{MSE}_{a^\perp_{xy}} = 9.40 \times 10^{-13}$, and $\text{MSE}_{a^\perp_{yy}} = 9.56 \times 10^{-13}$ (in units of m^4 s^{-4}). Spatial trends in the magnitude of $a^\perp$ are accurately predicted by the apNN (cf. Fig. 3.14). The individual points for the various components generally lie all along the ideal line $\tilde{a}^\perp = a^\perp_\theta$ (shown as the diagonal dashed lines in Fig. 3.13). A perusal of Fig. 3.13 shows that the diagonal (normal stress) components of $a^\perp$ are generally better predicted using apNN than the off-diagonal (shear stress) components. Indeed, the error distributions for the prediction of the normal stress components are more markedly peaked



(a) $\nu_{t\theta}^\dagger$

(b) $\tilde{\nu}_{t\theta}^\dagger$

Figure 3.12: *A priori* contours of $\nu_t^\dagger$ for the test case: (a) ground-truth value from DNS and (b) value predicted by the EVNN. The molecular kinematic viscosity for this case is $\nu = 5 \times 10^{-6}$ m^2 s^{-1}.

around zero (error) than that for the shear stress component [cf. Figs. 3.13(a) and (f) with Fig. 3.13(d)].

Comparing the predictions by the EVNN (Figs 3.11 and 3.12) to the predictions by the apNN (Figs 3.13 and 3.14), we see that the apNN generally out-performs the EVNN in terms of predicting their respective targets. This outcome agrees with our experience training and testing several times with these models—$\nu_t^\dagger$ is harder to predict than $a^\perp$. This discrepancy indicates that the information content embodied in the RANS fields for the prediction of $a^\perp$ is larger than that for the prediction of $\nu_{t\theta}^\dagger$.

3.4.2 *A posteriori* assessment

The fields predicted by the EVNN and apNN were injected into the RANS equations. A similar procedure as Section 3.2.3 was utilized, whereby a modified PIMPLE solver in OpenFOAM v2006 was employed to inject the predicted fields from the EVNN ($\tilde{\nu}_t^\dagger$) and apNN ($a^\perp$) into the RANS equations, and iteratively reach a converged solution for the mean velocity field.

Figure 3.15 shows the contours of the velocity magnitude for the converged solution. The converged results agree well with the flow fields coming from the highly resolved DNS

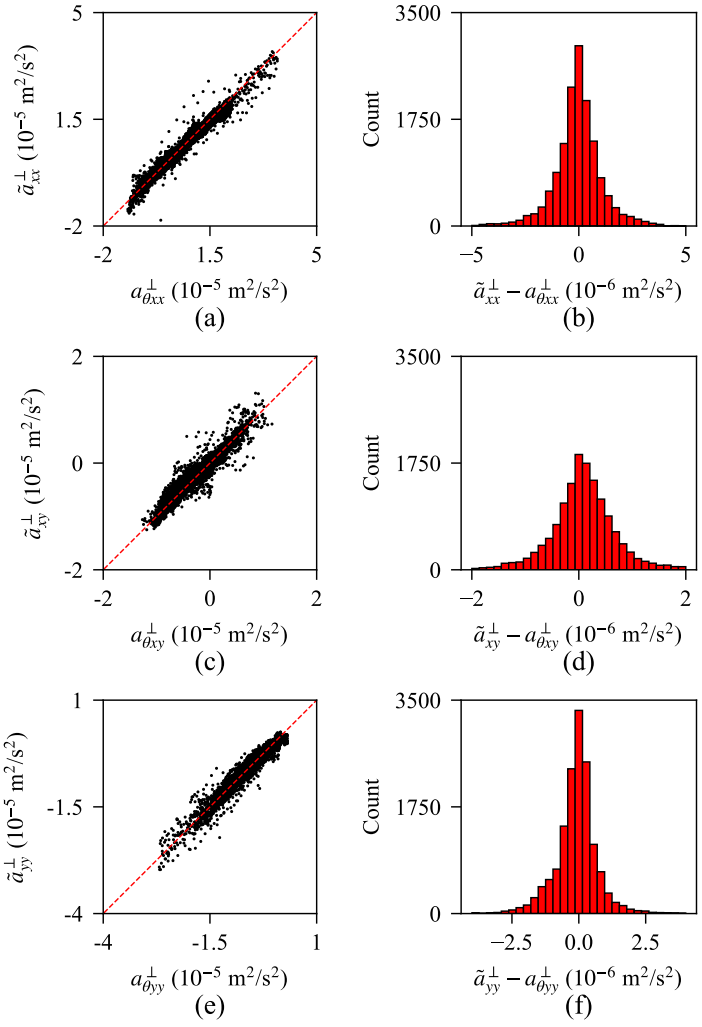


Figure 3.13: *A priori* prediction accuracy for the apNN on the $\alpha = 1.2$ test case. The prediction accuracy for each component of $a^{\perp}$ is shown: (a), (c), (e) plots of the predicted value (ordinate) vs the ground-truth value obtained from DNS (abscissa) and (b), (d), (f) distribution of the $a^{\perp}$ prediction errors.

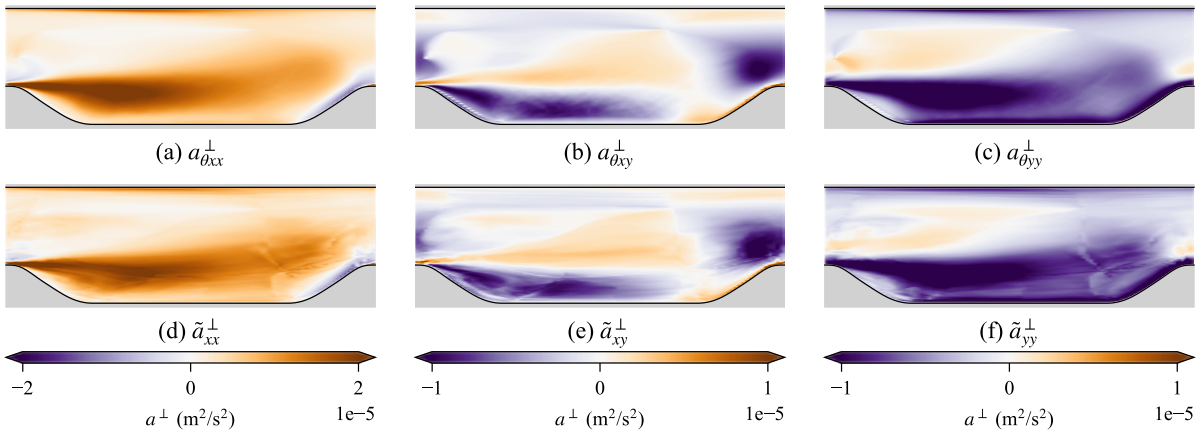


Figure 3.14: *A priori* contours of each component of $a^{\perp}$ for the test case: (a), (b), (c) ground-truth value obtained from DNS and (d), (e), (f) value predicted by the apNN.

simulation. In particular, the size of the separated region is well predicted, as well as several other important features of the flow: namely, boundary layers along the top and bottom wall, reattachment of the flow, and acceleration over the rightward hill.

Examining several slices of the velocity field in Figs 3.16 and 3.17, we see that both the U and V velocity components are captured well compared to the DNS fields. While the base RANS model (the green line in Figs 3.16 and 3.17) fails to predict the sharp transition in the U profiles above the separated region, the ML augmented solution predicts these transitions accurately. Furthermore, the U profiles after reattachment show much better agreement with the DNS solution than the base RANS model. Along the top wall, the ML augmented solution slightly underpredicts the U profile, while the base RANS model overpredicts the U profile.

While the test flow here is dominated by the streamwise flow, the V profiles in Fig. 3.17 shed light on key areas that ML augmentation improves the solution. The base RANS models under-predicts the bulge in V within the separated region, thereby indicating an over-prediction of separation. The tendency of commonly used RANS models to over-predict separation is widely known. Another key area of improvement in the solution concerns the acceleration of the flow over the right hill—the RANS model greatly under-predicts the V component along the right hill, while the corrected solution exhibits a much more accurate estimate compared to DNS.

Figure 3.18 compares the error distributions in U and V from the base RANS model to

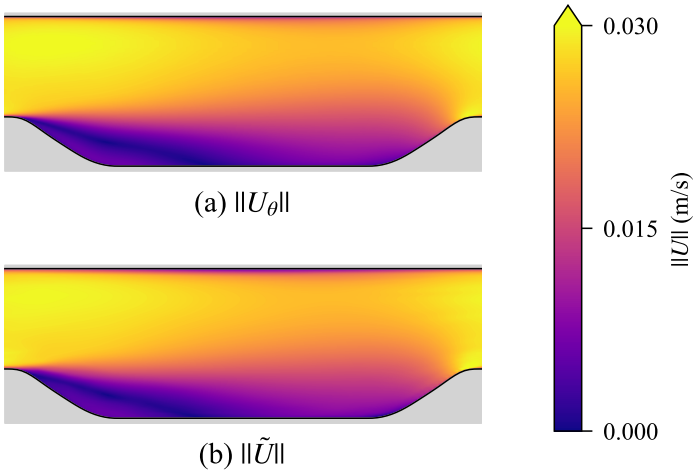


Figure 3.15: *A posteriori* contours of velocity magnitude after injecting the model predictions into the RANS equations: (a) DNS flow field and (b) result predicted by the present study. Here, $\|\tilde{U}\|$ is the magnitude of the velocity vector arising from the final converged solution, after injecting the model predictions for $\nu_t^\dagger$ and $a^\perp$.

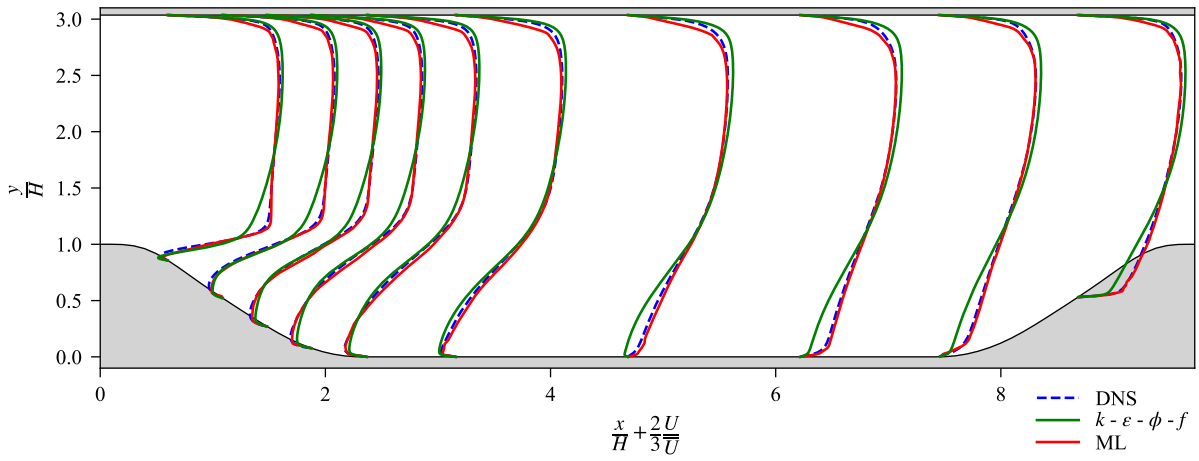


Figure 3.16: *A posteriori* samples of the U velocity component along several lines throughout the flow field.

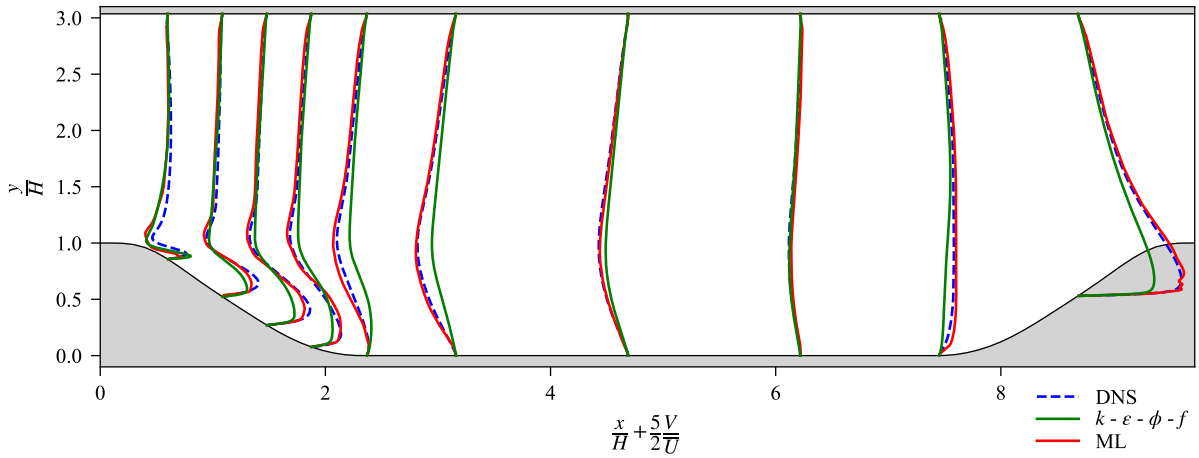


Figure 3.17: *A posteriori* samples of the V velocity component along several lines throughout the flow field.

the ML augmented solution. The base RANS model displays a tendency to over-predict the U component, demonstrated by a left skewed U error histogram. While the augmented solution contains smaller error magnitudes, it also contains this same skewed tendency. The V histograms are both roughly symmetric, with the augmented solution being slightly more skewed than the base RANS model.

Since the linear part of a continues to evolve as the solution converges, the full anisotropy tensor is an *a posteriori* quantity. Figure 3.19 compares the predicted anisotropy tensor magnitude to the DNS value. The prediction for a generally agrees well with a_θ. A slight increase in the magnitude of a as the flow accelerates over the right hill is captured, as well as the large increase in a as the flow separates over the left hill. This accurate prediction of a confirms the assumption in the present open loop framework that if $a^\perp$ and $\nu_t^\dagger$ are accurately predicted, $\vec{U}$ can evolve such that $\vec{U} \approx \vec{U}_\theta$, and therefore $S \approx S_\theta$. Finally, given $a^\perp \approx a_\theta^\perp$, $\nu_t^\dagger \approx \nu_{t\theta}^\dagger$, and $S \approx S_\theta$, we have $a \approx a_\theta$, which is the result shown in Fig. 3.19.

The conditioning problem for data-driven closures is often described as an amplification of errors in the closure term. It was of interest to determine whether this amplification was present in the solution after correcting $\nu_t^\dagger$ and $a^\perp$. Figure 3.20 compares the relative errors in the converged velocity field to the relative errors in the closure term, a. The errors in U are *reduced* compared to the errors in a. This reduction indicates that a well-conditioned closure formulation can also benefit the solution by suppressing errors in the closure term, leaving more room for errors in the machine learning model predictions. Clearly, the issue of

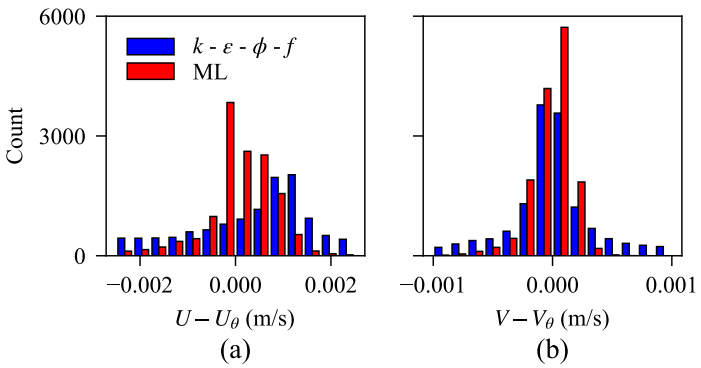


Figure 3.18: Error distribution for the U and V velocity components after injecting the neural network model predictions for $\nu_t^\dagger$ and $a^\perp$ into the RANS equations: (a) error in the U velocity component and (b) error in the V velocity component.

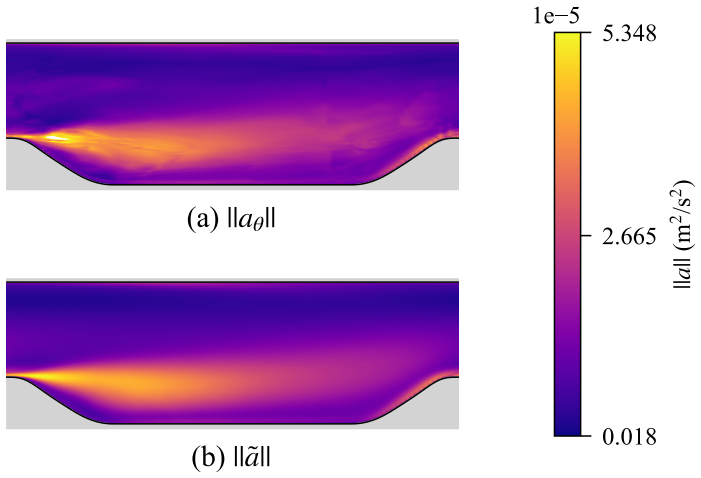


Figure 3.19: *A posteriori* contours of the anisotropy tensor magnitude: (a) ground-truth value obtained from DNS and (b) value predicted by the present study.

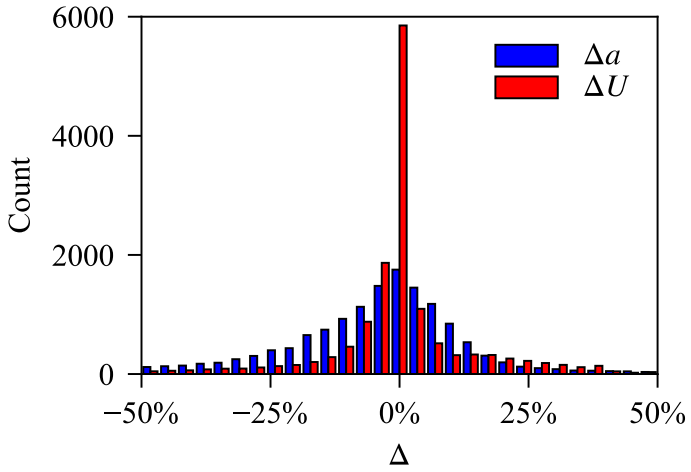


Figure 3.20: *A posteriori* comparison of conditioning error in U to conditioning errors in a. Relative error is calculated in an identical manner to Fig. 3.1.

conditioning deserves more attention in data-driven turbulence modelling. We recommend that a similar conditioning analysis be completed each time a new decomposition of the closure term is presented, or each time a new injection framework is proposed.

An important flow phenomenon in the periodic hills test case is separation. For the $\alpha = 1.2$ test case, the flow completely separates over the left hill, and reattaches along the bottom wall. The reattachment point is identified by the change in sign of the wall shear stress $\nu\frac{\partial U_n}{\partial n}$, where n is the wall-normal direction, and U_n is the velocity parallel to the wall. Figure 3.21 shows the predicted wall shear stress along the reattaching region of the bottom wall.

The reattachment point x_r in Fig. 3.21 is the location where the wall shear stress changes signs. The base RANS model predicts a delayed reattachment point, a result of over-predicting the separation. The ML corrected velocity field reattaches slightly earlier than the DNS field. Nevertheless, the predicted reattachment point is much closer after applying the ML augmentation. The relative error in x_r for the base RANS model is 20%, while the error in the ML corrected solution is significantly reduced to 8.9%.

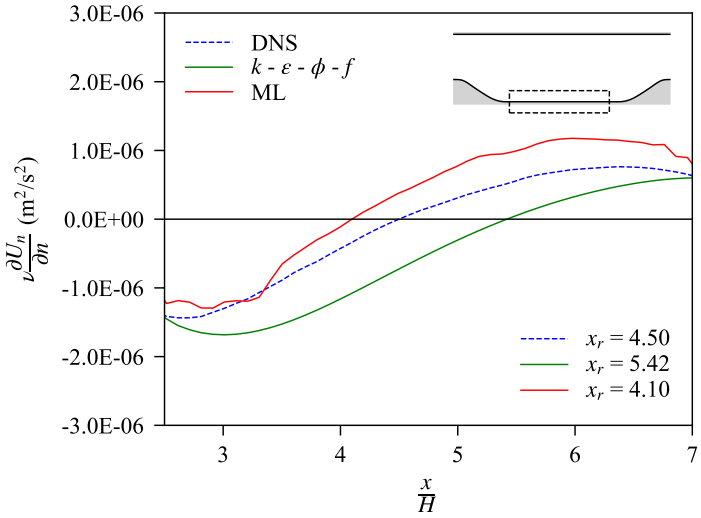


Figure 3.21: Wall shear stress for a portion of the bottom wall. The area considered is shown in the top right corner of the figure. The reattachment position (i.e., the location at which the wall shear stress changes sign) is summarized in the bottom right corner.

3.4.3 Extrapolation test case

After observing an improvement of the mean velocity field for the $\alpha = 1.2$ test case, it was of interest to determine whether this improvement would also be observed for a more challenging test case. The $\alpha = 1.2$ test case is an interpolation test case in the sense that values of α smaller and larger than 1.2 were part of the training set. A recent update to the original periodic hills dataset was provided by Xiao et al. [44]. This extended periodic hills dataset includes several additional geometries. While the original dataset only included variations in the slope steepness, the updated dataset now also includes variations in both the domain height and the length of the flat portion between the hills. For the extrapolation test, a case with $\alpha = 0.5$, $y_{\max} = 2.024H$, and a flat length of $4.071H$ between the hills was selected. This case is outside the bounds contained within the training dataset. As a consequence, this challenging test case is considered an extrapolation in terms of the case geometry.

Figure 3.8 compares the selected test case geometry to the steepest and shortest training case. This figure shows that the test case is much shorter in both height and length than the $\alpha = 0.5$ training case. This $\alpha = 0.5$ case is the most aggressive training case in terms of separation, and therefore the selected extrapolation test case is a challenging case for

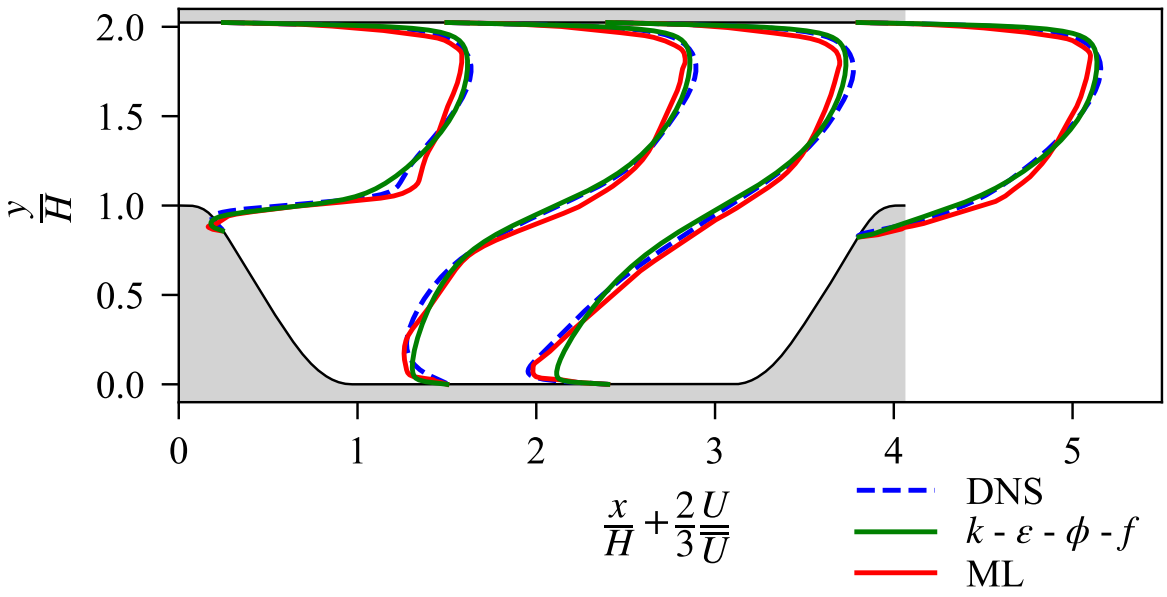


Figure 3.22: *A posteriori* samples of the U velocity component along several lines throughout the flow field for the extrapolation test case.

the model.

For the extrapolation test case, the same procedure used in Section 3.4.2 was used to predict the $\nu_t^\dagger$ and $a^\perp$ fields from a converged base RANS solution using the k-ε-ϕ-f model. These predicted fields were then injected into the momentum equation in the same manner as described in Section 3.4.2. Figure 3.22 compares the resulting velocity fields for the extrapolation test case.

Figure 3.22 shows that the nearly uniform velocity field improvement observed for the interpolation test case does not carry forward for the extrapolation test case. The machine learning augmented solution for the extrapolation test case shows limited improvement over the base RANS model near the upper wall. The training dataset lacked any variation in domain height, and therefore the severely reduced domain height of the test case is accounted for inaccurately by the model. However, the training dataset did contain a variety of separated flows, and the model is still able to provide improved prediction of the velocity fields in the separated region.

Further improvements to the generalization performance of the model may be possible

through the inclusion a richer training dataset. The periodic hills dataset used in the present study was selected for the purpose of developing and demonstrating the proposed model framework. For this reason, it was desired to limit the computational demands during training and testing. However, the dataset presented by McConkey et al. [79] contains several other flow configurations. The inclusion of a broad mix of flow cases into the training dataset is an important step toward developing a "universal" machine learning driven turbulence model, a challenging task that is not within the scope of the present work.

3.4.4 Interpreting the data-driven corrective closure model

Given the rich input feature set used by the two neural networks in the present work, it was of interest to determine which features were found to be important by the model. In deep learning, this analysis is referred to as "interpretability"—taking a look inside the neural network black box. For this purpose, Shapley additive explanation (SHAP) values are commonly used to interpret machine learning models. The SHAP package by Lundberg and Lee [86] was used to collect the SHAP values for a range of $\nu_t^\dagger$ predictions by the EVNN. The EVNN was selected for an interpretability analysis, because it is the principal neural network in the present model. The eddy viscosity prediction is a critical injected component. SHAP values are used to determine the relative incremental contribution of each input feature on the prediction, while accounting for non-linear interactions between features.

Summation of the SHAP values for all predictions can be used to determine the global relative importance of each input feature. Figure 3.23 shows the relative importance of each input feature used in the EVNN. On a global basis, the use of ϕ as an input feature (made possible by the use of the ϕ-f model) was highly valuable, as the EVNN ranked it as the most important feature. Furthermore, the heuristic scalars (q_1, q_2, q_3) also ranked relatively high in terms of importance. The rest of the input features consists of various combinations of the strain rate, rotation rate, k gradient, and v^2 gradient. The additional invariant I_2 used to extract additional features in the present work also added useful input features. Interestingly, there is no general correlation between the usefulness of I_1 and I_2 for a given basis tensor—the feature $I_1(B_7)$ ranked third, but $I_2(B_7)$ ranked twenty-fifth. The opposite is true for B_5—the second invariant ranked much higher than the first invariant.

Figure 3.24 shows the relative local SHAP values for the EVNN at several important points in the flow. The SHAP values for the eight most important features from Fig. 3.23, normalized by the maximum value at each location, are exhibited in this figure. The locations are annotated in Fig. 3.24(a), and are summarized as follows: (b) the bulk flow region

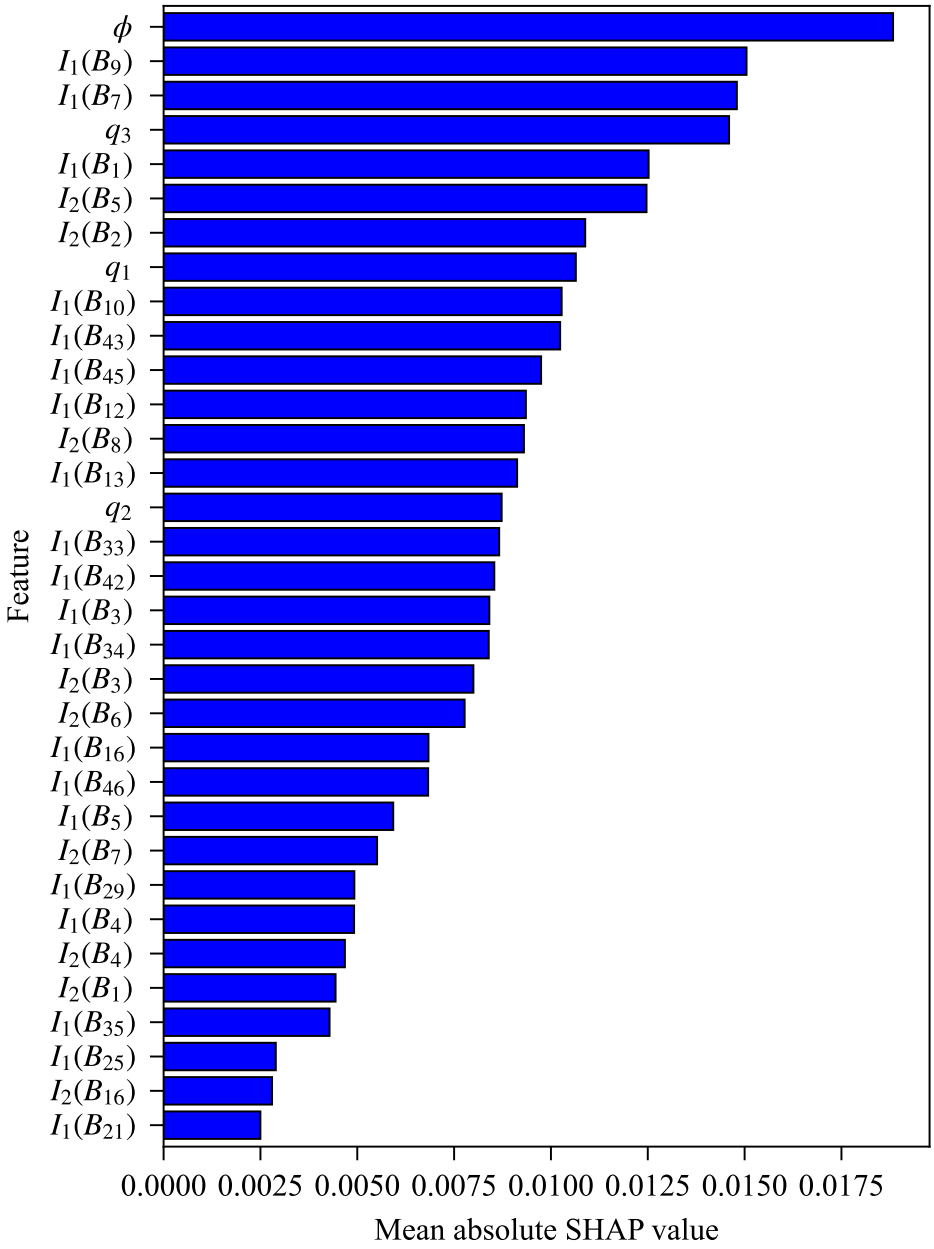


Figure 3.23: Mean absolute SHAP values for each feature over the entire test case. This value is used to measure the importance of each feature. The values are sorted from highest relevance (ϕ) to lowest relevance ($I_1(B_{21})$). The SHAP values in this plot were calculated using the EVNN.

above the separated region; (c) the upper wall above the reattached and developing flow; (d) the accelerating flow above the right hill; (e) the lower wall just after reattachment; and, (f) the separated region. A blue colour indicates a positive influence on the $\nu_t^\dagger$ prediction, while a red colour indicates a negative influence. The sign indicates the direction each input feature "nudges" the output prediction, while the magnitude indicates the amount of influence.

Figure 3.24 shows that while some regions of the flow may feature similar phenomena (e.g., (b) and (g)), the relative importance of each feature is distinct. Though locations (c), (d), and (e) are all near the boundary layer, the magnitude of the q_1 importance is relatively low. Though q_1 is an input feature designed to detect near-wall regions, q_1 does not strongly influence $\tilde{\nu}_t^\dagger$ near the wall. At the top wall (c), the anisotropy measure ϕ is the most important feature. On the contrary, as the flow develops along the bottom wall at (e), ϕ is less important than some higher-order gradients of the mean field. In the separated region (f), the ratio of total to normal Reynolds stress (q_3) is significantly more important than the other input features. This observation justifies the heuristic basis for including q_3. As the flow recirculates along the bottom wall, it is possible that the EVNN has learned to associate a particular change in q_3 with separation. The anisotropy measure ϕ was not found to be as important in the separated region, but was important for the near-wall points (c) and (e).

3.5 Conclusion

Applying machine learning to develop data-driven turbulence closures is a promising method to improve the accuracy of RANS simulations. However, a major issue for such models is the conditioning of the RANS equations after injecting a prediction for the closure term. If the framework is ill-conditioned, any errors made in the closure term could be amplified, and result in large errors in the converged mean field.

The decomposition of τ (and, in turn, a) proposed in this work was shown to have good conditioning, through an analysis similar to Brener *et al.*'s [76]. By injecting the training labels (calculated from DNS), rather than the model predictions, the resulting mean field provides the "upper limit" achievable at testing time. We demonstrated that decomposing a into an optimal eddy viscosity ($\nu_t^\dagger$) and a remaining non-linear part ($a^\perp$) produced a well-conditioned closure. We also motivated the inclusion of the non-linear part of a through evaluating the best prediction by a purely linear model.

We impose the often relaxed requirement of Galilean invariance on all input features. The input feature set was augmented by using an additional invariant to the one previously

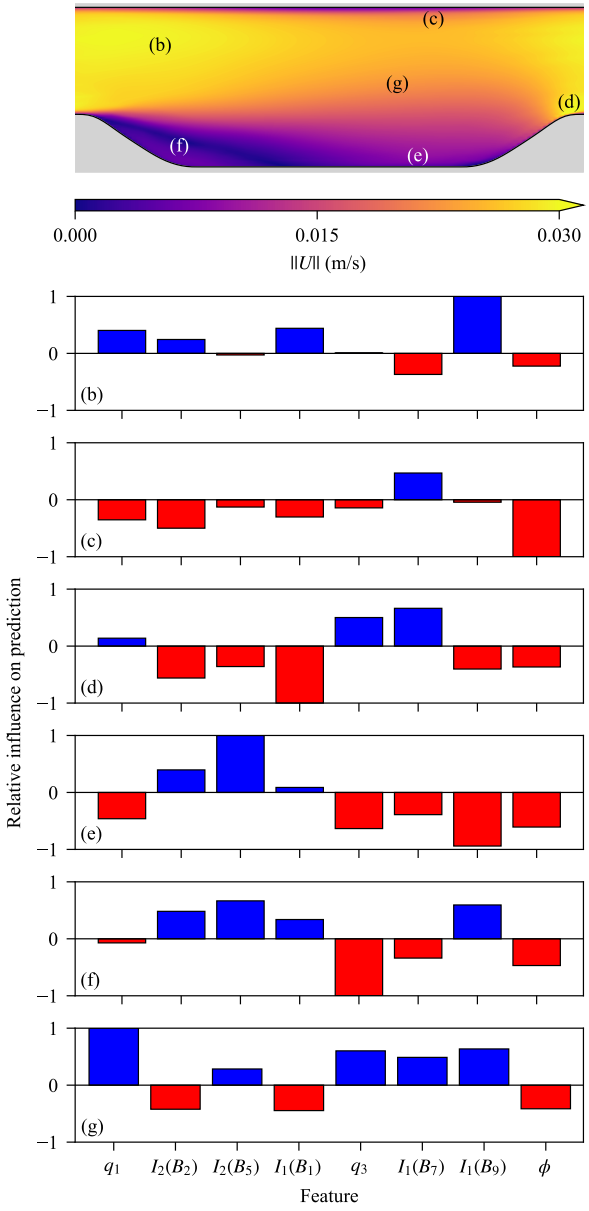


Figure 3.24: Relative feature importance at six locations in the flow: (a) locations in the flow domain and (b)–(g) bar plots showing the relative feature importance for the eight most important features in Fig. 3.23. The relative influence on the prediction is calculated by normalizing the SHAP values by the highest SHAP value within the top eight features at the given location.

used in the literature. The minimal integrity basis for the strain rate, rotation rate, k gradient, and p gradient consists of 47 tensors, and the first invariant yields a scalar for each of these basis tensors. For a complex three-dimensional flow, using the second tensor invariant can effectively double the number of input features arising from this tensor basis to 94. For the two-dimensional flow considered here, we showed that many of these basis tensors are zero (Appendix B), an outcome not previously discussed in detail. However, the input feature set used in the present work is still one of the richest feature sets applied for data-driven turbulence modelling to date. To determine which input features were found to be important, an interpretability framework (SHAP) was also applied to explain the predictions for the optimal eddy viscosity. Using this framework, it was found that the three heuristic scalars q_1, q_2 and q_3 ranked high along with ϕ (normalized wall-normal Reynolds stress) for the prediction of the optimal eddy viscosity.

Using one neural network to predict $\nu_t^\dagger$ and another to predict $a^\perp$, we demonstrated that these quantities are predictable with reasonable accuracy for a periodic hills dataset. Even after introducing model error into the predictions, the resulting mean fields agree well with the DNS results. *A posteriori* predictions of both the velocity field and the anisotropy tensor were demonstrated to agree well with the DNS results. Furthermore, the framework proposed here has a *smaller* error in the velocity field than the closure prediction. This result is significant because it shows that under certain conditions, the RANS equations can suppress rather than amplify errors in the closure prediction.

In the present work, we also demonstrated that machine learning augmentation can be used to improve even a sophisticated RANS model. The base turbulence model used was the ϕ-f model, a type of v^2-f model with three transport equations and one elliptic relaxation equation. The majority of previous studies has focused on simpler turbulence closure models, such as the standard k-ε model or the k-ω model. This result indicates that machine learning augmentation has the potential to improve the results from even more sophisticated and recent RANS models, such as the elliptic blending (EB) k-ε-lag model [87]. Since the field ϕ was found to be a highly useful feature, we conclude that more sophisticated models may benefit even more from machine learning augmentation than the simpler RANS models. The wider set of mean fields for forming a complex closure relationship may provide a richer description of the flow, yielding more information for a machine learning model to use in predicting quantities of interest. This idea could be extended to develop an augmented Reynolds stress transport model, where an even broader set of mean fields are available.

Future work includes further optimizing the non-linear part of the anisotropy tensor $a^\perp$. While the present work imposes strict invariance quantities for the input features, the direct prediction of the components of $a^\perp$ means that the outputs do not possess the

same invariance quantities. Possible options for remedying this discrepancy include using a tensor basis neural network (TBNN) for $a^{\perp}$ to construct neural network for $a^{\perp}$ that is invariant with respect to coordinate transformations (e.g., rotations, reflections). Further work also includes applying this framework to a broader set of flows. While the periodic hills dataset used here sufficiently demonstrated the merits of this framework, using a wider range of flows could provide additional useful information, especially when interpreting the closure model. For example, an analysis of the importance of various features for distinct flows could guide the input feature selection for a sophisticated three-dimensional machine learning closure model. Finally, another avenue for investigation involves implementation of the deep structured neural network for turbulence closure modelling within a closed-loop framework. For this iterative procedure, each iterative RANS computation produces updated values of the features, triggering the neural network to give an updated Reynolds stress tensor which is then injected into the next RANS computation until convergence.

Chapter 4

On the generalizability of machine-learning-assisted anisotropy mappings for predictive turbulence modelling

Associated reference:

R. McConkey, E. Yee, and F. S. Lien, "On the Generalizability of Machine-Learning-Assisted Anisotropy Mappings for Predictive Turbulence Modelling", *International Journal of Computational Fluid Dynamics* 36(7), 555-577 (2023), https://doi.org/10.1080/10618562.2022.2113520.

Abstract

Several machine learning frameworks for augmenting turbulence closure models have been recently proposed. However, the generalizability of an augmented turbulence model remains an open question. We investigate this question by systematically varying the training and test sets of several models. An optimal three-term tensor basis expansion is used to develop a model-agnostic data-driven turbulence closure approximation. Then, hyperparameter optimization was performed for a random forest, a neural network, and an eXtreme Gradient Boosting (XGBoost) model. We recommend XGBoost for data-driven turbulence closure modelling owing to its low-tuning cost and good performance. We also find that

machine learning models generalize well to new parametric variations of flows seen in the training dataset, but lack generalizability to new flow types. This generalizability gap suggests that machine learning methods are most suited for developing specialized models for a given flow type, a problem often encountered in industrial applications.

4.1 Introduction

Reynolds-averaged Navier-Stokes (RANS) simulations of turbulent flows are frequently used to help design industrial systems [88]. Turbulence modelling focuses on the approximation of the unclosed term in the RANS mean momentum transport equation: namely, the Reynolds stress tensor. Though eddy-viscosity based models are perhaps the lowest fidelity turbulence closure methods in common use, they are still the "workhorse" of industrial simulations [5]. Higher-fidelity RANS closure methods such as Reynolds stress transport models (RSTM) are often bypassed in favour of large-eddy simulation (LES) due to various reasons such as numerical stability concerns. However, the major issue with LES is the high computational cost [6]. While technological advances help expand the speed and scale of available computational resources, the large difference in the computational cost between RANS and LES implies that RANS will continue to be used for the foreseeable future [5]. Unfortunately, eddy-viscosity based RANS models have several well-known deficiencies that affect their accuracy [3, 31, 89].

While the Reynolds stress tensor is the unclosed term in the RANS mean momentum equation, the anisotropic portion of this tensor is the portion responsible for momentum transport. Therefore, the majority of closure models are formulated in terms of the anisotropy tensor, rather than the Reynolds stress tensor. The isotropic component of the Reynolds stress tensor is related to the turbulent kinetic energy, and can be combined with pressure to form the modified pressure. The remaining anisotropic portion of the Reynolds stress tensor is of principal interest in closure modelling, since it is the unclosed influence of turbulence on mean momentum balance.

A promising method to overcome the deficiencies in RANS modelling is the application of machine learning to model the turbulence closure relationships [17, 12]. Though these closure relationships have been the focus of decades of RANS turbulence modelling, they remain the principal weakness in industrial RANS. Recent advances in machine learning [23, 33, 35] and the recent availability of open-source turbulence datasets have accelerated the progress in this area [24, 25, 26, 79]. Machine learning has also been used to augment RSTM [90, 91] as well as subgrid-scale modelling in LES [92]. Within the scope of machine learning augmented eddy-viscosity modelling, a number of open questions remain.

How should the model predictions be injected into the RANS equations? How well do these models generalize? What machine learning algorithm is best suited for this problem? Investigations in this area have overwhelmingly focused on the first question of injecting the machine-learnt Reynolds stress tensor into the RANS framework—this question is undoubtedly the most important starting point. However, few studies have addressed the latter two questions. Perhaps the next most important question that needs to be addressed is the generalizability of data-driven turbulence closure models.

In this study, we evaluate models from three classes of machine learning algorithms for regression on tabular data: namely, neural network models, random forest models, and gradient boosting models—more specifically, eXtreme Gradient Boosting (XGBoost) [29] models. To compare these models, we propose a new model-agnostic data-driven turbulence closure framework. We also perform a systematic generalizability investigation where we vary the flows contained in the training dataset. Our main findings are to recommend XGBoost for data-driven turbulence closure modelling, and to demonstrate a generalizability gap that exists within machine learning turbulence closures. This work is limited to an *a priori* investigation (i.e., no machine-learnt predictions of the Reynolds stresses are injected into the RANS model) in order to allow for a more in-depth analysis of the generalizability and algorithm selection.

The model comparison investigation in the present work mirrors typical machine learning studies where several models are compared on an identical problem. These types of studies are not common within data-driven turbulence modelling, as most studies are focused on presenting new holistic injection frameworks, and the majority of effort is therefore invested in implementing and testing the injected fields. The machine learning algorithm used for the turbulence closure mapping is almost always chosen to be a neural network or random forest [23, 93, 39, 68], despite the machine learning community recommending gradient boosted tree ensembles (e.g., XGBoost, CatBoost, LightGBM) for tabular data regression [94]. In terms of model structure, our framework aims to produce a model-agnostic tensor basis closure framework. For the present work, this allows us to compare various algorithms for predicting the basis tensor coefficients. However, decoupling the machine learning algorithm from the turbulence closure approximation also allows us to take advantage of new tabular data algorithms. The rapid pace of machine learning innovation means that new tabular data algorithms are likely to be developed, and a model-agnostic closure framework allows modular "swapping" of the machine learning algorithm itself. Other augmented closure frameworks such as the tensor basis neural network (TBNN) [23] and the tensor basis random forest (TBRF) [35] are not model-agnostic, and require more adaptation as machine learning algorithms evolve.

This paper is structured as follows: Section 4.2 describes the machine learning method-

ology including introducing the closure framework (4.2.2), the machine learning algorithms (4.2.3), input features (4.2.4), dataset (4.2.5), hyperparameter optimization procedure (4.2.6), and code (4.2.7). Section 4.3 first presents the calculated optimal basis coefficients (4.3.1) and then summarizes the machine learning model predictions for these coefficients (4.3.2). The generalizability investigation is presented in Section 4.3.2, with additional results in Appendix D. Conclusions are given in Section 4.4.

4.2 Methodology

The RANS continuity and mean momentum equations are given by Equations (4.1) and (4.2):

$$\nabla \cdot \vec{U} = 0 \ , \tag{4.1}$$

$$\nabla \cdot (\vec{U}\vec{U}) = -\nabla p + \nu \nabla^2 \vec{U} - \nabla \cdot \tau \ , \tag{4.2}$$

where $\vec{U} \equiv (U, V, W)$ is the steady-state mean velocity field, ν is the molecular kinematic viscosity of the fluid, and p is the pressure. Here, the unclosed term τ is the Reynolds stress tensor. Predicting the Reynolds stress tensor as a function of the other mean flow variables in Equation (4.2) is the turbulence closure problem. Often, the Reynolds stress tensor is decomposed into an isotropic ($(2k/3)\delta_{ij}$) and an anisotropic (a) component where k is the turbulent kinetic energy (TKE) and δ_{ij} is the Kronecker delta function. Then, the mean momentum equation is written as

$$\nabla \cdot (\vec{U}\vec{U}) = -\nabla \left(p + \frac{2}{3}k \right) + \nu \nabla^2 \vec{U} - \nabla \cdot a \ , \tag{4.3}$$

where $p' \equiv (p + 2k/3)$ is the modified pressure and a is the dimensional anisotropy tensor. Moreover, a can be non-dimensionalized as $b = a/(2k)$, where b the non-dimensional anisotropy tensor (referred to simply as the anisotropy tensor in the present work). In view of this, the mean momentum equation is given by

$$\nabla \cdot (\vec{U}\vec{U}) = -\nabla \left(p + \frac{2}{3}k \right) + \nu \nabla^2 \vec{U} - \nabla \cdot (2kb) \ . \tag{4.4}$$

The turbulence closure problem is now focused on the prediction of b and k. For the majority of data-driven closure modelling, k is modelled using the familiar transport and scale-determining equation, while b is the focus of a complex closure relationship. Section 4.2.1 provides more details concerning the modelling of b.

4.2.1 Machine learning for RANS closure modelling

Several recent investigations have presented various ways to augment RANS simulations using machine learning [62, 23, 33, 35, 66]. In the present work, we focus on generalized eddy-viscosity modelling. A seminal work in this field was the introduction of the tensor basis neural network (TBNN) by Ling et al. [23]. The goal of the TBNN is to predict the anisotropy tensor b using the general ten-term tensor basis expansion originally proposed by Pope [28]:

$$b = \sum_{n=1}^{10} g^{(n)} T^{(n)} \ , \tag{4.5}$$

where $g^{(n)}$ are scalar coefficients for the following basis tensors $T^{(n)}$ $(n = 1, 2, \ldots, 10)$:

$$\begin{aligned}
T^{(1)} &= S \ , & T^{(6)} &= R^2 S + S R^2 - \tfrac{2}{3} I \mathrm{tr}(S R^2) \ , \\
T^{(2)} &= SR - RS \ , & T^{(7)} &= RSR^2 - R^2 SR \ , \\
T^{(3)} &= S^2 - \tfrac{1}{3} I \mathrm{tr}(S^2) \ , & T^{(8)} &= SRS^2 - S^2 RS \ , \\
T^{(4)} &= R^2 - \tfrac{1}{3} I \mathrm{tr}(R^2) \ , & T^{(9)} &= R^2 S^2 + S^2 R^2 - \tfrac{2}{3} I \mathrm{tr}(S^2 R^2) \ , \\
T^{(5)} &= RS^2 - S^2 R \ , & T^{(10)} &= RS^2 R^2 - R^2 S^2 R \ .
\end{aligned}$$

Here, I is the identity tensor, $\mathrm{tr}(\ \cdot\ )$ is the trace operator, and the basis tensors are constructed from the non-dimensional mean strain- and rotation-rate tensors defined, respectively, as follows:

$$S \equiv \frac{T_t}{2} \left(\nabla U + \nabla U^{\mathrm{T}} \right) \ , \tag{4.6}$$

and

$$R \equiv \frac{T_t}{2} \left(\nabla U - \nabla U^{\mathrm{T}} \right) \ , \tag{4.7}$$

where T_t is a turbulent time scale (cf. Equation (4.17)) and the superscript T denotes tensor transposition.

In Ling et al.'s investigation [23], the machine learning algorithm used was a deep neural network model. This algorithm was also used in subsequent TBNN-based investigations conducted by Song et al. [39] and by Zhang et al. [37]. However, more recent investigations have often relied on random forest predictors. Kaandorp and Dwight [35] developed the tensor basis random forest (TBRF) which is essentially the random forest analogue to the TBNN proposed by [23]. Other notable work in augmented eddy-viscosity modelling includes the investigation conducted by Wu et al. [33] which uses a model that predicts the components of an eigendecomposition of the anisotropy tensor. McConkey et

al. [93] decomposed the anisotropy tensor into optimal linear and nonlinear components. Two neural networks (one per component) were then used to learn the turbulence closure relationship for the anisotropy tensor on a periodic hills dataset.

In parallel with frameworks for predicting various forms of the anisotropy tensor b, Cruz et al. [75] proposed the prediction of the Reynolds force vector (RFV) $\nabla \cdot \tau$ rather than τ itself. Since the divergence of the Reynolds stress tensor (viz., the vector $\nabla \cdot \tau$) appears in the mean momentum equation, the closure model can be simplified if this vector is predicted directly. This framework avoids propagating any errors associated with τ through the divergence operator, and has been demonstrated to be another promising way to augment RANS simulations. Recently, Brener et al. [95] further improved this strategy, showing that it is a promising approach for data-driven closure modelling. In terms of the machine learning algorithm, Brener et al.[95] chose to use a random forest methodology. However, a disadvantage of the RFV approach is that the training dataset needs to contain various derivatives of the Reynolds stress tensor, quantities which are often not included in direct numerical simulation (DNS) datasets.

Aside from the closure relationship itself, which has varied significantly between investigations, the injection procedure is another key ingredient for a data-driven closure. Perhaps the most important issue for injection is the conditioning problem. If the RANS equations are ill-conditioned, errors in the predicted turbulence closure term can be amplified and result in large errors in the predicted velocity field. While originally thought to be solved by using an implicit treatment of the eddy viscosity [77], Brener et al. [76] showed recently that the conditioning problem can be resolved by incorporation of the information concerning the high-fidelity velocity field. As this finding is relatively recent, few injection procedures currently make use of this key result [96, 93, 95]. Though injection into the momentum equation is not considered in the present work, we base our closure and machine learning procedure on a one-time correction (open-loop) closure framework [81]. In an open-loop closure framework, the machine learning model is called to correct the closure term based on a converged RANS solution. Then, these fields are injected and fixed in the RANS equations until a new, ideally more accurate solution for the velocity field is obtained. Most previous work in augmented RANS modelling has focused on one-time correction. Closed-loop frameworks have also been proposed [66], though the machine learning procedure and injection process varies for each of these frameworks. The key contributions of the present paper (namely, algorithm selection and generalizability) often receive less attention than the injection formulation, despite their importance to the long-term goals of data-driven turbulence modelling.

4.2.2 Optimal tensor basis expansion for the Reynolds stress anisotropy

The core assumption of the eddy-viscosity paradigm is that the anisotropy tensor is related to the mean velocity gradients [3]. The most common linear-eddy viscosity model (LEVM) is formulated as

$$b = g^{(1)}T^{(1)} = -C_\mu S \ , \tag{4.8}$$

where C_μ is a global constant (closure) coefficient (usually, $C_\mu = 0.09$).

However, other relationships have been proposed. These include nonlinear eddy-viscosity models (NLEVM) involving higher-order terms of the mean strain and rotation rate tensors. Motivated by deficiencies in the LEVM approximation, several NLEVM models have been previously proposed (e.g. Craft et al. [97], Lien et al. [98], Shih et al. [99]), with most NLEVM models containing quadratic or cubic terms. In traditional turbulence modelling, the inclusion of additional terms means tuning additional (closure) coefficients, a process that becomes increasingly difficult as the number of terms increases. Pope [28] showed that a complete basis for three-dimensional flows involves ten tensors, up to fifth order in terms of strain- and rotation-rate tensors. A fifth-order eddy-viscosity model with ten closure coefficients $g^{(n)}$ $(n = 1, 2, \ldots, 10)$ would be insuperably difficult to tune using the conventional turbulence modelling paradigm.

The investigation conducted by Ling et al. [23] revitalized the turbulence closure modelling by using a neural network to predict the ten coefficients in the ten-term "general nonlinear eddy-viscosity model" proposed by Pope [28]. This ten-term model has been used widely in frameworks with a tensor basis expansion at the core of the closure relationship.

While the ten-term tensor basis used in several previous studies [23, 35, 39] forms a complete basis for the anisotropy tensor for three-dimensional turbulent flows, a basis with fewer terms is sufficient to form a complete basis in two-dimensional flows. Gatski and Jonger [100] showed that a tensor basis with just three terms is sufficient to model exactly the anisotropy tensor for two-dimensional (2D) flows, and this three-term basis can be interpreted as a least-squares approximation of the complete ten-term basis form for the anisotropy tensor associated with general three-dimensional (3D) flows [100]. All of the classical nonlinear eddy-viscosity models contain less than ten terms, making them simpler than the general case and easier to tune. However, the three-term basis has received little attention in machine learning augmented turbulence closure modelling, despite the simplicity and theoretical justification for it, and despite the fact that the majority of investigations have focused on two-dimensional flows (which only require a three-term basis). We therefore use a three-term tensor basis for the anisotropy tensor. This tensor

basis expansion for b assumes the following form:

$$b = g^{(1)}T^{(1)} + g^{(2)}T^{(2)} + g^{(3)}T^{(3)} \tag{4.9}$$

$$= g^{(1)}S + g^{(2)}(SR - RS) + g^{(3)}(S^2 - \tfrac{1}{3}I\mathrm{tr}(S^2)) \ . \tag{4.10}$$

Using such a basis, the closure problem simplifies to the determination (specification) of the basis tensor coefficients $g^{(n)}$ in Equation (4.9). By far, the most common approach is the linear-eddy viscosity model. Within this framework, a linear-eddy viscosity model is generated by specifying a global closure constant $g^{(1)} = -C_\mu$ (typically, $C_\mu = 0.09$), and setting $g^{(2)} = g^{(3)} = 0$. However, there are two principal deficiencies with this approach: namely, a strictly linear relation between the anisotropy tensor and the mean strain-rate tensor often breaks down [3], and the optimal values of the $g^{(n)}$ coefficients are functions of the spatial coordinates—so, assuming that these coefficients are global constants introduces predictive inaccuracy. The latter disadvantage applies to both linear models (where techniques such as damping functions help adjust C_μ near the wall) and several nonlinear models [99, 98, 97].

To allow both a spatially-varying set of coefficients $g^{(n)}$ and a nonlinear representation for b in terms of the mean velocity gradients, we propose an *optimal* tensor basis expansion. In this expansion, the optimal coefficients $g^{(n)}$ are the solution of the following constrained multiple objective least-squares problem:

$$\begin{aligned} &\min_{g^{(n)}}\Big(||b_\theta - \sum_n g^{(n)}T^{(n)}||^2 + \lambda||g^{(n)}||^2\Big) \ , \quad \text{subject to} \\ &\quad g^{(1)} \leq 0 \ , \\ &\quad -\tfrac{1}{3} \leq \sum_n g^{(n)}T^{(n)}_{ii} \leq \tfrac{2}{3} \quad i = 1,2,3 \ , \\ &\quad -\tfrac{1}{2} \leq \sum_n g^{(n)}T^{(n)}_{ij} \leq \tfrac{1}{2}, \quad i,j = 1,2,3; \ i \neq j \ , \end{aligned} \tag{4.11}$$

for $n = 1, 2, 3$, $\| \cdot \|$ denotes the Euclidean (L2) norm, the subscript θ indicates a quantity obtained from either DNS or LES, and λ is a tuning (regularization) parameter that adjusts the relative importance of each component of the objective function. Based on a trade-off study, we used $\lambda = 1 \times 10^{-5}$, which provided a good balance between an accurate approximation of b_θ and well-behaved coefficients.

It is noted that the formulation in Equation (4.11) can also be extended to an arbitrary tensor basis. In words, the formulation in Equation (4.11) seeks to obtain the solution to the following optimization problem: namely, find the solution that represents the best

compromise between minimizing the errors in the approximation of the anisotropy tensor while constraining the magnitude of the coefficients $g^{(n)}$ to be as small as possible (viz., shrinking the values of these coefficients toward zero). In particular, the objective function includes an L2 regularization (penalty) term that constrains the norm squared of the optimized coefficients $g^{(n)}$. We found this regularization term was crucial for stabilizing and reducing the magnitude of these coefficients, thereby increasing their useability as machine learning prediction targets.

The following constraints are also applied in Equation (4.11): namely, (1) ensure a non-negative eddy viscosity for simulation stability and (2) ensure a realizable anisotropy tensor through the application of realizability constraints on the various components of b. The first constraint is motivated by the fact that the term $g^{(1)}T^{(1)} = -C_\mu S$ is usually treated implicitly in a RANS simulation. This implicit treatment is achieved by the use of an "effective" viscosity, $\nu_{\text{eff}} = \nu + \nu_t$, where ν is the molecular kinematic viscosity and ν_t is the eddy (turbulent) viscosity. Using the common turbulent timescale $T_t = k/\varepsilon$, we have $\nu_t = C_\mu \frac{k^2}{\varepsilon}$. The constraint $g^{(1)} \leq 0$ guarantees $\nu_t \geq 0$ (assuming k is positive), an important target for stability in a RANS simulation. The remaining constraints arise from bounds on the anisotropy tensor itself in order to guarantee realizability [58].

The optimal tensor basis expansion shown here is also invariant. The invariance of such tensor basis expansions motivated the use of the tensor basis neural network (TBNN) by Ling et al. [23] and the subsequent application of the tensor basis random forest (TBRF) by Kaandorp and Dwight [35]. However, our framework differs in one major way from these previous tensor basis models: namely, the optimal coefficients are calculated before training, and the model is trained to predict these coefficients directly. In principle, this simplifies the machine learning procedure by isolating the problem of optimizing $g^{(n)}$ and the model prediction of these optimal coefficients via the input features. This optimization problem is treated implicitly within the machine learning procedure of the TBNN and TBRF—therefore, control and insight into this optimization problem is lost. Further investigation will be required to determine the advantages of isolating this optimization problem from the machine learning process.

In the calculation of $g^{(n)}$ in Equation (4.11), the only quantity that comes from DNS or LES is the anisotropy tensor b_θ. Therefore, the optimal $g^{(n)}$ coefficients approximate b_θ using $T^{(n)}$ from RANS, because only $T^{(n)}$ from RANS will be available at injection time. An alternative method to calculate the optimal $g^{(n)}$ coefficients would be to solve Equation (4.11) using $T_\theta^{(n)}$ (DNS or LES velocity gradients). However at injection time, the error $\epsilon = T_\theta^{(n)} - T^{(n)}$ will then be propagated and result potentially in larger errors in b. Therefore, using $T^{(n)}$ from RANS in Equation (4.11) embeds an implicit error-correction into the optimal $g^{(n)}$ coefficients, so that we do not rely on the implicit assumption that

$T^{(n)} \approx T_\theta^{(n)}$ when the data-driven turbulence closure model is injected into the RANS model.

4.2.3 Machine learning models

A number of machine learning models were used in the present work to predict the optimal tensor basis coefficients $g^{(1)}$, $g^{(2)}$, and $g^{(3)}$. As discussed in Section 4.2.2, these scalars were chosen as machine learning targets due to their optimal and realizable representation of the anisotropy tensor, and the embedded invariance of this representation.

Previous data-driven closure investigations have used a variety of machine learning model types. The usual data-driven turbulence closure modelling problem is an example of a "tabular data" regression problem. Within the turbulence modelling community, the majority of studies utilize neural networks and random forests for solving this tabular data problem. However, recent investigations in the broader machine learning field have revealed that neural networks are often not the optimal model type for tabular data. While neural networks excel at complex tasks such as computer vision and language processing, they are often outperformed by gradient boosting models on tabular data [94]. Specifically, XGBoost is a popular tree-based gradient boosting framework for tabular data [29]. XGBoost is an ensemble learning method in which a decision tree ensemble is created. As implied by the name, this ensemble is created by gradient boosting techniques. In gradient boosting, weak learners (in this case, decision trees) are repeatedly added to the ensemble to reduce the prediction errors of previous learners. Due to their ease of training and effective ensemble predictions, gradient boosting methods are highly popular for tabular data [94].

Despite the widespread use of XGBoost (and other similar implementations of gradient boosting machines such as LightGBM [101] and CatBoost [102]) for tabular data within the broader machine learning community, these models are not popular within the turbulence modelling community. To explore viability for data-driven closure modelling, we include XGBoost as a model candidate for predicting the optimal tensor basis coefficients. In addition to XGBoost, we consider also a deep feedforward neural network (multi-layer perceptron or MLP) and a random forest model—the two main model types currently in use for data-driven turbulence closure modelling. Finally, we consider a simple super-ensemble of these three model types using two weighting schemes: namely, an equal weighting and a score-based weighting. The model types evaluated in the present study are summarized in Table 4.1.

After training these three model types, two forms of stacking were used to combine

Table 4.1: Types of machine learning methodologies for turbulence closure modelling used in the present study.

Model	Type
Neural network	Fully connected, feedforward MLP
Random forest	Tree-based ensemble model
XGBoost	Gradient boosted tree ensemble

the individual model predictions. This stacked ensemble consisted of three models (viz., the optimized or fully-trained version of each model type summarized in Table 4.1). The ensemble predictions were calculated using:

$$g_{\text{ens}}^{(n)} = \sum_{i=1}^{N} w_i g_i^{(n)} \ , \qquad n = 1, 2, 3 \ , \tag{4.12}$$

where w_i are the ensemble weights. Two weight calculation techniques were used. The first was equal weighting of each ensemble member, so $w_i = 1/N$ where N is the number of members (models) in the ensemble ($N = 3$ in the present study). The second was a cross-validation score-based weight of the three models. In this case, the weights were calculated as follows:

$$w_i = \frac{N-1}{N} - \frac{\mathrm{C}_i}{\sum_{i=1}^{N} \mathrm{C}_i} \ , \tag{4.13}$$

where C_i is the cross-validation score for model i ($i = 1, 2, \ldots, N$ with $N = 3$).

These two ensembles are considered "simple stacks". Further investigation is required to determine the feasibility and advantages of using deep stacking (where the weights are determined locally by a more sophisticated machine learning algorithm which itself could be a gradient boosting machine) for turbulence closure modelling.

4.2.4 Input feature selection

As discussed in Section 4.2.2, careful selection of the machine learning targets guarantees framework invariance and stability. Invariance and stability of the input features used is equally important. If both the target quantities and input features are invariant, the overall framework is invariant (under coordinate transformations and Galilean boosts).

To provide a fair comparison between the various model types, all models used the same input feature set. This input feature set consisted of a number of invariant scalars,

combined with a set of heuristic scalars. The procedure for generating the invariant scalar set was discussed in detail by McConkey et al. [93]. In summary, these invariant scalars arise from the first and second invariants of a set of basis tensors formed by combining the mean strain rate, mean rotation rate, turbulent kinetic energy gradient, and pressure gradient. As discussed in Section 4.2.5, the present study uses the k-ω (shear stress transport) SST model as the base RANS turbulence closure model. Therefore, the pressure gradient tensor was used in the present work, rather than the v^2 (or, wall-normal stress) gradient tensor used by McConkey et al. [93]. Invariant scalars that were shown to be identically zero by McConkey et al. [93] for two-dimensional flows were eliminated, as all of the flows used in the present study are two-dimensional in nature.

The heuristic scalars were selected from a set of scalars used by Kaandorp and Dwight [35]. These scalars included the ratio of excess rotation to strain rate (q_1), the wall-distance based Reynolds number (q_2), and the ratio of turbulent time scale to mean strain timescale (q_3). These three heuristic scalars are defined as follows:

$$q_1 = \frac{||R^2|| - ||S^2||}{2||S^2||} \ , \tag{4.14}$$

$$q_2 = \min\left(\frac{\sqrt{k}y_w}{50\nu}, 2\right) \ , \tag{4.15}$$

$$q_3 = \frac{k}{\varepsilon}||S|| \ , \tag{4.16}$$

where y_w is the wall-normal distance, ε is the turbulent kinetic energy dissipation rate, and $\| T\|$ is the Frobenius norm of a tensor T. Table 4.2 summarizes the input feature set used.

As described in Section 4.2.5, the base turbulence closure model for this study was the k-ω SST model. The turbulent timescale used to non-dimensionalize the strain- and rotation-rate tensors is determined from

$$T_t = \max\left(6\sqrt{\frac{\nu}{\varepsilon}}, \frac{k}{\varepsilon}\right) \ . \tag{4.17}$$

This timescale appears in the invariants that correspond to the strain- and rotation-rate tensors in Table 4.2 (cf. Equations (4.6) and (4.7)). The first argument in Equation (4.17) (Kolmogorov timescale) was added to ensure the stability of T_t near the wall. The TKE dissipation rate ε was calculated using

$$\varepsilon = \beta^* k\omega \ , \tag{4.18}$$

Table 4.2: Input feature set used for all machine learning models. The first and second matrix invariants of a matrix A are defined as $I_1(A) \equiv \mathrm{tr}(A)$ and $I_2 \equiv \frac{1}{2}[(\mathrm{tr}(A))^2 - \mathrm{tr}(A^2)]$, respectively. Expressions for q_1, q_2, and q_3 are given in Equations (4.14), (4.15), and (4.16), respectively.

I_1	I_2	Scalars
S^2	S^2	q_1
R^2	S^3	q_2
A_p^2	R^2	q_3
A_k^2	A_p^2	–
R^2S^2	A_k^2	–
A_p^2S	R^2S	–
$A_p^2S^2$	R^2S^2	–
A_k^2S	R^2SRS^2	–
$A_k^2S^2$	A_pA_k	–
A_pA_k	–	–
A_pRS	–	–
$A_p^2SRS^2$	–	–
A_k^2RS	–	–
A_kSRS^2	–	–
A_pA_kS	–	–
$A_pA_kS^2$	–	–
RA_pA_k	–	–
RA_pA_kS	–	–
RA_kA_pS	–	–
$RA_pA_kS^2$	–	–
$RA_kA_pS^2$	–	–

with $\beta^* = 0.09$.

As shown in Table 4.2, after combining the invariant and heuristic scalars, the input feature set consists of 33 features. This feature set is relatively large compared to some other data-driven closure investigations [23, 35]. In machine learning, a large input feature set can lead to overfitting, and a common regularization technique is the elimination of irrelevant input features. However, a brief input feature pruning study during the hyperparameter optimization procedure showed that the validation set performance deteriorated with the elimination of one or more input features from this set. Therefore, it is possible that the availability of even more input features would improve the model performance. A possible method for augmenting the input feature set may be the use of additional nonzero invariant scalars if the flow is three-dimensional. While several of these scalars were found to be zero for two-dimensional flows [93], the full scalar set may be used as a starting point for fully three-dimensional flows.

4.2.5 Dataset and preprocessing

A major goal of this study is to investigate the generalizability of various machine learning models for turbulence closure modelling. The training dataset therefore contained several types of flows, included in the dataset compiled by McConkey et al. [79]. This dataset consists of several parametrically varying turbulent flows, with collocated RANS and high-fidelity DNS or LES fields. The dataset is designed to be used for generating machine learning mappings between RANS input features and highly accurate closure quantities. Figure 4.1 presents an overview of the cases in this curated dataset.

The k-ω SST turbulence model was chosen as the base RANS model [53]. This model is popular for simulating separated industrial flows, and provides a reasonably accurate base set of input features. Details of the domain, boundary conditions, mesh (including mesh independence), and numerical schemes for generating the RANS dataset can be found in McConkey et al. [79].

A total of 895,640 points were available in the base dataset, with all the input features described in Section 4.2.4 pre-calculated. The pre-processing steps included calculating the optimal tensor basis coefficients, removing outliers, and scaling the input features. The results for the optimal tensor basis coefficient calculations are described in Section 4.3.1. For each point in the dataset, a multiple objective least-squares solver was used to calculate the optimal $g^{(1)}$, $g^{(2)}$, and $g^{(3)}$ scalar coefficients for the tensor basis expansion of the anisotropy tensor (cf. Section 4.2.2).

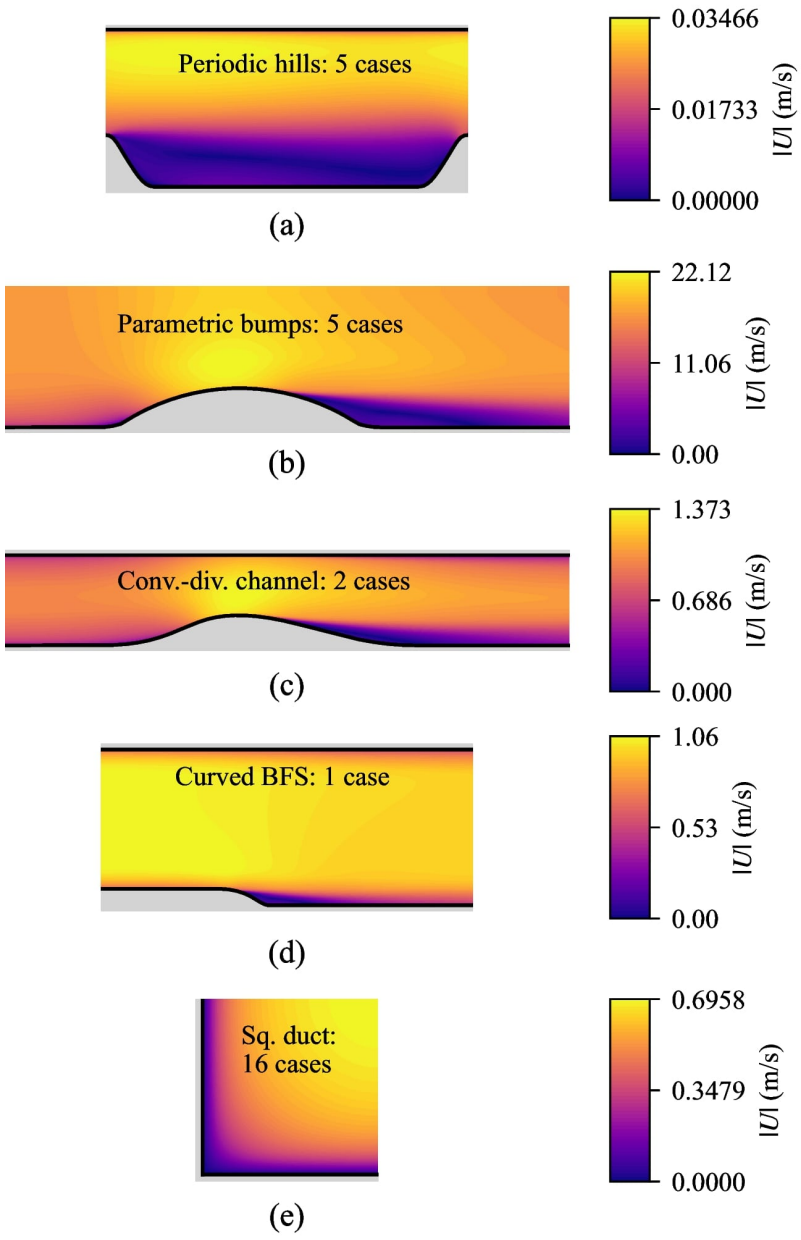


Figure 4.1: The flows used in the training dataset. The RANS fields for these flows were generated in McConkey et al. [79]. The high-fidelity fields were generated for (a) the periodic hills by Xiao et al. [44]; (b) the parametric bumps by Matai and Durbin [46]; (c) the converging-diverging channel by Marquillie et al.[57] and Schiavo et al. [48]; (d) the converging backward-facing step (CBFS) by Bentaleb et al. [49]; and, (e) the square duct flow by Pinelli et al. [45].

After calculating the optimal coefficients, several criteria were used to remove outliers. Though the optimization procedure ensured accuracy in the approximation of the anisotropy tensor for the majority of the points, a number of points with a large error in the reconstruction of b still remained. Therefore, the first criterion used for the elimination of outliers was to remove all points corresponding to an absolute error in any component of b, $|b_{ij} - b_{ij\theta}|$ greater than 0.05 (for reference, $-2/3 \leq b_{11} \leq 1/3$). The second criterion for outlier removal was based on the magnitude of the basis tensor coefficients themselves. For some locations in the flow, the optimization procedure produced optimal coefficients several orders of magnitude larger than the mean. The thresholds used for the elimination of large $g^{(n)}$ values (outliers) were determined through a trial-and-error process and by inspecting the resulting distributions for $g^{(n)}$ (see Section 4.3.1). Ultimately, the criteria for this magnitude-based elimination were $g^{(1)} < -0.20$, $|g^{(2)}| > 0.10$, and $|g^{(3)}| > 0.20$. Finally, the square duct case required special treatment in terms of outlier removal. Though the dataset contained points across the entire duct cross section, the duct case is symmetrical about two axes. To avoid including additional data for one flow type that was effectively augmented by reflection, only one quadrant of the square duct cross section was retained. In other words, only one quadrant of the square duct flow was kept in the training dataset in order to eliminate effects of data augmentation through reflection. The error magnitude criteria was also not applied to the square duct coefficients. The square duct case is the only case which contains non-zero b_{13} and b_{23}, and therefore all points from the square duct were included due to their unique value. After applying all these criteria for outlier removal, a total of 663,254 points remained in the dataset.

The last step of the pre-processing was scaling of the input features. The scaling method selected was MinMax scaling, where all features are scaled to lie between 0 and 1. Scaling the features to be $O(1)$ helps eliminate issues caused by different orders of magnitudes between each input feature, which is especially important for the neural network model. Although feature scaling has little effect on the results of the tree-based models (viz., random forest and XGBoost), the same scaled input features were used for all the models. The parameters for the MinMax scaler changed with each training set, to consistently ensure that the input features were between 0 and 1.

4.2.6 Training, hyperparameter optimization, and evaluation

The training procedure used was intertwined with the hyperparameter optimization. To test model generalizability, one hold-out test case was first selected from each flow type (see Table 4.3). Then, the remaining cases were used to train and optimize the hyperparameters of each machine learning model. To examine the generalizability of these models in detail,

Table 4.3: Training and test datasets: periodic hill PHLL; parametric bump BUMP; square duct DUCT; converging-diverging channel CNDV; and, curved backward-facing step CBFS.

	Name	Cases
Training sets	PHLL4	PHLL_case_0p5, PHLL_case_0p8, PHLL_case_1p0, PHLL_case_1p5
	BUMP4	BUMP_h20, BUMP_h26, BUMP_h31, BUMP_h42
	DUCT15	DUCT_1100, DUCT_1150, DUCT_1250, DUCT_1300, DUCT_1350, DUCT_1400, DUCT_1500, DUCT_1600, DUCT_1800, DUCT_2205, DUCT_2400, DUCT_2600, DUCT_2900, DUCT_3200, DUCT_3500
	FULL	PHLL4, BUMP4, DUCT15, CNDV_12600
Testing sets	Periodic hill test case	PHLL_case_1p2
	Bump test case	BUMP_h38
	Square duct test case	DUCT_2000
	Converging-diverging channel test case	CNDV_20580
	Curved backward facing step test case	CBFS_13700

several distinct training datasets were formed. These training datasets are summarized in Table 4.3.

The main strategy that was used for forming the various training sets in Table 4.3 was designed to address the following question: How well does a model trained on one flow generalize to another flow? Despite the important implications of this question for developing a general data-driven turbulence closure model, relatively few studies have examined the generalization performance in this manner. For example, splitting the training data in this way allowed us to test if a model trained purely on periodic hill cases generalized to a parametric bumps case, and vice-versa. In Table 4.3, the FULL training set represents the "complete" training set—it contains data from every type of flow except the curved backward facing step (CBFS). As the CBFS data only contain a single case, it represents a hold-out test case for a model trained on the FULL training data set.

After splitting the data into the training and test sets summarized in Table 4.3, a hyper-

parameter optimization procedure was completed. Machine learning models are sensitive to the values of the hyperparameters which define important model configuration variables (e.g., the number of layers in a neural network). Since the optimal hyperparameters depend on the training dataset itself, a hyperparameter optimization procedure must be completed for each new training dataset.

A grouped four-fold cross-validation (CV) hyperparameter optimization procedure was completed using each training set. This procedure first splits the training dataset into four folds. Each fold is used once as a validation set while the other three folds are used as training sets. The purpose of the validation set is to evaluate the generalization error (variance) after training. A model with a given set of hyperparameters is trained on each training set (three folds), and evaluated on the validation set (the remaining fold). The cross-validation score for a given hyperparameter set is then the average of the four validation set scores. This cross-validation is completed for a number of hyperparameter sets. Then, the optimal hyperparameter set is the one with the best cross-validation score. The purpose of cross-validation is to ensure that the selection of the validation set does not affect the choice of hyperparameters—the average of several validation set choices are used to select the hyperparameters. We recommend using grouped cross-validation for this purpose, to ensure that data points from the same flow case are not present in both the training and validation sets. Grouped cross-validation prevents data leakage and ensures the validation set provides a true measure of the generalization error. For example, in machine learning for medical applications, grouped cross-validation is used when multiple observations come from the same patient. In this case, four-fold cross validation was selected, as the PHLL4 and BUMP4 training sets contain four cases, which was the minimum number of cases in any of the training sets.

Each machine learning model type (neural network, random forest, and XGBoost) has a different set of hyperparameters that need to be optimized. Full details of the hyperparameter search space are given in Appendix C. After specifying the search space for each model, the scikit-opt hyperparameter tuning library was used. The hyperparameter tuner was called to complete 100 evaluations of various combinations of hyperparameter values in the search space using Bayesian optimization. Each evaluation in the search space involved the previously described four-fold cross validation procedure. At the end of the 100 evaluations, a model using the optimized hyperparameter set was fit on the entire training dataset.

The optimized hyperparameters are shown in Appendix C. For the XGBoost and random forest models, the optimized hyperparameter set produces relatively high-capacity models. While these high-capacity models have a tendency to overfit in some machine learning applications, these hyperparameters were selected for this application using grouped

cross-validation. Other tests using models with lower capacity had poorer cross-validation scores and generalization performance on new flows, increasing confidence in the cross-validated hyperparameter search. The high-capacity optimal hyperparameter sets indicate that there is relatively little difference between the validation sets and the training sets. Further work is required to see if this behavior holds for a more diverse 3D flow dataset.

In summary, the training and hyperparameter optimization procedure involved completing 100 iterations of a Bayesian optimizer search within a grouped four-fold cross validation procedure. This procedure was completed for each machine learning model type (see Table 4.1), and for each training dataset (see Table 4.3). At the end of the hyperparameter optimization procedure, an optimized version of each machine learning model type for each training dataset was then available for evaluation.

An optimized version of each model, together with two simple ensembles were evaluated on the hold-out test datasets. The purpose of this extensive evaluation was to determine the generalizability of each model type on completely new flows (viz., types of flows that were not present in the training dataset), and analyze the effects of including various flow types in the training dataset. Section 4.3 presents the results of this comprehensive evaluation.

4.2.7 Libraries and code

A number of open-source libraries were used for this investigation. Pandas was used as a general data handling library [103]. The optimization procedure for calculating the basis coefficients used the qpsolvers library [104]. For defining model pipelines and evaluating models, scikit-learn was used [85]. The scikit-opt library was used for hyperparameter tuning [105]. Neural network models were constructed using the high-level Keras [84] application programming interface (API) for TensorFlow 2, random forest models were created using scikit-learn [85], and gradient boosting models were created using XGBoost [29].

All code used for this investigation is available on GitHub as a Jupyter notebook: https://github.com/rmcconke/optimal_tensor_basis. All data used in this investigation comes from an open-source dataset described by McConkey et al. [79].

4.3 Results and discussion

4.3.1 Coefficients for optimal tensor basis expansion

The optimization problem described in Section 4.2.2 was solved using the qpsolvers library [104]. For each data point, optimal values for $g^{(1)}$, $g^{(2)}$, and $g^{(3)}$ were calculated. Figures 4.2 to 4.4 visualize these coefficients for one case from each of the flow types.

Figure 4.2 shows the distinct behavior of $g^{(1)}$ for each flow type. For the periodic hills (Figure 4.2(a)) $g^{(1)}$ is large in two distinct regions: namely, above the separated region of the flow and near the bottom of the recirculation zone. This same behavior is seen for the bump and the converging-diverging channel cases (Figures 4.2(b) and (c)). The coefficient $g^{(1)}$ is associated with the mean strain-rate tensor S in the tensor basis expansion of b. Therefore, it is not surprising that $g^{(1)}$ is large in near the boundaries of the separated regions of a flow, where the velocity gradients are large. Interestingly, Figure 4.2(c) shows that near the centerline of the channel upstream of the bump (where the mean strain rate goes to zero), $g^{(1)}$ is small in magnitude. However, along the centerline downstream of the bump, the magnitude of $g^{(1)}$ is large. The Reynolds stress anisotropy is high in this region after the flow accelerates over the bump, leading to the larger values of $g^{(1)}$ seen here. Figure 4.2(d) shows unusual behavior of $g^{(1)}$ near the top of the domain and along the centerline. As discussed later, unusual behavior of $g^{(2)}$ and $g^{(3)}$ was also observed for the curved backward-facing step (CBFS) case. Although the optimization procedure was able to find a solution for $g^{(n)}$ at all points, large magnitudes for these coefficients were observed for some regions of the CBFS case. Ultimately, these large magnitudes indicate that an accurate approximation of b while minimizing $g^{(n)}$ was not possible in these regions. The likely cause for these instabilities are the extremely small velocity gradients near the channel centerline for the curved backward-facing step. Compared to the other cases, the CBFS features a relatively uniform velocity profile near the channel centerline, where the velocity gradients are small in magnitude. Finally, examining Figure 4.2(e) shows an unsurprising result—the optimal representation of the anisotropy tensor for the square duct case involves almost no linear component and, therefore, the value of $g^{(1)}$ is small in magnitude. The inability of linear-eddy viscosity models to capture the corner vortices in a square duct is well-known [23].

The coefficients $g^{(2)}$ and $g^{(3)}$ represent the contribution of two basis tensors that are second-order in terms of the mean velocity gradients. For many flows, a purely linear-eddy viscosity model is unable to completely capture the behavior of the anisotropy tensor. Additional nonlinear terms in the tensor basis expansion of b are required to improve this approximation. Figure 4.3 shows the behavior of the optimal values for $g^{(2)}$. Whereas $g^{(1)}$

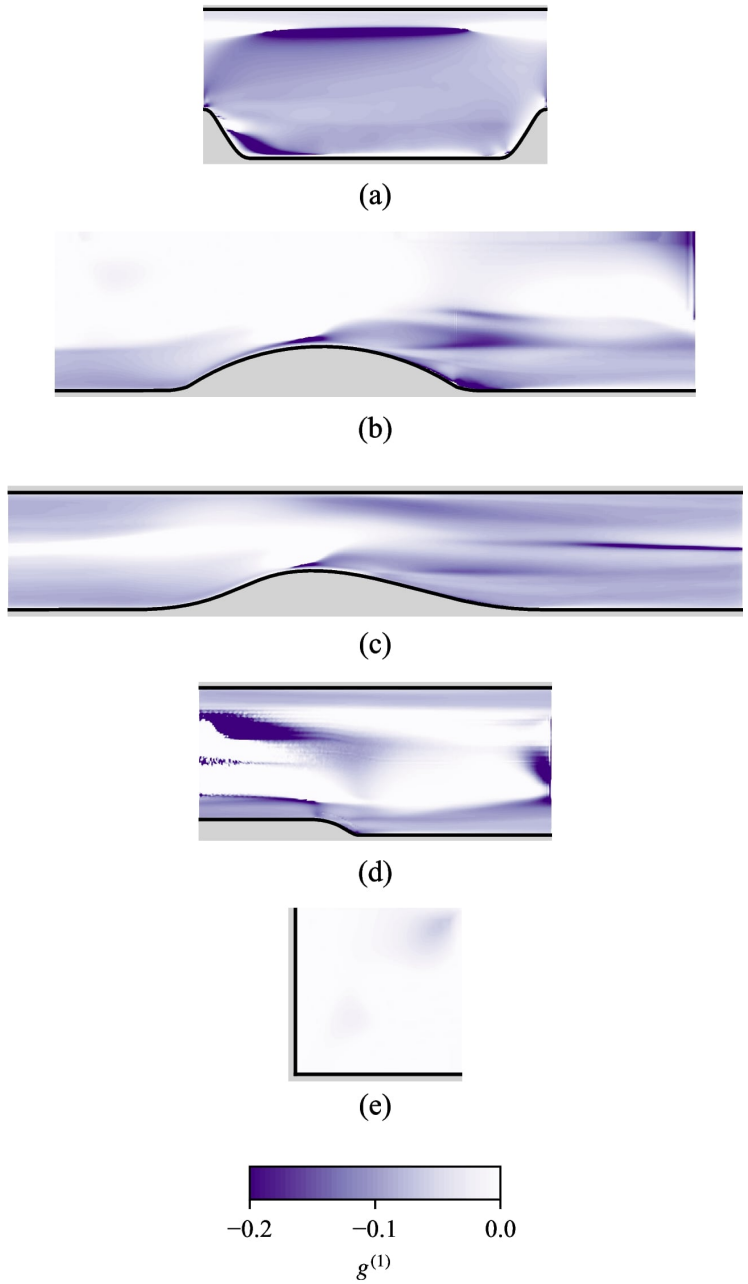


Figure 4.2: Optimal values for $g^{(1)}$ (cf. Equation (4.11)) for one selected case from each flow type: (a) the periodic hills $\alpha = 0.5$ case; (b) the parametric bump $h = 42$ mm case; (c) the converging-diverging channel at a Reynolds number of Re $= 12600$ case; (d) the curved backward-facing step case; and, (e) the square duct at Re $= 1150$ case.

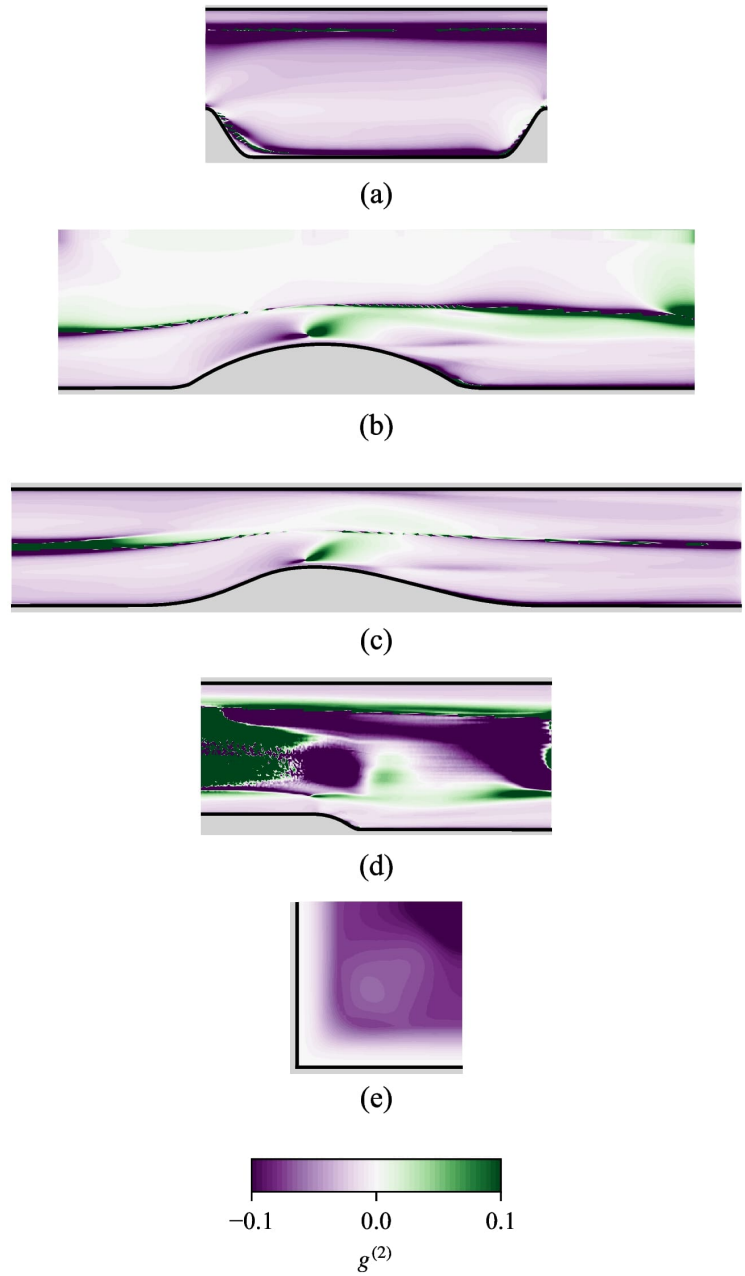


Figure 4.3: Optimal values for $g^{(2)}$ (cf. Equation (4.11)) for one selected case from each flow type: (a) the periodic hills $\alpha = 0.5$ case; (b) the parametric bump $h = 42$ mm case; (c) the converging-diverging channel at a Reynolds number of Re = 12600 case; (d) the curved backward-facing step case; and, (e) the square duct at Re = 1150 case.

is strictly negative to ensure stability, no stability constraints are required for the values of $g^{(2)}$. Figures 4.3(a), (b), and (c) show that $g^{(2)}$ is largest in magnitude near regions of the flow with high velocity magnitude. While the periodic hills case generally exhibits negative values of $g^{(2)}$, both the parametric bump and converging-diverging channel cases exhibit an equal distribution of positive and negative values of $g^{(2)}$. At the peak of the acceleration region in Figures 4.3(b) and (c), there is a rapid sign change of $g^{(2)}$, with $g^{(2)}$ being negative and positive upstream and downstream of the crest, respectively. Comparing Figure 4.3(a) to Figures 4.3(b), (c), (d), and (e), we see that the periodic hills case exhibits a distinctive behavior of $g^{(2)}$ with respect to the other flow cases—these flow cases generally exhibit small $g^{(2)}$ magnitudes near the wall, whereas the periodic hills case exhibits the opposite trend in $g^{(2)}$ near the lower wall. A careful examination of Figure 4.3(d) indicates that $g^{(2)}$ exhibits an unstable behavior for the CBFS case. Along with the larger magnitudes of $g^{(2)}$ near the inlet centerline, there is a rapid switch in the sign of $g^{(2)}$. In Figure 4.3(e), we see that the contribution of $T^{(2)}$ is magnified in the region of the square duct corner vortex. This result indicates that a greater contribution of the nonlinear term $T^{(2)}$ is required to capture the anisotropy of the corner vortex—a result that is in agreement with other investigations [23].

Figure 4.4 displays contours of $g^{(3)}$ for the various flow types studied herein. A careful perusal of Figure 4.4 shows that $g^{(2)}$ and $g^{(3)}$ have similar behaviors for some flows, but exhibit distinct behaviors for other flows. For the periodic hills flow (Figure 4.4(a)), $g^{(3)}$ changes sign within the separated region. The large positive values of $g^{(3)}$ near the top of the domain transition to large negative values near the bottom wall. For both the parametric bump and converging-diverging channel cases (Figures 4.4(b) and (c)), $g^{(3)}$ is small in magnitude as the flow accelerates over the obstacle. For these two cases, large positive values of $g^{(3)}$ are observed near the centerline of the domain. Whereas the parametric bump cases impose a zero-gradient (i.e., freestream) boundary condition at the top of the domain, the converging-diverging channel cases impose instead a wall-boundary condition at the top of the domain. This internal/external flow difference may explain the different behaviors of $g^{(3)}$ immediately upstream and downstream of the obstacle. A small negative $g^{(3)}$ region is found immediately upstream of the separation point, along the top of the obstacle. Figure 4.4(d) shows that similar to $g^{(2)}$, $g^{(3)}$ is large in magnitude near the middle of the domain and smaller magnitudes are observed for the CBFS near the top and bottom walls. However, $g^{(3)}$ does not experience the rapid sign switching observed in $g^{(2)}$ near the inlet. Finally, examining the $g^{(3)}$ field for the square duct case (Figure 4.4(e)) shows almost identical behavior to that of $g^{(2)}$, but with an opposite sign. The optimal value of $g^{(3)}$ increases near the region of the corner vortex, which agrees with previous results that nonlinear terms in the tensor basis expansion of the anisotropy tensor are

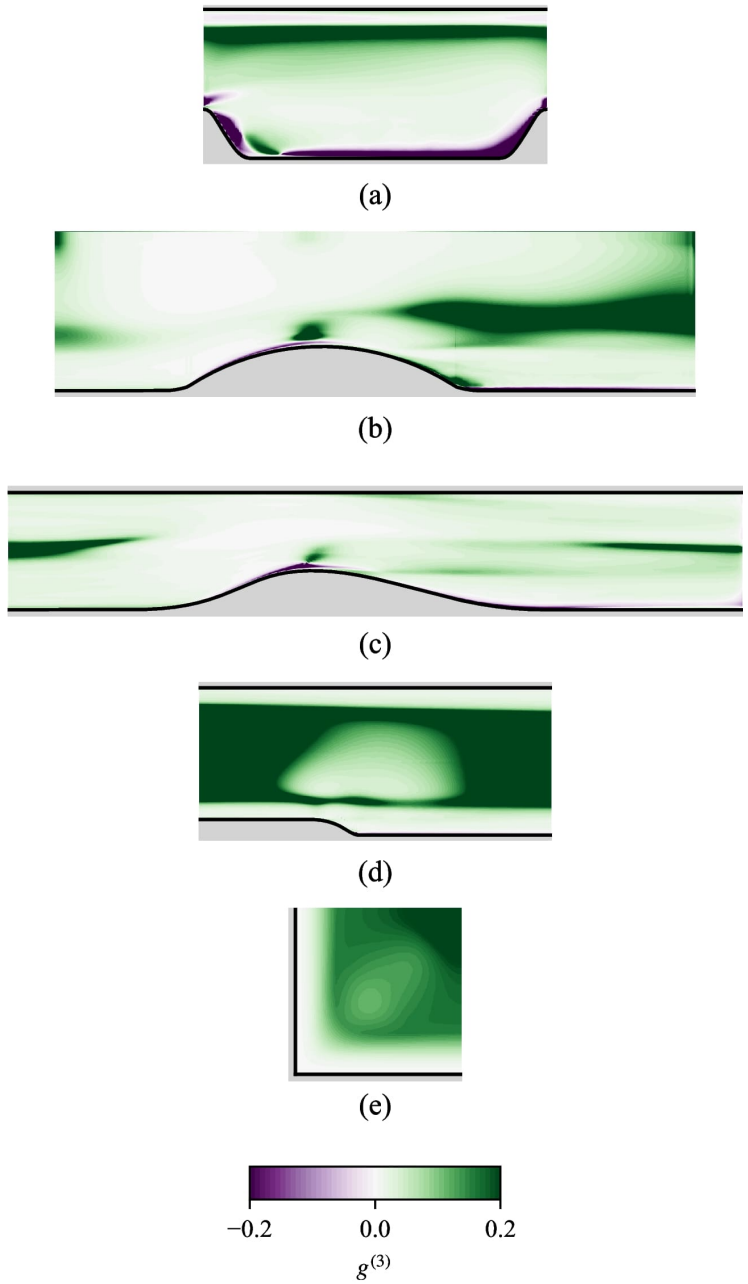


Figure 4.4: Optimal values for $g^{(3)}$ (cf. Equation (4.11)) for one selected case from each flow type: (a) the periodic hills $\alpha = 0.5$ case; (b) the parametric bump $h = 42$ mm case; (c) the converging-diverging channel at a Reynolds number of Re = 12600 case; (d) the curved backward-facing step case; and, (e) the square duct at Re = 1150 case.

needed to capture the corner vortex.

Examining the data distribution shape is another important tool for comparing the various flows in this dataset. Figure 4.5 presents a visualization of the data for each flow case, after removal of the outliers (Section 4.2.5). This figure uses each of the $g^{(n)}$ ($n = 1, 2, 3$) scalars as a coordinate to create a three-dimensional point cloud. The projection of this point cloud on the three coordinate planes help visualize the shape of these clouds. From Figure 4.5, it is immediately clear that the four 2D flows over obstacles have similar data distributions, despite the differences seen in their spatial behavior (cf. Figures 4.2 to 4.4). The periodic hills, parametric bumps, converging-diverging channel, and curved backward-facing step cases have more diverse optimal coefficients than the square duct cases.

Figure 4.6 displays the distributions of the optimal coefficients for each flow in the form of violin plots. These violin plots, which show a rotated continuous distribution generated using a kernel density estimation methodology, provide an informative comparison between the various flow cases. Figure 4.6 confirms the similarity between the 2D flows over an obstacle—while each distribution displays subtle differences, the overall shape of the distributions are similar. In Figure 4.6, we see that most of the $g^{(1)}$ distributions exhibit a peak value between about -0.05 and -0.10. For linear-eddy viscosity models, this coefficient is directly related to the closure coefficient C_μ as $g^{(1)} = -C_\mu$. This coefficient is applied globally, whereas the optimal $g^{(1)}$ value is a local quantity (viz., it has a dependence on the spatial coordinates). As shown in Figure 4.6(a), the usual value for this coefficient is approximately equal to the peak value in the optimal $g^{(1)}$ distributions, albeit a little larger in magnitude than the peak value in the $g^{(1)}$ distribution. While $g^{(1)}$ has a direct correspondence with C_μ, no analogous relationships to $g^{(2)}$ and $g^{(3)}$ exists in the classical turbulence closure models, since they represent the contribution to the anisotropy tensor arising from the nonlinear tensor basis terms $T^{(2)}$ and $T^{(3)}$.

Compared to the other flow cases, the square duct case displays a distinctive behavior in all of the $g^{(n)}$ coefficients. The distribution for $g^{(1)}$ in the square duct case shows that nearly all values are close to zero. While for many points, the optimal values for $g^{(2)}$ and $g^{(3)}$ are close to zero, these coefficients associated with the nonlinear terms in the tensor basis expansion of the anisotropy tensor have much larger magnitude than that of $g^{(1)}$. Clearly, the linear term $T^{(1)}$ embodies virtually no information concerning b for the square duct cases, and the optimization procedure returns an approximation for b consisting almost entirely of the nonlinear contributions from $T^{(2)}$ and $T^{(3)}$. As previously mentioned, a likely reason for this lack of a linear contribution to the anisotropy tensor is due to the failure of a linear-eddy viscosity model to accurately represent the Reynolds stresses associated with this flow.

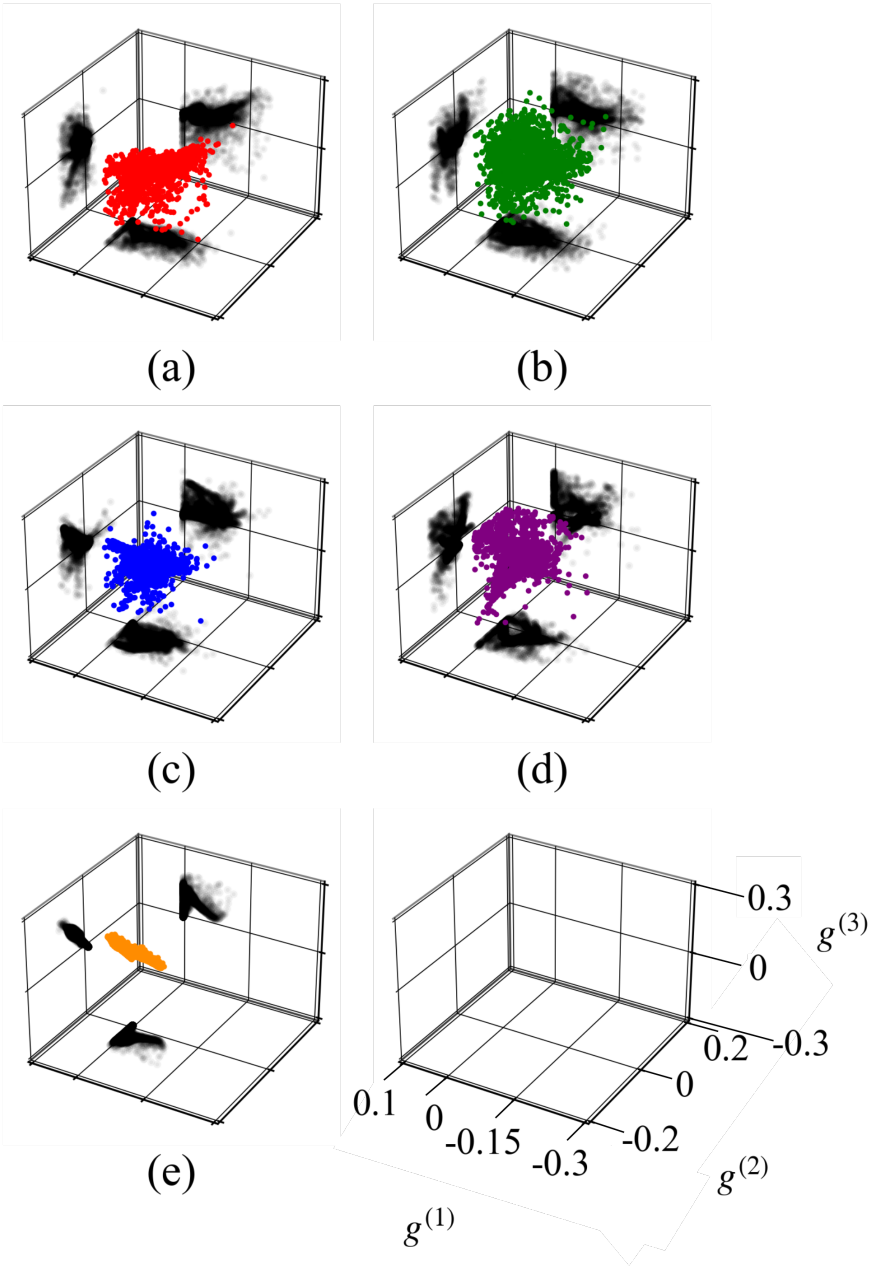


Figure 4.5: Three-dimensional scatter plot of the optimal $g^{(n)}$ coefficients (cf. Equation (4.11)) for (a) the periodic hills; (b) the parametric bumps; (c) the converging-diverging channels; (d) the curved backward-facing step; and, (e) the square duct.

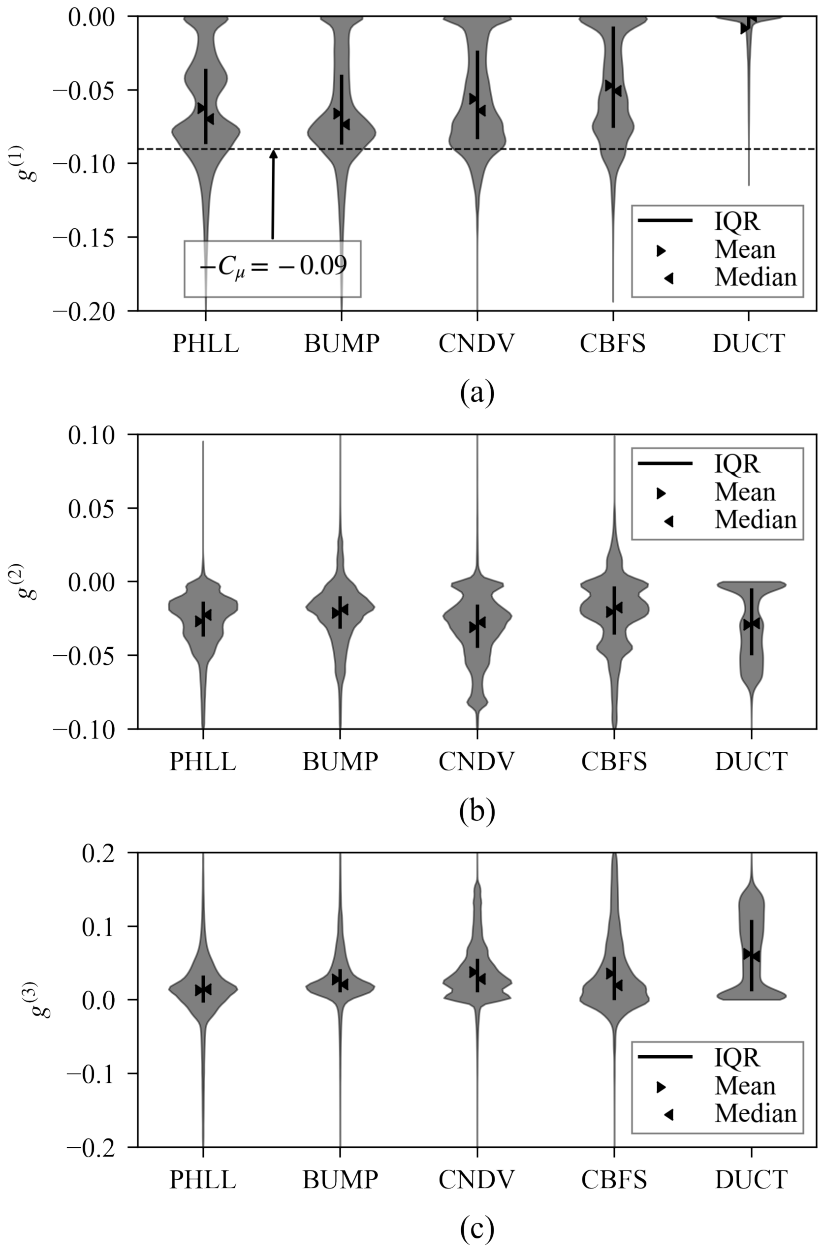


Figure 4.6: Violin plots showing the distributions of optimal $g^{(n)}$ coefficients: (a) $g^{(1)}$; (b) $g^{(2)}$; and, (c) $g^{(3)}$. Here, IQR denotes the interquartile range (viz., the difference between the 75th and 25th percentiles of the distribution).

4.3.2 Predictions of optimal tensor basis coefficients

After calculating and visualizing the optimal coefficients $g^{(n)}$ ($n = 1, 2, 3$), it is of interest to determine their predictability using various machine learning models. Furthermore, it is of interest to determine whether a model that was trained to predict the optimal coefficients for one flow would generalize to another flow. As previously noted, these coefficients are used in the present study as a surrogate for predicting the anisotropy tensor (and closing the RANS equations). Therefore, we believe that this generalizability investigation is applicable for many types of data-driven turbulence closure frameworks.

Table 4.4 shows a matrix of the mean-squared errors (MSE) for various models on various test cases. The models selected are the final versions of each model (viz., the fully-trained machine learning model) using the hyperparameters obtained from the hyperparameter optimization. The MSE is calculated over the three predicted coefficients (viz., $g^{(1)}$, $g^{(2)}$, and $g^{(3)}$).

For every model type, a model was trained on each of the four training datasets. Following this, the trained models were tested on each test set. This exhaustive evaluation allows the generalizability of each model to be determined. As seen in Table 4.4, all of the models trained (including the naive linear regression model used as a baseline for comparison) generalize well to new cases within the same flow (for example, a different periodic hills geometry). Good generalization (viz., low variance) is identified in this case by having similar CV and test set scores. Across varying flow types such as periodic hills, parametric bumps, and a square duct, Table 4.4 shows that models generalize well within the same flow type. For example, given a training dataset consisting of square duct flows, a machine learning model can be expected to generalize well to new square duct flows. This finding is not new, and has been reported in nearly every data-driven turbulence modelling study—most studies include a generalizability test case within the same flow type. However, a key finding of the present study comes from examining the generalization performance for new flow types. Table 4.4 shows that the MSE increases by roughly one order of magnitude when generalizing to a new flow type. This lack of generalizability holds true across the three machine learning model types and across all four training datasets.

Comparing the results in Table 4.4 for each machine learning model shows that the XGBoost model generally outperforms the random forest and neural network models. The CV and test set scores are generally lower for the XGBoost model, across various training and test datasets. Perhaps the most representative training dataset is the FULL training dataset—this contains every flow in the dataset except the CBFS case. Comparing the performance on the FULL training dataset, we see that the XGBoost model has a similar CV score to the random forest model (3.035×10^{-4} compared to 3.028×10^{-4}, respectively),

Table 4.4: Mean squared error (MSE) for various training and test datasets for each model type.

		Mean squared error					
Model	Training set	CV score	PHLL	BUMP	DUCT	CNDV20580	CBFS
Linear regression	PHLL4	7.402E-04	6.663E-04	1.063E+04	2.001E-03	5.004E+00	4.399E-03
	BUMP4	1.310E+02	1.015E-03	1.035E-03	1.200E-03	1.157E-03	1.409E-03
	DUCT15	2.272E-04	7.152E+7	9.328E+11	1.734E-04	6.833E+10	1.441E+08
	FULL	1.310E+02	1.015E-03	1.035E-03	1.200E-03	1.157E-03	1.409E-03
Neural network	PHLL4	6.365E-04	3.056E-04	3.046E+00	1.532E-03	2.062E-02	2.529E-03
	BUMP4	5.168E-04	1.496E-01	4.714E-04	1.001E-03	2.616E+01	6.430E-04
	DUCT15	4.130E-05	1.568E+06	4.848E+06	2.746E-05	8.671E+05	2.349E+06
	FULL	7.541E-04	1.088E-03	9.735E-04	7.337E-04	1.047E-03	9.081E-04
Random forest	PHLL4	1.296E-04	3.642E-05	1.154E-03	1.623E-03	1.506E-03	2.514E-03
	BUMP4	2.760E-04	1.172E-03	1.315E-04	9.068E-04	1.036E-03	8.232E-04
	DUCT15	1.215E-05	3.621E-03	3.238E-03	9.796E-06	1.407E-03	2.063E-03
	FULL	3.028E-04	9.667E-05	1.608E-04	1.951E-05	8.363E-04	1.016E-03
XGBoost	PHLL4	1.083E-04	3.064E-05	1.116E-03	1.495E-03	1.536E-03	1.865E-03
	BUMP4	2.484E-04	1.384E-03	1.262E-04	8.000E-04	1.083E-03	8.404E-04
	DUCT15	1.140E-05	3.434E-03	3.057E-03	6.729E-06	1.456E-03	2.545E-03
	FULL	3.035E-04	3.330E-05	1.360E-04	9.413E-06	8.718E-04	1.164E-03
Equal weight ensemble	FULL		1.907E-04	2.528E-04	9.122E-05	8.663E-04	9.216E-04
CV score weighted ensemble	FULL		7.506E-05	1.603E-04	2.092E-05	8.486E-04	1.013E-03

Table 4.5: Wall-clock time taken for the hyperparameter tuning procedure on the FULL training dataset. All central processing unit (CPU) based calculations (random forest and XGBoost) used an identical machine and number of cores, and the neural network calculations used a graphics processing unit (GPU).

Model type	Resource type	Wall time (s)
Neural network	GPU	30507
Random forest	CPU	70603
XGBoost	CPU	39285

and an improved CV score compared to the neural network model. In terms of test set scores, the XGBoost model outperforms the random forest model for the periodic hills, parametric bumps, and square duct test cases, while the random forest performs slightly better on the converging-diverging channel and curved backward-facing step test cases.

Table 4.5 compares the computational demands for the hyperparameter optimization procedure. As described in Section 4.2.6, each hyperparameter optimization involved 100 iterations of a Bayesian optimization procedure. Computations were performed on either an AMD Ryzen Threadripper 3970X 32-core central processing unit (CPU), or on an NVIDIA RTX 3090 graphics processing unit (GPU). For the tree-based models, the hyperparameter tuning procedure was completed entirely on a CPU, while the neural network hyperparameter tuning utilized computations on a GPU. Table 4.5 shows that the XGBoost model was nearly twice as fast for the hyperparameter optimization as the random forest model, despite a similar or better model performance shown in Table 4.4. The neural network model tuning was completed faster than the tree-based model tunings, likely due to the use of a GPU to accelerate the calculations. However, the XGBoost model hyperparameter tuning time (viz., 39,285 s) was still similar to that of the neural network model (viz., 30,507 s), despite the use of a CPU versus a GPU. The XGBoost model tuning can be further accelerated by the use of a GPU—this is expected to close or eliminate altogether the differences in times required for hyperparameter tuning of the XGBoost and neural network models.

Comparing the three fields for each test case across the five models is not possible within the scope of this paper. A selection is presented here, with further visualizations shown in Appendix D. Figures 4.7, 4.8, and 4.9 show the true and predicted $g^{(1)}$ field for the periodic hill, parametric bump, and square duct test cases, respectively, for the models trained on the combined (FULL) dataset. While the results in Table 4.4 allow easy comparison of MSE magnitudes between models and test cases, visualizing the predicted $g^{(n)}$ fields further

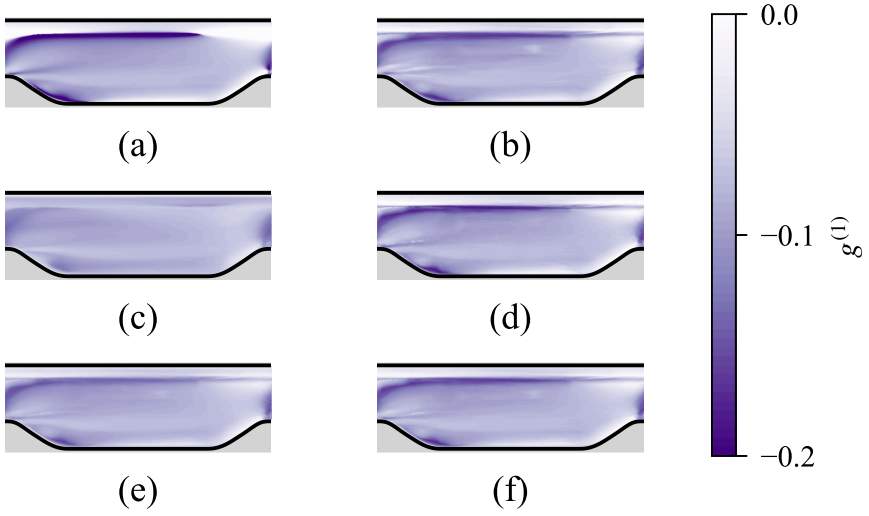


Figure 4.7: The $g^{(1)}$ fields for the periodic hills test case: (a) true field; (b) random forest model prediction; (c) neural network model prediction; (d) XGBoost model prediction; (e) uniform weighted ensemble prediction; and, (f) CV score weighted ensemble prediction.

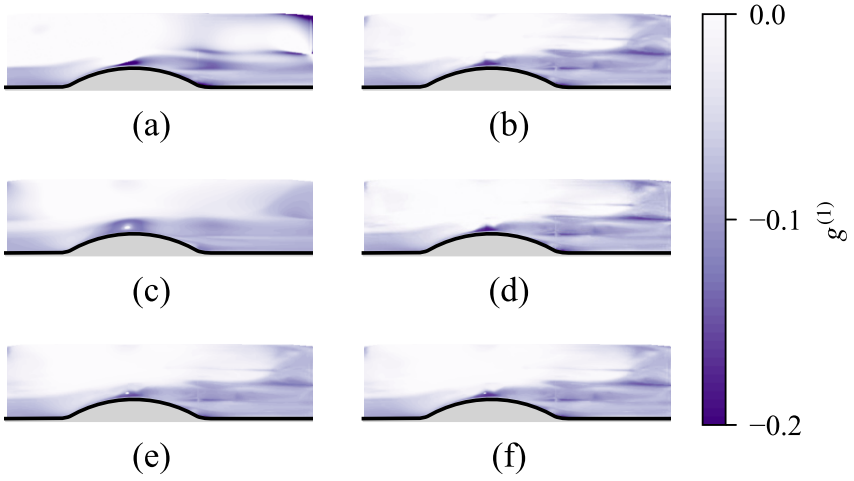


Figure 4.8: The $g^{(1)}$ fields for the parametric bump test case: (a) true field; (b) random forest model prediction; (c) neural network model prediction; (d) XGBoost model prediction; (e) uniform weighted ensemble prediction; and, (f) CV score weighted ensemble prediction.

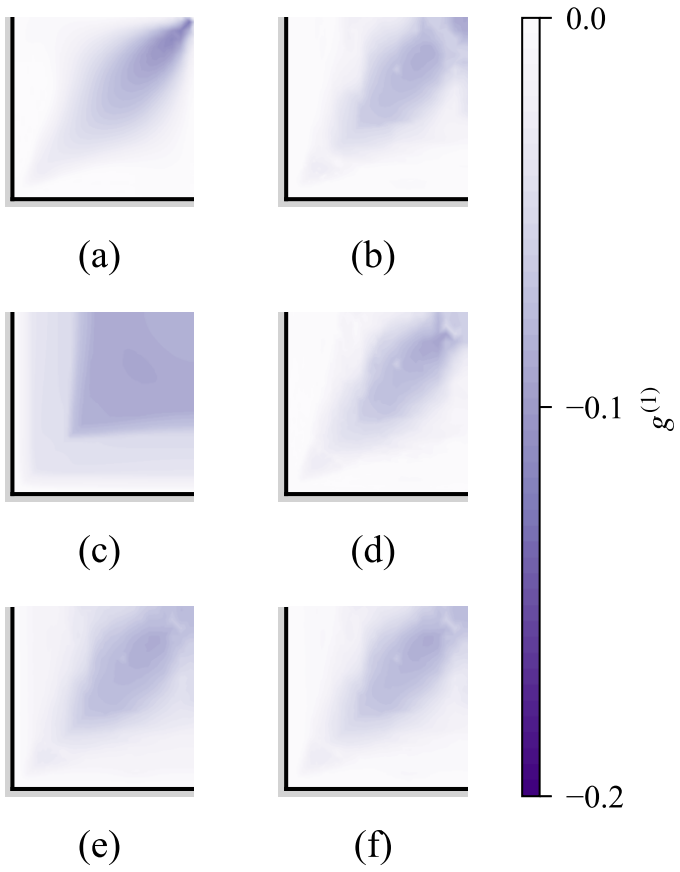


Figure 4.9: The $g^{(1)}$ fields for the square duct test case: (a) true field; (b) random forest model prediction; (c) neural network model prediction; (d) XGBoost model prediction; (e) uniform weighted ensemble prediction; and, (f) CV score weighted ensemble prediction.

facilitates the assessment of the performance of each model. The conclusion obtained from a careful examination of Figures 4.7, 4.8, and 4.9 is the same: namely, the tree-based models predict the $g^{(n)}$ fields with reasonable accuracy. The neural network model predictions are inferior to those provided by the tree-based models. In particular, from Figure 4.7, the neural network model does not predict the sharp increase in $g^{(1)}$ above the separated region. Furthermore, from a perusal of Figure 4.8, it is seen that the neural network model predictions of the $g^{(n)}$ fields are in close agreement with the associated label (true) fields—however the sharp increase in $g^{(1)}$ near the top of the parametric bump is smoother than that exhibited in the corresponding label field. In contrast, both tree-based models predict this sharp increase in $g^{(1)}$ accurately. Finally, Figure 4.9 shows that the neural network model overpredicts $g^{(1)}$ throughout most of the square duct quadrant. The tree-based models accurately capture the shape of this high $g^{(1)}$ region, whereas it is smoother and larger in the neural network model prediction. In Figures 4.7, 4.8, and 4.9, the two simple ensembles predict the $g^{(1)}$ field well—which, perhaps, is not too surprising as these two ensemble predictions are obtained simply by averaging over the other three individual predictors.

Based on the results in Tables 4.4 and 4.5, and the visualizations in Figures 4.7 to 4.9, we recommend XGBoost for data-driven turbulence closure modelling involving tabular data. These results are in agreement with recent findings in the broader machine learning community [94]. In terms of MSE magnitude, the XGBoost model performed slightly better than the random forest model for the majority of the test cases, but the difference is minimal. Both tree-based models outperformed the neural network model. The visualizations in Figures 4.7 to 4.9 also show minimal differences in the fields predicted by the random forest and XGBoost models. However, with the same number of optimizer evaluations, the XGBoost model was faster to optimize—using GPU-based computations is expected to further accelerate this procedure.

Despite poor performance of the neural network shown in Figures 4.7, 4.8, and 4.9, neural networks have been used successfully in previous data-driven closure investigations [23, 96, 93, 39]. Our intention is not to dispute the use of neural networks for data-driven closure modelling, but rather to demonstrate that XGBoost can achieve similar or better performance given the same attention and computational resources. For example, we expect that the neural network model here could be improved by further iterations of the hyperparameter optimizer or exploring various input feature transformations.

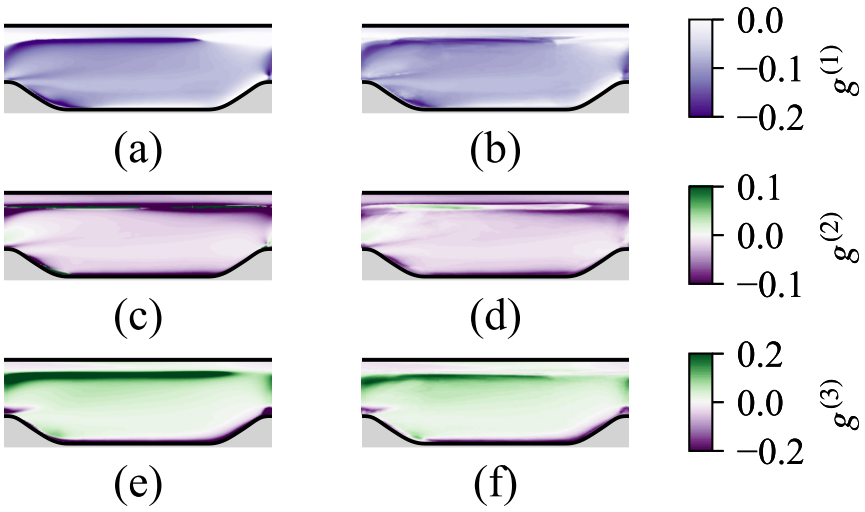


Figure 4.10: Generalizability test on the periodic hills test case for the XGBoost model. Here, the training dataset is the periodic hills PHLL4 data: (a) true $g^{(1)}$ field; (b) predicted $g^{(1)}$ field; (c) true $g^{(2)}$ field; (d) predicted $g^{(2)}$ field; (e) true $g^{(3)}$ field; and, (f) predicted $g^{(3)}$ field.

Generalizability

Having selected XGBoost as the recommended model type, we focus now on an analysis of the generalizability performance. In Table 4.4, we can deduce a rule of thumb that the MSE in $g^{(n)}$ $(n = 1, 2, 3)$ is on the same order as the CV score for a new test case within a training flow type, and is approximately within one order of magnitude on a new type of flow. Some improvement in generalizability is expected if more flow types are added to the training dataset.

Figure 4.10 shows the hold-out test set performance for the XGBoost model trained on the periodic hills dataset and tested on a periodic hills flow. The predictions for all $g^{(n)}$ component fields are nearly identical to the corresponding label field. This accuracy is also reflected by the low MSE in Table 4.4 for the XGBoost model with the periodic hills training dataset. Figure 4.10 demonstrates that the model generalizes well to a new case within the same flow type.

In order to visualize the generalization gap arising from the application of a data-driven turbulence closure model to a new flow, Figure 4.11 shows the hold-out test dataset performance for the same periodic hills trained XGBoost model, but using the parametric

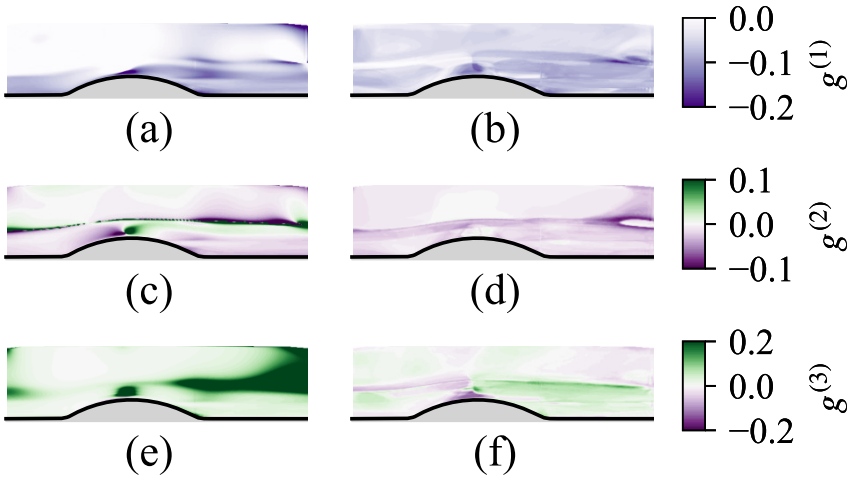


Figure 4.11: Generalizability test on the parametric bump test case for the XGBoost model. Here, the training dataset is the periodic hills PHLL4 data: (a) true $g^{(1)}$ field; (b) predicted $g^{(1)}$ field; (c) true $g^{(2)}$ field; (d) predicted $g^{(2)}$ field; (e) true $g^{(3)}$ field; and, (f) predicted $g^{(3)}$ field.

bump test case. The training dataset for this model only contained the periodic hills data. It is evident that the model clearly does not generalize well to this new type of flow. This is in spite of the fact that these 2D separated flows are similar to one another and have similar $g^{(n)}$ distributions (see Figure 4.6). Despite these similarities, the periodic hills trained XGBoost model does not generalize well in the prediction of the $g^{(n)}$ fields over a parametric bump. The predictions for the $g^{(1)}$ field are less sharp than the corresponding label field, and there is a distinct sign switching behavior in the $g^{(2)}$ field that is not predicted by the model at all. For the $g^{(3)}$ field, although the sign switching is predicted correctly by the model, the large magnitude of $g^{(3)}$ in the region behind the parametric bump is underpredicted. Table 4.4 shows that the CV score for this model is 1.083×10^{-4}, whereas the score for the parametric bump test case is 1.116×10^{-3} (roughly one order of magnitude worse). In contrast, the score for the periodic hills test case (cf. Figure 4.10) is 3.064×10^{-5}. A careful perusal of the figures in Appendix D further supports this inability of the periodic hills trained XGBoost model to generalize to new types of flow.

It is of interest to determine whether the inability to generalize between the periodic hills training dataset and the parametric bump test case would hold for the reverse case. Figure 4.12 shows the performance of the parametric bump trained XGBoost model on the periodic hills hold-out test case. The generalization performance for the parametric bump

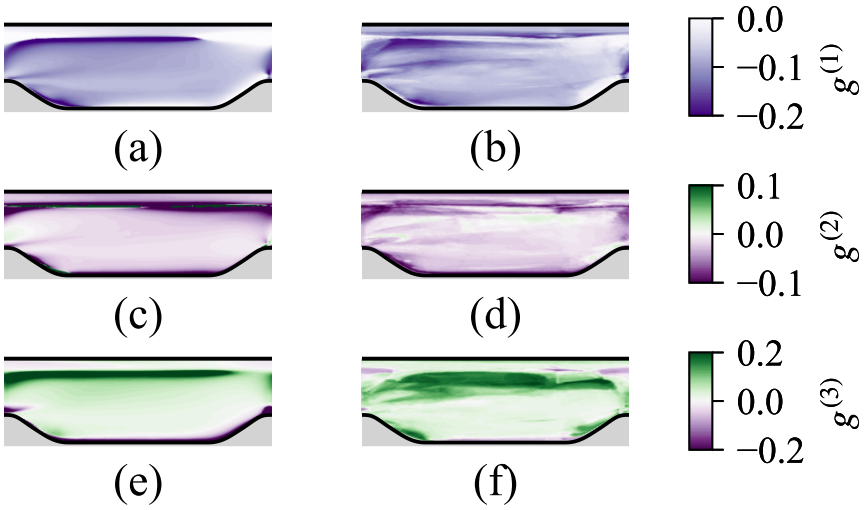


Figure 4.12: Generalizability test on the periodic hills test case for the XGBoost model. Here, the training dataset is the parametric bump BUMP4 set: (a) true $g^{(1)}$ field; (b) predicted $g^{(1)}$ field; (c) true $g^{(2)}$ field; (d) predicted $g^{(2)}$ field; (e) true $g^{(3)}$ field; and, (f) predicted $g^{(3)}$ field.

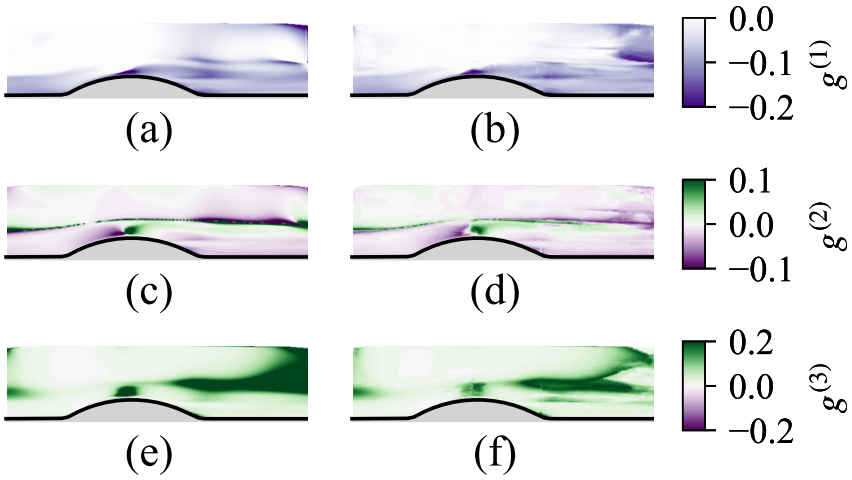


Figure 4.13: Generalizability test on the parametric bump test case for the XGBoost model. Here, the training dataset is the parametric bump BUMP4 set: (a) true $g^{(1)}$ field; (b) predicted $g^{(1)}$ field; (c) true $g^{(2)}$ field; (d) predicted $g^{(2)}$ field; (e) true $g^{(3)}$ field; and, (f) predicted $g^{(3)}$ field.

training dataset to periodic hills test dataset appears better than that for the reverse case. For the $g^{(1)}$ field, an increase in magnitude above the separation region is predicted. For the $g^{(2)}$ field, the same increase is predicted, in addition to an increase near the lower wall. The $g^{(3)}$ field predictions exhibit similar characteristics. However, the shape and extent of these regions are not nearly as accurate as the predictions provided by the periodic hills trained XGBoost model.

In Table 4.4, the CV MSE for this model is 2.484×10^{-4}, whereas that for the periodic hills test MSE is 1.172×10^{-3} (viz., approximately an order of magnitude worse). Despite this poor performance on the periodic hills test case, the model generalizes well to the hold-out parametric bump test case—Figure 4.13 shows that the predicted and corresponding label fields are nearly identical for the parametric bump trained XGBoost model applied to the hold-out parametric bump test case. The MSE for the parametric bump hold-out test case is 1.262×10^{-4} which is on the same order of magnitude as the CV MSE.

Whereas Figures 4.10 to 4.13 demonstrate the generalization gap for two similar 2D flows, Appendix D contains several other visualizations (including the generalization performance for the random forest and neural network models). The poor generalization performance shown here extends to all generalizability tests in the present study.

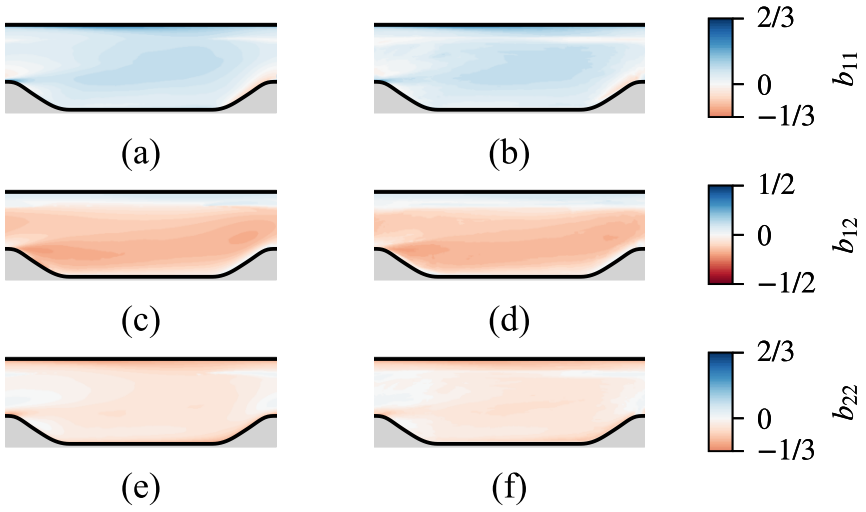


Figure 4.14: Components of b calculated using the $g^{(n)}$ coefficients from Figure 4.10, and the tensors from the base k-ω SST model. Here, the training dataset is the periodic hills PHLL4 set: (a) true b_{11} field; (b) predicted b_{11} field; (c) true b_{12} field; (d) predicted b_{12} field; (e) true b_{22} field; and, (f) predicted b_{22} field.

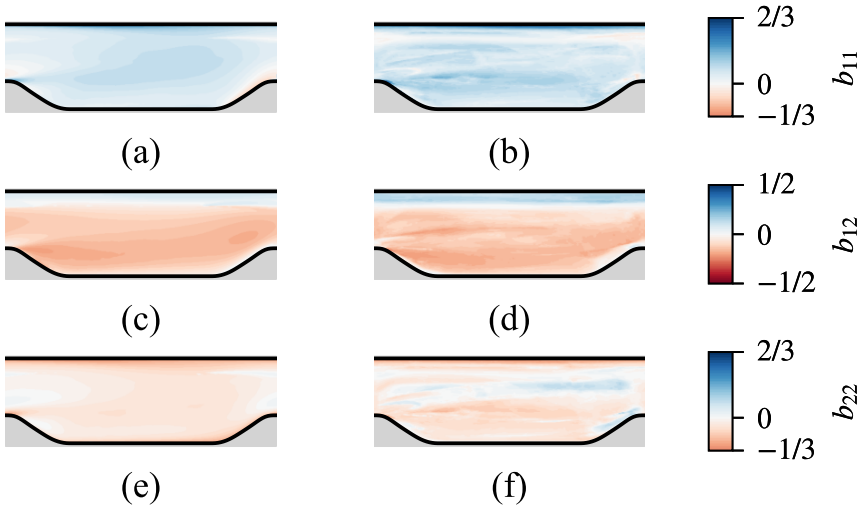


Figure 4.15: Components of b calculated using the $g^{(n)}$ coefficients from Figure 4.12, and the tensors from the base k-ω SST model. Here, the training dataset is the periodic hills BUMP4 set: (a) true b_{11} field; (b) predicted b_{11} field; (c) true b_{12} field; (d) predicted b_{12} field; (e) true b_{22} field; and, (f) predicted b_{22} field.

To further examine the generalizability issue, the anisotropy tensor itself was calculated using the predicted and true $g^{(n)}$ coefficients (Section 4.3.1). Figures 4.14 and 4.15 compare the predicted and true values for each component of b for the XGBoost model, for two training sets. While the coefficients $g^{(n)}$ were used in this study as a surrogate for an invariant closure quantity, the anisotropy tensor $b = \sum_{n=1}^{3} g^{(n)} T^{(n)}$ is the closure term. Figure 4.14 shows that the coefficients from Figure 4.10 results in an accurately predicted anisotropy tensor. This is expected—accurate predictions of $g^{(n)}$ should lead to accurate computations of b. However, Figure 4.15 shows that when $g^{(n)}$ are predicted poorly, the prediction of b can be inaccurate. The coefficients in Figure 4.12 are predicted poorly as a results of the generalization error, which translates to errors in b (Figure 4.15). While b_{11} is predicted with minimal error, b_{12} is overpredicted near the top of the domain. The prediction for b_{22} shows a strong positive value above the right hill, which is not present in the true b_{22} field.

4.4 Conclusions

The confluence of machine learning and turbulence closure modelling has led to recent advances in improving the accuracy of RANS simulations. A wide range of closure frameworks and machine learning models have been used to develop sophisticated methods for approximating the Reynolds stress tensor and closing the mean momentum equation in RANS modelling. The goals of this work were to investigate the performance of multiple machine learning model types for use in data-driven turbulence closure modelling, and to test the generalizability of these augmented closure models.

Towards these goals, we proposed a representative "invariant closure framework" that was model-agnostic. This representative closure is an optimal three-term tensor basis expansion of the anisotropy tensor, which forms a complete basis for 2D flows (and a least-squares approximation of a basis for 3D flows). The key closure parameters are three scalar coefficients, one for each basis tensor. The data-driven turbulence closure framework was then formulated as predicting optimal values of these coefficients from a variety of flow input features. We then performed hyperparameter tuning and evaluation of a random forest model, a neural network model, and an XGBoost model. To test generalizability on new flows, we restricted the training dataset to a particular flow type, and then systematically tested the trained model on other flow types. This process was repeated across three different flow types in order to determine the effect of the training dataset on the generalization of the trained turbulence closure model.

Our findings indicate that the XGBoost model achieves a good balance between performance and computational cost. While the raw cross-validation and test dataset performance of the XGBoost model was only slightly better than that of the tuned random forest model, the computational cost was much lower for the XGBoost model tuning process. Both tree-based models outperformed the neural network model. These results are in agreement with findings in the broader machine learning community, which indicate that gradient boosted tree-based ensemble models are often faster and more accurate than neural network predictors for tabular data problems. In the present work, the neural network model was tuned in less wall-clock time than the XGBoost model. However, the XGBoost model training could be accelerated in future studies through the use of GPU computations.

Though the machine learning models generalized well to new parameter variations within the same flow type (e.g., geometry or Reynolds number), the generalization performance on new flows was not nearly as good. These results point towards a slightly more pessimistic outlook for developing a "universal turbulence model" using machine learning

methods. While a machine learning model can be trained to predict a closure quantity to high accuracy within the same flow type, the model generalization on new types of flows is not reliable.

The results in the present study were limited to strictly *a priori* tests—the predicted closure fields were not injected to update the velocity field. We limited the investigation in this manner in order to allow a more in-depth study of the generalization performance, without introducing other factors such as stability issues after injection. We believe the findings around lack of generalizability will also extend to velocity fields after injection (if they were to converge). If the predicted closure quantities are inaccurate and/or unstable, accuracy of the resulting velocity fields cannot be anticipated.

On a more optimistic note, the generalization gap can be reduced by incorporating a richer and broader set of flows in the training dataset. In the present work, a model trained on a variety of flows was able to generalize well within new variations of all flows in the training dataset. As the wide availability of open-source datasets in turbulence increases, the ability to incorporate a broader set of flows in a training dataset also increases. While the dataset used in the present work is perhaps the most diverse single-format dataset investigated to date, it only consists of five different 2D flows. We therefore recommend that more DNS and LES datasets be released publicly in order to accelerate data-driven modelling.

Furthermore, the lack of generalizability shown in the present work is not surprising. For decades, turbulence modellers have battled with the "no free lunch theorem"—a sophisticated model tuned for one flow often does not generalize well to other flows. Despite this, models highly suited for particular flow types are still widely used. For many industrial applications, a universal turbulence model (if such a model even exists) is not required—a tailored turbulence model for the particular application is what is needed. As an example, the Spalart-Allmaras one-equation model has received widespread use for external aerodynamics simulations [106], where it performs well despite lack of generalizability to some other flows. This suggests that many industrial users prefer an accurate purpose-built turbulence model over a less accurate "universal" turbulence model. Several investigations have already developed problem-specific data-driven closures with good success [73, 107]

We believe the use in machine learning methods for turbulence modelling simply offers a new set of tools and paradigms for developing turbulence closures that are highly accurate within a specific flow type. Nearly all investigations into data-driven RANS closures demonstrate excellent performance, and in many cases the resulting velocity fields can be almost identical to DNS results. Machine learning methods offer industrial users a way to leverage large (internal or public) datasets to develop accurate models for their own use

cases. For example, an automotive user can develop a model which is highly accurate for predicting separated flow over vehicles, but that does not generalize well for flows through rotating turbomachinery. Another user focused on rotating flows can use machine learning to develop a well-tuned turbulence closure model for rotating flows that does not necessarily generalize well to aerodynamic flows. Whereas hand-tuning nonlinear or sophisticated RANS models in the past was expensive or difficult, machine learning methods offer a data-driven alternative to develop turbulence closure models for specific use cases.

Chapter 5

Realizability-informed machine learning for turbulence anisotropy mappings

Associated reference:

R. McConkey, E. Yee, and F. S. Lien, "Realizability-Informed Machine Learning for Turbulence Anisotropy Mappings", *arXiv* (2024), https://arxiv.org/abs/5672858.

Abstract

Within the context of machine learning-based closure mappings for RANS turbulence modelling, physical realizability is often enforced using ad-hoc postprocessing of the predicted anisotropy tensor. In this study, we address the realizability issue via a new physics-based loss function that penalizes non-realizable results during training, thereby embedding a preference for realizable predictions into the model. Additionally, we propose a new framework for data-driven turbulence modelling which retains the stability and conditioning of optimal eddy viscosity-based approaches while embedding equivariance. Several modifications to the tensor basis neural network to enhance training and testing stability are proposed. We demonstrate the conditioning, stability, and generalization of the new framework and model architecture on three flows: flow over a flat plate, flow over periodic hills, and flow through a square duct. The realizability-informed loss function is demonstrated to significantly increase the number of realizable predictions made by the model when

generalizing to a new flow configuration. Altogether, the proposed framework enables the training of stable and equivariant anisotropy mappings, with more physically realizable predictions on new data. We make our code available for use and modification by others.

5.1 Introduction

It is prohibitively expensive to resolve all relevant scales of turbulence for industrially relevant flows. Even with increasing computational capacity, Kolmogorov microscale-resolving techniques such as Direct Numerical Simulation (DNS) will be out of reach for decades [6]. In order to enable the practical simulation of turbulent flows, a variety of techniques are currently in use, from Reynolds-Averaged Navier Stokes (RANS), Detached Eddy Simulation (DES), and Large Eddy Simulation (LES). Each technique comes with its own advantages, with the user's available computational resources being the primary consideration. RANS, a technique which models turbulence as a single-scale phenomenon, remains the most popular industrial technique due to the computational costs of scale resolving simulations [5].

In recent years, a new menu item for industrial turbulence modelling is emerging. Machine learning has been used to augment RANS, DES, LES, and DNS, with various objectives, such as accelerating simulation times [20], infer optimal coefficient fields [18], model calibration [72], and turbulence model augmentation [12, 17]. Within the context of RANS simulations, a promising new menu item is the ability to train a specialized turbulence model with a given industrial dataset. Significant attention has been given to the development of machine learning-augmented closure models. Specifically, flow-specific sensitization of the stable but often inaccurate linear eddy viscosity relationship via machine learning is an area of major interest. In their seminal work, Ling et al. [23] proposed a neural network architecture based on a tensor basis expansion of the anisotropy tensor, referred to as a tensor basis neural network (TBNN). This architecture was extended to random forests by Kaandorp [34], and Kaandorp and Dwight [35]. The TBNN architecture has been used in several studies, such as by Song et al. [39] and Zhang et al. [37]. Further modifications to the TBNN framework within the context of simple channel flows have been proposed by Cai et al. [108, 109]. Specifically, Cai et al. studied issues related to non-unique mappings between the closure term and input features for plane channel flow, an issue reported by Liu et al. [66]. This issue primarily occurs when a small input feature set is used, such as the 5 invariants (several of which are zero for 2D flows) used in Ling et al.'s original TBNN [23]. Several strategies have been proposed to address the issue of a non-unique mapping, including incorporation of additional input features [33], and ensembling

through a divide and conquer approach by Man et al. [110]. In the present investigation, we address the issue related to non-uniqueness of the mapping via use of a rich input feature set. In an effort to make these mappings more transparent, McConkey et al. [93], and Mandler and Bernhard [111] analyzed predictions made via an anisotropy mapping using input feature importance metrics such as SHapley Additive exPlanations (SHAP) values. Interpretability analysis sheds light on which RANS features contain more information about different flow physics, which helps guide future input feature selection [111]. TBNN-type models are not the only architecture in use—others include the eigenvalue reconstruction technique proposed by Wu et al. [33, 112], and the Reynolds force vector approach proposed by Cruz et al. [75], and further investigated by Brener et al. [95] and Amarloo et al. [113].

Here, we consider the specific problem of training data-driven anisotropy mappings between RANS and a higher fidelity closure term from LES or DNS. Several open questions remain in the area of machine learning anisotropy mappings for RANS, with perhaps the most popular question being whether a "universal turbulence model" could be produced via machine learning [22, 114]. However, when it comes to generalizability of these mappings, the no-free-lunch theorem is at play as demonstrated in McConkey et al. [115]. Accurate sensitization of the anisotropy mapping for a given flow comes at the cost of generalization to completely new flows. Nevertheless, numerous studies show that the sensitized mapping generalizes well within the same flow type. While this lack of new flow generalizability means that a machine learning augmented turbulence model is off the menu for some applications, it remains on the menu for many industrial applications. For example, an industrial user with LES data can leverage these techniques to develop an augmented turbulence model sensitized to a flow of interest. Our opinion is that this lack of generalizability is due to a lack of sufficient data. Despite numerous datasets for this purpose being available, we believe the entire space of possible mappings between RANS input features and higher fidelity closure terms is still sparsely covered by available data. For the foreseeable future, these mappings will therefore be limited to flow-specific sensitization.

While the generalizability question is perhaps the most popular one, several other key issues remain in training anisotropy mappings for RANS. In this investigation, we address several important remaining questions. We formulate an injection framework which unites tensor basis neural network (TBNN)-type model architectures with the more stable optimal eddy viscosity-based techniques. This unification enables stable use of the equivariance-enforcing TBNN within the momentum equation, without the use of stabilizing blending factors [35]. The proposed framework also has the advantage of producing a well-conditioned solution without the use of an optimal eddy viscosity, an often-unstable quantity that is difficult to predict via a machine learning model. We also target the prob-

lem of producing realizable predictions via a TBNN-type architecture via a physics-based loss function penalty (viz., incorporation of a learning bias in the framework). We also further investigate the invariant input feature sets commonly used for TBNN architectures, and provide new insights as to which input features are appropriate for use in flows with certain zero gradient directions. We demonstrate good generalization performance of the "realizability-informed" TBNN. We also demonstrate that the realizability-informed loss function greatly reduces non-realizable predictions when generalizing.

The present work is the first to unite TBNN-type frameworks (e.g. Ling et al. [23], Kaandorp [34], Kaandorp and Dwight [35], Man et al. [110]) with optimal eddy viscosity frameworks (e.g. Wu et al. [33], Brener et al. [76], and McConkey et al. [93]). The advantage here is maintaining a stable injection environment [77, 76], while also retaining the simplicity, elegance, and implicit equivariance of the TBNN architecture. Whereas most existing techniques to enforce realizability of the neural network involve an ad-hoc post-processing step, our technique leverages physical realizability as an additional training target, thereby embedding an additional physics-based (learning) bias into the model. This idea extends the learning bias proposed by Riccius et al. [116]. Lastly, despite widespread use of minimal integrity basis input features for 2D flows, and flows through a square duct, the present investigation is the first to systematically examine these input features for flow through a square duct. This examination leads to several unsuitable input features being identified, and a codebase for investigating other flows of interest.

This manuscript is organized as follows. Section 5.2 describes the novel techniques in detail, including the injection framework (5.2.1–5.2.2), realizability-informed loss function (5.2.3), and improved TBNN architecture (5.2.4). Details on the datasets, input features, hyperparameters, implementation, and code availability are given in Section 5.2.5. Two primary results are presented in Section 5.3: generalization tests of the realizability-informed TBNN (5.3.1), and an examination of how realizability-informed training produces more realizable predictions when generalizing (5.3.2). Conclusions and future work are discussed in Section 5.4.

5.2 Methodology

5.2.1 Reynolds stress tensor decompositions used in data-driven closure modelling frameworks

The steady-state, Reynolds-Averaged Navier-Stokes momentum equations for an incompressible, Newtonian fluid are given by

$$U_i \frac{\partial U_j}{\partial x_i} = -\frac{1}{\rho}\frac{\partial P}{\partial x_j} + \nu \frac{\partial^2 U_j}{\partial x_i \partial x_i} - \frac{\partial \tau_{ij}}{\partial x_i} \ , \tag{5.1}$$

where U_j is the mean velocity, P is the mean pressure, ρ is the fluid density, ν is the kinematic viscosity, and τ_{ij} is the Reynolds stress tensor.

The continuity equation also applies to this flow:

$$\frac{\partial U_i}{\partial x_i} = 0 \ . \tag{5.2}$$

Together, Equations 5.1 and 5.2 are unclosed. In the most general case, there are four equations (continuity + 3 momentum), and 10 unknowns: P, 3 components of U_i, and 6 components of τ_{ij}. The goal of turbulence closure modelling is to express τ_{ij} in terms of P and U_i.

The Reynolds stress tensor can be decomposed into isotropic (hydrostatic) and anisotropic (deviatoric) components:

$$\tau_{ij} = \frac{2}{3} k \delta_{ij} + a_{ij} \ , \tag{5.3}$$

where $k = \frac{1}{2}\tau_{kk}$ is half the trace of the Reynolds stress tensor, and a_{ij} is the anisotropy tensor. The isotropic component ($\frac{2}{3}k\delta_{ij}$) in Equation 5.3 can be absorbed into the pressure gradient term in Equation 5.1 to form a modified pressure:

$$U_i \frac{\partial U_j}{\partial x_i} = -\frac{1}{\rho}\frac{\partial}{\partial x_j}\left(P + \frac{2}{3}\rho k\right) + \nu \frac{\partial^2 U_j}{\partial x_i \partial x_i} - \frac{\partial a_{ij}}{\partial x_i} \ . \tag{5.4}$$

Equation 5.4 is the form of the RANS momentum equation commonly used in RANS turbulence closure modelling. Several data-driven closure modelling frameworks are based on modelling the Reynolds stress tensor itself, or it's divergence (as in Brener et al. [95]), which imply the use of Equation 5.1. In our work, we model the Reynolds stress anisotropy tensor a_{ij}, implying the use of Equation 5.4.

In an eddy viscosity hypothesis, the anisotropy tensor a_{ij} is postulated to be a function of the mean strain rate S_{ij} and rotation rate R_{ij} tensors:

$$a_{ij} = a_{ij}(S_{ij}, R_{ij}) \ , \tag{5.5}$$

$$S_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \ , \tag{5.6}$$

$$R_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i}\right) \ . \tag{5.7}$$

The eddy viscosity hypothesis implies the following important constraints:

- The anisotropy tensor can be predicted locally and instantaneously. In Equation 5.5, there is no temporal dependence.
- The anisotropic turbulent stresses are caused entirely by mean velocity gradients.

The most common eddy viscosity hypothesis is the linear eddy viscosity hypothesis:

$$a_{ij} = -2\nu_t S_{ij} \ , \tag{5.8}$$

where ν_t is the eddy viscosity, a scalar. This linear hypothesis draws direct analogy from the stress-strain rate relation for a Newtonian fluid. Along with the important constraints implied by invoking an eddy viscosity hypothesis, the linear eddy viscosity hypothesis implies the following:

- The anisotropy tensor is aligned with the mean strain rate tensor.
- The mapping between mean strain rate and anisotropic stress is isotropic, in that it can be represented using a single scalar (ν_t).

More general non-linear eddy viscosity hypotheses have been used in several models. The primary advantage of these models is that they permit a misalignment of the principal axes of a_{ij} and S_{ij}, which occurs for even simple flows. Applying Cayley-Hamilton theorem to Equation 5.5, Pope derived the most general expression for a non-linear eddy viscosity model:

$$a_{ij} = 2k \sum_{n=1}^{10} g_n \hat{T}_{ij}^{(n)} \ , \tag{5.9}$$

where $n = 1, 2, ...10$ indexes the scalar coefficients g_n, and the following basis tensors:

$$\hat{T}_{ij}^{(1)} = \hat{S}_{ij} \ , \qquad \hat{T}_{ij}^{(6)} = \hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lj} + \hat{S}_{ik}\hat{R}_{kl}\hat{R}_{lj} - \tfrac{2}{3}\hat{S}_{kl}\hat{R}_{lm}\hat{R}_{mk}\delta_{ij} \ ,$$
$$\hat{T}_{ij}^{(2)} = \hat{S}_{ik}\hat{R}_{kj} - \hat{R}_{ik}\hat{S}_{kj} \ , \qquad \hat{T}_{ij}^{(7)} = \hat{R}_{ik}\hat{S}_{kl}\hat{R}_{lm}\hat{R}_{mj} - \hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{R}_{mj} \ ,$$
$$\hat{T}_{ij}^{(3)} = \hat{S}_{ik}\hat{S}_{kj} - \tfrac{1}{3}\hat{S}_{kl}\hat{S}_{lk}\delta_{ij} \ , \qquad \hat{T}_{ij}^{(8)} = \hat{S}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{S}_{mj} - \hat{S}_{ik}\hat{S}_{kl}\hat{R}_{lm}\hat{S}_{mj} \ ,$$
$$\hat{T}_{ij}^{(4)} = \hat{R}_{ik}\hat{R}_{kj} - \tfrac{1}{3}\hat{R}_{kl}\hat{R}_{lk}\delta_{ij} \ , \qquad \hat{T}_{ij}^{(9)} = \hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{S}_{mj} + \hat{S}_{ik}\hat{S}_{kl}\hat{R}_{lm}\hat{R}_{mj} - \tfrac{2}{3}\hat{S}_{kl}\hat{S}_{lm}\hat{R}_{mo}\hat{R}_{ok}\delta_{ij} \ ,$$
$$\hat{T}_{ij}^{(5)} = \hat{R}_{ik}\hat{S}_{kl}\hat{S}_{lj} - \hat{S}_{ik}\hat{S}_{kl}\hat{R}_{lj} \ , \qquad \hat{T}_{ij}^{(10)} = \hat{R}_{ik}\hat{S}_{kl}\hat{S}_{lm}\hat{R}_{mo}\hat{R}_{oj} - \hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{S}_{mo}\hat{R}_{oj} \ .$$

The tensors $\hat{S}_{ij}$ and $\hat{R}_{ij}$ above are the non-dimensionalized strain rate and rotation rate tensors. In Pope's original work, these tensors are given by

$$\hat{S}_{ij} = \frac{k}{\varepsilon} S_{ij} \ , \tag{5.10}$$

$$\hat{R}_{ij} = \frac{k}{\varepsilon} R_{ij} \ . \tag{5.11}$$

$$\tag{5.12}$$

However, other normalization constants are possible, and in several non-linear eddy viscosity models, different basis tensors use different normalization constants. To our knowledge, varying the basis tensor normalization has not been explored in data-driven turbulence closure modelling.

With Pope's general expression for the anisotropy tensor, Equation 5.4 becomes

$$U_i \frac{\partial U_j}{\partial x_i} = -\frac{1}{\rho}\frac{\partial}{\partial x_j}\left(P + \frac{2}{3}\rho k\right) + \nu \frac{\partial^2 U_j}{\partial x_i \partial x_i} - \frac{\partial}{\partial x_i}\left(2k \sum_{n=1}^{10} g_n \hat{T}_{ij}^{(n)}\right) \ . \tag{5.13}$$

This is the form of the momentum equation used in several studies which aim to augment the closure relationship via machine learning. For example, in Ling et al.'s "tensor basis neural network" (TBNN) investigation [23], Equation 5.13 was used. Kaandorp [34], and Kaandorp and Dwight [35] also used this form of the momentum equation. This closure term is highly expressive, in that ten different combinations of S_{ij} and R_{ij} can be used to represent the anisotropy tensor. However, a major disadvantage with numerically solving Equation 5.13 is that the closure term is entirely explicit, greatly reducing numerical

stability. For this reason, Kaandorp and Dwight needed to implement a blending function, which blends the fully explicit closure term in Equation 5.13 with the more stable implicit closure term treatment made possible with a linear eddy viscosity hypothesis. Assuming the closure term takes the form $a_{ij} = -2\nu_t S_{ij}$, Equation 5.4 can be written as

$$U_i \frac{\partial U_j}{\partial x_i} = -\frac{1}{\rho}\frac{\partial}{\partial x_j}\left(P + \frac{2}{3}\rho k\right) + \frac{\partial}{\partial x_i}\left[(\nu + \nu_t)\frac{\partial U_j}{\partial x_i}\right]. \tag{5.14}$$

Equation 5.14 has the major advantage of increasing diagonal dominance of the U_j coefficient matrix obtained from discretization of this equation, via the eddy viscosity. However, this closure framework only permits the inaccurate linear eddy viscosity closure approximation.

In the present work, we propose the following hybrid treatment of the closure term:

$$U_i \frac{\partial U_j}{\partial x_i} = -\frac{1}{\rho}\frac{\partial}{\partial x_j}\left(P + \frac{2}{3}\rho k\right) + \frac{\partial}{\partial x_i}\left[(\nu + \nu_t)\frac{\partial U_j}{\partial x_i}\right] - \frac{\partial}{\partial x_i}\left(2k\sum_{n=2}^{10} g_n \hat{T}_{ij}^{(n)}\right). \tag{5.15}$$

In Equation 5.15, the $n = 1$ (linear) term has been separated and receives implicit treatment, while the remaining $n = 2, 3, ...10$ terms grant an opportunity for a machine learning model to provide rich representation of the non-linear part of a_{ij}. As we will discuss, the separation of the linear term requires special treatment during training and closure term injection.

5.2.2 Conditioning analysis

Various decompositions of the Reynolds stress tensor were investigated by Brener et al.'s conditioning analysis [76]. Conditioning analysis for data-driven turbulence closure frameworks is important, since ill-conditioned momentum equations have the potential to amplify errors in the predicted closure term. Brener et al. [76] concluded that an optimal eddy viscosity approach is necessary to achieve a well-conditioned solution, since it incorporates information about the DNS mean velocity field. In the present work, we demonstrate that the following closure decomposition:

$$a_{ij} = -2\nu_t^{\mathrm{R}} S_{ij}^{\theta} + a_{ij}^{\perp} \tag{5.16}$$

also achieves a well conditioned solution. We follow the nomenclature of Durisamy et al. [17] in that the θ superscript indicates a quantity that comes from a high-fidelity source such as DNS. The R superscript indicates a quantity taken from the corresponding baseline RANS simulation.

The decomposition in Equation 5.16 permits an augmented turbulence closure framework that treats the machine learning correction only in an explicit term in the momentum equation. Separating the machine learning model prediction has several advantages. It allows the model correction to be easily "turned off" in unstable situations. It also is more interpretable—rather than correcting both the eddy viscosity and also injecting an explicit correction term, the correction is contained entirely within an explicit term in the momentum equation. Lastly, it avoids the necessity for an optimal eddy viscosity to be computed from high-fidelity data. Though there are methods to increase the practicality of computing the optimal eddy viscosity [93], this quantity is often unstable and difficult to predict via a machine learning model.

Figures 5.1 and 5.2 demonstrate a conditioning test similar to the tests conducted by Brener et al. [76]. Several different decompositions of the DNS Reynolds stress tensor are injected into a RANS simulation, to identify which decompositions permit a well-conditioned solution. Comparing subfigures (b) and (c) in both Figures 5.1 and 5.2, we can see that the proposed decomposition of the Reynolds stress tensor (Equation 5.16) achieves an equally well-conditioned solution as the optimal eddy viscosity framework. To our knowledge, this is the first result in the literature showing that a well-conditioned solution can be achieved without the use of an optimal eddy viscosity to incorporate information about the DNS velocity field [76]. We further confirm the findings of Brener et al. [76] with respect to the requirement that the closure decomposition includes information about the DNS mean velocity field in order to achieve a well-conditioned solution. We confirm Wu et al.'s [112] notion that implicit treatment helps address the ill-conditioning issue, but using the DNS strain rate tensor S_{ij}^{θ} to calculate the explicitly injected $a_{ij}^{\perp}$.

5.2.3 Realizability-informed training

The Reynolds stress tensor is symmetric positive semidefinite. A set of constraints on the non-dimensional anisotropy tensor b_{ij} arise from this property, as determined by Banerjee et al. [58]. These constraints are:

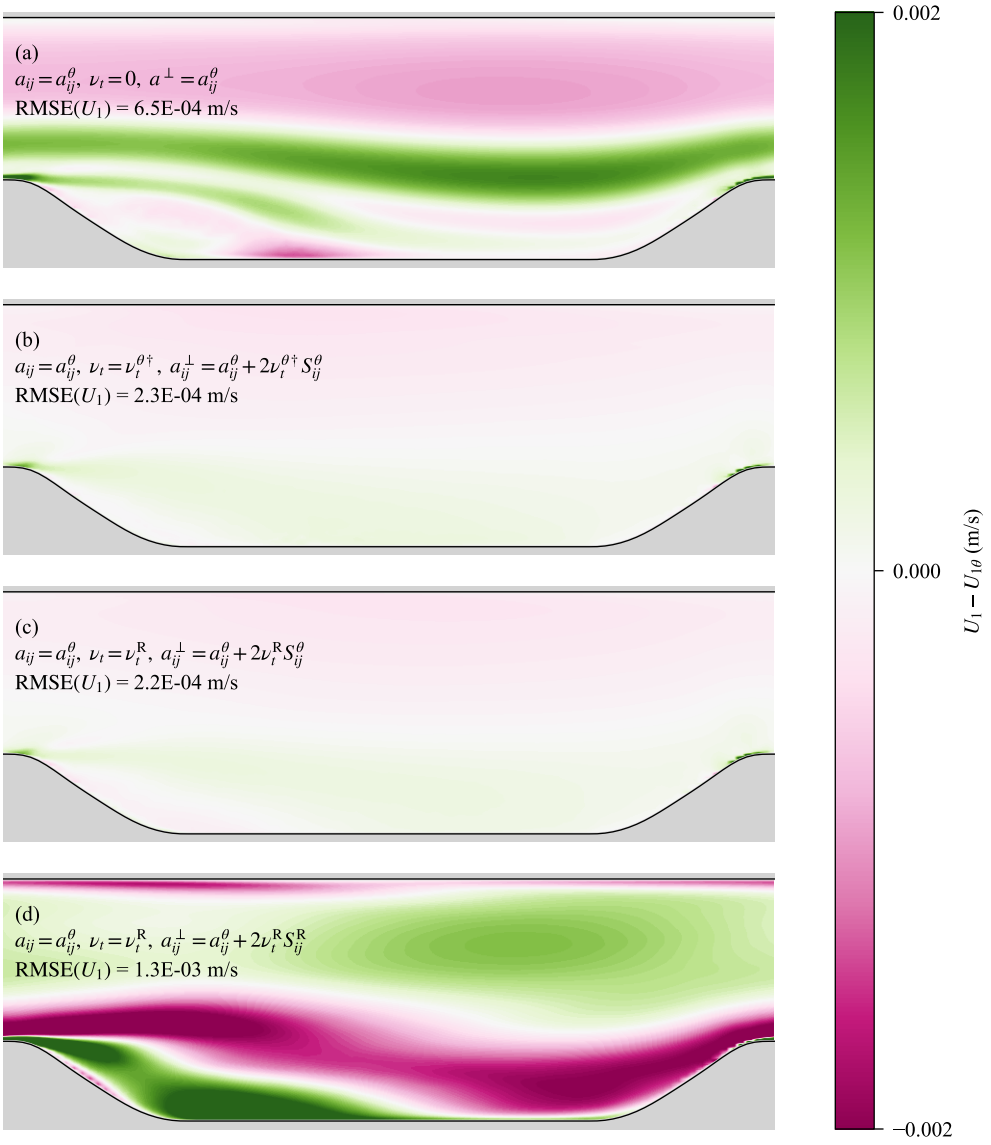


Figure 5.1: Conditioning test comparing the errors in U_1 after injecting: (a) the DNS anisotropy tensor fully explicitly, (b) the optimal eddy viscosity (implicitly), and the remaining non-linear part of the anisotropy tensor calculated using S_{ij}^{θ} (explicitly), (c) the eddy viscosity estimated by the k-ω SST model (implicitly), and the remaining non-linear part of the anisotropy tensor calculated using S_{ij}^{θ} (explicitly), and (d) the eddy viscosity estimated by the k-ω SST model (implicitly), and the remaining non-linear part of the anisotropy tensor calculated using S_{ij}^{R} (explicitly).

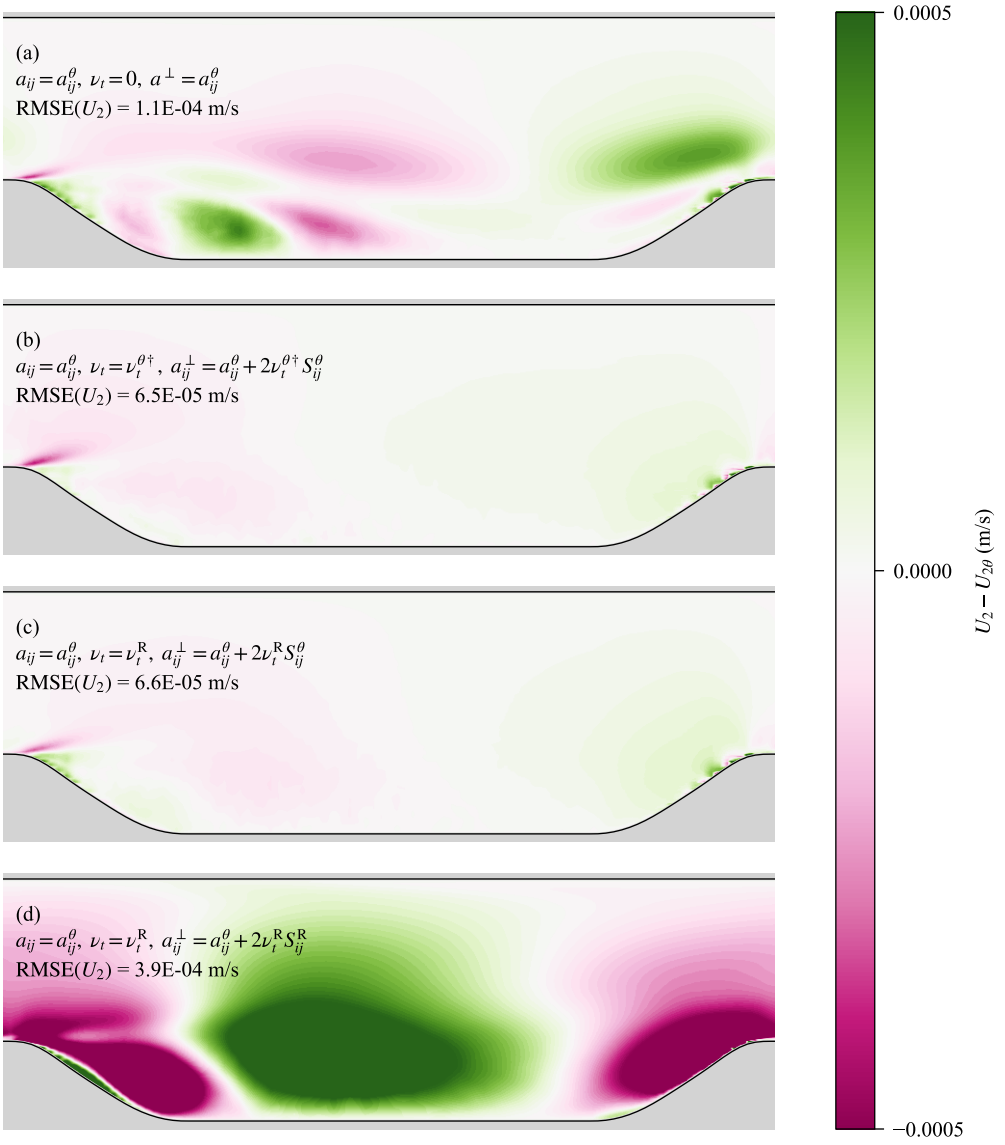


Figure 5.2: Conditioning test comparing the errors in U_2 after injecting: (a) the DNS anisotropy tensor fully explicitly, (b) the optimal eddy viscosity (implicitly), and the remaining non-linear part of the anisotropy tensor calculated using S_{ij}^{θ} (explicitly), (c) the eddy viscosity estimated by the k-ω SST model (implicitly), and the remaining non-linear part of the anisotropy tensor calculated using S_{ij}^{θ} (explicitly), and (d) the eddy viscosity estimated by the k-ω SST model (implicitly), and the remaining non-linear part of the anisotropy tensor calculated using S_{ij}^{R} (explicitly).

$$-\frac{1}{3} \leq b_{ij} \leq \frac{2}{3} \, , i = j \, , \tag{5.17}$$

$$-\frac{1}{2} \leq b_{ij} \leq \frac{1}{2} \, , i \neq j \, , \tag{5.18}$$

$$\lambda_1 \geq \frac{3|\lambda_2| - \lambda_2}{2} \, , \tag{5.19}$$

$$\lambda_1 \leq \frac{1}{3} - \lambda_2 \, , \tag{5.20}$$

where the nondimensional anisotropy tensor b_{ij} is calculated by

$$b_{ij} = \frac{a_{ij}}{2k} \, , \tag{5.21}$$

and the eigenvalues of b_{ij} are given by $\lambda_1 \geq \lambda_2 \geq \lambda_3$.

A given Reynolds stress tensor is physically realizable if it satisfies these constraints. While the physical realizability of the closure term may seem an important constraint for turbulence models, many commonly used turbulence models such as the k-ε model [51], k-ω [117], and k-ω shear stress transport (SST) model [118, 53] do not guarantee physical realizability.

Based on the widespread acceptance and popularity of non-realizable turbulence models, it is fair to say that realizability is not a hard constraint on a new turbulence model. Nevertheless, the true Reynolds stress tensor is realizable, and if a machine learning model is able to learn to predict realizable closure terms, it may be more physically accurate. Unfortunately, Pope's tensor basis expansion for the anisotropy tensor (and machine learning architectures based on this expansion) do not provide a means to achieve realizability. To enforce realizability, a variety of ad-hoc strategies have been used, including in the original TBNN paper by Ling et al. [23]. Most of these strategies involve postprocessing predictions by the TBNN, such shrinking the predicted anisotropy tensor in certain directions until it is physically realizable [64].

In the present work, we propose including a penalty for violating realizability constraints in the loss function. In a similar spirit of physics-informed neural networks (PINNs), we term the use of this loss function "realizability-informed training". Whereas PINNs encourage the model to learn physics by penalizing violations of conservation laws, realizability-informed TBNN's learn to predict physically realizable anisotropy tensors.

The realizability penalty $\mathcal{R}(b_{ij})$ is given as follows:

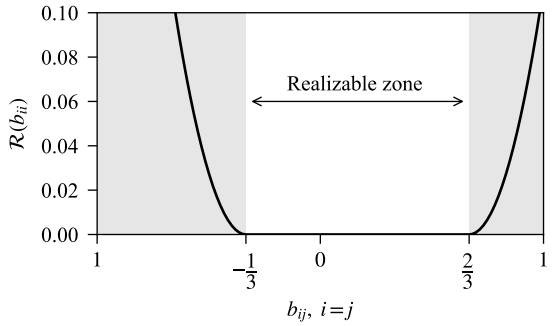
$$\begin{aligned}\mathcal{R}(\tilde{b}_{ij}) = \frac{1}{6}\Bigg\{ & \sum_{i=j}\left(\max\left[\tilde{b}_{ij}-\frac{2}{3},-\left(\tilde{b}_{ij}+\frac{1}{3}\right),0\right]\right)^2 \\ & + \sum_{i\neq j}\left(\max\left[\tilde{b}_{ij}-\frac{1}{2},-\left(\tilde{b}_{ij}+\frac{1}{2}\right),0\right]\right)^2\Bigg\} \\ & + \frac{1}{2}\Bigg\{\left(\max\left[\frac{3|\lambda_2|-|\lambda_2|}{2}-\lambda_1,0\right]\right)^2 + \\ & \left(\max\left[\lambda_1-\left(\frac{1}{3}-\lambda_2\right),0\right]\right)^2\Bigg\}\ ,\end{aligned} \tag{5.22}$$

where the $\sim$ above a symbol denotes a model prediction for the quantity associated with the symbol.

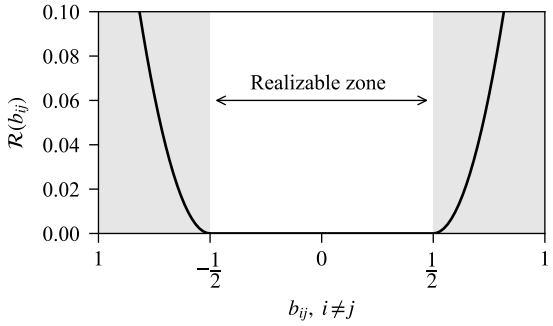
Equation 5.22 can be thought of as the mean squared violation in the components of $\tilde{b}_{ij}$, plus the mean squared violation in the eigenvalues of $\tilde{b}_{ij}$. To help visualize this penalty function, Figure 5.3 shows the penalties incurred by violating various realizability constraints on the anisotropy tensor.

It should be noted that in the same way that a PINN's prediction cannot be guaranteed to satisfy a conservation law, a realizability-informed TBNN cannot be guaranteed to predict a physically realizable anisotropy tensor. The goal is that the incorporation of a physics-based loss at training time will encode into the model a tendency to predict physically realizable anisotropy tensors. At training time, when the model predicts a b_{ij} component outside the realizability zone, then it is penalized in two ways: the error in this prediction will be non-zero (since all label data are realizable), and the realizability penalty will be non-zero. Therefore, the realizability penalty serves as an additional drive towards the realizable, true b_{ij} value. The merits of this approach are demonstrated in Section 5.3.2.

Another important metric in the loss function is the error-based penalty $\mathcal{E}(\tilde{b}_{ij})$. When the model predicts a certain anisotropy tensor, it is evaluated against a known high-fidelity anisotropy tensor available in the training dataset. Typically, mean-squared error loss functions are used to train machine learning augmented closure models. However, we propose the following modifications to the loss function of a TBNN:



(a)



(b)

Figure 5.3: Realizability penalty for the (a) diagonal components of b_{ij}, and (b) off-diagonal components of b_{ij}.

1. Since b_{ij} is a symmetric tensor, we propose to sum the squared errors as follows:

$$\mathcal{E}(\tilde{b}_{ij}) = \frac{1}{6}\left\{ \sum_{\substack{ij\in\{11,12,13,\\22,23,33\}}} (\tilde{b}_{ij} - b_{ij}^{\theta})^2 \right\} . \tag{5.23}$$

   Calculating the squared-error in this way avoids double-penalizing the off-diagonal components, a situation which arises when summing over all components of b_{ij}.

2. Though the TBNN model predicts b_{ij}, we propose to use a loss function based on the error in a_{ij}. Near the wall, $\hat{T}_{ij}^{(n)} \to 0$, but $b_{ij} \nrightarrow 0$.

   The vanishing of $\hat{T}_{ij}^{(n)}$ with non-vanishing b_{ij} causes instabilities during training, leading to $g_n \to \infty$ here. Since a_{ij} is the tensor injected into the momentum equation, its accurate prediction should the focus of the training process. To dimensionalize b_{ij}, the turbulent kinetic energy k must be used. This leads to the following error-based loss:

$$\mathcal{E}(\tilde{a}_{ij}) = \frac{1}{6}\left\{ \sum_{\substack{ij\in\{11,12,13,\\22,23,33\}}} (2k^{\theta}\tilde{b}_{ij} - 2k^{\theta}b_{ij}^{\theta})^2 \right\} = \frac{2(k^{\theta})^2}{3}\left\{ \sum_{\substack{ij\in\{11,12,13,\\22,23,33\}}} (\tilde{b}_{ij} - b_{ij}^{\theta})^2 \right\} , \tag{5.24}$$

   or stated more simply,

$$\mathcal{E}(\tilde{a}_{ij}) = (2k^{\theta})^2\mathcal{E}(\tilde{b}_{ij}) . \tag{5.25}$$

   The reason for the use of k^{θ} in Equation 5.24 and Equation 5.25 is discussed in Section 5.2.4.

The final loss function includes an error-based metric and a realizability-violation penalty. This loss function $\mathcal{L}$ is given by

$$\mathcal{L} = \frac{1}{N}\sum_{p} \frac{(2k_p^{\theta})^2}{\sum_{k=1}^{s} Z_k^2 \mathcal{I}_{C_k}(p)} \left(\mathcal{E}(\tilde{b}_{ij,p}) + \alpha\mathcal{R}(\tilde{b}_{ij,p}) \right) , \tag{5.26}$$

where α is a factor used to control the relative importance of the realizability penalty. N is the total number of points in the dataset. The points are indexed by $p = 1, 2, \ldots, N$.

The cases in the dataset are indexed $k = 1, 2, \ldots, s$. $\mathcal{I}_{C_k}(p)$ is an indicator function:

$$\mathcal{I}_{C_k}(p) = \begin{cases} 1, \text{if } p \in C_k, \\ 0 \text{ otherwise.} \end{cases} \tag{5.27}$$

For a given point p, $\mathcal{I}_{C_k}(p)$ selects all points which come from the same case as p. The denominator for all points from the same case C_k is the same—normalization is applied on a case-by-case basis. A given point is normalized by the mean-squared Frobenius norm of the anisotropy tensor over all the points from the case it comes from. The mean Frobenius norm over a case is given by:

$$Z_k = \frac{1}{|C_k|} \sum_{p \in C_k} \|a_{ij}(p)\|_F \ , \qquad k = 1, 2, \ldots, s \ ,$$

where $|C_k|$ is the cardinality of the case C_k (viz., the number of points in the kth case C_k). Z_k is simply the average of the Frobenius norm $\| \ \cdot \ \|_F$ of the anisotropy tensor a_{ij} for all points p (viz., $a_{ij}(p)$) in case C_k. The objective of this denominator in Equation 5.26 is to promote a more balanced regression problem, since data points from various cases or flow types may have $||a_{ij}||$ that differ by orders of magnitude. Normalization on a case-by-case basis is made in an effort to normalize all a_{ij} error magnitudes to a similar scale.

In this study, we use $\alpha = 10^2$ to encode a high preference for realizable results in Section 5.3. Lower values of α will reduce the penalty applied to realizability violations, which may be necessary for flows in which the anisotropy tensor is difficult to predict via a TBNN. As discussed, the multiplicative term $(2k^\theta)^2$ is used to formulate the loss function in terms of predicting a_{ij} rather than b_{ij}. However, $\mathcal{R}(\tilde{b}_{ij})$ is also multiplied by $(2k^\theta)^2$ to ensure that the realizability penalty and mean-squared error in a_{ij} are of similar scales.

5.2.4 Neural network architecture

Motivated by improving the training and injection stability, as well as generalizability, we propose several modifications to Ling et al.'s [23] original tensor basis neural network (TBNN).

The original TBNN is shown in Figure 5.4. At training time, this network predicts the non-dimensional anisotropy tensor b_{ij}. All basis tensors used in this prediction at training time come from RANS, and during training the prediction $\tilde{b}_{ij}$ is evaluated against a known value of b_{ij}^θ from a high-fidelity simulation. At injection time, the network is

used in this same configuration to predict $\tilde{b}_{ij}$. $\tilde{b}_{ij}$ is injected into a coupled system of equations consisting of the continuity/momentum equations (explicit injection), as well as the turbulence transport equations. This system of equations is iterated around a fixed $\tilde{b}_{ij}$, to obtain an updated estimate for the turbulent kinetic energy k, and therefore an updated estimate for $\tilde{a}_{ij} = 2k\tilde{b}_{ij}$.

The modified TBNN is shown in Figure 5.5. This TBNN relies on the same tensor basis expansion as the original TBNN. However, the linear term has been modified in this expansion. Whereas the original TBNN uses

$$\hat{T}_{ij}^{(1)} = \frac{k^{\mathrm{R}}}{\varepsilon^{\mathrm{R}}} S_{ij}^{\mathrm{R}} \ , \tag{5.28}$$

our modified TBNN uses

$$\hat{T}_{ij}^{(1)} = \frac{\nu_t^{\mathrm{R}}}{k^{\theta}} S_{ij}^{\theta} \ . \tag{5.29}$$

Further, while the original TBNN calculates the linear component of b_{ij} as

$$b_{ij}^{\mathrm{L}} = g_1 \hat{T}_{ij}^{(1)} = g_1 \frac{k}{\varepsilon} S_{ij} \ , \tag{5.30}$$

our modified TBNN uses

$$b_{ij}^{\mathrm{L}} = -\hat{T}_{ij}^{(1)} = -\frac{\nu_t}{k} S_{ij} \ , \tag{5.31}$$

where the superscript R denotes a quantity that comes from the original RANS simulation.

These changes are motivated by the following:

1. At injection time, we use implicit treatment of the linear term $\hat{T}_{ij}^{(1)}$ to formulate Equation 5.15 in a stable manner. After injection, S_{ij} will continue to evolve. In a similar spirit as optimal eddy viscosity frameworks, we therefore use S_{ij}^{θ} to compute the linear component at training time. In optimal eddy viscosity based frameworks, using $S_{ij} = S_{ij}^{\theta}$ at training time helps drive $U_j \to U_j^{\theta}$ at injection time (the cause of this behavior is currently unknown). In Ling et al.'s [23] original TBNN, all basis tensors (and therefore b_{ij}) remain fixed after injection. We also fixed $\nu_t = \nu_t^{\mathrm{R}}$ at injection time (viz., no further evolution of the eddy viscosity is permitted).

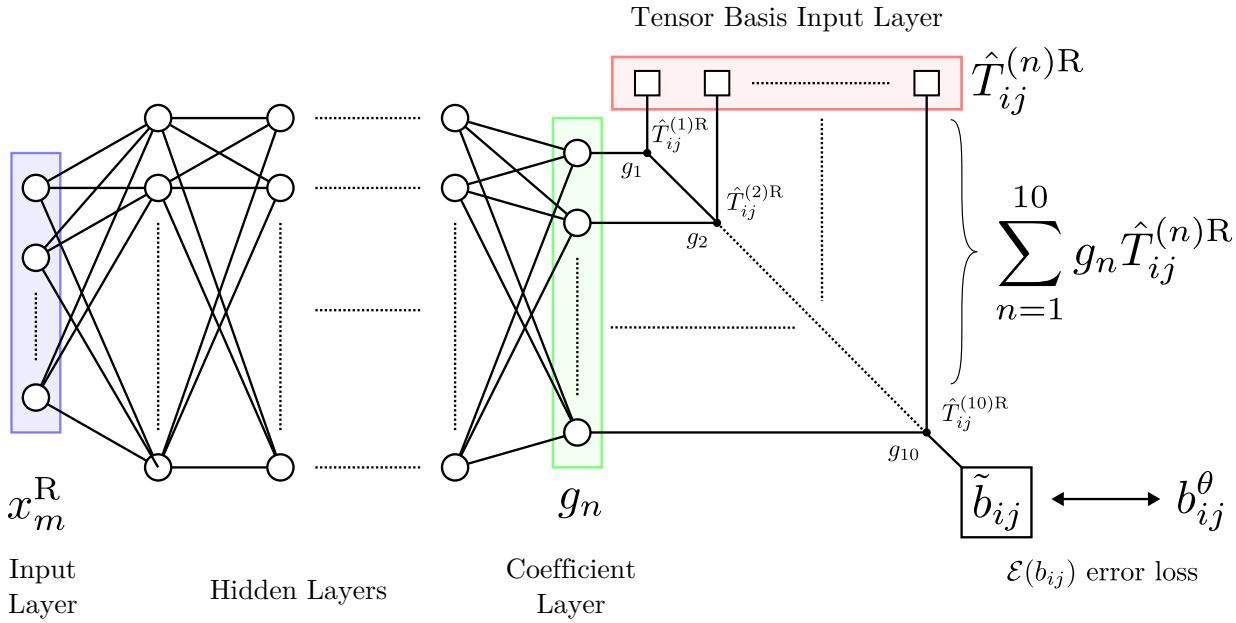


Figure 5.4: Model architecture and training configuration for the original TBNN proposed by Ling et al. [23].

2. At training time, we dimensionalize $\tilde{b}_{ij}$ using k^{θ}: $\tilde{a}_{ij} = 2k^{\theta}\tilde{b}_{ij}$. At evaluation time, we do not have k^{θ}. However, the need for k^{θ} is avoided, since *we only use the non-linear part of* $\tilde{b}_{ij}$ *at test time*. The reason we only need $\tilde{b}_{ij}^{\perp}$ at test time is that the training process has been designed to use the linear part of b_{ij} estimated by the RANS turbulence model, and augment this by the TBNN's equivariant prediction for $\tilde{b}_{ij}^{\perp}$.

3. Using ν_t/k to normalize the basis tensors and fixing $g_1 = -1$ in Equation 5.31 results in the RANS prediction for b_{ij}^{L} being implicitly used in the TBNN. Therefore, the TBNN learns to correct b_{ij} using $b_{ij}^{\perp}$ in a way that allows a realizability-informed training process, and fully implicit treatment of the linear term at injection time.

Lastly, the use of k^{θ} to dimensionalize $\tilde{b}_{ij}$ is also enabled by our use of a separate neural network to correct k^{R} at injection time. This neural network is called the k-correcting neural network (KCNN), and is a simple fully-connected feed-forward neural network that predicts a single output scalar Δ:

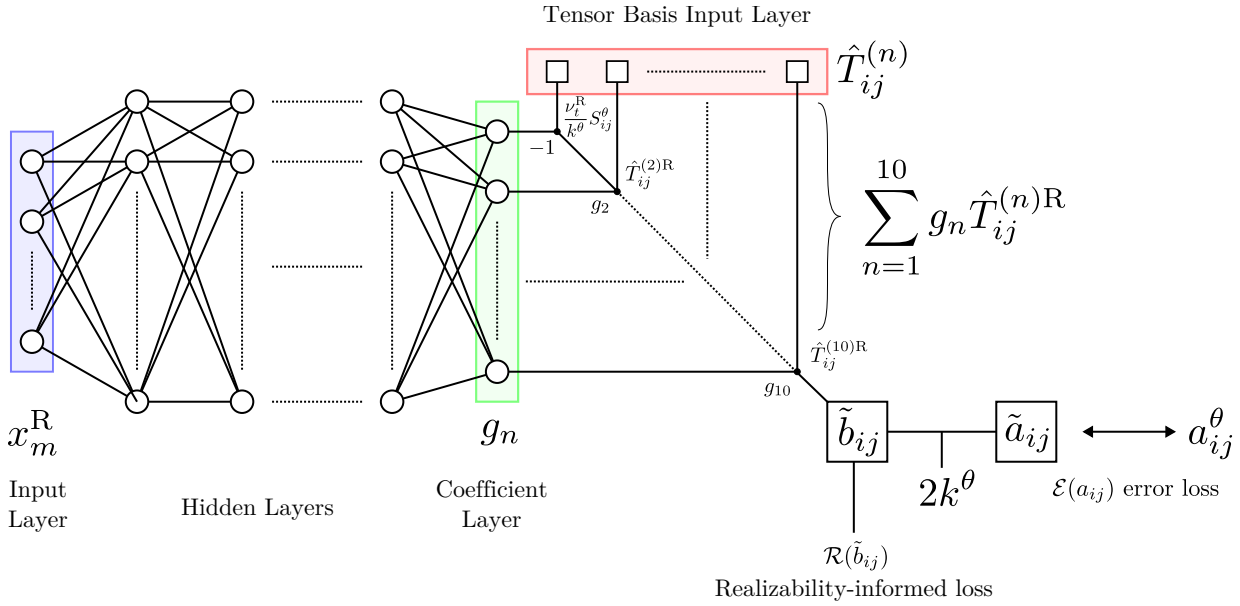
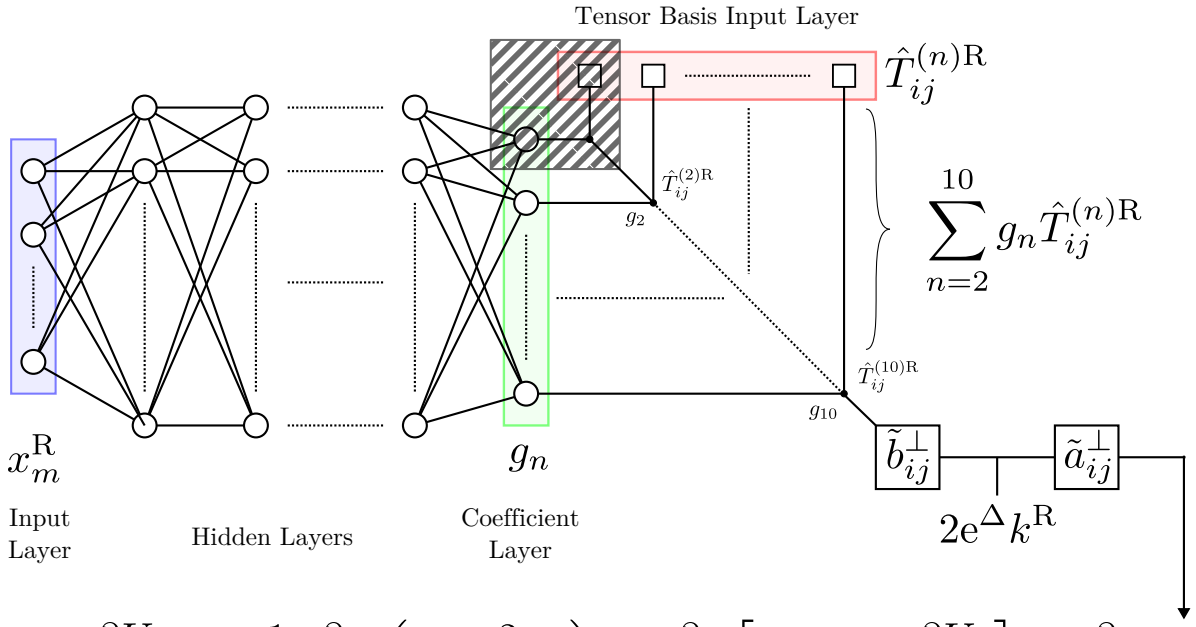


Figure 5.5: Model architecture and training configuration proposed in the present investigation. Note that during training, S_{ij}^θ is used in $\hat{T}_{ij}^{(1)}$, whereas all other quantities come from the original RANS simulation.



$$U_i \frac{\partial U_j}{\partial x_i} = -\frac{1}{\rho}\frac{\partial}{\partial x_j}\left(P + \frac{2}{3}\rho k\right) + \frac{\partial}{\partial x_i}\left[(\nu + \nu_t^{\mathrm{R}})\frac{\partial U_j}{\partial x_i}\right] - \frac{\partial}{\partial x_i}\left(\tilde{a}_{ij}^{\perp}\right)$$

Figure 5.6: Test-time injection configuration proposed in the present investigation. The hatching over g_1 and $\hat{T}_{ij}^{(1)}$ indicates that they are not used at injection time.

$$\Delta = \log\left(\frac{k^{\theta}}{k^{\mathrm{R}}}\right) \ , \tag{5.32}$$

such that at injection time, an updated estimate for k can be obtained $\tilde{k} = \mathrm{e}^{\Delta} k^{\mathrm{R}}$, without the need to re-couple the turbulence transport equations. The KCNN shares the same input features as the TBNN.

Together, the KCNN and TBNN predict the anisotropy tensor in the following manner:

$$\tilde{a}_{ij} = 2\left(\mathrm{e}^{\Delta} k^{\mathrm{R}}\right) \sum_{n=1}^{10} g_n \hat{T}_{ij}^{(n)} \ . \tag{5.33}$$

With the linear component of $\tilde{a}_{ij}$ being treated implicitly during injection, the entire closure framework is summarized as explicit injection of the following term into the momentum equation:

$$\tilde{a}_{ij}^{\perp} = 2\left(\mathrm{e}^{\Delta} k^{\mathrm{R}}\right) \sum_{n=2}^{10} g_n \hat{T}_{ij}^{(n)\mathrm{R}} \ . \tag{5.34}$$

5.2.5 Machine learning procedure

The KCNN and TBNN architectures proposed in Section 5.2.4 were implemented in PyTorch, along with the proposed realizability-informed loss function (Equation 5.26). These models were trained using an open-source dataset for data-driven turbulence modelling [79]. The objective of this study is to train and evaluate models trained on various flows, to determine whether the proposed realizability-informed loss function and architecture modifications are significantly beneficial. All code is available on Github [119].

Datasets

The training flows consist of flow over periodic hills [44], flow through a square duct [45], and flow over a flat plate with zero pressure gradient [25]. These flows are selected because they contain several challenging physical phenomena for RANS, including separation, reattachment, and Prandtl's secondary flows. The flat plate case is also included, to demonstrate how machine learning can improve the anisotropy estimates within the boundary layer. For each flow type, both a hold-out validation set, and a hold-out test

Table 5.1: Datasets used for training, validation, and testing. Varied parameters are defined in Section 5.3.1.

	Flat plate	Square duct	Periodic hills
Data points	1,396	147,456	73,755
Parameter varied	Re_θ	Re_H	α
Training set	1000, 2000, 2540, 3270, 3630, 3970	1100, 1150, 1250, 1350, 1400, 1500, 1600, 2205, 2400, 2600, 2900, 3500	0.5, 1.0, 1.5
Validation set	1410, 3030, 4060	1300, 1800, 3200	0.8
Test set	3630	2000	1.2

set are selected. The validation set is used during training to help guide when to stop training in order to prevent overfitting, but the validation set loss is not back-propagated through the network to update weights and biases. While we hold-out an entire case for the test set (the usual procedure in data-driven turbulence modelling), we also generate the validation sets by holding out entire cases at a time. We recommend this method for generating validation sets in data-driven turbulence modelling—it is analogous to grouped cross-validation, a practice used in machine learning where several data points come from a single observation. Here, we consider each separate flow case as a single observation, each containing many data points. It is therefore prudent to ensure that two data points from the same observation are not used in both the training and test set.

Table 5.1 outlines the three training/validation/test splits considered. The objective in splitting the dataset this way is to determine whether a realizability-informed model can generalize to a new case for a given flow. As discussed in McConkey et al. [115], machine learning-based anisotropy mappings do not generalize well to entirely new flows. However, they can be used to dramatically enhance the performance of a RANS simulation for a given flow type. In this same spirit, we aim to test how our modifications improve the generalizability of the learned TBNN anisotropy mapping to an unseen flow, albeit within the same class of flow.

Input features

The input features here are all derived from the baseline RANS k-ω SST simulation. The input features form the vector x_m^{R}. The superscript R has been dropped in this section to avoid crowded notation, but it applies to all quantities discussed in Section 5.2.5.

The input features to the TBNN must be Galilean invariant in order to generate an appropriately constrained anisotropy mapping. Most data-driven anisotropy mapping investigations use a mixture of heuristic scalars and scalars systematically generated from a minimal integrity basis for a set of gradient tensors. We emphasize that all scalars must be Galilean invariant—without this criteria, the RANS equations will lose Galilean invariance. Despite the importance of this constraint, several data-driven anisotropy mappings include scalars like the turbulence intensity, which breaks Galilean invariance.

We use a mixture of heuristic scalars and scalars systematically generated from a minimal integrity basis [33]. We use the following heuristic scalars:

$$q_1 = \min\left(\frac{\sqrt{k}y_w}{50\nu}, 2\right) \ , \tag{5.35}$$

$$q_2 = \frac{k}{\varepsilon}\sqrt{\sum_i \sum_j |S_{ij}|^2} \ , \tag{5.36}$$

$$q_3 = \frac{\sqrt{\sum_i \sum_j |\tau_{ij}|^2}}{k} \ , \tag{5.37}$$

$$q_4 = \frac{\sqrt{k}}{0.09\omega y_w} \ , \tag{5.38}$$

$$q_5 = \frac{500\nu}{y_w^2\omega} \ , \tag{5.39}$$

$$q_6 = \min\left(\max\left(q_4, q_5\right), \frac{2.0k}{\max\left(\frac{y_w^2}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}\right)}\right) \ , \tag{5.40}$$

with $\varepsilon = 0.09\omega$, and y_w is the distance to the nearest wall.

These input features correspond to: the wall-distance based Reynolds number (q_1), the ratio of turbulent time scale to mean strain timescale (q_2), the ratio of total Reynolds stress to k (q_3), and different blending scalars used within the k-ω SST model (q_4, q_5, q_6).

The full list of integrity basis tensors is given in Appendix E. We use the first invariant (the trace, A_{ii}) of the following tensors: $B_{ij}^{(1)}$, $B_{ij}^{(3)}$, $B_{ij}^{(4)}$, $B_{ij}^{(6)}$, $B_{ij}^{(7)}$, $B_{ij}^{(16)}$, and $B_{ij}^{(35)}$. We use the second invariant, $\frac{1}{2}\left((A_{ii})^2 - A_{ij}A_{ji}\right)$, of the following tensors: $B_{ij}^{(3)}$, $B_{ij}^{(6)}$, $B_{ij}^{(7)}$, $B_{ij}^{(8)}$. These input features were hand-picked from the full set of 94 invariants listed in Appendix E [33, 93]. As discussed in McConkey et al. [93], many of the invariants are zero for 2D flows. However, different conditions cause different invariants to be zero. For example, in the present study, there are a different set of zero invariants for flow through over periodic hills, and flow through a square duct. This difference occurs because different components of $\partial()/\partial x_j$ are to be zero. We have performed a systematic investigation using a symbolic math toolbox (sympy [120]) to determine which invariants are zero for the duct case, and general 2D flows. We make the results and code available in Appendix E and on Github [121], respectively. Input features used in this investigation were selected based on the results in Appendix E. Therefore, the input features are not uniformly zero on any of the considered flows.

To ensure all input features are of the same magnitude during training, they are scaled according to the following formula:

$$x_m^{\mathrm{R}} = \frac{X_m^{\mathrm{R}} - \mu_m}{\sigma_m} \ , \tag{5.41}$$

where x_m^{R} is the input feature vector for the neural network, X_m^{R} is the raw input feature vector from the RANS simulation, μ_m is a vector containing the mean of each input feature over the entire training dataset, and σ_m is a vector containing the standard deviation of each input feature over the entire training dataset.

When making predictions on the hold-out validation and test sets, the mean and standard deviation values from the training data are used to avoid data leakage.

Hyperparameters and training procedure

The hyperparameters for each neural network were hand-tuned based on validation set performance. The hidden layers for all TBNNs and KCNNs are fully connected, feed-forward layers with Swish activation functions [122]. The appropriate hyperparameters vary between flows, since each dataset contains a different number of data points, and the anisotropy mapping being learned is distinct. Table 5.2 shows the final hyperparameters used. All training runs used a mini-batch size of 32, except the periodic hills TBNN run, which used a mini-batch size of 128.

Table 5.2: Hyperparameters selected for each model.

Model	Dataset	Learning rate	Epochs	Hidden layers	Neurons per layer
TBNN	Flat plate	$5(10)^{-4}$	7959	4	20
KCNN	Flat plate	$2.5(10)^{-4}$	2478	5	30
TBNN	Square duct	$5(10)^{-4}$	100	7	30
KCNN	Square duct	$5(10)^{-4}$	1150	7	30
TBNN	Periodic hills	$1(10)^{-5}$	19140	7	30
KCNN	Periodic hills	$1(10)^{-6}$	13138	11	30

The AMSGrad Adam optimizer [123] was found to achieve better performance than the standard Adam optimizer [124] for training TBNNs. The learning rate and number of epochs for each optimizer is given in Table 5.2. Satisfactory performance was achieved with a constant learning rate; for training on more complex flows we recommend the use of learning rate scheduling to achieve better performance. The training/validation loss curves for each model are shown in Figure 5.7.

5.3 Results

5.3.1 Generalization tests

It was of interest to determine how well the trained models generalize to unseen flows. This section demonstrates generalization results for flow over a flat plate with zero pressure gradient, flow through a square duct, and flow over periodic hills.

For all cases, the original RANS solution was generated using OpenFOAM v2212, assuming an isothermal, incompressible, and Newtonian fluid. Simulation parameters such as solver, schemes, and solution methodology for the zero pressure gradient flat plate case were identical to those discussed in McConkey et al. [79].

It should be noted that the predictions shown for the TBNN/KCNN use S_{ij}^{θ} for predicting the linear part of the anisotropy tensor. The nature of the proposed TBNN training process is to utilize S_{ij}^{θ} during training, so that the remaining non-linear part can be extracted during injection.

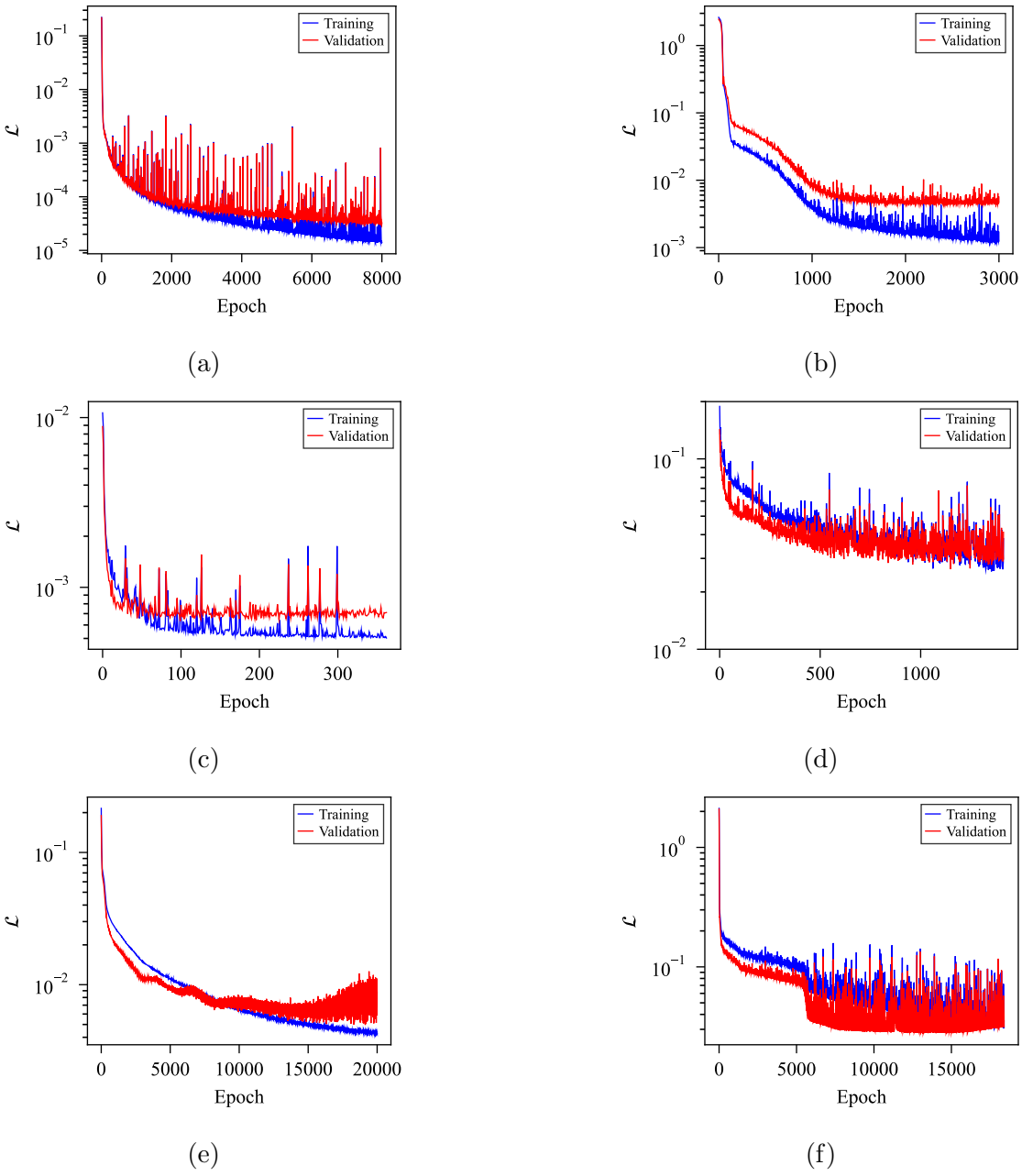


Figure 5.7: Loss function vs epoch for the (a) TBNN and (b) KCNN models on the flat plate dataset; the (c) TBNN and (d) KCNN models on the square duct dataset; and the (e) TBNN and (f) KCNN models on the periodic hills dataset.

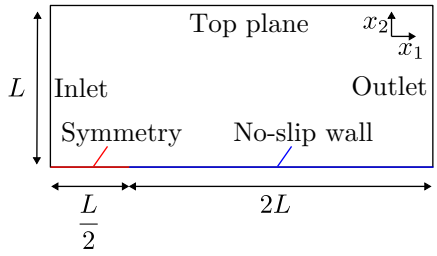


Figure 5.8: Computational domain for the zero pressure gradient flat plate boundary layer case.

Flat plate

This case features a developing turbulent boundary layer on a flat plate with zero pressure gradient, based on the NASA "2DZP" validation case [25]. Figure 5.8 shows the domain for the flat plate case. The NASA-provided meshes are sufficient to resolve the viscous sublayer region, with a total number of cells $N \approx 200,000$. However, this mesh was further refined to increase the number of solution data points available for training and testing. The goal of this case is to learn how the anisotropy tensor evolves in a turbulent boundary layer, therefore substantial mesh refinement was required to generate data points in this region. The total number of cells in the mesh is $N = 4,673,130$. The plate-length Reynolds number is $Re_L = 5(10)^6$ to match the NASA reference data. The reference data for this case consists of a series of wall-normal profiles, for various Re_θ, defined as

$$Re_\theta = \frac{U_\infty \theta}{\nu} \ , \tag{5.42}$$

where the momentum thickness θ is given as

$$\theta = \int_0^\infty \frac{U_1}{U_\infty}\left(1 - \frac{U_1}{U_\infty}\right) dx_2 \ , \tag{5.43}$$

and U_∞ is the free-stream velocity. The following boundary conditions and fluid properties are used for the domain in Figure 5.8. At the inlet boundary, U_j, k , and ω are uniform: $U_j = (69.4, 0, 0)$ m/s, $k = 1.08(10)^{-3}$ m^2/s^2, $\omega = 8675$ s^{-1}, and P is zero normal gradient. At the outlet, P is zero, and all other variables are zero normal gradient. At the symmetry plane, all variables are zero normal gradient, and normal velocity is zero. At the top plane, all flow variables are zero normal gradient. At the no-slip wall, $U_j = 0$, $k = 0$, $\omega = 6\nu/\beta_1 y^2$ (here, $\beta_1 = 0.075$), and P is zero normal gradient. A kinematic viscosity of $\nu = 1.388(10)^{-5}$ m/s^2 was used.

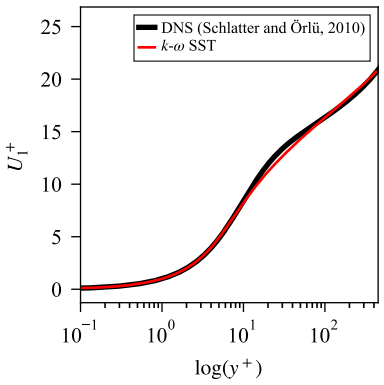


Figure 5.9: Streamwise velocity profile in the $Re_\theta = 3630$ boundary layer predicted by the k-ω SST model, compared to the reference data from Schlatter and Örlü [125]. Here, $U_1^+ = U_1/u_\tau$, where $u_\tau = \sqrt{\tau_w/\rho}$, is the friction velocity, and τ_w is the wall shear stress. A density of 1 kg/m^3 was used to be consistent with OpenFOAM's kinematic units. The wall-normal coordinate is $y^+ = x_2 u_\tau/\nu$.

DNS reference data for a developing turbulent boundary layer comes from Schlatter and Örlü [125]. This dataset contains a variety of turbulent boundary layer profiles, at various Re_θ shown in Table 5.1. As discussed in Section 5.2.5, the TBNN model was trained on various Re_θ, with $Re_\theta = 3,630$ serving as a hold-out test case. The results shown in this section are for this hold-out test case.

The baseline k-ω SST model performs well for the test case flow in terms of predicting the mean velocity profiles. Figure 5.9 shows a sample mean velocity profile predicted by the baseline k-ω SST model, and Figure 5.10 shows the predicted evolution of Re_θ along the plate. The excellent performance of the k-ω SST model demonstrated by Figures 5.9 and 5.10 is expected. This case features a fully attached boundary layer with zero pressure gradient, which is one of the fundamental calibration scenarios for RANS models. The zero pressure gradient turbulent boundary layer is considered a "solved problem" for modern RANS models [114].

While the mean velocity profile is predicted well, Figure 5.11 shows that the evolution of the near-wall anisotropy tensor is not predicted well. For this reason, the model architecture discussed in Section 5.2.4 was used to correct the anisotropy tensor in the near-wall region. This test also aims to determine whether the input feature set is sufficiently expressive to enable predicting the evolution of the anisotropy tensor within a boundary layer. Figure 5.11 shows the wall-normal profiles of various anisotropy tensor

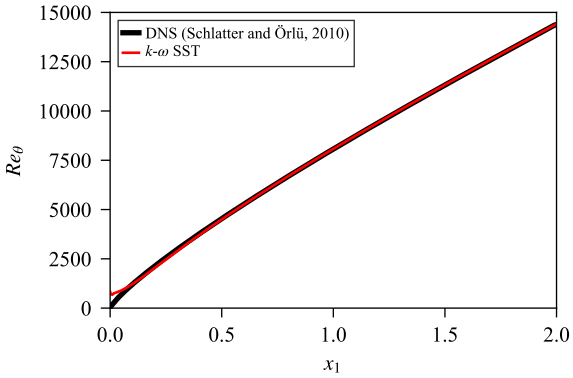
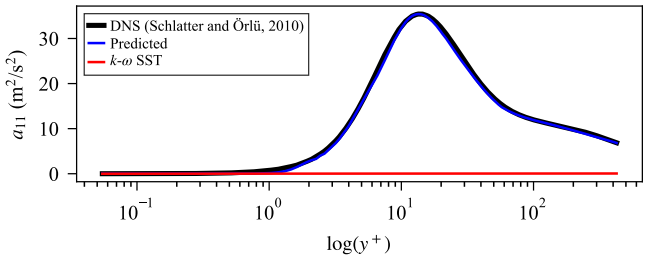


Figure 5.10: Momentum thickness Reynolds number growth along the flat plate as predicted by the k-ω SST model, and the reference DNS data from Schlatter and Örlü [125].

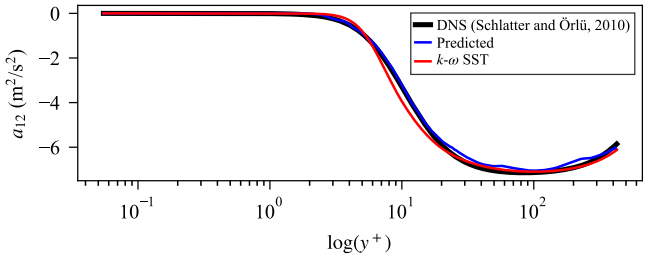
components predicted by the k-ω SST model, the ML-augmented k-ω SST model, and the reference DNS simulation for the hold-out test case. As discussed in Section 5.2.5, these models were trained on flat plate data at various values of Re_θ. The results in Figure 5.11 are designed to test these models on input features from an unseen boundary layer profile, to determine whether the learned anisotropy mapping was generalizable.

Figure 5.11a shows the predicted evolution of a_{11} in the turbulent boundary layer. The baseline k-ω SST model predicts $a_{11} = 0$, since $\partial U_1/\partial x_1 \approx 0$ in the boundary layer. However, the DNS data clearly shows that a_{11} is non-zero in the boundary layer. The TBNN/KCNN model combination is able to correct the the a_{11} term to a high degree of accuracy in the boundary layer on this test case, indicating that the anisotropy mapping for the a_{11} component generalizes well. Similar evolutions of a_{22} (Figure 5.11c) and a_{33} (Figure 5.11d) are observed in the DNS data. Again, the k-ω SST model predicts $a_{11} = a_{22} = a_{33} = 0$, which is not physically correct. The TBNN/KCNN models are able to correct the baseline prediction to a high degree of accuracy on this unseen boundary layer profile.

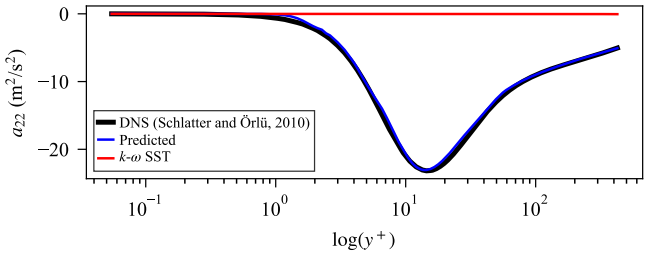
Figure 5.11b shows the predicted evolution of a_{12}. The baseline RANS model predicts the evolution of a_{12} well, and this is likely the reason that the mean velocity profile of U_1 is predicted well (see Figure 5.9). While the TBNN/KCNN is not needed to correct this off-diagonal component, it is able to correct minor inaccuracies in the k-ω SST model predictions in the buffer region ($5 \leq y^+ \leq 30$). Nevertheless, the baseline k-ω SST model achieves a satisfactory accuracy level for this flow. As discussed, this is expected, given that low Reynolds number RANS models are able to predict a zero pressure gradient boundary



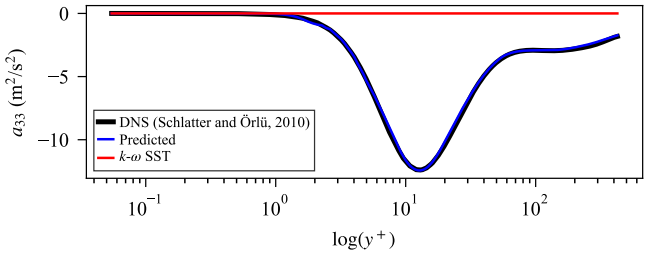
(a)



(b)



(c)



(d)

Figure 5.11: Evolution of (a) a_{11}, (b) a_{12}, (c) a_{22}, and (d) a_{33} in the $Re_\theta = 3630$ boundary layer, as predicted by the DNS data from Schlatter and Örlü [125], the TBNN/KCNN model, and the baseline k-ω SST model.

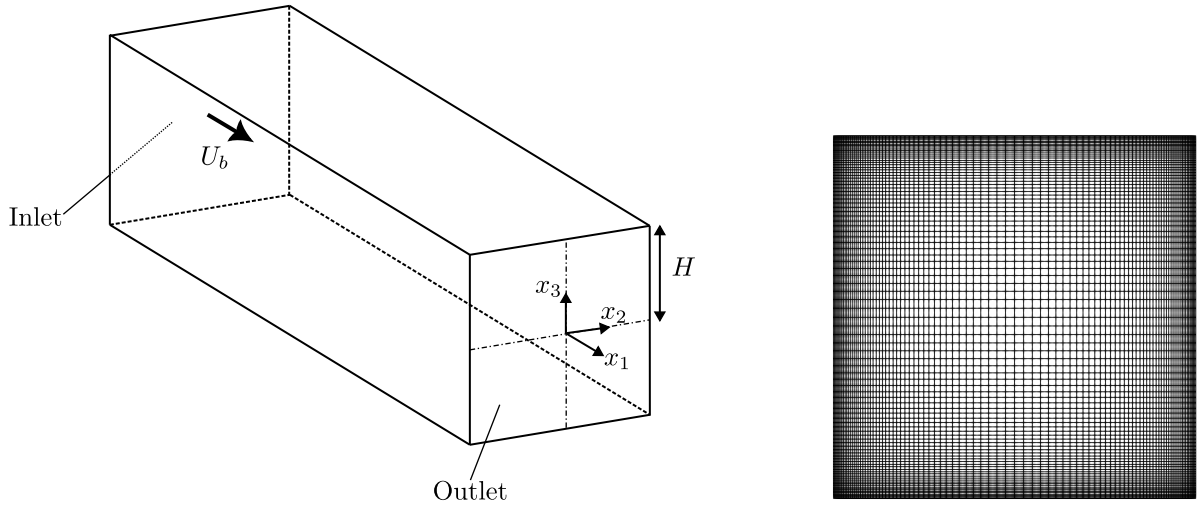


Figure 5.12: Computational (a) domain and (b) RANS mesh for the square duct flow.

layer with a high degree of accuracy. Figure 5.11 demonstrates that this performance is the result of an accurate prediction of a_{12} by the k-ω SST model.

Square duct

Turbulent flow through a square duct is a challenging case for RANS models, since linear eddy viscosity models cannot predict the secondary flows that occur in the cross-sectional plane. The goal of the square duct test case is to determine whether the proposed closure framework could enable the k-ω SST model to predict these Prandtl secondary flows [126]. The square duct DNS dataset generated by Pinelli et al. [45] was used as reference data, and the RANS data from McConkey et al.'s [79] curated dataset was used.

Figure 5.12 shows the computational setup and mesh for the square duct case. The mesh is designed to achieve $y^+ \leq 1$ for all square duct cases. As discussed in Section 5.2.5, the duct half-height H Reynolds number varies between cases, calculated by:

$$Re_H = \frac{U_b H}{\nu} \ , \tag{5.44}$$

where U_b is the mean (bulk) cross-sectional velocity. A kinematic viscosity of $\nu = 5(10)^{-6}$ m^2/s was used for all square duct cases. With the geometry fixed as shown in Figure 5.12,

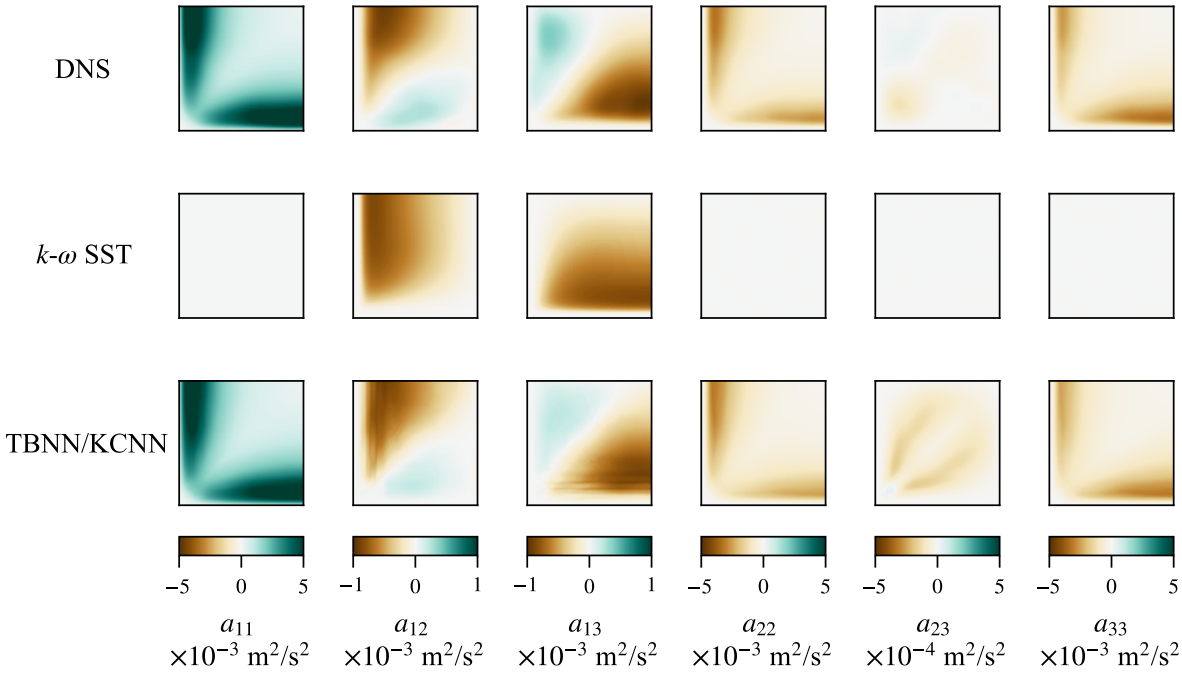


Figure 5.13: Components of a_{ij} predicted by the DNS data from Pinelli et al. [45], the k-ω SST model, and the TBNN/KCNN model from the present investigation. Shown here are contours of a_{ij} components in the lower left $x_2 \leq 0,\ x_3 \leq 0$ quadrant.

the bulk velocity was adjusted to vary the Reynolds number. More details on the computational setup for the square duct case are provided by McConkey et al. [79]. The boundary conditions are periodic at the inlet/outlet, and no-slip walls were applied along the sides of the duct. As discussed in Section 5.2.5, the modified TBNN was trained on several values of Re_H, with $Re_H = 2,000$ serving as a hold-out test case.

Figure 5.13 shows the components of the anisotropy tensor a_{ij} predicted by RANS, DNS, and the TBNN/KCNN models for the square duct test case. The k-ω SST model is a linear eddy viscosity model, and therefore predicts zero a_{ij} where S_{ij} is zero. Figure 5.13 shows that a_{11}, a_{22}, a_{23}, and a_{33} are all non-zero in the duct, and that the k-ω SST model is unable to capture this behavior. The TBNN/KCNN models predict an accurate evolution of almost all anisotropy tensor components across the duct cross section (viz. a_{11}, a_{12}, a_{13}, a_{22}, and a_{33} are all predicted well on this test case). The anisotropy tensor component a_{23} is not predicted well, likely because it is at least an order of magnitude smaller than

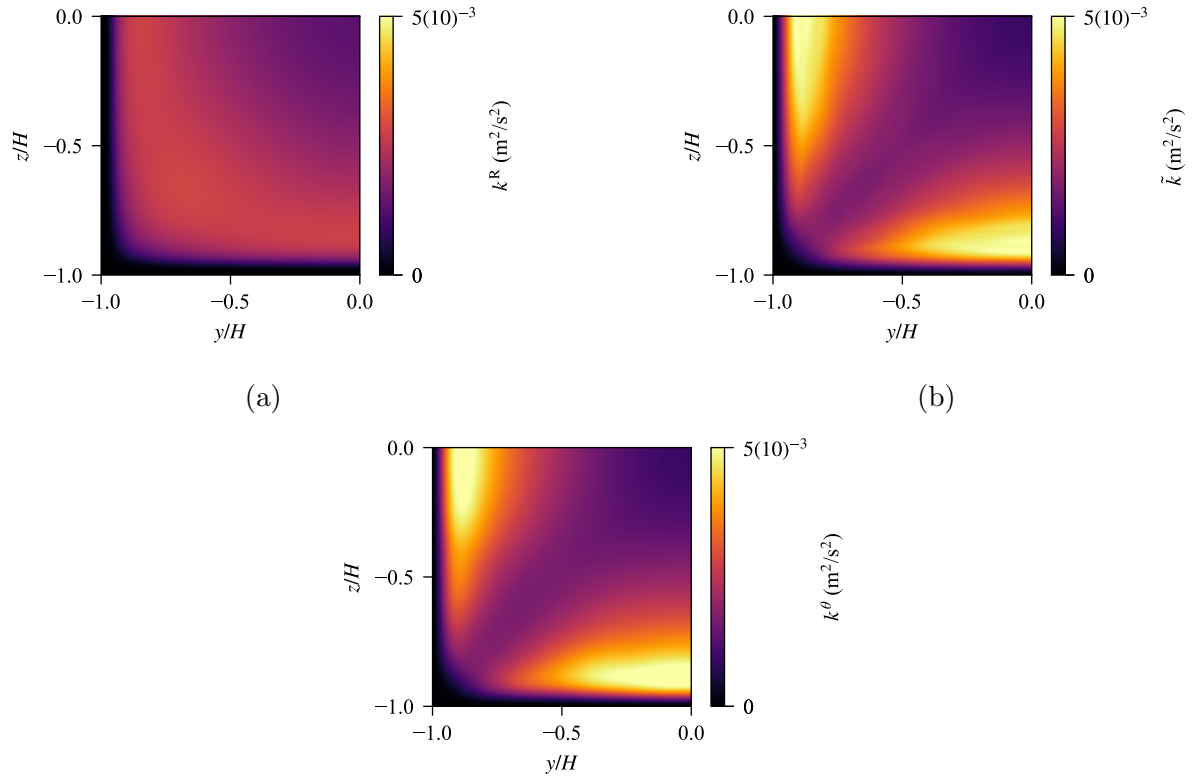
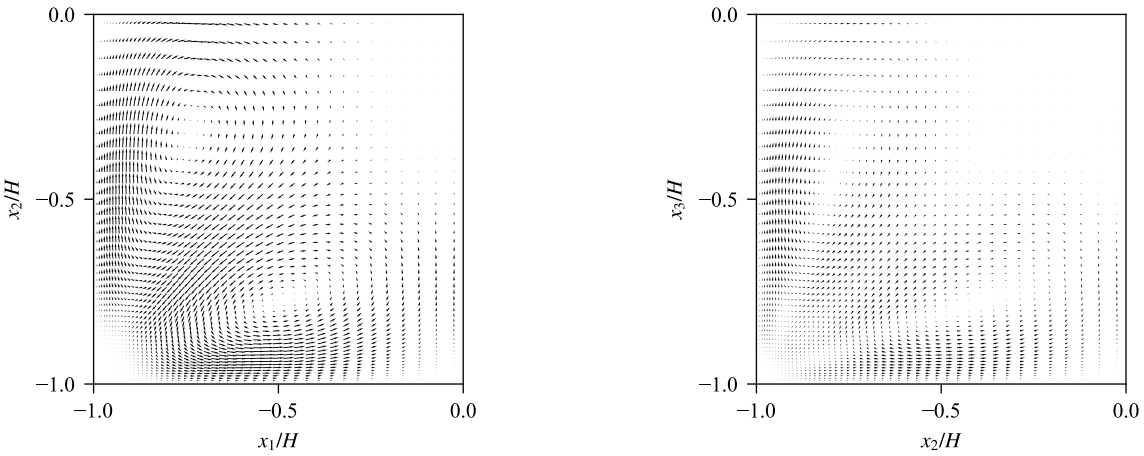


Figure 5.14: Contours of turbulent kinetic energy k predicted by (a) the k-ω SST model, (b) the KCNN in the present investigation, and (c) the DNS data from Pinelli et al. [45]. Shown here is the lower left $x_2 \leq 0,\ x_3 \leq 0$ quadrant.

the other components, and therefore errors in a_{23} are not penalized as heavily in the loss function.

Figure 5.14 shows the turbulent kinetic energy k after being corrected by the KCNN model for the square duct test case. Accurate prediction of k is critical to an accurate estimate of a_{ij}, since $a_{ij} = 2kb_{ij}$. The k-ω SST model generally under-predicts k. After correction via the KCNN, the k field is predicted well compared to the DNS data. The primary feature in the k field that is absent from the k-ω SST prediction is the high-k region along the side walls of the duct. The KCNN introduces a correction to the baseline RANS field, and is able to predict this high-k region.

Ultimately, it is the goal of the proposed framework to improve the estimated mean



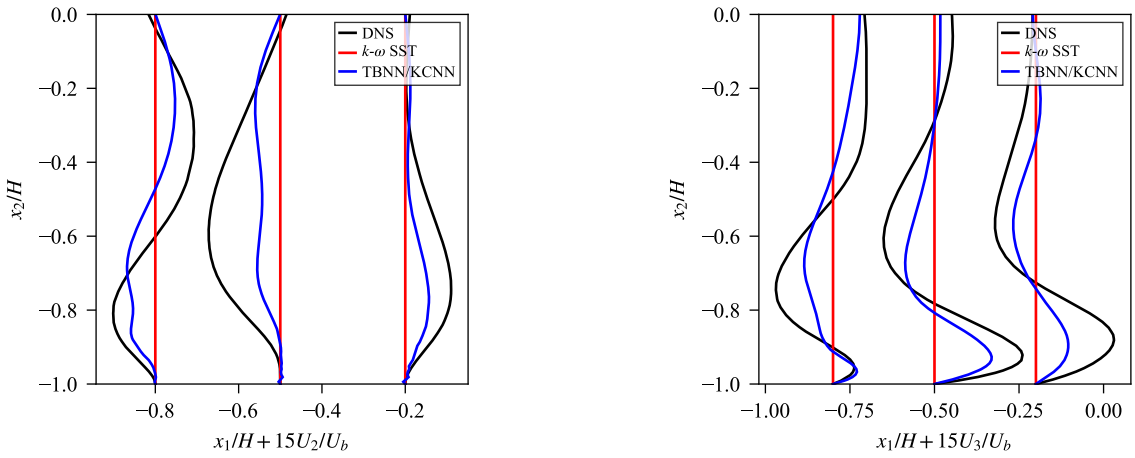
(a) (b)

Figure 5.15: Velocity vectors predicted by (a) the DNS data from Pinelli et al. [45] and (b) injecting the TBNN/KCNN predictions into the RANS momentum equations. Shown here is the lower left $x_2 \leq 0,\ x_3 \leq 0$ quadrant.

fields in the RANS simulation. To determine whether the corrected closure term would produce corrected mean velocity fields, the predicted $\tilde{a}_{ij}^{\perp}$ was injected into the RANS momentum equation as shown in Equation 5.15. The momentum and continuity equations converged around the fixed $\tilde{a}_{ij}^{\perp}$ until numerical convergence was achieved. In OpenFOAM v2212, a modified version of the PIMPLE solver was implemented for the purpose of this injection. The PIMPLE solver was used to incorporate an unsteady term into the system of equations during iteration, to promote stability. Though this unsteady term affects the solution during convergence, the simulation ultimately achieved a steady state condition, thereby reducing this unsteady term to zero.

Figure 5.15 shows that the TBNN/KCNN model is able to produce secondary flows after injecting $\tilde{a}_{ij}^{\perp}$ into the momentum equation. This *a posteriori* prediction of the mean field is ultimately the main prediction of interest for a ML-augmented RANS closure framework. Whereas the original k-ω SST model does not predict formation of any secondary flows in the duct, Figure 5.15 shows that the ML-augmented k-ω SST model predicts corner vortices.

To further examine the ability of the ML-augmented k-ω SST model to predict secondary flows in the duct test case, profiles of U_2 and U_3 are plotted in Figure 5.16. While the ML-augmented model is able to produce this non-linear feature, the corner vortex



(a) (b)

Figure 5.16: Profiles of the in-plane velocity components (a) U_2 and (b) U_3, as predicted by the k-ω SST model, the injected TBNN/KCNN predictions, and the DNS data from Pinelli et al. [45]. Shown here is the lower left $x_2 \leq 0,\ x_3 \leq 0$ quadrant.

strengths are reduced compared to the reference DNS data. Both the U_2 and U_3 components are under-predicted. Nevertheless, the k-ω SST model (which predicts $U_2 = U_3 = 0$) has clearly been improved via a ML-augmented correction to the closure term in the momentum equation. From Figure 5.13, it would appear that the good prediction of the normal stress anisotropy (the primary mechanism responsible for the streamwise vorticity determined by U_2 and U_3) should provide good predictions of the streamwise vorticity. However, once the secondary flow is set in motion by this normal stress anisotropy, it is the secondary (rather than primary) shear stress component a_{23} (generated by the presence of the secondary flow itself) that is required to maintain this flow and from Figure 5.13, this secondary component of the shear stress is not well predicted. Therefore, the underprediction of U_2 and U_3 is likely due to inaccurate prediction of a_{23}.

Periodic hills

Flow over periodic hills is used as a popular benchmark case for turbulence modelling given the challenging physics of boundary layer separation in an adverse pressure gradient, reattachment along the bottom wall, and acceleration of the flow before reentering the domain. For the purpose of data-driven turbulence modelling, a variety of periodic hills

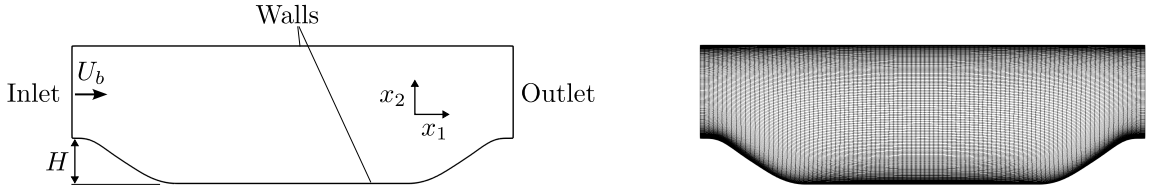


Figure 5.17: Computational domain and mesh for the periodic hills case, with $\alpha = 1.2$.

data has been made available. In this study, we use the $Re_H = 5,600$ configuration, simulated using DNS by Xiao et al. [44]. Xiao et al.'s data was included in McConkey et al.'s [79] curated dataset, which is the primary data source for this study.

The geometry and mesh for the periodic hills case are shown in Figure 5.17. For all periodic hills cases, the hill height-based Reynolds number is 5,600, calculated by

$$Re_H = \frac{U_b H}{\nu} \ , \tag{5.45}$$

where U_b is the bulk (mean) velocity at the domain inlet. The hill geometry is varied between cases, based on the hill steepness α. Further details on the computational setup for the baseline RANS periodic hills simulations are provided in McConkey et al. [79]. The boundary conditions are periodic at the inlet/outlet, and no-slip walls at the top and bottom of the domain were imposed. As discussed in Section 5.2.5, the TBNN and KCNN models were trained on several hill steepness values, with $\alpha = 1.2$ being used as a hold-out test set.

Figure 5.18 shows the components of the anisotropy tensor a_{ij} predicted by RANS (k-ω SST), DNS, and the ML-augmented RANS simulation. The improvement in all components of a_{ij} is clear. The baseline k-ω SST under-predicts all components, with severe under-prediction of a_{11}, a_{22}, and a_{33}. The a_{12} prediction by the k-ω SST showcases similar trends to the DNS data, but the overall magnitude is lower. However, after correction, key features of all a_{ij} fields are captured when the TBNN/KCNN augment the k-ω SST model. In particular, the higher magnitudes of the diagonal a_{ij} (normal stress) components are captured by the augmented model.

As was done for the square duct test case (Section 5.3.1), a modified PIMPLE solver was used to inject the predicted $\tilde{a}_{ij}$ into the RANS momentum equation for the periodic hills test case. The numerical setup for the periodic hills injection was identical to the square duct case. Figure 5.19 compares the mean velocity fields before and after the corrected closure term is used within the RANS simulation. Figure 5.20 compares the errors in the

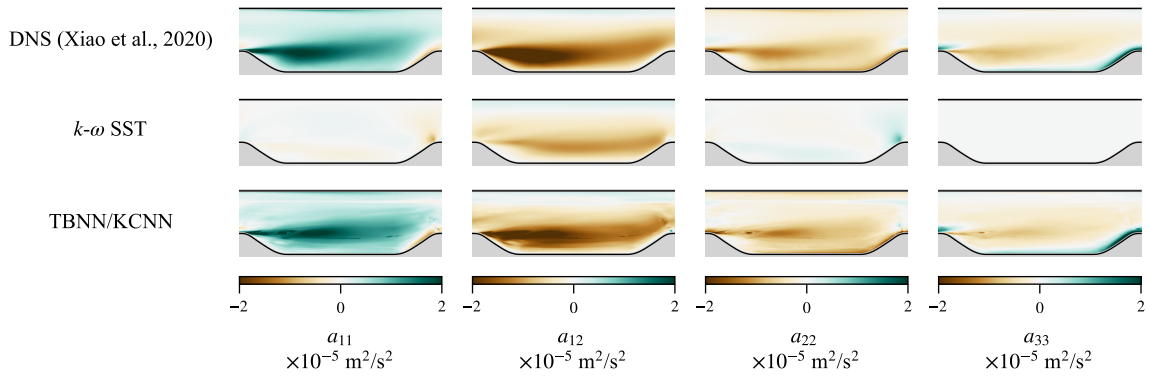


Figure 5.18: Contours of non-zero a_{ij} components predicted by the DNS data from Xiao et al. [44], the k-ω SST model, and the TBNN/KCNN model predictions from the present investigation.

velocity components U_1 and U_2 estimated by the k-ω SST model, and the *a posteriori* (post-injection) TBNN/KCNN-augmented SST model.

As seen in Figure 5.19, the primary feature of this flow is a recirculation zone which appears immediately after the left hill. The recirculation zone is most clearly visualized by examining the U_1 fields. The k-ω SST model over-predicts the size of this recirculation zone. After correction via injecting the TBNN/KCNN predictions, the recirculation zone size closely matches the DNS data. In the U_2 field, a region with $U_2 < 0$ is seen immediately above this recirculation region. The baseline k-ω SST model under-predicts the downward velocity here, leading to delayed reattachment, and a longer recirculation zone. After correction, the magnitude of U_2 in this shear layer more closely matches the DNS data. On the right hill, the upward acceleration of the flow under the favourable pressure gradient is also under-predicted by the k-ω SST. Here, the injected $\tilde{a}_{ij}$ is able to better capture the strength of this upward acceleration.

Figure 5.20 more closely examines the improvments offered by augmenting the k-ω SST model via the TBNN/KCNN. It can be seen that the overall magnitudes of the errors in U_1 and U_2 are significantly reduced after injecting the TBNN/KCNN model predictions. In particular, error in U_1 is reduced in the reattachment region along the bottom wall, and the bulk flow above this region. Error in U_2 is reduced in the previously identified shear layer above the recirculation region, and the accelerating region before the outlet.

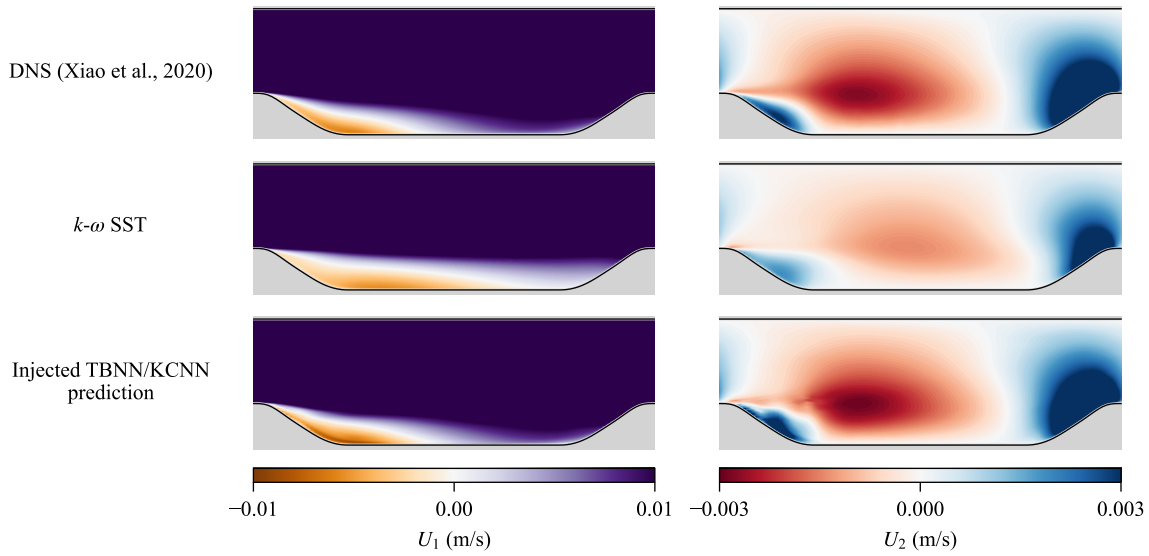


Figure 5.19: Contours of U_1 and U_1 predicted by the DNS data from Xiao et al. [44], the k-ω SST model, and the injected TBNN/KCNN model predictions from the present investigation.

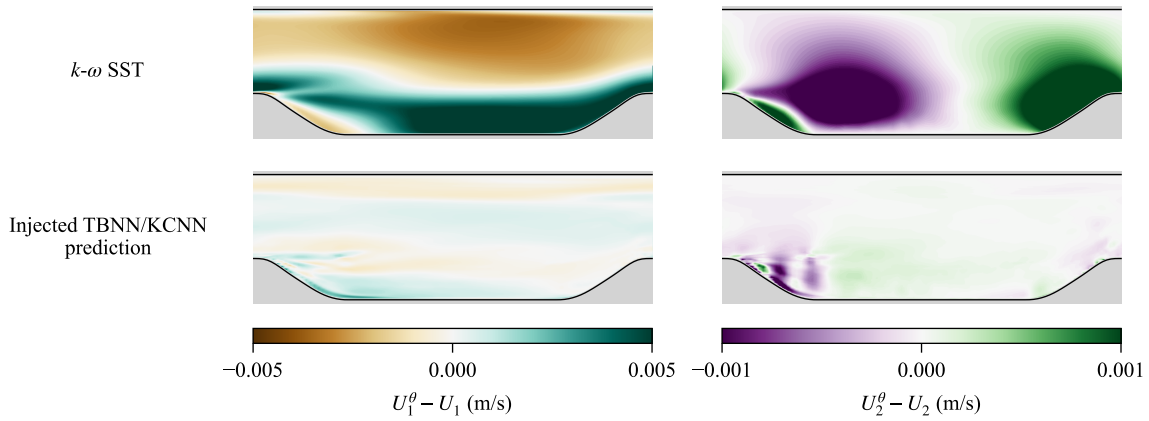


Figure 5.20: Contours of error in U_1 and U_2 predicted by the k-ω SST model and the injected TBNN/KCNN model predictions, as compared to the DNS data from Xiao et al. [44].

Table 5.3: Comparison of mean squared error and number of non-realizable predictions when training with and without a realizability-informed loss function.

Dataset	α	MSE (a_{ij})	% non-realizable
Flat plate	0	$3.5(10)^{-3}$	0.6%
	10^2	$7.4(10)^{-3}$	0%
Square duct	0	$2.2(10)^{-9}$	9.2%
	10^2	$2.6(10)^{-9}$	0.4%
Periodic hills	0	$3.9(10)^{-13}$	1.7%
	10^2	$6.3(10)^{-13}$	0.3%

5.3.2 Impact of realizability-informed training

To determine the impact that including a realizability-informed penalty has on the training process, the closure term predictions for the three test cases were examined in greater detail. Two loss functions were used: one with $\alpha = 0$ (representing no realizability penalty), and one with $\alpha = 10^2$ (representing an exaggerated realizability penalty term in the loss function). The objective of this test was to determine whether including the realizability penalty during training promotes better generalization of the closure mapping to unseen flows.

All TBNN hyperparameters were fixed to those given in Table 5.2. Since the realizability-informed training procedure only applies to the TBNN, a perfect prediction of k via the KCNN was assumed for calculating error in $\tilde{a}_{ij}$.

Table 5.3 compares the mean-squared error in $\tilde{a}_{ij}$ on the hold-out test set, with and without realizability penalties being used in training. It should be noted that similar to the *a priori* tests in Section 5.3, the linear component used when visualizing b_{ij} comes from S^{θ}_{ij}, as is the configuration when training the TBNN. This linear component is the one used when training the TBNN, and therefore its use here provides the most fair assessment of how the proposed loss function promotes more physically realizable results.

As seen in Table 5.3, the realizability-informed loss function significantly reduces realizability violations on unseen flows. Even on hold-out test cases, the model is able to predict b_{ij} without any realizability violations. In some cases, a small tradeoff in a_{ij} occurs—this tradeoff is expected, as for some difficult points the realizability-informed loss function involves a tradeoff between error and realizability. However, in all cases, realizability-informed training also reduces the error in b_{ij}. This error reduction in b_{ij} is expected, since all b_{ij} reference data are realizable. In the case of predicting b_{ij} accurately, the gradients of

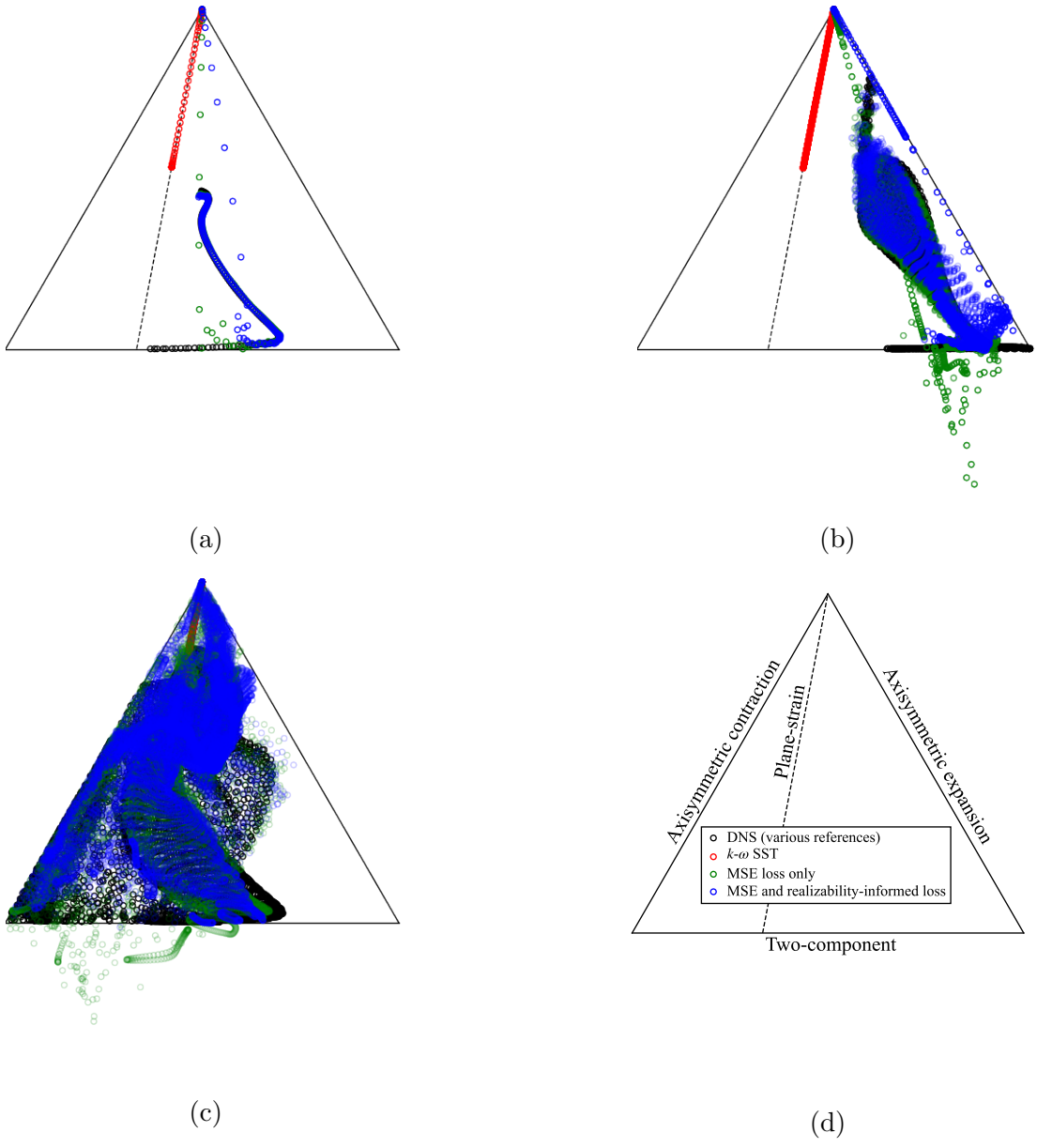
the realizability penalty $\mathcal{R}(b_{ij})$ further push the predictions towards an accurate prediction of b_{ij}, compared to a purely mean-squared error gradient.

To further visualize the realizability of the TBNN predictions, the barycentric map of Banerjee et al. [58] was used. In the barycentric map, the eigenvalues of a given b_{ij} are mapped into a triangle, the bounds of which represent the limiting realizable behaviors of turbulent fluctuations. This triangle is useful to spatially visualize realizability violations, and types of turbulent flow physics predicted by the baseline turbulence model, the ML-augmented turbulence model, and DNS. Further details on the construction of this mapping are given in Banerjee et al. [58], and Emory and Iaccarino [127].

Figure 5.21 compares the realizability of the predictions made by a model trained on only a mean-squared error loss function to a model trained on the realizability-informed loss function for the three flows in the present study. All predictions are for the hold-out test set for each flow, representing a generalization test. The goal of the realizability-informed loss function is to encode a preference for realizable predictions into the model when generalizing outside of the training dataset. Figure 5.21 shows a clear improvement in the realizability of the TBNN predictions. This visualization supports the results in Table 5.3 in that a realizability-informed model has significantly lower realizability violations when predicting the anisotropy tensor on a new flow. Nearly all of the predictions from the realizability-informed model fall within the realizable boundaries, while the model trained only on mean-squared error predicts several realizability-violating results when generalizing to new cases of complex flows such as the duct and periodic hills cases. Also, the violations of physical realizability for the realizability-informed model (when they do occur) are not as severe (as measured from the magnitude of deviation outside the barycentric map) as those obtained from only a mean-squared error loss function.

Figure 5.21 also shows that for all flows in the present study, the original k-ω SST model predicts plane-strain turbulence. All flows in the present study have a strain rate tensor which results in at least one zero eigenvalue of b_{ij}, for the linear eddy viscosity approximation ($b_{ij} = -\nu_t/kS_{ij}$), and therefore plane-strain turbulence is predicted for all flows by the k-ω SST model.

As discussed in Section 5.2.3, realizability-informed learning function does not guarantee a fully realizable prediction. Rather, realizability-informed learning encodes a preference for realizable predictions into the model predictions. This avoids the need for ad-hoc post-processing of the predicted anisotropy tensor, while also encoding a physics-based (learning) bias into the TBNN. Strict realizability can be further enforced by post-processing any non-realizable predictions by the TBNN, as the anisotropy tensor can still be accessed and assessed for realizability in the proposed framework (albeit, with an evolving



(a) (b) (c) (d)

Figure 5.21: Projection of the predicted b_{ij} from the k-ω SST model, the TBNN/KCNN without realizability-informed training (MSE loss only), the TBNN/KCNN with realizability-informed training, and the DNS reference data onto the barycentric triangle. (a) shows the flat plate data, (b) shows the square duct data, and (c) shows the periodic hills data. Points outside the triangle are not realizable.

linear component).

5.4 Conclusion

The objectives of this study were to propose a physics-informed loss function for training TBNN anisotropy mappings, a new TBNN architecture, and a new injection framework that accommodates implicit treatment of the linear anisotropy component within a TBNN-type architecture. This framework addresses an issue related to the stability of injected TBNN predictions, an issue that has been reported by Kaandorp and Dwight [35], and consistent with our own experience. This framework also addresses the issue of realizability within the context of predicting the anisotropy tensor from RANS input features.

The results here indicate that the proposed model architecture generalizes well to new flow configurations, and the predicted anisotropy tensor can be injected in a highly stable manner. During the injection procedure in the present investigation, the CFD solver remained stable, even when testing erratic model predictions. While this finding corroborates findings from others that frameworks which leverage implicit treatment of the linear anisotropy component via an eddy viscosity are numerically stable [77, 76, 66], a novelty in the present investigation is the avoidance of using an optimal eddy viscosity. With the proposed decomposition of a_{ij} (Section 5.2.1), a simpler (and more straightforward) baseline k-ω SST eddy viscosity is used. The core modification that allows a well-conditioned solution in this case is the use of S_{ij}^{θ} within $\hat{T}_{ij}^{(1)}$ in the TBNN training process. As a result, all corrections induced by the TBNN are contained in an explicit term in the momentum equation. Future work includes investigating how new-flow generalization can be improved by a blending factor that multiplies this explicit term, thereby allowing the correction to be turned off. For example, a statistics-based scalar or a separate machine learning model could be used to predict a blending factor. This blending factor could reduce or eliminates corrections when a test data point has significant departure from the training dataset, and erroneous predictions are likely.

While the realizability-informed loss function does not strictly guarantee physical realizability of the predictions, the results in Section 5.3.2 indicate that the model retains a realizability bias when generalizing. The realizability-informed loss function is not only applicable to the framework and architecture proposed in this study—it could be used anytime an anisotropy mapping is generated via machine learning. The use of the realizability-informed loss function in the present investigation promoted better realizability of predictions by a TBNN, but we fully expect the bias induced by this loss function to also be beneficial for tensor basis random forests (e.g., Kaandorp and Dwight [35]), or non-tensor

basis frameworks (e.g., Wu et al. [33]). Since high-quality DNS anisotropy tensor data is realizable, the realizability penalty can be viewed as an additional boost to the loss function gradient towards the true value, when a non-realizable prediction is made. Future work will investigate how the realizability-informed loss function performs with training data generated by LES, which is not guaranteed to be realizable.

The KCNN used in the proposed framework to correct k could also be replaced by coupling the k equation to the momentum equation, as exists in the original turbulence model, and the TBNN approach originally proposed by Ling et al. [23]. At training time, k^{θ} is used to dimensionalize b_{ij} in the present study, since k is corrected via the KCNN. This direct correction produced satisfactory results in the present work, but it is possible that generalization could be further enhanced by the re-coupling the k equation with an updated closure term. This enhanced generalization would result from the fact that a physics-based coupled equation system is used to correct k, rather than a simple multiplicative corrector (the KCNN in the present investigation). However, this coupling of an additional partial differential equation introduces the possibility for instability, an issue which is common in machine learning anisotropy modelling. Future work will investigate the merits of this route.

Ultimately, this investigation demonstrates that with sufficient modifications, TBNN-type anisotropy mappings can be injected in a stable and well-conditioned manner. Further, with appropriate physics-based loss function penalties, the mapping can be sensitized to more physically informative targets than mean-squared error. While industrial use of machine learning-based anisotropy mappings is currently not widespread, the continual development of techniques which increase the practicality of training and injecting model predictions will help accelerate more widespread use. Machine learning-augmented turbulence closure modelling is an aid that the turbulence modelling community can use to help bridge the current computational gap between RANS, and widespread use of LES [5].

Chapter 6

Turbo-RANS: straightforward and efficient Bayesian optimization of turbulence model coefficients

Associated reference:

R. McConkey, N.Kalia, E. Yee, and F. S. Lien, "Turbo-RANS: Straightforward and Efficient Bayesian Optimization of Turbulence Model Coefficients", *International Journal of Numerical Methods for Heat and Fluid Flow* (2024), https://doi.org/10.1108/HFF-12-2023-0726.

Abstract

Industrial simulations of turbulent flows often rely on Reynolds-averaged Navier-Stokes (RANS) turbulence models, which contain numerous closure coefficients that need to be calibrated. In this work, we address this issue by proposing a semi-automated calibration of these coefficients using a new framework (referred to as turbo-RANS) based on Bayesian optimization. We introduce the generalized error and default coefficient preference (GEDCP) objective function, which can be used with integral, sparse, or dense reference data for the purpose of calibrating RANS turbulence closure model coefficients. Then, we describe a Bayesian optimization-based algorithm for conducting the calibration of these model coefficients. An in-depth hyperparameter tuning study is conducted to recommend efficient settings for the turbo-RANS optimization procedure. We demonstrate that the performance of the k-ω shear stress transport (SST) and Generalized k-ω (GEKO)

turbulence models can be efficiently improved via turbo-RANS, for three example cases: predicting the lift coefficient of an airfoil; predicting the velocity and turbulent kinetic energy fields for a separated flow; and, predicting the wall pressure coefficient distribution for flow through a converging-diverging channel. This work is the first to propose and provide an open-source black-box calibration procedure for turbulence model coefficients based on Bayesian optimization. We propose a data-flexible objective function for the calibration target. Our open-source implementation of the turbo-RANS framework includes OpenFOAM, Ansys Fluent, STAR-CCM+, and solver-agnostic templates for user application.

6.1 Introduction

Turbulent flows are found in many engineering applications such as automotive, aerospace, chemical, nuclear, and wind energy. Engineering activities such as design, optimization, safety assessment, maintenance, and failure investigation in these industries frequently require predicting the behaviour of turbulent flows. Predicting turbulent flow is a necessary, but complex problem, that is often addressed using experimental or computational approaches. While experimental approaches often produce the most physically accurate results, they are expensive and limited in the quantities of interest (information) they can provide. For these reasons, predictions obtained using computational fluid dynamics (CFD) are attractive due to their low cost, capability to rapidly explore a (frequently large) design space, and ability to access detailed information about practically all aspects of the flow.

Many industrial CFD simulations rely on solving the Reynolds-averaged Navier-Stokes (RANS) equations, which are more computationally affordable than conducting large-eddy simulation (LES) [5]. RANS models have been considered an industrial workhorse for the past several decades. However, there are several well-known deficiencies with the application of these methods. A major source of error in RANS is the turbulence model, which often relies on a single-point, linear-eddy-viscosity closure model (e.g., turbulence closure models such as k-ε [51], k-ω [117], Spalart-Allmaras [106]). The past several decades of experience with RANS have resulted in the development of sophisticated turbulence closure models such as the k-ω shear stress transport (SST) model [118], the k-ε-ϕ-f model [52], and the recently-developed generalized k-ω (GEKO) model [128]. However, structural limitations that arise as a result of the assumptions and approximations underpinning single-point RANS closure models affect their predictive performance for in various engineering and industrial applications [31, 3].

Recently, there has been widespread interest in using data-driven approaches to improve and calibrate RANS models [12, 17]. Because of the ability to automate traditional physics-based and heuristic turbulence modeling, more advanced models can be potentially developed. Machine learning has been used to develop complex non-linear eddy viscosity models, which can provide good approximations for highly-resolved LES and direct numerical simulation (DNS) closure terms for use with the RANS modelling framework for incompressible flows [23, 93, 35, 33], and compressible flows [129, 130]. Additionally, machine learning can be used to generate LES subgrid closure models [92] and accelerate DNS through the application of super-resolution models [20]. While some approaches aim to use highly complex machine learning models to replace or augment the turbulence closure relation, another promising approach is using reference data to calibrate closure coefficients in an existing turbulence model. Most RANS turbulence closure models contain a number of tuneable coefficients, which can be calibrated to improve the RANS predictive performance for certain classes of flows. The problem of calibrating the turbulence model coefficients so that a prediction is in better conformance with the reference data can be formulated as an optimization problem. To this purpose, a number of previous studies have proposed the use of genetic algorithms [131, 132], Nelder-Mead simplex algorithm [133], optimal Latin hypercube method [134], and the global search method [135] for addressing this problem. Bayesian optimization was demonstrated to be an effective method for calibrating model coefficients in multiscale dynamical systems by Nadiga et al. [136]. As discussed by Diessner et al. [137], the Bayesian optimization methodology is particularly promising for addressing many fluid mechanics problems involving model coefficient calibration because it is efficient at optimizing expensive functions. For example, Morita et al. [138] demonstrated the efficient performance of Bayesian optimization for applications to shape optimization and to optimal icing control problems. Additionally, zonal coefficient optimization can lead to improved predictive performance, as demonstrated by the zonal eddy viscosity approach of Matai and Durbin [72].

Naturally, different optimization problems will require different optimization techniques. Some techniques are more suited for high-dimensional problems, noisy problems, or problems where the gradient of the objective function is analytically available. In our view, the primary consideration for the turbulence model calibration problem is the computational cost associated with evaluating the objective function. If we formulate the objective function as an error measure between our reference data and the associated predictions of this data for a given set of model coefficients, then every evaluation of the objective function will require a new CFD calculation using the new turbulence model coefficients. The cost of a CFD calculation varies with the specific problem, but for an industrially relevant flow, a single three-dimensional (3D) RANS calculation can easily require on the

order of hundreds to thousands of central processing unit (CPU) hours. Therefore, this optimization problem demands a technique that can carefully and efficiently select new query points (viz., new values for the tuneable model coefficients at which the evaluation of the predictive performance will occur next). Bayesian optimization is a popular choice for optimizing computationally expensive objective functions. For example, Bayesian optimization is commonly used in neural network hyperparameter tuning, where each evaluation of the objective function typically involves an expensive training run [139].

Bayesian approaches have been applied for calibrating turbulence models in a number of previous studies. In particular, Bayesian inference has received attention because it can provide information on uncertainty in a RANS model prediction. Maruyama et al. [140] applied Bayesian inference to infer model coefficients of the Spalart-Allmaras (SA) model for a transonic flow over a wing. SA model coefficients were also inferred for several transonic airfoil problems in an investigation conducted by Da Ronch et al. [141], who found that a properly calibrated set of model coefficients improved the predictive accuracy on transonic test cases outside of the reference data used for the calibration. Ray et al. [142] developed an analytical model for the optimal coefficients of the k-ε turbulence model for a jet in a cross flow and also performed Bayesian calibration of the turbulence model coefficients. In particular, these investigators hypothesized that for many flows, model-form errors in RANS may be significantly smaller than coefficient calibration errors. A similar investigation, involving the calibration of the SA model for a jet flow with a concomitant determination of the uncertainty in the tuned model coefficients, was undertaken by Li et al. [143]. Edeling et al. [144] analyzed the error in the Launder-Sharma k-ε turbulence model for wall-bounded flows and applied Bayesian inference for calibrating the model coefficients. An earlier investigation conducted by Oliver and Moser [145] applied Bayesian uncertainty quantification to four eddy-viscosity models. Recently, Fischer et al. [146] coupled a neural network with Bayesian optimization to predict optimal values of coefficients of the generalized k-ω (GEKO) model [128] for a trench-shaped film-cooling flow. Another popular approach is the field inversion machine learning approach proposed by Singh et al. [18] which is often applied to infer local values of the calibration coefficients.

In this work, we introduce the **tu**ning pa**r**ameters using **B**ayesian **o**ptimization**-RANS** (turbo-RANS) platform. More specifically, turbo-RANS is a framework for applying Bayesian optimization to the RANS coefficient calibration problem. Bayesian optimization is particularly suited to problems where an evaluation of the objective function is computationally expensive—this is certainly the case for the turbulence model calibration problem in RANS. We formulate an objective function for use with turbo-RANS that is versatile in terms of the types of reference data that may be used. For three different model coefficient calibration problems, we demonstrate that turbo-RANS can be applied to tune

turbulence model coefficients in a computationally efficient manner. Another contribution of the present work is the provision of an open-source implementation of our proposed calibration framework. We illustrate the application of this framework on several example codes. Templates and source code have been made available in a GitHub repository [30]. We also investigate how the hyperparameters associated with the Bayesian optimization can be chosen and, in the course of this investigation, we provide general recommendations for the settings of these various hyperparameters that lead to a computationally efficient and stable calibration of the RANS model coefficients.

A number of previous studies have applied a Bayesian framework to the turbulence model coefficient calibration problem. However, most of these previous studies have focused on the problem as an inference problem. For example, given a set of RANS calculations and some reference data, Bayesian inference can provide a best estimate plus uncertainty (BEPU) of the turbulence closure coefficients. However, this methodology is computationally prohibitive for use with very expensive RANS computations (forward model predictions). Indeed, the computational demands of the RANS model predictions make it practically impossible as a tool for sampling from the posterior distribution using Markov chain Monte Carlo (MCMC) to produce a sequence of dependent draws from the posterior. In the case where one is under the constraint of a very limited computational budget (typically less than about 100 RANS model calculations), we advocate the application of Bayesian optimization to the problem: given previous knowledge of the problem, what is the next most efficient point to query? The use of the Bayesian optimization approach here results in a naturally computationally efficient optimization procedure.

The Bayesian optimization approach allows the exploration and exploitation tradeoff to be tightly controlled during optimization. Moreover, the objective function proposed here is flexible and pragmatic and, as a result, is well-suited for use with Bayesian optimization. Whereas many previous studies used simple objective functions (e.g., root-mean-squared error), the objective function proposed here produces a well-posed optimization problem and can simultaneously be used with different types of reference data. We demonstrate that using this objective function for calibration leads generally to an accurate prediction of various quantities of interest derived from the turbulent flow. Finally, we believe that the complexity of the implementation and the ease of use (or lack thereof) are often overlooked factors that limit the widespread use of Bayesian calibration. To address these factors, we present a straightforward methodology and provide an open-source implementation that can be immediately used in a variety of CFD contexts.

This paper is organized as follows. In Section 6.2, we highlight relevant concepts of Bayesian optimization. We then discuss the formulation of the generalized objective function used in the turbo-RANS framework, providing a rationale for this formulation. The

turbo-RANS framework is then described in detail along with how this framework is implemented. In Section 6.3, we apply turbo-RANS to calibrate coefficients in the k-ω SST and GEKO turbulence closure models for three different types of reference data: namely, integral parameter data, dense DNS data, and sparse wall measurement data. In Section 6.4, we perform an extensive study of how to choose the hyperparameters associated with the Bayesian optimization used in the RANS model calibration. Based on this hyperparameter investigation, we provide recommended settings for the hyperparameters that appear in the Bayesian optimization schema, including those used in the Gaussian process regression. Finally, Section 6.5 provides a conclusion—here, we contextualize the results obtained in this study and provide suggestions for future work.

6.2 Background and Methodology

This section is organized as follows. Sections 6.2.1–6.2.3 provide a high-level overview of Bayesian optimization. Section 6.2.4 explains the proposed generalized error and default coefficient preference (GEDCP) objective function, which we recommend for optimizing turbulence model coefficients. Section 6.2.5 provides details and a discussion of the turbo-RANS framework. Implementation of this framework is discussed in Section 6.2.6.

6.2.1 Bayesian optimization

We summarize the main ideas underpinning Bayesian optimization and Gaussian process regression as they are applied in the turbo-RANS framework. Bayesian optimization is described in detail in Garnett [139], and Gaussian process regression is explained in detail in Rasmussen and Williams [147].

The goal of Bayesian optimization is to maximize an objective function $f(x)$ with respect to a vector of parameters x. Since this objective function can be computationally expensive to compute, we create a surrogate model of the objective function that is significantly less expensive to compute. It is useful for the surrogate model to return both the best estimate $\mu(x)$ of the objective function and the uncertainty $\sigma(x)$ in this best estimate. We introduce a utility function $\alpha(x) \equiv \alpha(\mu(x), \sigma(x))$ that combines the information embodied in the predicted objective function value and uncertainty in this value into a single scalar-valued function (see Section 6.2.3). The next x value queried is the point that maximizes the utility function $\alpha(x)$. This utility function allows us to control the tradeoff between exploration and exploitation. Exploration is the evaluation of the

objective function in new regions far away from the currently available data (i.e., points with high uncertainty $\sigma(x)$). Exploitation is the evaluation of the objective function in regions where it is expected to be favourable based on past data (i.e., points with a high expected value of $\mu(x)$). The primary advantage of Bayesian optimization is the ability to control this exploration/exploitation tradeoff, which produces an optimization procedure that is efficient in terms of the number of objective function evaluations required to find the optimal value of x.

6.2.2 Gaussian process regression

In Bayesian optimization, Gaussian process regression is typically used to construct a surrogate model of the objective function. Given a dataset $\mathcal{D}$ consisting of a set of values of the objective function $f(x)$ at points x ($\big\{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))\big\}$), we can fit a Gaussian process regression model $\mathcal{GP}(x)$ and use it as a surrogate model for $f(x)$. A Gaussian process regression model fits a Gaussian process to $f(x')$ for all x' in the domain of definition of the function. The Gaussian process is *conditioned* on the given data $\mathcal{D}$ consisting of $f(x)$ measured at points x, so $f(x') \sim N\big(\mu(x'), k(x, x'; \theta)\big)$ where $N\big(\mu(x'), k(x, x'; \theta)\big)$ is the Gaussian process with mean function $\mu(x')$ and $k(x, x'; \theta)$ is the covariance function between the pair of points x and x'. The covariance function between the pair of points depends on the hyperparameters θ. A complete description of Gaussian process regression is provided in the book by Rasmussen and Williams [147].

In this work, the covariance function $k(x, x')$ (suppressing the vector θ of hyperparameters from the notation) used for Gaussian process regression is the Matern kernel which has the following form:

$$k(x_i, x_j) = \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l_k} d(x_i, x_j) \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{l_k} d(x_i, x_j) \right) \ , \tag{6.1}$$

where $k(x_i, x_j)$ is the covariance of the points x_i and x_j, ν is a kernel parameter (hyperparameter), $\Gamma(\nu)$ is the gamma function evaluated at ν, l_k (hyperparameter) is the kernel length scale (or characteristic length scale of the Gaussian process), $d(x_i, x_j)$ is the Euclidean distance between x_i and x_j, and $K_\nu(\ \cdot\ )$ is the modified Bessel function of order ν. More details are available in the scikit-learn Matern kernel documentation [85]. The critical hyperparameters of the Matern covariance function (kernel) are ν and l_k, which are discussed below.

The behavior of the Matern kernel depends on the value of the hyperparameter ν. When $\nu = 1/2$, this kernel reduces to the absolute exponential kernel. When $\nu \to \infty$,

the Matern kernel reduces to the squared exponential kernel. For $\nu = 3/2$ and $5/2$, the associated Gaussian process is once- and twice-differentiable, respectively.

The length scale l_k of the Matern kernel is a hyperparameter that needs to be optimized for a given dataset $\mathcal{D}$. In this work, we use scikit-learn for the Gaussian process regression, which automatically fits this hyperparameter to the available data. However, an initial guess for l_k needs to be specified, and tuning this initial guess can lead to a more efficient fitting of the Gaussian process which, in turn, improves the subsequent Bayesian optimization performance [148]. We reparameterize this initial guess in terms of a relative length scale l. The l_{k,c_i} length scale parameter associated with a model coefficient c_i is determined as follows:

$$l_{k,c_i} = l(c_{i,\text{upper}} - c_{i,\text{lower}}) \ , \tag{6.2}$$

where l_{k,c_i} is an initial guess for the kernel length scale associated with coefficient c_i. Moreover, $c_{i,\text{upper}}$ and $c_{i,\text{lower}}$ are the upper and lower bounds of the domain of definition for the coefficient c_i, respectively. Also, l_k is the length scale vector with components $\left(l_{k,c_1}, l_{k,c_2}, ..., l_{k,c_N}\right)$. This vector defines an anisotropic Matern covariance function (kernel) in the sense that the length scale associated with each model coefficient c_i can have a different value. The motivation behind parameterizing l_k as a vector is to specify a scale-appropriate initial guess for l_k (viz., an appropriate scale for each model coefficient). For example, two model coefficients $c_1 \in [0.5, 1.5]$ and $c_2 \in [0.001, 0.005]$ will require two different length scales. Eq. (6.2) will apply the same relative length scale to both dimensions, resulting in scale-appropriate length scales for each dimension. Finally, we note that the hyperparameter vector θ for the Matern covariance function (kernel) has the explicit form $\theta = (\nu, l_k^{\text{T}})^{\text{T}}$ where the superscript "T" denotes vector transposition.

6.2.3 Utility functions

Upper confidence bound (UCB) utility function

The upper confidence bound (UCB) utility function features a clear exploration/exploitation tradeoff that is controlled by the κ hyperparameter. The UCB utility function is given by

$$\alpha_{\text{UCB}}(x) = \mu(x) + \kappa\sigma(x) \ , \tag{6.3}$$

where $\alpha_{\text{UCB}}(x)$ is the utility function value at point x, $\mu(x)$ is the expected value of the objective function, and $\sigma(x)$ is the standard deviation at point x (viz., $\sigma(x) = k(x,x)^{1/2}$). Both $\mu(x)$ and $\sigma(x)$ are readily available from the Gaussian process regression of the objective function. Larger values of κ lead to more exploration of points in the parameter

space with large uncertainty, and smaller values of κ lead to more exploitation of regions in the parameter space where the expected value $\mu(x)$ is known to be large.

Probability of improvement (POI) utility function

The probability of improvement (POI) utility function returns the probability that a given point x will result in an improvement of the objective function value. The POI hyperparameter ξ specifies a minimum probability of improvement. The POI utility function is given by

$$\alpha_{\text{POI}}(x) = \text{CDF}\left(\frac{\mu(x) - \mu^\star - \xi}{\sigma(x)}\right) \ , \tag{6.4}$$

where $\alpha_{\text{POI}}(x)$ is the utility function value at point x, $\text{CDF}(z)$ is the cumulative distribution function of the standard Gaussian distribution (viz., a Gaussian distribution with zero mean and unit variance), $\mu(x)$ is the expected value of the objective function, $\mu^\star$ is the best objective function value observed so far, and $\sigma(x)$ is the standard deviation at x. Again, $\mu(x)$ and $\sigma(x)$ are immediately available from the Gaussian process regression of the objective function. Larger values of ξ lead to a more explorative behavior, whereas smaller values of ξ lead to a more exploitative behavior.

Expected improvement (EI) utility function

The expected improvement (EI) utility function improves on the POI utility function. Whereas the POI utility function finds the point with the highest probability of providing any improvement, the EI utility function finds the point with the highest expected improvement. The hyperparameter ξ is also used to control the exploration/exploitation tradeoff for the EI utility function. The EI utility function has the following form:

$$\alpha_{\text{EI}}(x) = \text{CDF}\left(\frac{\mu(x) - \mu^\star - \xi}{\sigma(x)}\right)(\mu(x) - \mu^\star - \xi) + \text{PDF}\left(\frac{\mu(x) - \mu^\star - \xi}{\sigma(x)}\right)\sigma(x) \ , \tag{6.5}$$

where all variable definitions are the same as those of the POI function (Section 6.2.3), and $\text{PDF}(z)$ is the probability density function for a standard Gaussian distribution. As in the case of the POI utility function, smaller ξ values lead to more exploitation, whereas larger ξ values lead to more exploration, in the parameter space.

6.2.4 Generalized Error and Default Coefficient Preference objective function

Objective function design is important for optimization. In general, the purpose of turbo-RANS is to reduce the error in a RANS prediction by adjusting the turbulence model coefficients. Therefore, the objective function should represent errors in the RANS predictions that ensure a well-posed optimization problem. In view of this, we formulate an objective function for use with turbo-RANS and demonstrate its utility.

We formulate a functional form for an objective function, which will be referred to henceforth as the generalized error and default coefficient preference (GEDCP) function. This function aims to be both highly versatile in terms of the types of reference data that can be used in the optimization and well-posed in terms of the process of the optimization itself. This function computes an error measure that can be used to assimilate dense reference data, sparse reference data, or integral reference data (or any combination of these types of data), making it adaptable generally for any kind of calibration problem. For example, RANS coefficients can be calibrated so that the predicted velocity field matches sparse wind-tunnel velocity data or so that the predicted force coefficients match those obtained from experimental data. For unsteady simulations, the coefficients can be calibrated based on an experimental Strouhal number or on a time-averaged force coefficient. The GEDCP function can also combine different types of reference data into a single objective function. For example, coefficients can be calibrated to simultaneously reduce errors in sparse reference data and in reference data for an integral parameter (e.g., lift or drag coefficient).

An optional term is included in the GEDCP function that embeds a preference for the "default" values of the turbulence model coefficients. In practice, including this term serves two goals: namely, it embeds prior knowledge of the problem into the objective function, and it often results in the optimization problem having a single global optimum. For an example that motivates the first goal, consider the prior knowledge that $C_\mu = 0.09$ is commonly used in linear eddy viscosity models. This prior knowledge may be an important preference that is relevant for some applications. This default value is based on the correct ratio of $\nu_t \varepsilon / k^2$ in a turbulent channel flow away from the wall [3]. Here, ν_t is the eddy viscosity, ε is the dissipation rate, and k is the turbulent kinetic energy. If the optimized C_μ deviates from this value, there must be a corresponding significant reduction in the misfit error for this state of affairs to be acceptable. This preference penalty must also increase with increasing deviation from the default value of the coefficient, so if the nominally optimal value for C_μ is very far from 0.09, a large reduction in the fitting error to the reference data must occur for this to be acceptable. The second goal of the preference

term is to ensure that the optimization problem has a single (unique) solution. In our experience, an error-based objective function based solely on the misfit between the model predictions and the reference data for calibrating RANS model coefficients often exhibits a saturation in the misfit. In other words, if the coefficient exceeds a certain value, the misfit error remains constant and, as a result, the optimization problem does not have a unique solution. Saturation or asymptotic behavior in the misfit can arise from the nature of the physics in the problem or from the presence of minimum, maximum, hyperbolic tangent, and other functions that are used in various combinations in many turbulence models. With a preference for the default value in a model coefficient, a small slope is introduced into the objective function, which drives coefficients towards the default value of the coefficient if this circumstance arises.

The GEDCP objective function, which depends on the model coefficients $\{c_1, c_2, \ldots, c_N\}$ that are to be calibrated against the available reference data, is formulated as follows:

$$f_{\text{GEDCP}}(c_1, c_2, \ldots, c_N) = -(\lambda_{\text{F}} E_{\text{F}} + \lambda_{\text{I}} E_{\text{I}})(1 + \lambda_p p)\big|_{c_1, c_2, \ldots, c_N} , \tag{6.6}$$

$$E_{\text{F}}(c_1, c_2, \ldots, c_N) = \sum_{\phi \in \Phi} \text{MAPE}(\phi)\big|_{c_1, c_2, \ldots, c_N} = \sum_{\phi \in \Phi} \frac{1}{N_\phi} \sum_{i=1}^{N_\phi} \left| \frac{\phi_i - \tilde{\phi}_i(c_1, c_2, \ldots c_N)}{\phi_i} \right| , \tag{6.7}$$

$$E_{\text{I}}(c_1, c_2, \ldots, c_N) = \sum_{\psi \in \Psi} \left| \frac{\psi - \tilde{\psi}(c_1, c_2, \ldots, c_N)}{\psi} \right| , \tag{6.8}$$

$$p(c_1, c_2, \ldots, c_N) = \frac{1}{N} \sum_{n=1}^{N} \left| \frac{c_{n,\text{default}} - c_n}{c_{n,\text{default}}} \right| . \tag{6.9}$$

The variables and terms in Eqs. (6.6)–(6.9) are defined below.

In Eq. (6.6), $f_{\text{GEDCP}}(c_1, c_2, ..., c_N)$ is the objective function which depends on the tuneable coefficients $c_1, c_2, \ldots, c_N$; E_{F} is the field error; E_{I} is the integral parameter error; p is the default coefficient preference term; and, λ_{F}, λ_{I}, and λ_p are regularization parameters that control the amount of preference (relative importance) ascribed to each term. The negative sign in Eq. (6.6) converts the problem of minimizing $E_{\text{F}}, E_{\text{I}}$, and p to a maximization problem, which is the convention used for Bayesian optimization.

The reference data used for calibration of the model coefficients consist of a set of field values (e.g., velocity, pressure) and integral parameters (e.g., lift coefficient). The set of calibration fields is specified by Φ, and the individual calibration fields are denoted by $\phi \in \Phi$. Individual reference values for a given calibration field are denoted by ϕ_i.

In a similar manner, the set of reference integral parameters is specified by Ψ, and the individual integral parameters are denoted by $\psi \in \Psi$. As an example, given calibration data consisting of 15 locations with pressure measurements P, 50 locations with measured velocity components U and V, a reference lift coefficient c_l, and a reference drag coefficient c_d, we have the following sets: $\Phi = \{P, U, V\}$, $N_P = 15$, $N_U = N_V = 50$, and $\Psi = \{c_l, c_d\}$.

The field error E_{F} [Eq. (6.7)] is a summation of the mean absolute percentage errors (MAPE) in each field ϕ over the set of fields in the set Φ for which reference data are available for calibration. Here, $i = 1, 2 \ldots, N_\phi$ indexes the points where reference data corresponding to the field ϕ are available. Furthermore, ϕ_i is the reference datum associated with the field ϕ at a given location and $\tilde{\phi}_i(c_1, c_2, \ldots, c_N)$ is the predicted value of the field at this location obtained using the set of model coefficients $\{c_1, c_2, \ldots, c_N\}$. This term can accommodate arbitrarily sparse and/or dense reference data and involves the determination of MAPE over all available reference data for a given field. Moreover, the use of MAPE implies that the influence of the reference data is not affected by the scale of ϕ_i. For example, a 25% error at a point with low velocity in the boundary layer is penalized equally to that of a 25% error at a point with high velocity. The use of MAPE here is also motivated so that varying scales of $\phi \in \Phi$ are treated equally; for example, the velocity specified in units of m s^{-1} can have a different scale than that associated with the pressure in units of Pa. This equal treatment would not hold true if a mean-squared error (MSE) or mean-absolute error (MAE) were to be used in the formulation of the GEDCP objective function.

The integral parameter error E_{I} [Eq. (6.8)] is a summation of the percentage error in each integral parameter for which reference data are available. Here, Ψ is the set of integral parameters. Again, the percentage error is used so that varying scales associated with different quantities of interest within Ψ are treated equally. For example, consider the calibration of an aerodynamic simulation to match reference data for the lift c_l and drag c_d coefficients. A given prediction using model coefficients $c_1, c_2, \ldots, c_N$ will provide an estimate for the lift $\tilde{c}_l(c_1, c_2, \ldots, c_N)$ and drag $\tilde{c}_d(c_1, c_2, \ldots, c_N)$ coefficients. The use of a relative error implies that even though c_l and c_d can have different orders of magnitude, the percentage errors will be penalized equally.

The default coefficient preference term p is given by Eq. (6.9). This term can be interpreted as the mean relative deviation of all model coefficients from their corresponding default values. The use of a relative (rather than an absolute) deviation is again motivated by possible scale differences in the values of the model coefficients.

In Eq. (6.6), λ_{F}, λ_{I}, and λ_p can be adjusted to determine the preference for reference field data, reference integral data, and for default values of the model coefficients. We

recommend setting $\lambda_{\mathrm{F}} = \lambda_{\mathrm{I}} = 1$, and $\lambda_p = 1/2$. These settings retain some preference for the default values of the model coefficients while ensuring that the objective function is primarily influenced by the error-based terms E_{F} and E_{I}. If no reference field data is available (viz., $\Phi = \emptyset$), then $\lambda_{\mathrm{F}} = 0$. Similarly, the absence of reference integral parameter data (viz., $\Psi = \emptyset$) implies $\lambda_{\mathrm{I}} = 0$.

The multiplicative design incorporated in Eq. (6.6) is to allow deviations from the default values of the model coefficients, as long as a corresponding significant decrease in misfit (prediction) error is realized. If the optimizer begins to "stray" too far from the default values of the model coefficients with only a marginal or negligible reduction in the prediction error, then the default coefficient preference term will penalize this behavior. However, if this departure from the default values of the model coefficients results in a significant prediction error reduction, then the objective function will still reward the proposed departure in the optimization process. For example, in the event that the proposed model coefficients result in perfect predictions of the quantities of interest (viz., $E_{\mathrm{F}}, E_{\mathrm{I}} \to 0$), then the default coefficient preference term has no effect. As long as the calibration of the model coefficients results in some error in the fitting process (which is expected for the majority of use cases), then the default coefficient preference term will influence the objective function.

As part of the hyperparameter optimization in this work (see Section 6.4), the GEDCP loss functions for two example problems (see Sections 6.3.1– 6.3.2) were computed on a dense grid of RANS predictions associated with a grid of values of the model coefficients. This computation allows for the visualization of the GEDCP loss functions for two typical model coefficient calibration problems.

Fig. 6.1 shows the GEDCP loss function associated with the calibration of the a_1 coefficient of the k-ω SST turbulence closure model. The calibration of this model coefficient was conducted in order to improve the predictive performance of the turbulence model for the lift coefficient of an airfoil. More details of this example problem are provided in Section 6.3.1. Over a wide range of values of a_1, the GEDCP objective function for this case exhibits a single global optimum point and is a relatively straightforward function to optimize. For this example, turbo-RANS is able to find the optimum value of $a_1 = 0.28$ in approximately 9 evaluations of the objective function (see Section 6.3.1 for more detailed results). In the range of $0.34 < a_1 < 0.37$, a small negative slope can be seen in $f_{\mathrm{GEDCP}}(a_1)$. This small negative slope is due to the presence of the default coefficient preference term in the GEDCP objective function. Without the inclusion of this default coefficient preference term, an objective function that involves only a misfit term would have a zero slope here and, as a result, the optimizer would not be "guided away" from the upper bound for a_1 when exploring this plateau in the objective function.

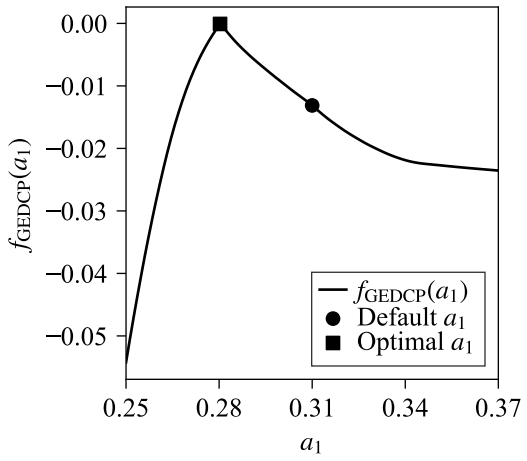


Figure 6.1: Objective function [Eq. (6.16)] for the airfoil case. The objective function is computed for a set of 50 RANS predictions with a_1 taking values in the interval $[0.25, 0.37]$.

A more involved example that justifies the inclusion of the default coefficient preference term in the objective function is exhibited in Fig. 6.2. The GEDCP objective function shown here corresponds to optimizing the two model coefficients a_1 and β^* of the k-ω SST turbulence closure model in order to improve the predictive performance of the model for a flow over periodic hills. The misfit error used in this example is a summation over the entire flow field because dense DNS data is available for this case. More details on this case are provided in Section 6.3.2. Again, a single local maximum over a wide range of values of the model coefficients exists in this case. Without the inclusion of the default coefficient preference term, the objective function would asymptotically saturate (viz., achieve a maximum value) along a line defined by $\beta^* \approx 0.08$ in the range $0.36 < a_1^* < 0.60$. With a pure misfit error-based objective function, in spite of the fact that there is virtually no improvement in the predictive accuracy for increasing values of a_1, the optimizer in this case nevertheless recommends that the best estimate of $a_1 = 0.60$ (i.e., this value for a_1 maximizes the objective function). This large value for a_1 is not justified (or even reasonable) because there is only a marginal improvement in the predictive performance for values of a_1 greater than about 0.34. The GEDCP objective function addresses this problem through the inclusion of a small preference for the default values of the model coefficients. In so doing, a single (unique) solution is obtained for the optimization problem, resulting in reasonable values for the model coefficients: namely, $\beta^* = 0.076$ and $a_1 = 0.34$. This optimum is found in approximately 20 evaluations of the objective function (see Section 6.3.2 for more results).

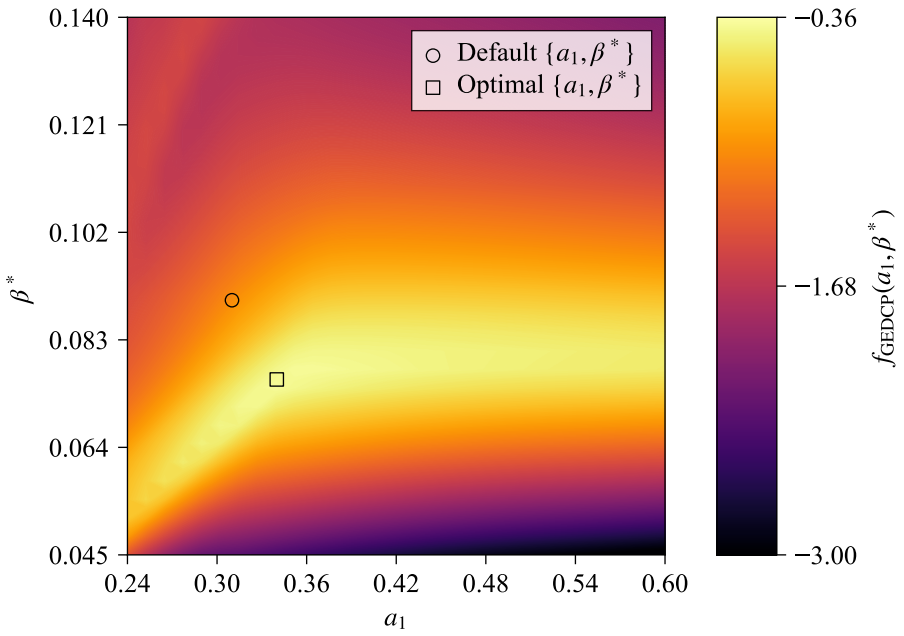


Figure 6.2: Objective function [Eq. (6.18)] for the periodic hills case. The objective function is computed from a set of 900 RANS predictions with $a_1 \in [0.24, 0.60]$ and $\beta^* \in [0.045, 0.140]$.

6.2.5 turbo-RANS framework

Fig. 6.3 shows the turbo-RANS framework, along with details of the probe-and-suggest subroutines. Various steps of the algorithms incorporated in this framework will be described below.

To initialize the optimization process, information must be provided by the user as to which model coefficients are to be optimized. This information includes the default values for the model coefficients and their lower and upper bounds. Hyperparameters (see Section 6.4) used in the optimization, as well as other settings, must also be specified by the user as part of the initialization. It is noted that the hyperparameters can be modified during each iteration of the optimization process. The first point probed is the point corresponding to the default values for the model coefficients. If the chosen optimization problem does not include default values for the model coefficients, then this first default coefficient probe is skipped in the initialization.

After probing the default values for the model coefficients, the sampling loop is executed. In the framework, n_s is the number of sample points that are probed before the Bayesian optimization loop is entered. If n_s is too small, errant behavior of the Gaussian process regression can occur (see Section 6.4). At this stage, values of the model coefficients are sampled using Sobol sampling to produce a Sobol sequence of points, which promotes better uniformity of the sampled values compared to a purely random sampling [149]. After a new point is obtained using Sobol sampling, this point is probed. The sampling loop continues until a number of points n_s (prespecified by the user) have been probed.

The Bayesian optimization loop begins after a default point and a sufficient number of additional sample points have been probed. In the Bayesian optimization loop, the optimizer considers the past history and suggests a new value of the model coefficients (viz., a point) to be probed. Following this, the suggested new point is probed. The probe-suggest loop continues until a convergence criterion is satisfied. This convergence criterion is specified by the user. We envision that turbo-RANS loops that involve relatively inexpensive CFD calculations can be stopped when the best objective function value and/or the best set of model coefficients do not provide any further improvements (changes) in the optimization process after a certain number of iterations (early stopping). For problems corresponding to more expensive CFD calculations, an upper bound on the computational budget can be prescribed *a priori* by the user (e.g., repeat the loop until a pre-specified number of CPU hours has been expended in the computation). With either the convergence-based or budget-based stopping criteria, the Bayesian optimization algorithm will result in an efficient optimization in terms of the number of objective function evaluations (viz., CFD calculations). After the stopping criteria are satisfied, the final rec-

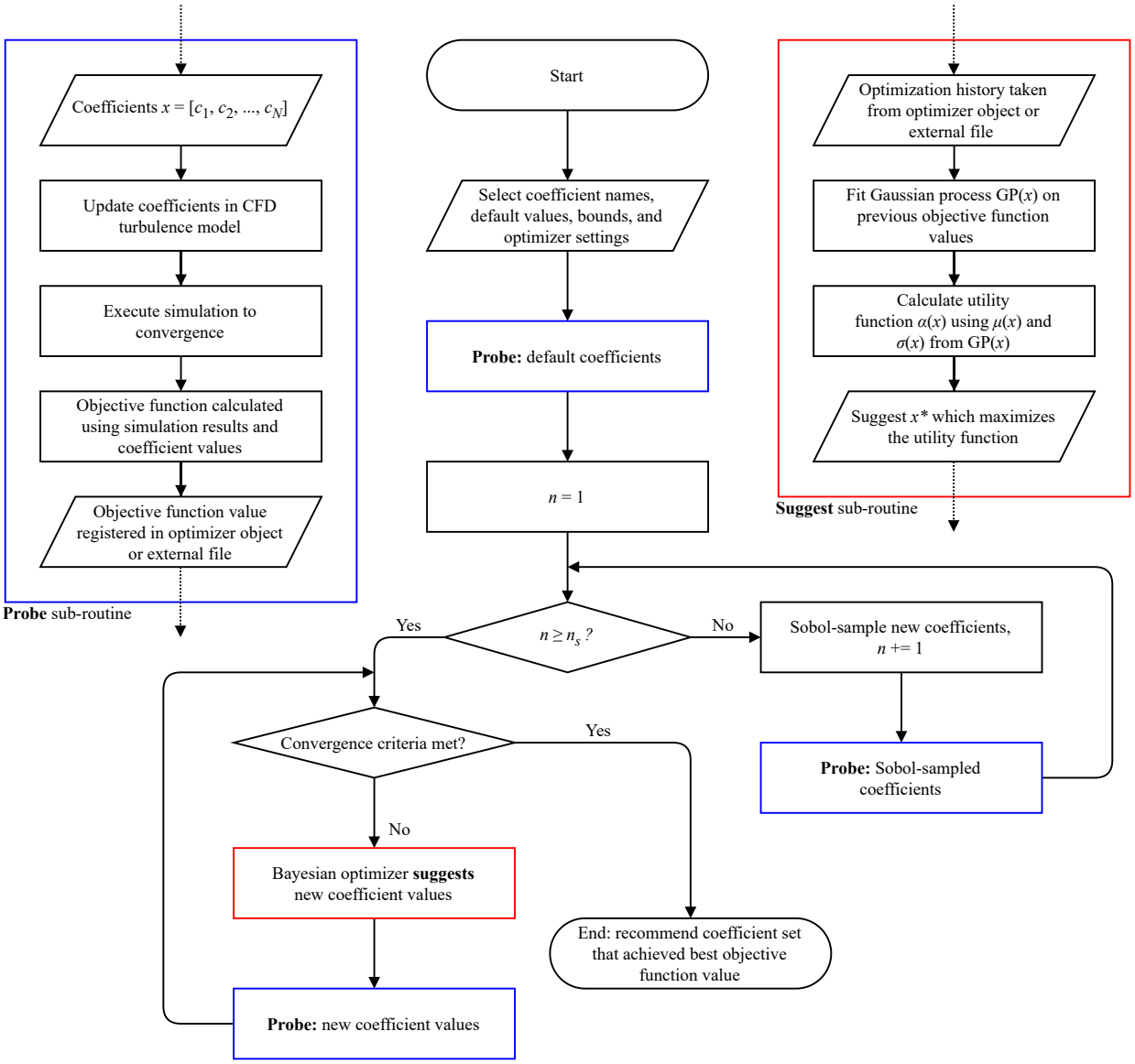


Figure 6.3: The turbo-RANS framework summarized in an algorithm flowchart.

ommended model coefficient set is the set that achieves the maximum value of the objective function.

Probing a set of values of the model coefficients consists of running a RANS calculation with these coefficients, computing the objective function, and registering the objective function value so that it can be used in future iterations. The probe sub-routine in Fig. 6.3 details these steps. The convergence criteria for the RANS calculations are also specified (prescribed) by the user. We recommend the standard practice of evaluating both the residual values of various flow quantities and monitoring an integral parameter to assess convergence. In terms of actual implementation, this process is typically automated using scripts that modify the turbulence model coefficients, execute a RANS calculation, and monitor the convergence of this calculation. To reduce computational demands, we recommend using the converged flow fields obtained from the default coefficient probe (viz., the default values of the model coefficients) as initial conditions for future probe RANS calculations. This re-use of the converged flow field from the default coefficient probe greatly reduces the computational costs of the turbo-RANS algorithm, since the flow fields start near convergence. The post-processing of RANS calculations into fields that are compatible with the GEDCP objective function (Section 6.2.4) is also typically automated. For example, this post-processing can include computing force coefficients or sampling the solution field at the locations where the reference data is available. Details of the objective function registration are provided in Section 6.2.6, and in the turbo-RANS documentation [30].

The suggest sub-routine exhibited in Fig. 6.3 addresses two sub-optimization problems: namely, (1) fitting a Gaussian process using all previously probed points obtained in modelling of the objective function; and, (2) maximizing the utility function. These optimization problems are already implemented in the dependencies for turbo-RANS. The suggest sub-routine executes almost instantaneously, and therefore the main computational expense in the turbo-RANS algorithm is conducting a RANS calculation with a given set of turbulence model coefficients. The computational time of the suggest sub-routine increases with the number of points previously sampled, as the cost of the Gaussian process regression increases with an increasing number of points. However, in our experience, the computational cost of the suggest sub-routine is negligible compared to the cost of conducting the RANS calculations required for each point probed (undertaken in the probe sub-routine). For large parameter spaces and a large number of previously evaluated points defining the objective function hypersurface (e.g., greater than 100), the Gaussian process regression may perform slowly. However, even in the worst case, we have observed that the cost of the suggest sub-routine remains below 1% of the total cost associated with the RANS calculations required for each probed point.

6.2.6 Implementation

The core Bayesian optimization package used in turbo-RANS is the Python library bayesian-optimization [148]. This library is widely used in many Bayesian optimization applications and includes implementations of the UCB, POI, and EI utility functions. The underlying Gaussian process regression library used in bayesian-optimization is from scikit-learn [85]. A series of convenient Python interfaces to these two libraries has been provided in the turbo-RANS GitHub repository [30]. The repository includes examples and templates for a wide variety of use cases, including the following:

- two examples based on OpenFOAM: flow over an airfoil (Section 6.3.1), and flow over periodic hills (Section 6.3.2);
- one example based on Ansys Fluent: flow through a converging-diverging channel (Section 6.3.3);
- an example script for STAR-CCM+ users;
- templates for optimizing OpenFOAM simulations;
- solver-agnostic templates for optimizing CFD simulation Python scripts; and,
- solver-agnostic templates for optimizing CFD simulation shell scripts.

Full documentation for the code is provided in the GitHub repository [30]. It should be noted that the parallelization of Bayesian optimization is non-trivial. While a RANS calculation with a given set of model coefficients is typically parallelized, the primary turbo-RANS algorithm is currently restricted to running in a sequential manner. This restriction means that only a single RANS calculation is conducted at a time, rather than probing multiple sets of model coefficients to enable multiple RANS calculations to be undertaken simultaneously. It is assumed that the RANS calculation step is by far the most computationally expensive component of the turbo-RANS algorithm since the acquisition of a new point in the Bayesian optimization occurs nearly instantaneously. If desired, the user can opt to implement a parallelized Bayesian optimization by modifying the turbo-RANS example scripts. The code's verbose file input and output structure facilitates implementing parallelized Bayesian optimization but, for now, this functionality is left for a future effort.

6.3 Demonstrations

6.3.1 Example 1: Flow over an airfoil

Description

RANS calculations are often used in external aerodynamics, such as in aerospace, wind engineering, and automotive engineering. Accurate prediction of integral quantities such as force coefficients is typically desired in these applications. The purpose of this example is to demonstrate how turbo-RANS can be used to calibrate turbulence model coefficients using integral parameter reference data.

Computational setup

We calculate the steady-state, incompressible, turbulent flow over a NACA 0012 airfoil at a 10.12° angle of attack and a chord-based Reynolds number of $Re_c = 6 \times 10^6$. The case details are taken from the NASA turbulence modelling resource "2D NACA 0012 Airfoil Validation" 2DN00 case [25]. OpenFOAM v2206 was used—the RANS calculations were conducted using the semi-implicit method for pressure-linked equations (SIMPLE) solver (simpleFoam) in OpenFOAM. A second-order upwind scheme was used for the discretization of the convective terms in the momentum equation, and the central difference scheme was used for the discretization of the diffusion terms. A first-order upwind scheme was used for the discretization of the convective terms in the turbulence transport equations. The geometric agglomerated algebraic multigrid (GAMG) solver was used to obtain the pressure, and the preconditioned bi-conjugate gradient (PBiCGStab) solver was used to solve all the other equations. The full OpenFOAM case setup is available in the turbo-RANS GitHub repository [30].

The turbulence model used for this case is the k-ω shear stress transport (SST) model. The OpenFOAM v2206 variant of this model is the updated 2003 Menter version [53], with production terms from Menter and Esch [88, 50]. The following model equations are provided in Menter et al. [53] (k is the turbulent kinetic energy and ω is the specific

dissipation rate):

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial(\rho U_i k)}{\partial x_i} = \tilde{P}_k - \beta^* \rho k \omega + \frac{\partial}{\partial x_i}\left[(\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_i}\right] , \tag{6.10}$$

$$\frac{\partial(\rho \omega)}{\partial t} + \frac{\partial(\rho U_i \omega)}{\partial x_i} = \alpha \frac{\tilde{P}_k}{\nu_t} - \beta \rho \omega^2 + \frac{\partial}{\partial x_i}\left[(\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_i}\right] + 2(1 - F_1)\rho \sigma_{\omega 2} \frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i} , \tag{6.11}$$

$$\nu_t = \frac{a_1 k}{\max(a_1 \omega, S F_2)} , \tag{6.12}$$

where the definitions of the blending functions F_1 and F_2, and limited production term $\tilde{P}_k$ have been omitted as they are not relevant to the present discussion. Given that the coefficients are a blend of the coefficients of the k-ε and k-ω turbulence closure models, the k-ω SST model has 10 model coefficients: a_1, β^*, σ_{k1}, σ_{k2}, α_1, α_2, β_1, β_2, $\sigma_{\omega 1}$, and $\sigma_{\omega 2}$. Default values for these coefficients are given as: $a_1 = 0.31$, $\beta^* = 0.09$, $\sigma_{k1} = 0.85$, $\sigma_{k2} = 1$, $\alpha_1 = 5/9$, $\alpha_2 = 0.44$, $\beta_1 = 3/40$, $\beta_2 = 0.0828$, $\sigma_{\omega 1} = 0.5$, and $\sigma_{\omega 2} = 0.856$. Here, the subscript 1 denotes the k-ω model, and the subscript 2 denotes the k-ε model. The default values of these coefficients arise from the application of various heuristics, such as tuning the behavior of certain terms to be in conformance with the characteristics of specific types of (usually canonical) flows. For many industrially relevant flows, additional tuning of these model coefficients can increase predictive accuracy. For the airfoil case, we employ turbo-RANS to calibrate the a_1 coefficient of the k-ω SST turbulence model for this particular flow.

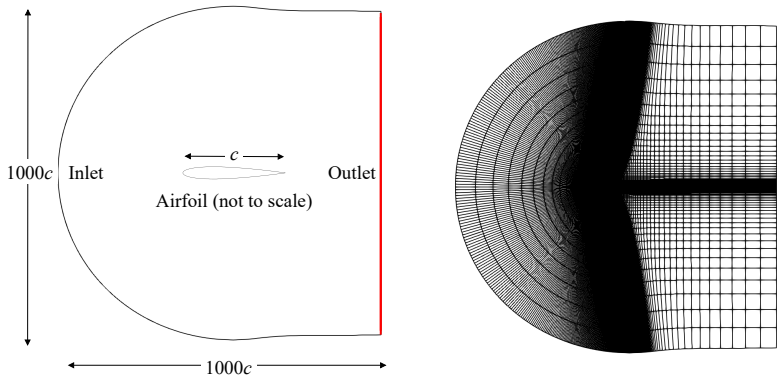
Fig. 6.4 shows the computational domain and mesh, which were taken from the NASA turbulence modelling resource [25]. The computational domain is relatively large. More specifically, the domain is $500c$ (c is the chord length) in any direction from the airfoil so as to negate blockage effects. Pressure at the inlet is zero gradient, and all other flow quantities at the inlet are prescribed as follows:

$$U_{\text{inlet}} = (50.68, 0, 9.05) \text{ m s}^{-1} , \tag{6.13}$$

$$k_{\text{inlet}} = 0.001075 \text{ m}^2 \text{ s}^{-2} , \tag{6.14}$$

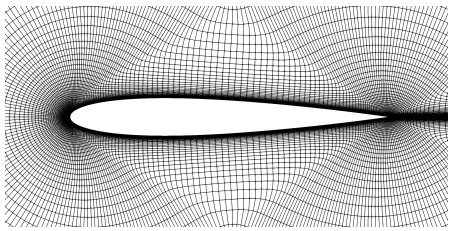
$$\omega_{\text{inlet}} = 0.3279 \text{ s}^{-1} . \tag{6.15}$$

In particular, turbulence quantities at the inlet are based on a 0.052% turbulence intensity and a turbulent length scale of 10% of the chord length. At the outlet plane of the computational domain, all quantities except pressure are zero-normal-gradient, and the gauge pressure is fixed at zero Pa.



(a) Computational domain and boundary names.

(b) Computational mesh.



(c) Close-up of computational mesh on the airfoil surface.

Figure 6.4: Computational setup for the airfoil example case.

The mesh for this case consists of 57,344 structured hexahedral cells, with approximately 129 points along the surface of the airfoil. With the default values for k-ω SST model coefficients, the y^+ wall-normal distance from the airfoil surface has values lying between 0.017 and 0.88, with a mean value of 0.30. These values imply that the RANS computation for this case is wall-resolved.

Optimization problem

The purpose of the airfoil example case is to optimize the a_1 model coefficient in the k-ω SST turbulence model in order to improve the accuracy of the model for prediction of the

lift coefficient c_l. The objective function for the airfoil case is given by

$$f_{\text{GEDCP}}(a_1) = -(E_{\text{I}})(1 + \tfrac{1}{2}p)\big|_{a_1} \ , \tag{6.16}$$

$$E_{\text{I}}(a_1) = \left| \frac{c_l - \tilde{c}_l(a_1)}{c_l} \right| \ . \tag{6.17}$$

Here, $f_{\text{GEDCP}}(a_1)$ is the GEDCP objective function [Eq. (6.6)], with $\lambda_{\text{I}} = 1$, and $\lambda_p = 1/2$. This objective function primarily aims to minimize the relative error in the prediction of the integral parameter c_l while retaining a minor preference for the default value of the model coefficient: namely, $a_1 = 0.31$. In the present work, we have computed this objective function over a grid of values of a_1 so that it can be visualized (cf. Fig. 6.1). No field quantities are used in the objective function for the airfoil case. The parameter space for the model coefficient for the airfoil case is $a_1 \in [0.25, 0.37]$. The coefficient a_1 was calibrated so that the predictions matched the reference lift coefficient of $c_l = 1.0707$, the latter of which comes from the fully-turbulent experimental measurement for this quantity obtained by Ladson [150] at a Mach number of 0.15, where the suction and pressure side transition points were fixed at a distance of $0.05c$ from the leading edge, using No. 80-W grit carborundum strips.

Results

The turbo-RANS framework was applied to the airfoil example case, allowing a maximum of ten iterations in the optimization process. The first iteration was the default coefficient run (viz., RANS calculation using the default value for a_1). Subsequently, four additional Sobol-sampled iterations were completed. Therefore, five of these ten iterations correspond to iterations of the Bayesian optimization procedure. Each iteration corresponds to a RANS calculation with a given value for the a_1 model coefficient. The computational time was greatly reduced by using the converged fields from the default coefficient run as initial fields for the remaining RANS calculations involving the new proposals for the value of a_1. Fig. 6.5 displays the convergence history over the ten iterations.

In nine iterations, the optimization process attained a maximum value of the objective function, and the value of a_1 corresponding to this maximum (optimum) objective function value yielded a predicted value for the lift coefficient of $\tilde{c}_l = 1.0706$. This predicted value for the lift coefficient is in excellent conformance with the experimental value for the lift coefficient of $c_l = 1.0707$. The airfoil example calibration results are summarized in Table 6.1. While the default coefficient performs reasonably well at predicting the experimental value for the lift coefficient, this example demonstrated that turbo-RANS can efficiently calibrate

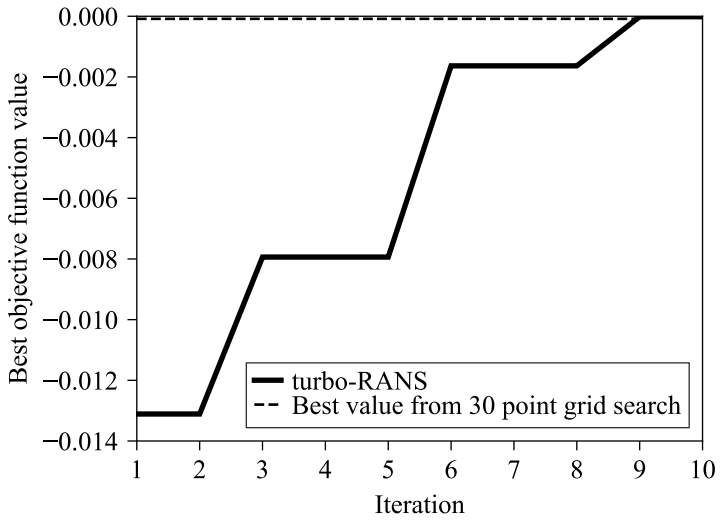


Figure 6.5: Convergence history for the optimization process for the airfoil example case.

Table 6.1: Summary of the airfoil lift coefficient calibration case.

Case	a_1	c_l
Experimental calibration data [150]	—	1.0707
k-ω SST, default coefficient	0.31	1.0847
k-ω SST, turbo-RANS optimized	0.28	1.0706

the model coefficient so that the predicted value of the lift coefficient provides an essentially perfect agreement with the corresponding experimental measurement of this coefficient.

6.3.2 Example 2: Periodic hills

Description

Predicting the evolution of separated, turbulent flow is a major challenge for RANS turbulence models [31]. Non-equilibrium effects, non-local effects, and history effects are frequently present for this type of flow. Unfortunately, commonly used RANS turbulence models such as the k-ω SST model often do not accurately predict these flow phenomena. Nevertheless, these models are frequently applied to separated turbulent flows since separation occurs in many industrial flows. A canonical example of separation is turbulent

flow over periodic hills. In this flow, a periodic domain is used so that the flow repeatedly separates from the smooth surface of one hill, reattaches, accelerates up a slope of the hill immediately downstream of it, and then separates again from the surface of this second hill. When we apply single-point RANS turbulence closure models such as the k-ω SST model to this type of flow, we can calibrate the model coefficients so that the effects of the poorly captured physics are at least mimicked. Otherwise, the turbulence model will be unable to accurately predict the general characteristics of the flow. In this example, we demonstrate how turbo-RANS can be used to calibrate turbulence model coefficients so that the predicted field quantities are in better conformance with the associated reference data.

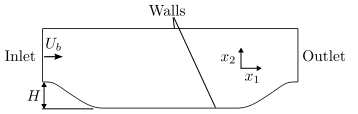
Computational setup

We calculate the steady-state, incompressible, turbulent flow over a series of periodic hills at a height-based Reynolds number of $Re_H = 5,600$. Direct numerical simulation data and OpenFOAM meshes are provided for this flow by Xiao et al. [44]. The software, solver, and schemes are identical to the airfoil example case (cf. Section 6.3.1). The k-ω SST turbulence model is also calibrated in this example [Eqs. (6.10)–(6.12)].

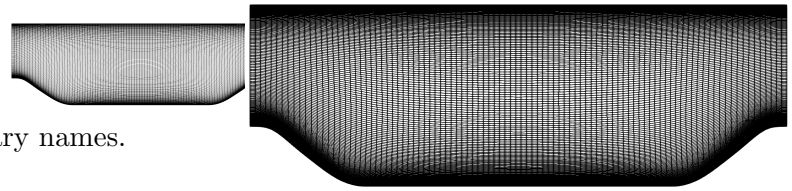
The computational domain and mesh for this case are taken from Xiao et al. [44]. The computational domain is two-dimensional (2D)—the three-dimensional (3D) DNS data has been averaged along the spanwise direction. The mesh consists of 14,751 hexahedral cells. Boundary conditions and mesh convergence for the RANS calculation, in this case, are discussed in McConkey et al. [79]. In short, periodic boundary conditions are imposed at the inlet and outlet planes of the computational domain. A no-slip wall boundary condition is imposed on the top and bottom walls of the domain. Along these no-slip walls, $0.01 \leq y^+ \leq 0.80$, with an average value for y^+ along the top and bottom walls of 0.71 and 0.20, respectively. This range of values for y^+ along the top and bottom walls implies that the RANS calculations conducted here are wall-resolved. Figure 6.6 shows the computational domain and mesh for the periodic hills case.

Optimization problem

For the case of the periodic hills, the GEDCP objective function is used, but flow field quantities are used in the misfit error term of the function. The purpose is to calibrate the a_1 and β^* model coefficients of the k-ω SST turbulence model so that the predictions of the magnitude of the velocity $\vec{U}$ and k fields are in better conformance with the available



(a) Domain and boundary names.



(b) Computational mesh.

Figure 6.6: Computational setup for the periodic hills example case.

DNS reference data. No integral parameters are included in the objective function for this example. In consequence, the objective function assumes the following form:

$$f_{\text{GEDCP}}(a_1, \beta^*) = -(E_{\text{F}})(1 + \tfrac{1}{2}p)\big|_{a_1,\beta^*} \ , \tag{6.18}$$

$$\begin{aligned} E_{\text{F}}(a_1, \beta^*) &= \sum_{\phi \in \{|\vec{U}|, k\}} \text{MAPE}(\phi) \\ &= \sum_{\phi \in \{|\vec{U}|, k\}} \frac{1}{N_\phi} \sum_{i=1}^{N_\phi} \left| \frac{\phi_i - \tilde{\phi}_i(a_1, \beta^*)}{\phi_i} \right| \ . \end{aligned} \tag{6.19}$$

This objective function encapsulates a tradeoff between reducing the MAPE in the prediction of $|\vec{U}|$ and k while retaining a minor preference for the default value of the model coefficients: namely, $a_1 = 0.31$ and $\beta^* = 0.09$. The parameter space corresponding to the model coefficients for the case of the periodic hills is $a_1 \in [0.24, 0.60]$ and $\beta^* \in [0.045, 0.14]$. The DNS reference data has been interpolated onto the RANS mesh so that $N_\phi = 14,751$. Given that every point in the RANS calculation has corresponding reference data, this example demonstrates how turbo-RANS can be used in an application involving highly dense reference data.

Results

A fixed computational budget of 30 iterations was prescribed for the optimization. The default values of the model coefficients were used for the first iteration. For the next nine iterations, Sobol sampling was used to generate the nine additional points. Subsequently, Bayesian optimization was used for the remaining 20 iterations.

Fig. 6.7 illustrates the convergence of the optimization process for the periodic hills example case. After approximately 20 iterations, the Bayesian optimization has found the

Table 6.2: The values of MAPE (lower is better) before and after turbo-RANS calibration of the model coefficients in the k-ω SST turbulence model for flow over periodic hills.

	Mean absolute percentage error (MAPE)	
Coefficients	k-ω SST, default coefficients	k-ω SST, turbo-RANS
$\{a_1, \beta^*\}$	$\{0.31, 0.09\}$	$\{0.34, 0.076\}$
$\lvert\vec{U}\rvert$	0.522	0.120
k	1.285	0.850

maximum (optimum) value (dashed line corresponding to the maximum value obtained using a 30×30 grid search in the parameter space). At the end of the optimization process, Bayesian optimization found model coefficients whose associated objective function value was slightly larger (better score) than the maximum (best score) obtained in the grid search—naturally, this optimal value occurs between the grid points used in the exhaustive grid search.

The predicted velocity and turbulent kinetic energy fields obtained using RANS calculations based on the default and optimized model coefficients were compared to the DNS reference data. Figs. 6.8 and 6.9 display the streamwise U and vertical V components of the velocity along several vertical sampling lines in the computational domain, respectively. The RANS predictions obtained using the turbo-RANS-optimized model coefficients agree very well with the DNS data, demonstrating a significant improvement in comparison to the corresponding RANS predictions obtained using the default values for the model coefficients. The default values of the model coefficients lead to predictions that overestimate the flow separation and, as a result, do not accurately predict the velocity after the occurrence of the separation. In the bulk flow over the separated flow region, the velocity predicted by the RANS models using the default values of the model coefficients is also high compared to the DNS reference data. After calibrating the a_1 and β^* model coefficients, the velocity here agrees well with the DNS reference data. Both the default and optimized RANS predictions exhibit discrepancies in the near-wall quantities directly along the separated bump. Here, the breakdown of assumptions made in modelling the RANS boundary layers is likely responsible for this discrepancy. Overall, the inclusion of the DNS U and V reference data through the GEDCP objective function [Eq. (6.18)] leads to a greatly improved conformance between the RANS predictions and the DNS reference data for this separated flow, as summarized in Table 6.2.

Fig. 6.10 compares vertical profiles of the turbulent kinetic energy k along various sampling lines in the computational domain. The k field was used in conjunction with the $|\vec{U}|$ field in the calibration of the model coefficients. While there is a clear improvement in

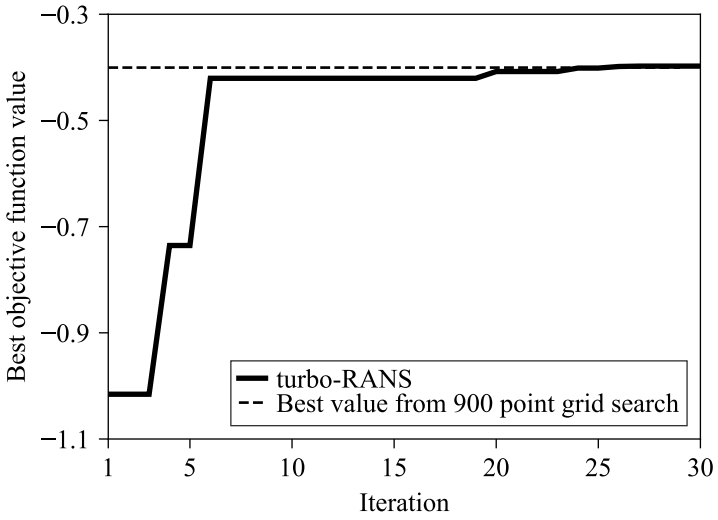


Figure 6.7: Convergence of the optimization process for the case of the periodic hills.

the k estimation in many regions of the flow, there is still a discrepancy between the DNS reference k and the RANS predicted k in the shear zone at the top of the separation region. Further accuracy improvements may be possible through the inclusion of additional model coefficients in the calibration procedure, such as the various coefficients that appear in the k-transport equation [Eq. (6.10)] and the ω-transport equation [Eq. (6.11)]. Nevertheless, the calibration procedure has generally resulted in an improved prediction of k for this separated flow, as summarized in Table 6.2.

While the reference data used in the GEDCP objective function was $|\vec{U}|$ and k, it was of interest to determine whether other flow quantities were also predicted more accurately. Even though the wall shear stress was not included as an explicit calibration target, Fig. 6.11 shows that the wall shear stress along the bottom wall is in much better conformance with the DNS data of the wall shear stress (not used in the calibration) after calibration of the model coefficients. In fact, the baseline k-ω SST turbulence model does not predict any reattachment of the flow along the bottom wall before the flow reaches the downstream hill. After calibration, the k-ω SST turbulence model now predicts reattachment, with the reattachment point in very good agreement with the DNS data. Fig. 6.11 demonstrates that calibrating turbulence model coefficients based on a specific set of flow quantities can lead to improvements in the predictive accuracy of other flow quantities not used in the calibration.

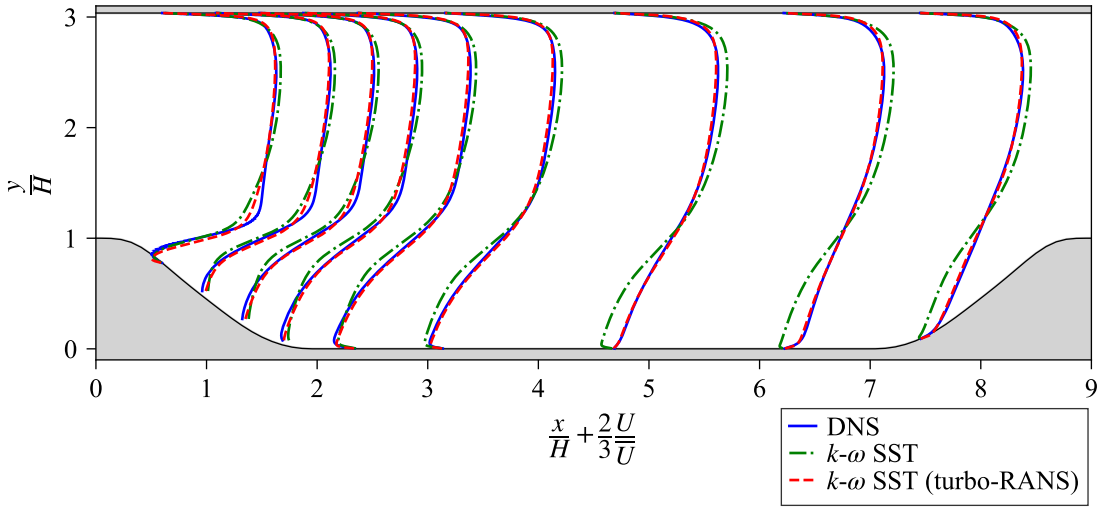


Figure 6.8: The streamwise (U) component of velocity along various vertical sampling lines for the case of the periodic hills.

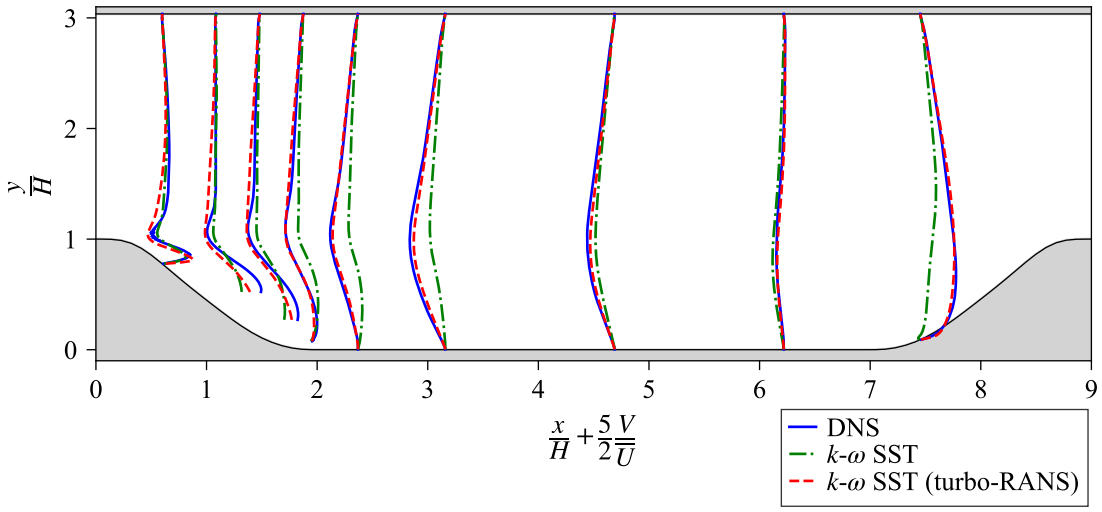


Figure 6.9: The vertical (V) component of velocity along various vertical sampling lines for the case of the periodic hills.

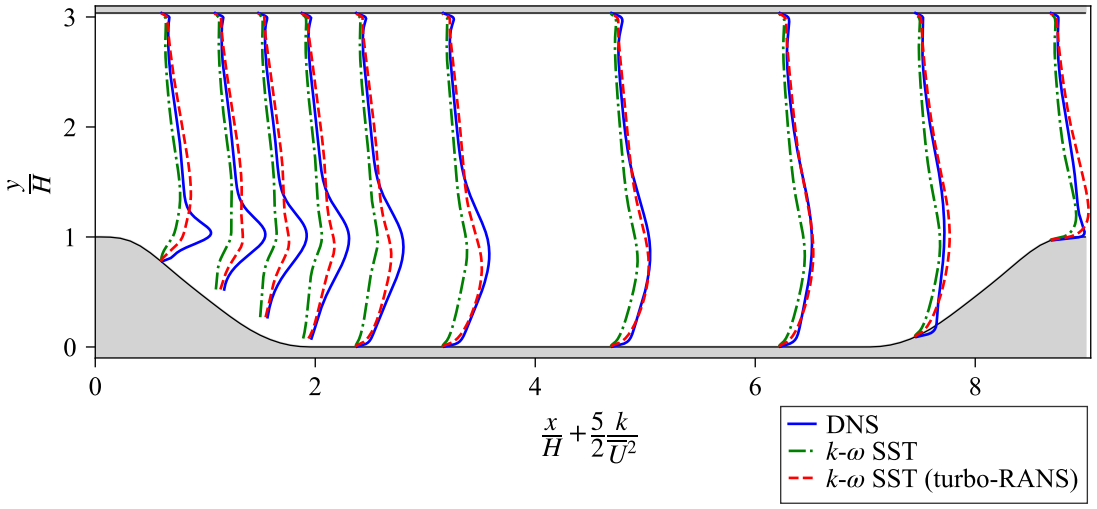


Figure 6.10: The turbulent kinetic energy k along various vertical sampling lines for the case of the periodic hills.

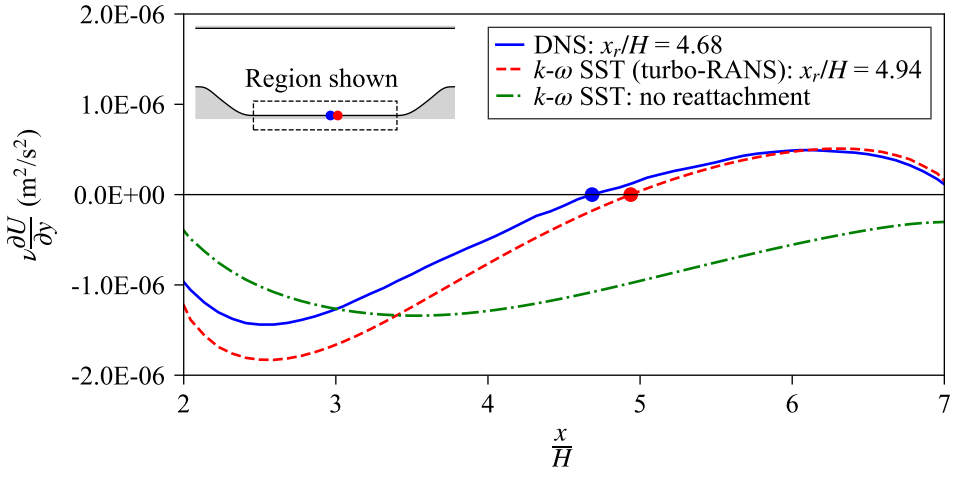


Figure 6.11: The wall shear stress along the bottom wall for the case of the periodic hills.

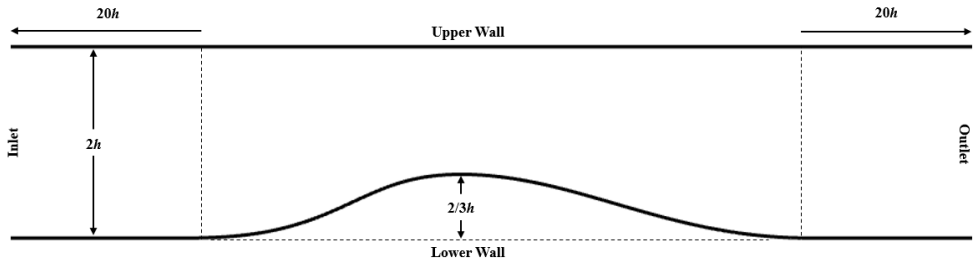


Figure 6.12: Computational domain for the converging-diverging channel.

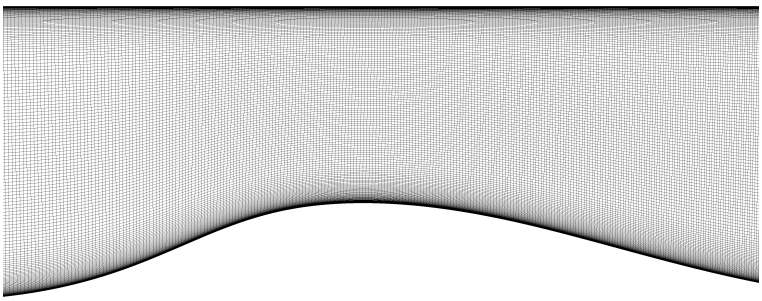


Figure 6.13: Mesh for the converging-diverging channel [79].

6.3.3 Example 3: Converging-diverging channel

GEKO turbulence model

The latest turbulence model that has been developed by Ansys, Inc. is the "generalized k-ω" (GEKO) two-equation model. The aim of this model is to provide flexibility for the user to fine-tune different coefficients for a RANS computation. GEKO has several parameters that can be fine-tuned to increase predictive accuracy. There are six parameters that can be tuned by the user. In this paper, the focus is on fine-tuning the model coefficients C_{SEP} and C_{NW} in order to produce more accurate predictions for flow in a converging-diverging channel. The exact equations by which these GEKO coefficients modify the underlying two-equation turbulence closure model are not provided by Ansys, Inc.; however, general guidance for their use is provided in the Ansys User Manual. The parameter C_{SEP} modifies the characteristics of a separation in the flow. According to the Ansys User Manual, "increasing C_{SEP} reduces eddy viscosity, which leads to more sensitivity to adverse pressure gradients for boundary layers and lower spreading rates for free shear flow" [128]. A change in the value of C_{SEP} has an influence on all classes of flows. The parameter C_{NW} modifies the near-wall characteristics of the flow that affect the nature of the wall boundary layer.

Table 6.3: Recommended range of values for the two GEKO coefficients C_{SEP} and C_{NW}.

Parameter	Minimum Value	Maximum Value	Default Value
C_{SEP}	0.7	2.5	1.75
C_{NW}	-2.0	2.0	0.50

The parameter C_{NW} generally has little or no effect on free shear flows. An increase in the value of C_{NW} leads to a larger wall shear stress. Here, we focus on calibrating global values for the GEKO model coefficients, although local values can also be accommodated through the application of user-defined functions (UDFs). These coefficients have a specified (recommended) range of values which is documented in the Ansys User Manual [128]. In accordance with this documentation, the model coefficients C_{SEP} and C_{NW} should take values in the range summarized in Table 6.3.

Computational setup

We calculate the steady-state, incompressible, turbulent flow through a converging-diverging channel using the GEKO turbulence closure model in a RANS framework. The DNS Reynolds number for this flow, based on $U_{\max}$ and h, is $Re_{U_{\max}} = 12,800$. Here, $U_{\max}$ is the maximum velocity in the fully-developed channel cross-section. The computational domain used for the 2D RANS calculation is shown in Fig. 6.12. The DNS reference data for this flow was obtained from Marquillie et al. [57, 151]. The mesh is adapted from McConkey et al. [79], and contains 610,720 grid cells as displayed in Fig. 6.13. Mesh independence with the selected mesh density near the converging section was demonstrated in McConkey et al. [79].

Boundary conditions and fluid properties for the RANS calculation are prescribed to match those of the DNS reference simulation. The inlet plane of the computational domain is placed at a distance of $20h$ upstream of the converging section, where h is the half-channel height. A Neumann pressure boundary condition is imposed at the inlet plane (viz., the gradient of the pressure in the streamwise direction is zero). The uniform streamwise velocity is prescribed at the inlet with a value of 0.845 m s^{-1}. This value was chosen to match the mass flow rate from the original RANS computation undertaken by McConkey et al. [79]. The inlet plane turbulent kinetic energy value is set to $k = 4.28421 \times 10^{-4}$ m^2 s^{-2}, and specific dissipation rate at the inlet plane is $\omega = 0.26993$ s^{-1}, in conformance with the computations conducted by McConkey et al. [79]. A fixed gauge pressure of zero Pa is imposed at the outlet plane of the computational domain, where all other flow quantities

are zero-gradient. The outlet plane is placed at a distance of $20h$ downstream of the bump. Lastly, a no-slip boundary condition was applied on all walls in the computational domain.

The fluid properties of the flow were specified as follows: density $\rho = 1$ kg m^{-3} and kinematic viscosity $\nu = 7.9365 \times 10^{-5}$ m^2 s^{-1}. The SIMPLE-consistent (SIMPLEC) algorithm was used for the pressure-velocity coupling. Convective terms in the momentum equation were discretized with a second-order upwind scheme. The convective terms in the k- and ω-transport equations were discretized with a first-order upwind scheme. The diffusion terms in the momentum and turbulence transport equations were discretized with a second-order central difference scheme.

Optimization problem

For the converging-diverging channel case, we use the GEDCP objective function to calibrate the model coefficients based on sparse reference data. Specifically, we calibrate C_{SEP} and C_{NW} based on a sparse set of measurements of the pressure coefficient along the bottom wall of the channel. The pressure coefficient is determined as follows:

$$C_p = \frac{P - P_0}{\frac{1}{2}\rho U_{\max}^2} \ . \tag{6.20}$$

Here, P is the pressure and P_0 is the pressure taken at a point downstream of the converging-diverging channel (approximately at a downstream distance of $12h$ from the start of the converging section).

No integral parameters are included in the objective function. The objective function used here is given by

$$f_{\text{GEDCP}}(C_{\text{SEP}}, C_{\text{NW}}) = -(E_{\text{F}})(1 + \tfrac{1}{2}p)\big|_{C_{\text{SEP}}, C_{\text{NW}}} \ , \tag{6.21}$$

$$\begin{aligned} E_{\text{F}}(C_{\text{SEP}}, C_{\text{NW}}) &= \text{MAPE}(C_p) \\ &= \frac{1}{N_{C_p}} \sum_{i=1}^{N_{C_p}} \left| \frac{C_{p,i} - \tilde{C}_{p,i}(C_{\text{SEP}}, C_{\text{NW}})}{C_{p,i}} \right| \ . \end{aligned} \tag{6.22}$$

Ten points along the bottom wall have been selected as sparse wall measurements ($N_{C_p} = 10$) used as the reference data in the calibration of the model coefficients. These points are indicated by the blue scatter points in Fig. 6.14. The misfit error term in the GEDCP objective function is the MAPE in C_p over the ten reference data points. For the upper and lower bounds of C_{SEP} and C_{NW}, we use the Ansys recommended limits of $0.75 \leq C_{\text{SEP}} \leq 2.5$ and $-2 \leq C_{\text{NW}} \leq 2$, as discussed previously in Section 6.3.3.

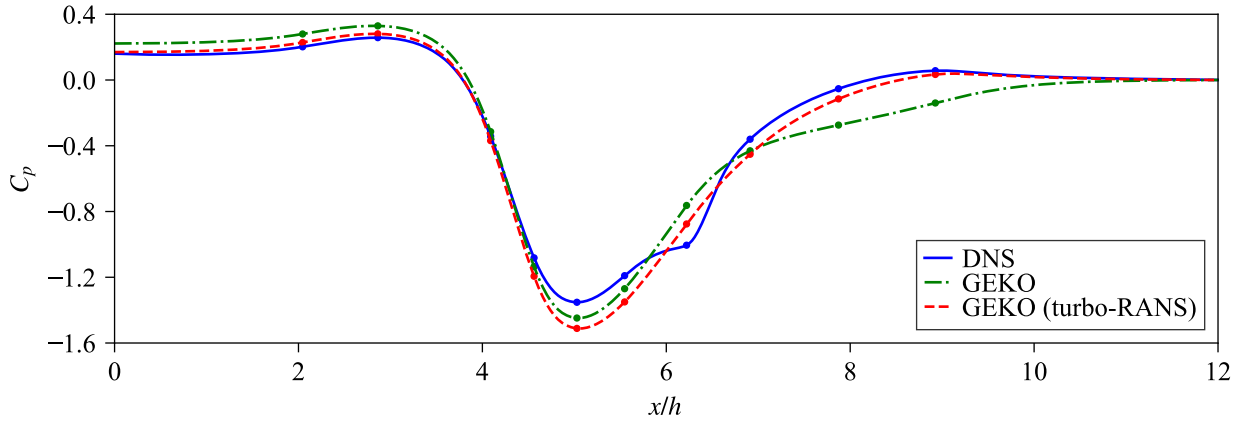


Figure 6.14: Pressure coefficient along the bottom wall for the converging-diverging channel case. The blue data points exhibit the locations of the measured pressure coefficients that form the reference data used in the calibration of the GEKO turbulence closure model.

Results

A Python script was created to modify and execute an Ansys Workbench journal file. A fixed computational budget of 30 iterations was used. The first iteration corresponds to the default values of the model coefficients. Subsequently, nine additional Sobol-sampled iterations are executed. The remaining 20 iterations use the Bayesian optimization schema. After 30 iterations, the recommended values for C_{SEP} and C_{NW} are those that maximize the objective function: $C_{\text{SEP}} = 0.7$, and $C_{\text{NW}} = 1.778$.

Fig. 6.14 compares the DNS data to the predictions obtained using the default and turbo-RANS-optimized GEKO models. After optimization, the streamwise profile of the pressure coefficient is in better conformance with the DNS data. In particular, in the diverging section of the channel, the turbo-RANS-optimized GEKO model better predicts the pressure recovery, in contrast to the default GEKO model which is seen to significantly under-predict the recovery. The optimized model exhibits a slightly greater error in its prediction of the minimum pressure at the top of the converging section (viz., at $x/h \approx 5$), but achieves greater predictive accuracy in every other region of the flow.

6.3.4 Discussion

The preceding three examples demonstrate the flexibility of the GEDCP objective function to work with various reference data types for the calibration of turbulence closure models. A mixture of integral, dense, and sparse reference data was used in these examples. The GEDCP objective function also permits the use of a mixture of reference data types for a given problem. For example, the airfoil case can be modified to include surface pressure coefficient reference data for use in calibration. As demonstrated by the case of the periodic hills, improving predictions of specific quantities of interest often results in improved predictions of other flow quantities (not used as reference data for the calibration). Moreover, when the GEDCP objective function is used as a calibration target in turbo-RANS, a computationally efficient optimization of turbulence model coefficients is possible. Sobol sampling, which occurred until iteration 5 for the airfoil case and iteration 10 for the periodic hills and converging-diverging channel cases, proved to be a highly effective sampling methodology. During the Sobol sampling period, a point close to the optimal value was almost always identified, and therefore the Bayesian optimization algorithm was well-positioned to determine whether to exploit this region or to explore the more sparsely sampled (albeit more uncertain) regions in the parameter space. For more complex flows, there is no guarantee that the Sobol sampling algorithm will happen to provide a point near the optimal value, and exploration within the Bayesian optimization loop may become more important. Section 6.4 will investigate and provide recommendations for the number of points to be sampled in the initialization phase of Bayesian optimization.

6.4 Hyperparameter optimization

The objective of hyperparameter optimization was to determine the optimal settings for Bayesian optimization in the turbo-RANS framework. In order to recommend suitable hyperparameters for typical applications involving the calibration of model coefficients of RANS turbulence closure models, two cases were investigated in this context: namely, flow over an airfoil; and, flow over periodic hills. The objective functions for the airfoil and periodic hills cases are given in the main article.

Several optimization runs were completed for these cases with various hyperparameter settings to determine which settings resulted in a solution using the smallest computational effort (i.e., the strategy that arrived at the solution using the smallest number of evaluations of the objective function). Ultimately, the selected hyperparameters allow the optimal RANS model coefficients to be obtained with the lowest computational effort compared to

other settings. The following hyperparameters were optimized in this study.

- **Utility function**. Three utility functions are included in the bayesian-optimization [148] Python library: upper confidence bound (UCB), expected improvement (EI), and probability of improvement (POI). See the main article for further explanation of these utility functions.
- **Exploration/exploitation tradeoff parameter**. The UCB function is parameterized by a tradeoff parameter κ, and both the EI and POI functions are parameterized by a tradeoff parameter ξ. The default value for the tradeoff parameter in bayesian-optimization for the UCB utility function is $\kappa = 2.576$. For both the EI and POI utility functions, the default value for the tradeoff parameter in bayesian-optimization is $\xi = 0$.
- **Gaussian process kernel parameters**. The Matern kernel is used in the underlying Gaussian process regression in turbo-RANS. Important parameters for this kernel are the characteristic length scale l and the parameter ν (see the main article for more details).
- **Number of initially sampled points**. Before calling the Bayesian optimization, it is important to condition the underlying Gaussian process regression surrogate model of the objective function on some initial points. We use Sobol sampling to randomly generate these points, which provide a uniform sampling of the parameter space [149].

The following procedure was used to select optimal hyperparameters:

1. Fix $l = 0.1$, $\nu = 5/2$, $n_s = 5$ for the airfoil case, and $n_s = 10$ for the periodic hills case.
2. Perform a grid search over both the utility function type and corresponding exploration/exploitation tradeoff parameter.
3. Select and fix the utility function and exploration/exploitation tradeoff parameter.
4. Perform a grid search over l and ν.
5. Select and fix values for l and ν.
6. Perform a grid search over n_s.

Table 6.4: Hyperparameter search space.

Hyperparameter	Values tested	Notes
Utility function	UCB, EI, POI	See the main article for discussion.
κ (UCB)	$0.5 \leq \kappa \leq 6$	Linear spaced interval, increment of $\Delta\kappa = 0.1$ (56 values)
ξ (EI, POI)	$0 \leq \xi \leq 0.5$	Logarithmic spacing, with 56 values
ν	$\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \infty$	See the main article for discussion.
l	$0.01 \leq l \leq 1.0$	14 values total, more concentrated around 0.1, then every 0.1 to 1.0
n_s	$3 \leq n_s \leq 15$	13 values total, every integer

The grid search space for each parameter is summarized in Table 6.4. In total, 237 hyperparameter combinations were tested. Given the random nature of the Gaussian process regression that underpins Bayesian optimization, the core turbo-RANS algorithm is not deterministic. However, the algorithm can be made deterministic in practice by setting the random seed in the code. To better select optimal hyperparameters, 30 random turbo-RANS optimization runs were completed for each hyperparameter setting, for a total of 7,110 optimization runs. Following this, the mean and standard deviation of the number of runs to reach convergence were determined. The mean number of runs, $\overline{n}$, is determined in accordance to

$$\overline{n} = \frac{1}{N} \sum_{i=1}^{N} n_i \ , \tag{6.23}$$

where N is the total number of turbo-RANS runs, i indexes each turbo-RANS run, and n_i is the number of iterations to reach convergence in the i-th run. The standard deviation, s, of the number of runs required to achieve convergence is given by

$$s = \left(\frac{1}{N} \sum_{i=1}^{N} \left(n_i - \overline{n} \right)^2 \right)^{1/2} \ . \tag{6.24}$$

The purpose of applying turbo-RANS is to tune turbulence model coefficients based on a case-specific objective function. To accelerate the hyperparameter tuning, the objective functions for the airfoil case and periodic hills case were evaluated *a priori* over a series of a_1 values for the airfoil case, and a grid of values of (a_1, β^*) for the periodic hills case. These *a priori* simulations permitted faster hyperparameter optimization since the value of f_{GEDCP} given a model coefficient suggestion can be determined by interpolation. This *a priori* evaluation of f_{GEDCP} on a grid of values of the model coefficients would not be used in practice with turbo-RANS. In a typical application, a new RANS prediction would

need to be computed with each model coefficient suggestion. In other words, since the hyperparameter optimization procedure required thousands of evaluations of f_{GEDCP}, it was more computationally efficient to "simulate a simulation" by interpolating between a set of previously computed RANS calculations. These *a priori* RANS calculations also allow visualization of the objective function. The objective function for the airfoil case is shown in the main article for a set of 50 RANS calculations for values of a_1 that span the parameter space. Similarly, the objective function for the periodic hills case is shown in the main article for a set of 900 RANS calculations for combinations of values of the two model coefficients (a_1, β^*) that span the parameter space.

Convergence criteria in turbo-RANS are determined by the user. For the hyperparameter optimization, the optimal coefficients and values of f_{GEDCP} were known because a large number of *a priori* RANS calculations were completed on a grid. For the airfoil case, the optimization procedure was considered convergent when the suggested value of the a_1 coefficient was within 0.5% of the optimal value of this coefficient. For the case of the periodic hills, the optimization procedure was considered convergent when the suggested values for the coefficients (a_1, β^*) resulted in an objective function f_{GEDCP} value that was within 5% of the optimum objective function value. The choice of convergence criteria for hyperparameter optimization is a special case since it relies on *a priori* knowledge of the objective function optimum. In practice, turbo-RANS convergence criteria can be set based on the change in the best objective function or best coefficients over a given window of iterations. Given the expensive nature of RANS calculations, we envision that many turbo-RANS optimization procedures will be run with a prescribed computational budget instead of a numerical convergence criteria (e.g., a 40-simulation optimization run can be prescribed *a priori*). The Bayesian optimization algorithm produces a computationally efficient result for either of these criteria.

6.4.1 Utility function

Fig. 6.15 shows the results of the utility function search for the airfoil case. In terms of purely minimizing the number of evaluations of the objective function, the UCB, EI, and POI utility functions all give roughly the same minimum mean number of evaluations of $\overline{n} = 7$. For the UCB and EI utility functions, the performance is stable over a wide range of values of κ and ξ. However, past a critical value of ξ, the EI utility function becomes too explorative, and the number of objective function evaluations for convergence increases sharply. The UCB utility function achieves stable performance for all values of κ, with a minor trend of an increasing $\overline{n}$ with an increasing value of κ. The POI utility function has an optimum ξ value. Below this optimum value, the performance of the POI

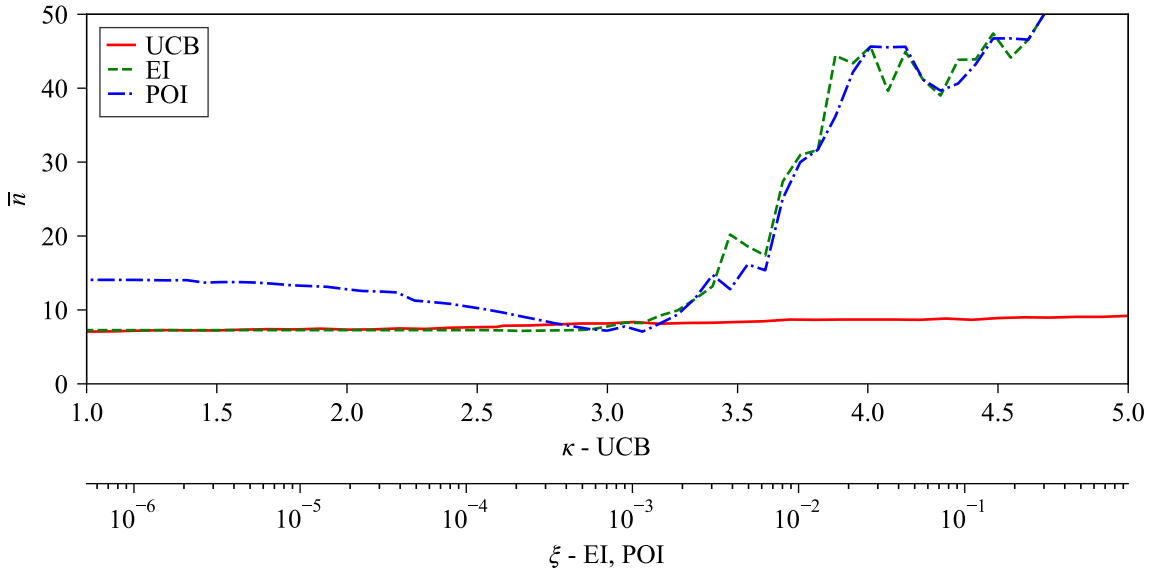
utility function is worse than that of the UCB or EI utility functions. Above this optimum value, the POI function becomes highly explorative, performing similarly to the EI utility function. Furthermore, the standard deviation of the 30 random runs is consistent for the UCB utility function, indicating that all the runs achieve convergence in a similar number of iterations. When the EI and POI utility functions become overly explorative, the standard deviation is seen to increase sharply.

Fig. 6.16 shows the hyperparameter optimization results for the periodic hills case. The general trends are similar to the airfoil case: the UCB utility function performs consistently, and the EI and POI functions have a critical ξ value beyond which the Bayesian optimization process becomes overly explorative. Similar trends to the airfoil case are also seen for the standard deviation. Given the two-coefficient optimization, a larger coefficient search space, and a more complex objective function, it is not surprising that the periodic hills case requires more iterations to find the optimal point than the airfoil case, with the minimum mean number of objective function evaluations being 20.

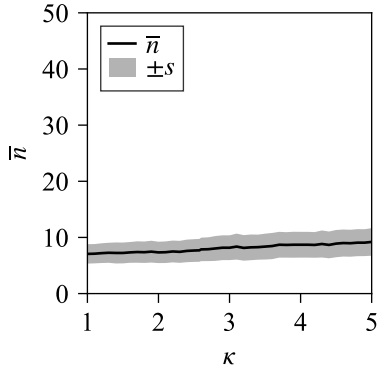
Comparing Figs. 6.15 and 6.16, we see that, when optimized, all three utility functions perform equally well. However, a major disadvantage of the POI utility function is the need to determine the optimum ξ, which is problem-specific: it is $\xi \approx 10^{-3}$ for the airfoil case and $\xi \approx 10^{-2}$ for the periodic hills case. Given their similar performance and low standard deviation, the EI and UCB utility functions are both acceptable choices for use in the acquistion of a new point in the Bayesian optimization process. However, the EI utility function will rapidly become overly explorative past a problem-dependent threshold. Therefore, for a general turbo-RANS use case, we recommend the application of the UCB utility function.

The optimal exploration/exploitation tradeoff parameter for the GEDCP objective function generally favours exploitation. However, when the κ or ξ parameters were too small (overly promoting exploitation), we observed that the Bayesian optimization process became "stuck" on a perceived local maximum. For the UCB utility function, the Bayesian optimization process can occasionally become stuck for values of $\kappa \lessapprox 1$. Indeed, these runs would not converge after hundreds of iterations, and therefore they were removed from the mean and standard deviation calculations. It is noted that these "stuck" runs are indicative of overly exploitative behavior. Therefore, we recommend a value of $\kappa = 2$, so that the optimization is slightly more (but not overly) exploitative than the default value for κ recommended in bayesian-optimization [148].

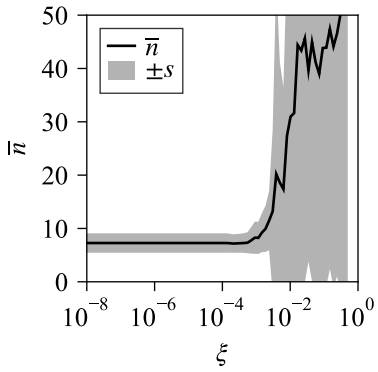
A strategy that can be used to ensure that the Bayesian optimization process does not become "stuck" at a particular query point selected by the utility function is to simply add some "noise" (jitter) to a selected query point. This will guarantee that the selected query



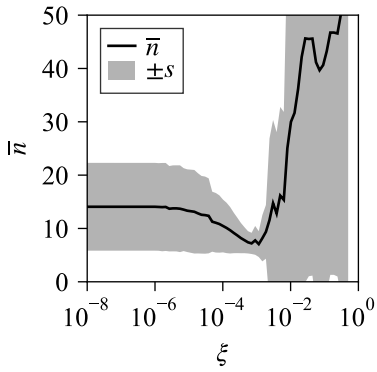
(a) Comparison of the mean value $\overline{n}$ of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter for the UCB, EI, and POI utility functions.



(b) Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter κ for the UCB utility function.

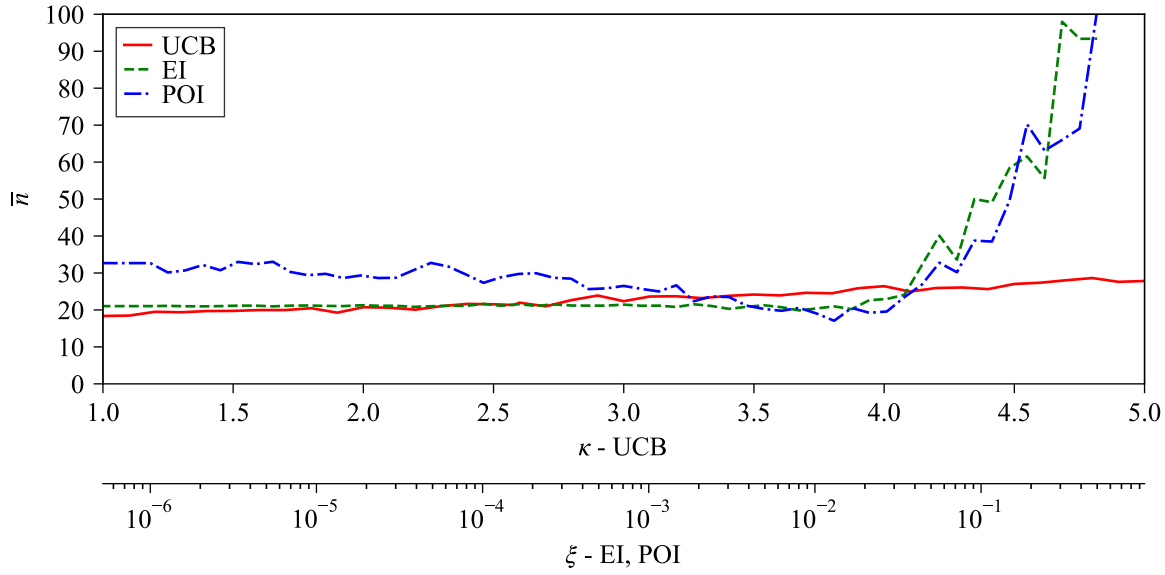


(c) Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter ξ for the EI utility function.

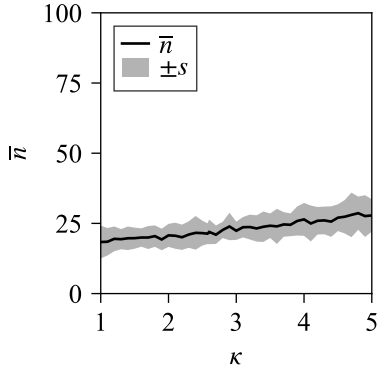


(d) Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter ξ for the POI utility function.

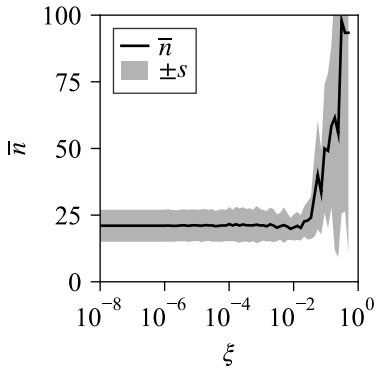
Figure 6.15: Utility function hyperparameter search results for the airfoil case.



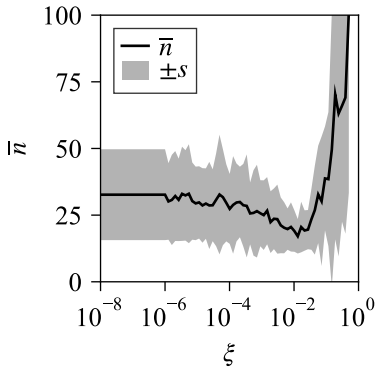
(a) Comparison of the mean value $\overline{n}$ of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter for the UCB, EI, and POI utility functions.



(b) Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter κ for the UCB utility function.



(c) Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter ξ for the EI utility function.



(d) Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence as a function of the exploration/exploitation tradeoff parameter ξ for the POI utility function.

Figure 6.16: Utility function hyperparameter search results for the periodic hills case.

point in the current iteration of the Bayesian optimization is different from a previously selected query point. We have not investigated this option in the current study, but it is a possible extension that will be considered in some future work on turbo-RANS.

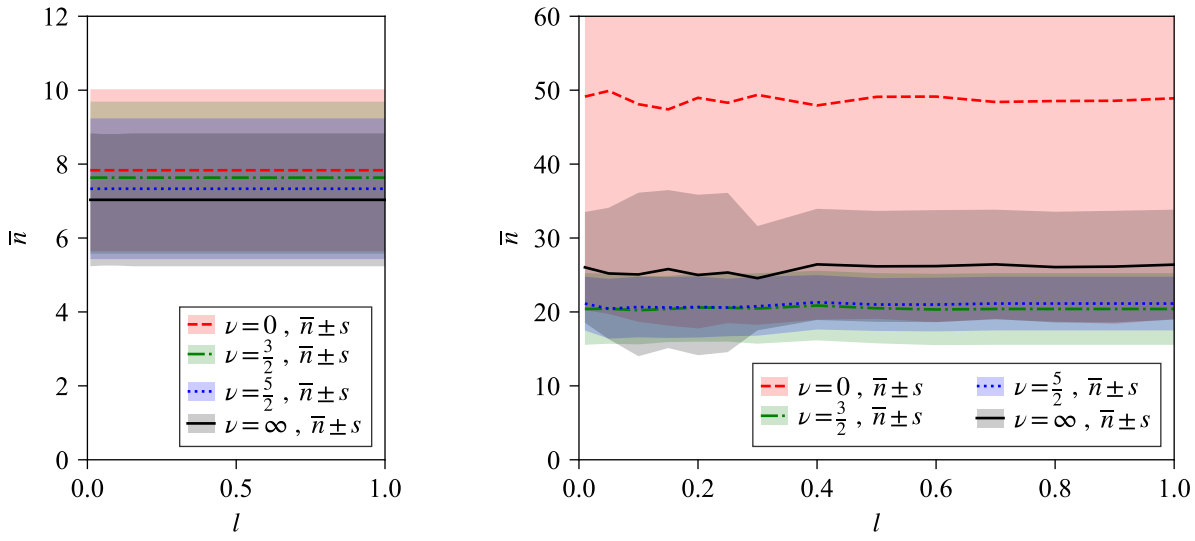
6.4.2 Matern kernel parameters

After selecting the UCB utility function with $\kappa = 2.0$, the Matern kernel parameters were optimized. A grid search was completed for values of ν and l in the following ranges: $\nu \in [0, \infty)$ and $l \in (0, 1]$. Fig. 6.17 shows the results of this grid search. Neither the airfoil nor the periodic hills cases show sensitivity to the length scale value. Performance in the airfoil case is not very sensitive to the ν value, with all values of ν achieving convergence of between 7 and 8 objective function evaluations on average. For the periodic hills case, it becomes clear that $\nu = 0$ does not perform well, whereas $\nu = 3/2$ and $5/2$ achieve similar performance. We recommend setting $\nu = 5/2$, which is the default value recommended for the scikit-learn Matern kernel [85]. This kernel returns a twice-differentiable function. For the length scale, we recommend a value of $l = 0.1$. This length scale value specifies the initial guess for conditioning the Gaussian process, and the final length scale inside the kernel is optimized internally in scikit-learn. Therefore, only a reasonable initial guess for this hyperparameter needs to be specified. For optimizing the Gaussian process parameters, using a sufficient number of sample points initially before the Bayesian optimization was much more important than the choice of the initial value for the length scale.

6.4.3 Number of initial samples

After selecting $\nu = 5/2$ and $l = 0.1$, the last hyperparameter to be optimized was the number of initial samples, n_s. As discussed in the main article, a certain number of points within the parameter space are obtained using Sobol sampling before calling the Bayesian optimization algorithm. The purpose of these points is to provide sufficient information to properly perform Gaussian process regression on the objective function. When n_s is too small, errant behavior of the Gaussian process can occur. This errant behavior results in subsequent slow optimization, with the Bayesian optimization process often getting immediately stuck. When n_s is too large, then the overall algorithm becomes computationally inefficient by overly sampling the parameter space prior to entering the efficient Bayesian optimization loop.

Fig. 6.18 shows the results of optimizing n_s for the airfoil and periodic hills cases. For the airfoil case, it is optimal to reduce the number of sample points as much as possible.



(a) Airfoil case.

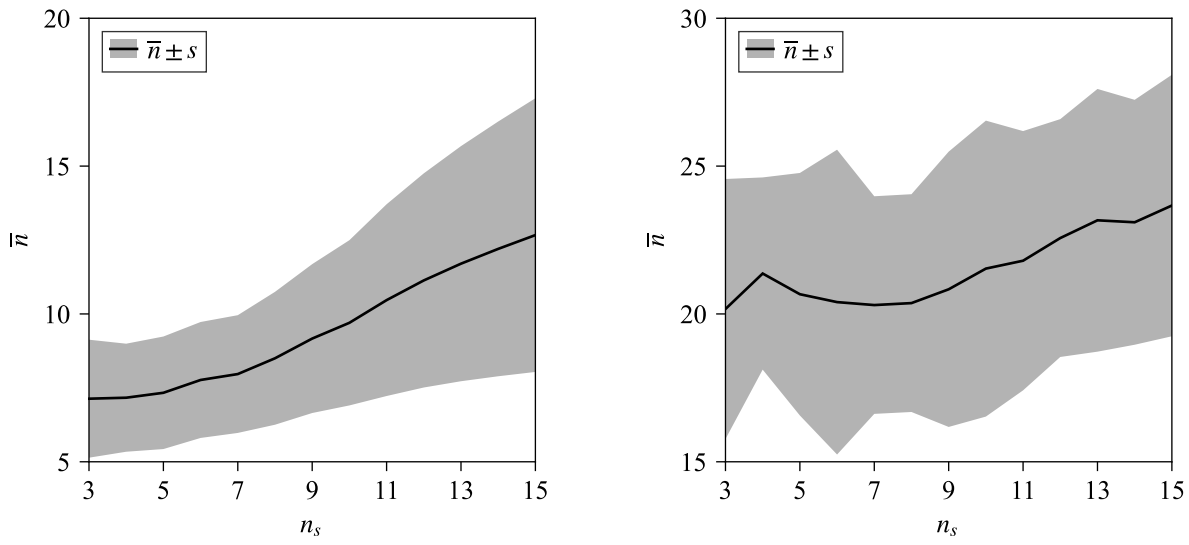
(b) Periodic hills case.

Figure 6.17: Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence for the Matern kernel for different values of the hyperparameter ν.

Below $n_s = 3$, the previously mentioned erratic behavior can occur, as the Gaussian process regression of the objective function is fit on too few points. For the periodic hills case, there is a local minimum at $n_s \approx 8$. These results show that the number of initial samples should be increased as the number of optimized parameters increases. Given our experience concerning the errant behavior of the Gaussian process regression with too few initial samples, a conservative default has been set in the turbo-RANS algorithm of $n_s = 10$. If the end-user wishes to increase performance further, we recommend first reducing the value of n_s, but generally keeping $n_s > 5$ for maintaining the stability of the Gaussian process regression.

6.4.4 Recommendations

The recommended hyperparameters for turbo-RANS are summarized in Table 6.5. The optimal hyperparameter settings likely change on a case-by-case basis, since the highly customizable GEDCP objective function varies depending on the type of reference data used in the calibration and on the default coefficient preference. Nevertheless, the two relatively different objective functions [Eqs. (6.16) and (6.18)] have similar optimal hyperparameters.



(a) Airfoil case.

(b) Periodic hills case.

Figure 6.18: Mean $\overline{n}$ and standard deviation s of the number of runs required to achieve convergence for various numbers n_s of initial sample points.

Moreover, the UCB utility function is relatively insensitive to κ for these typical objective functions. As the objective function increases in complexity and the number of optimized coefficients increases, the optimal value of κ changes slightly but remains close to the recommended default value of $\kappa = 2.576$ in the bayesian-optimization library. This value of κ represents a reasonable exploration and exploitation tradeoff for a general case, and the optimal turbo-RANS setting of $\kappa = 2$ indicates that a slight preference for exploitation is favourable for turbo-RANS. For different flows, settings in the GEDCP function can depend on the number of coefficients in the problem, so the core optimization problem can vary greatly. While we demonstrated here that the recommended setting (UCB, $\kappa = 2$) performs well for different problems, it may not perform well for all cases. The Bayesian optimization hyperparameters should be re-tuned if the convergence speed is unsatisfactory for a different objective function.

Table 6.5: Recommended hyperparameter settings.

Hyperparameter	Recommendation	Notes
Utility function	UCB	Expected improvement (EI) also performs well, but use $\xi \lessapprox 10^{-3}$.
κ	2.0	Minor efficiency gains can be made by lowering κ, but at the risk of becoming stuck.
ν	$\frac{5}{2}$	$\nu = \frac{3}{2}$ also performs well.
l	0.1	Since this is just the initial guess for the Gaussian process optimizer, the choice for l is often not significant.
n_s	10	For very simple optimization problems (e.g. single variable, under 10 iterations total), n_s can be lowered, at the risk of becoming stuck.

6.5 Conclusion

This work focused on an important and widespread optimization problem in engineering flow simulation: calibrating turbulence model coefficients. The major contributions of this work are to propose a framework for Bayesian optimization of turbulence model coefficients (turbo-RANS), propose a general objective function, demonstrate the predictive accuracy improvements that can be achieved, and recommend hyperparameters for future users of turbo-RANS. We presented the turbo-RANS framework, demonstrated how RANS models can be calibrated using various types of reference data to produce more accurate predictive results, and performed a systematic hyperparameter tuning for the Bayesian optimization process.

Even with perfectly calibrated model coefficients, there is an upper limit to the predictive accuracy that can be achieved with RANS. Errors associated with unavoidable assumptions and approximations (model errors) in RANS can severely degrade the accuracy of these predictions. For example, the assumptions of locality can be violated for many real-world flows [3]. Duraisamy et al. [17] list four main sources of error in RANS: errors arising from ensemble averaging (L1), errors in the functional representation of the Reynolds stress (L2), errors in the model functional form (L3), and errors in the model coefficients (L4). It is possible that for certain classes of flows, the calibration of model coefficients (which reduces the L4 error) can help offset some L2 and L3 errors. However, this effect is difficult to quantify. Nevertheless, even with perfectly calibrated model

coefficients, turbo-RANS is only as good as the "best RANS" solution for a particular problem, which is often limited by unrecoverable L1 and L2 errors. Although turbo-RANS can potentially improve the accuracy of a RANS prediction to some upper bound limit in their predictive accuracy for a given flow, this upper bound limitation in the accuracy will always exist due to implicit modelling assumptions and approximations used in RANS, which may not hold true for many industrially relevant (complex) flows.

Within the wider context of modern machine learning techniques for RANS, we believe that an important future research direction is comparing compare coefficient calibration to highly sophisticated data-driven algebraic Reynolds stress closure models. Although the process of calibrating model coefficients is comparatively simpler than training a non-linear eddy viscosity model, the resulting performance in the present work appears to compete with these machine learning models. Previous results for the periodic hills case by Wu et a., [33], McConkey et al. [93] and Kaandorp and Dwight [35] show similar predictive accuracy to the presently calibrated linear eddy viscosity models. We demonstrated here that simply by tuning model coefficients, there is significant performance to be leveraged from the physics represented in the RANS turbulence closure models. Even further performance can be leveraged from the Bayesian optimization of zonal coefficient values. Currently, Bayesian inference has been applied to infer local turbulence model coefficients, but Bayesian optimization applied to local or zonal model coefficients has not been explored to the authors' knowledge.

Despite years of optimization methods being applied to turbulence model calibration, and despite the significant accuracy improvements that are possible, these methods are not yet widely used in industrial settings. We believe this may be due to the complicated implementation and expert background required to perform Bayesian calibration. Therefore, by designing an easy-to-use software package, we hope to help spur industrial interest in this technique. All data and code from the present work can be found in the turbo-RANS GitHub repository [30]. We also include the 900 periodic hills RANS calculations used for the present hyperparameter tuning investigation, as broader investigations concerned with uncertainty quantification may find this data set useful.

Future work will include model coefficient calibration for more challenging flows. The presently proposed GEDCP objective function can be immediately applied to quantify error in an unsteady quantity of interest, such as a Strouhal number, time-averaged force coefficient, or root-mean-square (RMS) force coefficient. We also aim to explore optimal turbo-RANS hyperparameter tuning for more complex 3D flows and investigate calibrating a RANS turbulence model on a mixture of experimental and numerical reference data. turbo-RANS can also be extended to calibrate coefficients within a Reynolds-stress transport model, or other turbulence closure relations. Ultimately, the algorithm and GEDCP

objective function proposed here can be applied to calibrate any coefficient of interest in a RANS model.

Currently, CFD best practices such as a mesh independence study, careful numerical scheme selection, designing meshes appropriate for wall treatment, verification, and validation are missing an important step: coefficient calibration. Previously, this would have been a largely manual activity. However, turbo-RANS automates this activity in a highly computationally efficient manner. We envision that, along with validation, calibration can become a more frequently used practice in RANS. With the more widespread use of closure coefficient calibration, future RANS practitioners can potentially generate a coefficient database, and make an educated selection of coefficients to use in their specific application based on the nature of the physics present in their flow. To some extent, this is the idea behind the GEKO turbulence closure model [128]. However, the GEKO model still requires user tuning of the model coefficients. With sufficient reporting and the use of calibrated RANS models, a recommendation engine (e.g., a large language model) can be embedded into CFD software of the future. This model can be trained to recommend turbulence model coefficients given a description of the flow physics present and quantities of interest. Such a model relies on more reporting of calibrated closure coefficients, which we aim to make easier with turbo-RANS.

Chapter 7

Conclusion

This thesis presented two distinct techniques for improving the accuracy of RANS simulations: using machine learning to augment closure models via data-driven anisotropy mappings (Chapters 2–5), and using Bayesian optimization for calibrating turbulence model coefficients (Chapter 6).

The augmented closure modelling stream of this thesis began by generating an open-source dataset suitable for training data-driven anisotropy mappings (Chapter 2). This dataset is the first dataset to contain collated RANS and high-fidelity data for a variety of turbulent flows. It aims to provide convenience for future modellers and enable standardized benchmarking of new model architectures. This dataset was used in all of the other investigative chapters of this thesis, highlighting its utility for data-driven turbulence modelling. Chapter 3 proposed a novel closure framework. The results demonstrated in Chapter 3 indicate that the proposed closure relationship is well-conditioned after injection into the momentum equations, a crucial consideration for data-driven turbulence models. Additionally, SHAP values were applied to explain the predictions from a data-driven anisotropy mapping for the first time.

Chapters 4 and 5 investigated data-driven anisotropy mappings further. In Chapter 4, it was shown that these mappings do not generalize well to new flow types. This conclusion holds true for all three model types (random forest, XGBoost, and deep neural networks), and for all training flows. Nevertheless, it was shown that these models are able to generalize reasonably well to similar flows to those in the training dataset. This outcome is likely due to the "no free lunch" theorem, which has plagued turbulence modellers for decades: the more sensitized a RANS model becomes to a given flow type, the less it is able to generalize to a new flow. On an optimistic note, there are many applications

where a flow-specialized turbulence model is desired. Chapter 4 shows that the techniques along this stream in the thesis are ideal for users who wish to train a specialized machine learning model for a particular flow. Chapter 5 concludes this stream by proposing "realizability-informed" learning, which was demonstrated to improve the realizability of predicted anisotropy tensors when the model is generalizing to a new flow configuration. Additionally, the framework used in Chapter 5 has the stability and conditioning benefits of eddy viscosity-based frameworks, while retaining the equivariance enforced by using a tensor basis neural network. This "best of both worlds" approach is highly useful for the broader data-driven turbulence modelling community.

Though it is a stable and well-conditioned framework, the framework proposed in Chapter 3 has the disadvantages that the apNN is not equivariant, and the optimal eddy viscosity is a difficult quantity to predict. Additionally, the closure relationship used in Chapter 4 is likely ill-conditioned, since it does not contain information about the high-fidelity mean velocity field [76]. This closure relationship was selected because it is model agnostic and therefore enables direct comparison between machine learning model types. The framework, objective function, and model architectures in Chapter 4 are therefore ultimately recommended for future data-driven turbulence modelling.

The turbo-RANS framework presented in Chapter 6 is recommended when dense high-fidelity data is not available. Since turbo-RANS is highly data-flexible, it can be even be used in cases where the only reference data available is a single integral parameter, such as a target lift coefficient. Though turbo-RANS is designed to be efficient and data-flexible, it is highly constrained by the expressivity of the original turbulence model. Simply put, turbo-RANS is capable of efficiently finding the upper performance limit of a given turbulence model. However, the upper performance limit of a RANS turbulence model may still be relatively inaccurate. For example, the optimized GEKO model in Chapter 6 still contains error compared to the DNS data for flow in a converging-diverging channel. This upper performance limit is also highly dependent on the selected objective function, which is a user choice. The user defines how the performance of the turbulence model will be assessed, and turbo-RANS calibrates the coefficients accordingly. Nevertheless, turbo-RANS is a useful, straightforward, and computationally efficient tool for tuning RANS turbulence models. If the end user is not satisfied with the calibrated RANS model performance, then a richer data source may be required to facilitate the training of a data-driven anisotropy mapping.

7.1 Future work

For the data-driven anisotropy mapping stream, the following future research directions are recommended:

1. An investigation rooted in information theory to determine why these mappings struggle to generalize. For example, are the distributions of input features and targets highly distinct between different flows? This question remains a controversial topic in machine learning augmented closure modelling. In regular machine learning, we do not expect a model that predicts dog breeds to also predict cat breeds. Is a paradigm shift required in data-driven turbulence modelling, or do we just need more data?

2. Extending the augmented closure framework to unsteady flows. The eddy viscosity closure relationship does not contain any temporal dependence, and therefore there is no reason preventing a data-driven anisotropy mapping being used as a closure relationship within an unsteady simulation. However, there is a high potential for instability—any erroneous prediction could lead to a feedback loop, which causes the simulation to either be highly inaccurate or simply crash. Further, how should training data be gathered for such a mapping? Currently, training data is gathered by using a steady RANS simulation, and collocating time-averaged high-fidelity data for a statistically steady flow. This provides an equivalent "time-step", so that collocation is only necessary in space. However, for unsteady training data, this also requires temporal collocation of the RANS flow field, and an equivalent ensemble-averaged high-fidelity simulation. If both temporal and spatial collocation, it presents the opportunity for additional input features which involve time-derivatives of the current invariants and heuristic scalars. In other words, temporal effects could be captured in the closure relationship, significantly increasing the expressivity of the learned mapping.

3. Bayesian tensor basis neural networks for closure modelling. The closure framework, model architecture, and realizability-informed loss function proposed in Chapter 5 could be used to train a Bayesian neural networks, so that uncertainties in the predicted anisotropy tensor could also be predicted. These uncertainties in the anisotropy tensor can be propagated through the RANS model to provide uncertainties in quantities of interest predicted by the framework.

4. Kolmogorov-Arnold Networks (KANs) for data-driven anisotropy mappings. Recently, this model architecture has drawn significant interest as an alternative to

classical multilayer perceptrons (MLP) [152]. With the interesting ability to learn activation functions, the merits of using a tensor basis Kolmogorov-Arnold networks (TBKAN) could be investigated using the dataset from Chapter 2, along with the model architecture and closure framework in Chapter 5.

The following future research directions are recommended for turbo-RANS:

1. Using turbo-RANS to calibrate coefficients based on an unsteady target. For example, turbo-RANS could be used to calibrate an unsteady simulation in order to match a reference Strouhal number.

2. Demonstrating that turbo-RANS can calibrate Reynolds stress transport models (RSTM) and LES models (turbo-RSTM and turbo-LES). Closure models in these techniques also contain calibration coefficients. Though it was demonstrated for RANS calibration in Chapter 6, the core optimization algorithm can be used as an efficient methodology for tuning RSTM and LES calibration coefficients.

3. Parallelization of the Bayesian optimization procedure in turbo-RANS. Currently, it is assumed that each optimization iteration is completed sequentially. While the simulation during the probe subroutine can be parallelized in the usual manner for CFD simulations, the overall search procedure is still a series of CFD simulations. Parallelized Bayesian optimization algorithms need to be investigated for this purpose.

4. Dynamic adjustment of the optimization bounds. Currently, the optimization domain is fixed to the user's originally set bounds for each coefficient. An automatic tightening of these bounds based on the loss function history would lead to greater computational efficiency. Further, an automatic loosening of these bounds would reduce required user effort when the optimal coefficient lies on a bound. The implementation needs to be modified so that the user could supply nominal yet soft bounds for some coefficients, while also having the option to supply hard bounds where desired.

References

[1] S. Hosseini, R. Vinuesa, P. Schlatter, A. Hanifi, and D. Henningson. Direct numerical simulation of the flow around a wing section at moderate Reynolds number. *International Journal of Heat and Fluid Flow*, 61:117–128, 2016. ISSN: 0142727X. DOI: 10.1016/j.ijheatfluidflow.2016.02.001.

[2] P. Sagaut. *Large Eddy Simulation for Incompressible Flows*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN: 978-3-662-04697-5. DOI: 10.1007/978-3-662-04695-1.

[3] S. B. Pope. *Turbulent Flows*. 2000. ISBN: 9780521598866.

[4] P. A. Durbin and B. A. P. Reif. *Statistical Theory and Modeling for Turbulent Flows*. Wiley, 2010. ISBN: 9780470689318. DOI: 10.1002/9780470972076.

[5] F. D. Witherden and A. Jameson. Future directions of computational fluid dynamics. *23rd AIAA Computational Fluid Dynamics Conference, 2017*:1–16, 2017. DOI: 10.2514/6.2017-3791.

[6] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Marvriplis. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. Technical report NASA/CR-2014-218178, 2014.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. DOI: 10.1109/CVPR.2009.5206848.

[8] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung,

D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Ł. Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Ł. Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. d. A. B. Peres, M. Petrov, H. P. d. O. Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. GPT-4 Technical Report, 2023. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774.

[9] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pa-

cholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. ISSN: 0028-0836. DOI: 10.1038/s41586-021-03819-2. URL: https://www.nature.com/articles/s41586-021-03819-2.

[10] J. Schmidt, M. R. G. Marques, S. Botti, and M. A. L. Marques. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials*, 5(1):83, 2019. ISSN: 2057-3960. DOI: 10.1038/s41524-019-0221-0. URL: https://www.nature.com/articles/s41524-019-0221-0.

[11] M. Shehab, L. Abualigah, Q. Shambour, M. A. Abu-Hashem, M. K. Y. Shambour, A. I. Alsalibi, and A. H. Gandomi. Machine learning in medical applications: A review of state-of-the-art methods. *Computers in Biology and Medicine*, 145:105458, 2022. ISSN: 00104825. DOI: 10.1016/j.compbiomed.2022.105458. URL: https://linkinghub.elsevier.com/retrieve/pii/S0010482522002505.

[12] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52(1):1–32, 2020. ISSN: 0066-4189. DOI: 10.1146/annurev-fluid-010719-060214.

[13] M. Raissi, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 861:119–137, 2019. ISSN: 14697645. DOI: 10.1017/jfm.2018.872.

[14] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00314-5. URL: https://www.nature.com/articles/s42254-021-00314-5.

[15] H. Chen, X. He, L. Qing, Y. Wu, C. Ren, R. E. Sheriff, and C. Zhu. Real-world single image super-resolution: A brief review. *Information Fusion*, 79:124–145, 2022. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.09.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S1566253521001792.

[16] K. Fukami, K. Fukagata, and K. Taira. Super-resolution analysis via machine learning: a survey for fluid flows. *Theoretical and Computational Fluid Dynamics*, 37(4):421–444, 2023. ISSN: 14322250. DOI: 10.1007/s00162-023-00663-0.

[17] K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence Modeling in the Age of Data. *Annual Review of Fluid Mechanics*, 51(1):357–377, 2019. ISSN: 0066-4189. DOI: 10.1146/annurev-fluid-010518-040547. URL: https://www.annualreviews.org/doi/10.1146/annurev-fluid-010518-040547.

[18] A. P. Singh, K. Duraisamy, and Z. J. Zhang. Augmentation of Turbulence Models Using Field Inversion and Machine Learning. In *55th AIAA Aerospace Sciences Meeting*, Reston, Virginia. American Institute of Aeronautics and Astronautics, 2017. ISBN: 978-1-62410-447-3. DOI: 10.2514/6.2017-0993. URL: https://arc.aiaa.org/doi/10.2514/6.2017-0993.

[19] E. J. Parish and K. Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics*, 305:758–774, 2016. ISSN: 10902716. DOI: 10.1016/j.jcp.2015.11.012. URL: http://dx.doi.org/10.1016/j.jcp.2015.11.012.

[20] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021. ISSN: 1091-6490. DOI: 10.1073/pnas.2101784118. URL: http://dx.doi.org/10.1073/pnas.2101784118.

[21] M. Schmelzer, R. P. Dwight, and P. Cinnella. Discovery of Algebraic Reynolds-Stress Models Using Sparse Symbolic Regression. *Flow, Turbulence and Combustion*, 104(2-3):579–603, 2020. ISSN: 15731987. DOI: 10.1007/s10494-019-00089-x.

[22] K. Duraisamy. Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence. *Phys. Rev. Fluids*, 6:050504, 5, 2021. DOI: 10.1103/PhysRevFluids.6.050504. URL: https://link.aps.org/doi/10.1103/PhysRevFluids.6.050504.

[23] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016. ISSN: 0022-1120. DOI: 10.1017/jfm.2016.615. URL: https://www.cambridge.org/core/product/identifier/S0022112016006157/type/journal_article.

[24] John Hopkins Turbulence Database, 2020. URL: http://turbulence.pha.jhu.edu/.

[25] C. Rumsey. NASA Langley Research Center Turbulence Modeling Resource, 2021. URL: https://turbmodels.larc.nasa.gov/.

[26] European Research Community on Flow, Turbulence and Combustion Classic Collection Database. URL: http://cfd.mace.manchester.ac.uk/ercoftac/doku.php.

[27] R. McConkey, A. Mole, A. Skillen, A. Revell, E. Yee, and F.-S. Lien. Machine Learning Augmented Turbulence Modelling for Massively Separated Three-Dimensional Flows. In *Computational Fluids Conference*, Cannes, 2023.

[28] S. B. Pope. A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 72(02):331, 1975. ISSN: 0022-1120. DOI: 10.1017/S0022112075003382. URL: http://www.journals.cambridge.org/abstract_S0022112075003382.

[29] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM, 2016. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[30] R. McConkey and N. Kalia. turbo-RANS: straightforward & efficient optimization of turbulence model coefficients, 2024. URL: https://github.com/rmcconke/turbo-rans.

[31] D. C. Wilcox. *Turbulence Modelling for CFD*. DCW Industries, Inc., 2nd edition, 1994. ISBN: 0-9636051-0-0.

[32] P. Catalano and M. Amato. An evaluation of RANS turbulence modelling for aerodynamic applications. *Aerospace Science and Technology*, 7(7):493–509, 2003. ISSN: 12709638. DOI: 10.1016/S1270-9638(03)00061-0.

[33] J. L. Wu, H. Xiao, and E. Paterson. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Physical Review Fluids*, 7(3):1–28, 2018. ISSN: 2469990X. DOI: 10.1103/PhysRevFluids.3.074602.

[34] M. Kaandorp. Machine Learning for Data-Driven RANS Turbulence Modelling. *Delft University of Technology*, 2018. URL: https://repository.tudelft.nl/islandora/object/uuid%3Af833e151-7c0f-414c-8217-5af783c88474?collection=education.

[35] M. L. Kaandorp and R. P. Dwight. Data-driven modelling of the Reynolds stress tensor using random forests with invariance. *Computers & Fluids*, 202:104497, 2020. ISSN: 00457930. DOI: 10.1016/j.compfluid.2020.104497. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045793020300700.

[36] Y. Zhu and N. Dinh. A data-driven approach for turbulence modeling. *arXiv*, 2020. ISSN: 23318422. DOI: 10.48550/arXiv.2005.00426. URL: http://arxiv.org/abs/2005.00426.

[37] Z. Zhang, X. D. Song, S. R. Ye, Y. W. Wang, C. G. Huang, Y. R. An, and Y. S. Chen. Application of deep learning method to Reynolds stress models of channel flow based on reduced-order modeling of DNS data. *Journal of Hydrodynamics*, 31(1):58–65, 2019. ISSN: 1001-6058. DOI: 10.1007/s42241-018-0156-9. URL: http://link.springer.com/10.1007/s42241-018-0156-9.

[38] R. Fang, D. Sondak, P. Protopapas, and S. Succi. Neural network models for the anisotropic Reynolds stress tensor in turbulent channel flow. *Journal of Turbulence*, 21(9-10):525–543, 2020. ISSN: 1468-5248. DOI: 10.1080/14685248.2019.1706742. URL: https://www.tandfonline.com/doi/full/10.1080/14685248.2019.1706742.

[39] X. D. Song, Z. Zhang, Y. W. Wang, S. R. Ye, and C. G. Huang. Reconstruction of RANS model and cross-validation of flow field based on tensor basis neural network. *Proceedings of the ASME-JSME-KSME 2019 8th Joint Fluids Engineering Conference*:1–6, 2019. DOI: 10.1115/AJKFluids2019-5572.

[40] S. Taghizadeh, F. D. Witherden, and S. S. Girimaji. Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations. *New Journal of Physics*, 22(9):0–32, 2020. ISSN: 13672630. DOI: 10.1088/1367-2630/abadb3.

[41] O. Mesnard and L. A. Barba. Reproducible and replicable computational fluid dynamics: it's harder than you think. *Computing in Science and Engineering*, 19(4):44–55, 2017. ISSN: 15219615. DOI: 10.1109/MCSE.2017.3151254.

[42] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and H. Larochelle. Improving reproducibility in machine learning research (A report from the NeurIPS 2019 reproducibility program). *arXiv*, 2020. ISSN: 23318422. DOI: 10.48550/arXiv.2003.12206. URL: https://arxiv.org/abs/2003.12206.

[43] R. McConkey, E. Yee, and F.-S. Lien. Turbulence modelling using machine learning, 2021. DOI: 10.34740/kaggle/dsv/3256104. URL: https://www.kaggle.com/dsv/3256104.

[44] H. Xiao, J. L. Wu, S. Laizet, and L. Duan. Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations. *Computers and Fluids*, 200:104431, 2020. ISSN: 00457930. DOI: 10.1016/j.compfluid.2020.104431. URL: https://doi.org/10.1016/j.compfluid.2020.104431.

[45] A. Pinelli, M. Uhlmann, A. Sekimoto, and G. Kawahara. Reynolds number dependence of mean flow structure in square duct turbulence. *Journal of Fluid Mechanics*, 644:107–122, 2010. ISSN: 00221120. DOI: 10.1017/S0022112009992242.

[46] R. Matai and P. Durbin. Large-eddy simulation of turbulent flow over a parametric set of bumps. *Journal of Fluid Mechanics*, 866:503–525, 2019. ISSN: 0022-1120. DOI: 10.1017/jfm.2019.80. URL: https://www.cambridge.org/core/product/identifier/S0022112019000806/type/journal_article.

[47] J. P. Laval and M. Marquillie. Direct numerical simulations of converging-diverging channel flow. In *ERCOFTAC Series*. Volume 14, pages 203–209. Springer Netherlands, Dordrecht, 2011. ISBN: 9789048196029. DOI: 10.1007/978-90-481-9603-6_21. URL: http://link.springer.com/10.1007/978-90-481-9603-6_21.

[48] L. A. Schiavo, A. B. Jesus, J. L. Azevedo, and W. R. Wolf. Large eddy simulations of convergent-divergent channel flows at moderate Reynolds numbers. *International Journal of Heat and Fluid Flow*, 56:137–151, 2015. ISSN: 0142727X. DOI: 10.1016/j.ijheatfluidflow.2015.07.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0142727X15000880.

[49] Y. Bentaleb, S. Lardeau, and M. A. Leschziner. Large-eddy simulation of turbulent boundary layer separation from a rounded step. *Journal of Turbulence*, 13:1–28, 2012. ISSN: 14685248. DOI: 10.1080/14685248.2011.637923.

[50] OpenCFD Ltd. OpenFOAM: User Guide, 2019. URL: https://www.openfoam.com/documentation/guides/latest/doc/.

[51] B. Launder and D. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269–289, 1974. ISSN: 00457825. DOI: 10.1016/0045-7825(74)90029-2. URL: https://linkinghub.elsevier.com/retrieve/pii/0045782574900292.

[52] D. R. Laurence, J. C. Uribe, and S. V. Utyuzhnikov. A robust formulation of the v2-f model. *Flow, Turbulence and Combustion*, 73(3-4):169–185, 2005. ISSN: 13866184. DOI: 10.1007/s10494-005-1974-8.

[53] F. R. Menter, M. Kuntz, and R. Langtry. Ten years of industrial experience with the SST turbulence model. *Turbulence, Heat and Mass Transfer 4*:625–632, 2003.

[54] P. A. Durbin. Near-wall turbulence closure modeling without "damping functions". *Theoretical and Computational Fluid Dynamics*, 3(1):1–13, 1991. ISSN: 09354964. DOI: 10.1007/BF00271513.

[55] F.-S. Lien and G. Kalitzin. Computations of transonic flow with the v2-f turbulence model. *International Journal of Heat and Fluid Flow*, 22(1):53–61, 2001. ISSN: 0142727X. DOI: 10.1016/S0142-727X(00)00073-4.

[56] F. Liu. A thorough description of how wall functions are implemented in OpenFOAM. *Proceedings of CFD with OpenSource Software*:1–33, 2016.

[57] M. Marquillie, J. P. Laval, and R. Dolganov. Direct numerical simulation of a separated channel flow with a smooth profile. *Journal of Turbulence*, 9:1–23, 2008. ISSN: 14685248. DOI: 10.1080/14685240701767332.

[58] S. Banerjee, R. Krahl, F. Durst, and C. Zenger. Presentation of anisotropy properties of turbulence, invariants versus eigenvalue approaches. *Journal of Turbulence*, 8:1–27, 2007. ISSN: 14685248. DOI: 10.1080/14685240701506896.

[59] C. M. Rhie and W. L. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532, 1983. DOI: 10.2514/3.8284. URL: https://doi.org/10.2514/3.8284.

[60] A. J. Spencer and R. S. Rivlin. Isotropic integrity bases for vectors and second-order tensors - Part I. *Archive for Rational Mechanics and Analysis*, 9(1):45–63, 1962. ISSN: 00039527. DOI: 10.1007/BF00253332.

[61] J. N. Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017. ISSN: 14697645. DOI: 10.1017/jfm.2016.803.

[62] J. X. Wang, J. L. Wu, and H. Xiao. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. *Physical Review Fluids*, 2(3):1–22, 2017. ISSN: 2469990X. DOI: 10.1103/PhysRevFluids.2.034603.

[63] C.-W. Chang, J. Fang, and N. T. Dinh. Reynolds-averaged turbulence modeling using deep learning with local flow features: an empirical approach. *Nuclear Science and Engineering*, 194(8-9):650–664, 2020. DOI: 10.1080/00295639.2020.1712928. URL: https://doi.org/10.1080/00295639.2020.1712928.

[64] C. Jiang, R. Vinuesa, R. Chen, J. Mi, S. Laima, and H. Li. An interpretable framework of data-driven turbulence modeling using deep neural networks. *Physics of Fluids*, 33(5):055133, 2021. ISSN: 1070-6631. DOI: 10.1063/5.0048909. URL: https://doi.org/10.1063/5.0048909.

[65] S. Bhushan, G. W. Burgreen, W. Brewer, and I. D. Dettwiller. Development and validation of a machine learned turbulence model. *Energies*, 14(5), 2021. ISSN: 19961073. DOI: 10.3390/en14051465.

[66] W. Liu, J. Fang, S. Rolfo, C. Moulinec, and D. R. Emerson. An iterative machine-learning framework for RANS turbulence modeling. *International Journal of Heat and Fluid Flow*, 90(March):108822, 2021. ISSN: 0142727X. DOI: 10.1016/j.ijheatfluidflow.2021.108822. URL: https://doi.org/10.1016/j.ijheatfluidflow.2021.108822.

[67] P. A. Srinivasan, L. Guastoni, H. Azizpour, P. Schlatter, and R. Vinuesa. Predictions of turbulent shear flows using deep neural networks. *Physical Review Fluids*, 4(5):1–15, 2019. ISSN: 2469990X. DOI: 10.1103/PhysRevFluids.4.054603.

[68] Y. Yin, P. Yang, Y. Zhang, H. Chen, and S. Fu. Feature selection and processing of turbulence modeling based on an artificial neural network. *Physics of Fluids*, 32(10):105117, 2020. ISSN: 1070-6631. DOI: 10.1063/5.0022561. URL: https://doi.org/10.1063/5.0022561.

[69] R. Fang, D. Sondak, P. Protopapas, and S. Succi. Neural network models for the anisotropic Reynolds stress tensor in turbulent channel flow. *Journal of Turbulence*, 21(9-10):525–543, 2020. ISSN: 1468-5248. DOI: 10.1080/14685248.2019.1706742. URL: https://www.tandfonline.com/doi/full/10.1080/14685248.2019.1706742.

[70] X. Zhang, J. Wu, O. Coutier-Delgosha, and H. Xiao. Recent progress in augmenting turbulence models with physics-informed machine learning. *Journal of Hydrodynamics*, 31(6):1153–1158, 2019. ISSN: 18780342. DOI: 10.1007/s42241-019-0089-y.

[71] L. Zhu, W. Zhang, J. Kou, and Y. Liu. Machine learning methods for turbulence modeling in subsonic flows around airfoils. *Physics of Fluids*, 31(1), 2019. ISSN: 10897666. DOI: 10.1063/1.5061693. URL: http://dx.doi.org/10.1063/1.5061693.

[72] R. Matai and P. A. Durbin. Zonal Eddy Viscosity Models Based on Machine Learning. *Flow, Turbulence and Combustion*, 103(1):93–109, 2019. ISSN: 15731987. DOI: 10.1007/s10494-019-00011-5.

[73] Z. M. Nikolaou, C. Chrysostomou, Y. Minamoto, and L. Vervisch. Evaluation of a Neural Network-Based Closure for the Unresolved Stresses in Turbulent Premixed V-Flames. *Flow, Turbulence and Combustion*, 2020. ISSN: 15731987. DOI: 10.1007/s10494-020-00170-w. URL: https://doi.org/10.1007/s10494-020-00170-w.

[74] J. Tan, X. He, G. Rigas, and M. Vahdati. Towards explainable machine-learning-assisted turbulence modeling for transonic flows. *European Conference on Turbomachinery Fluid Dynamics and Thermodynamics*, 2021. DOI: 10.29008/ETC2021-490. URL: https://www.euroturbo.eu/publications/proceedings-papers/etc2021-490/.

[75] M. A. Cruz, R. L. Thompson, L. E. Sampaio, and R. D. Bacchi. The use of the Reynolds force vector in a physics informed machine learning approach for predictive turbulence modeling. *Computers and Fluids*, 192, 2019. ISSN: 00457930. DOI: 10.1016/j.compfluid.2019.104258.

[76] B. P. Brener, M. A. Cruz, R. L. Thompson, and R. P. Anjos. Conditioning and accurate solutions of Reynolds average Navier-Stokes equations with data-driven turbulence closures. *Journal of Fluid Mechanics*, 915:1–27, 2021. ISSN: 14697645. DOI: 10.1017/jfm.2021.148.

[77] J. L. Wu, R. Sun, S. Laizet, and H. Xiao. Representation of stress tensor perturbations with application in machine-learning-assisted turbulence modeling. *Computer Methods in Applied Mechanics and Engineering*, 346:707–726, 2019. ISSN: 00457825. DOI: 10.1016/j.cma.2018.09.010. URL: https://doi.org/10.1016/j.cma.2018.09.010.

[78] K. Hanjalić, M. Popovac, and M. Hadžiabdić. A robust near-wall elliptic-relaxation eddy-viscosity turbulence model for CFD. *International Journal of Heat and Fluid Flow*, 25(6):1047–1051, 2004. ISSN: 0142727X. DOI: 10.1016/j.ijheatfluidflow.2004.07.005.

[79] R. McConkey, E. Yee, and F.-S. Lien. A curated dataset for data-driven turbulence modelling. *Scientific Data*, 8(1):1–14, 2021. ISSN: 20524463. DOI: 10.1038/s41597-021-01034-2. URL: http://dx.doi.org/10.1038/s41597-021-01034-2.

[80] R. L. Thompson, G. Mompean, and L. Thais. A methodology to quantify the non-linearity of the Reynolds stress tensor. *Journal of Turbulence*, 11:1–27, 2010. ISSN: 14685248. DOI: 10.1080/14685248.2010.501799.

[81] J. Ho and A. West. Field Inversion and Machine Learning for turbulence modelling applied to three-dimensional separated flows. (August), 2021. DOI: 10.2514/6.2021-2903.

[82] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *Advances in Neural Information Processing Systems 30*:972–981, 2017. ISSN: 10495258. DOI: 10.48550/arXiv.1706.02515.

[83] A. Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2nd edition, 2019. ISBN: 1492032646.

[84] F. Chollet et al. Keras, 2015. URL: https://keras.io.

[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[86] S. M. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 31*, 2017. DOI: 10.48550/arXiv.1705.07874. URL: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[87] S. Lardeau and F. Billard. Development of an elliptic-blending lag model for industrial applications. In *54th AIAA Aerospace Sciences Meeting*, Reston, Virginia. American Institute of Aeronautics and Astronautics, 2016. ISBN: 978-1-62410-393-3. DOI: 10.2514/6.2016-1600. URL: https://arc.aiaa.org/doi/10.2514/6.2016-1600.

[88] F. Menter and T. Esch. Elements of Industrial Heat Transfer Predictions. In 16th Brazilian Congress of Mechanical Engineering, 2001.

[89] W. Layton and M. Schneier. Diagnostics for eddy viscosity models of turbulence including data-driven/neural network based parameterizations. *Results in Applied Mathematics*, 8:100099, 2020. ISSN: 25900374. DOI: 10.1016/j.rinam.2020.100099. URL: https://doi.org/10.1016/j.rinam.2020.100099.

[90] J. P. Panda and H. V. Warrior. Modelling the Pressure Strain Correlation in turbulent flows using Deep Neural Networks, 2021. DOI: 10.48550/arXiv.2103.00907. URL: http://arxiv.org/abs/2103.00907.

[91] J. P. Panda and H. V. Warrior. Evaluation of machine learning algorithms for predictive Reynolds stress transport modeling. 38, 2021. DOI: 10.48550/arXiv.2105.13641. URL: http://arxiv.org/abs/2105.13641.

[92] R. Maulik, O. San, A. Rasheed, and P. Vedula. Subgrid modelling for two-dimensional turbulence using neural networks. *Journal of Fluid Mechanics*, 858:122–144, 2019. ISSN: 14697645. DOI: 10.1017/jfm.2018.770.

[93] R. McConkey, E. Yee, and F.-S. Lien. Deep structured neural networks for turbulence closure modeling. *Physics of Fluids*, 34(3):035110, 2022. ISSN: 1070-6631. DOI: 10.1063/5.0083074. URL: https://pubs.aip.org/pof/article/34/3/035110/2845436/Deep-structured-neural-networks-for-turbulence.

[94] R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.11.011.

[95] B. P. Brener, M. A. Cruz, M. S. S. Macedo, and R. L. Thompson. An Invariant and Highly–Accurate Strategy for Data-Driven Turbulence Modelling. *SSRN Electronic Journal*, (March 2022):1–43, 2022. ISSN: 1556-5068. DOI: 10.2139/ssrn.4073177. URL: https://www.ssrn.com/abstract=4073177.

[96] R. McConkey, E. Yee, and F.-S. Lien. Deep learning-based turbulence closure with improved optimal eddy viscosity prediction. In *Proceedings of the 29th Annual Conference of the Computational Fluid Dynamics Society of Canada*, Online, 2021. URL: https://www.researchgate.net/publication/353496397.

[97] T. J. Craft, B. E. Launder, and K. Suga. Development and application of a cubic eddy-viscosity model of turbulence. *International Journal of Heat and Fluid Flow*, 17(2):108–115, 1996. ISSN: 0142727X. DOI: 10.1016/0142-727X(95)00079-6.

[98] F.-S. Lien, W. L. Chen, and M. A. Leschziner. Low-Reynolds-Number Eddy-Viscosity Modelling Based on Non-Linear Stress-Strain/Vorticity Relations. In *Engineering Turbulence Modelling and Experiments*. Volume 3, Elsevier Series in Thermal and Fluid Sciences, pages 91–100. Elsevier, Oxford, 1996. ISBN: 978-0-444-82463-9. DOI: 10.1016/B978-0-444-82463-9.50015-0. URL: https://www.sciencedirect.com/science/article/pii/B9780444824639500150.

[99] T. H. Shih, J. Zhu, and J. L. Lumley. A new Reynolds stress algebraic equation model. *Computer Methods in Applied Mechanics and Engineering*, 125(1-4):287–302, 1995. ISSN: 00457825. DOI: 10.1016/0045-7825(95)00796-4.

[100] T. B. Gatski and T. Jongen. Nonlinear eddy viscosity and algebraic stress models for solving complex turbulent flows. *Progress in Aerospace Sciences*, 36(8):655–682, 2000. ISSN: 03760421. DOI: 10.1016/S0376-0421(00)00012-9.

[101] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 31*:3147–3155, 2017. ISSN: 10495258. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[102] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems 32*:6638–6648, 2018. ISSN: 10495258. DOI: 10.48550/arXiv.1706.09516. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf.

[103] The pandas development team. pandas-dev/pandas: Pandas, 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[104] S. Caron, D. Arnström, S. Bonagiri, A. Dechaume, N. Flowers, A. Heins, T. Ishikawa, D. Kenefake, G. Mazzamuto, D. Meoli, B. O'Donoghue, A. A. Oppenheimer, A. Pandala, J. J. Quiroz Omaña, N. Rontsis, P. Shah, S. St-Jean, N. Vitucci, S. Wolfers, F. Yang, @bdelhaisse, @MeindertHH, @rimaddo, @urob, and @shaoanlu. qpsolvers: Quadratic Programming Solvers in Python, version 4.3.2, 2024. URL: https://github.com/qpsolvers/qpsolvers.

[105] T. Head, M. Kumar, H. Nahrstaedt, G. Louppe, and I. Shcherbatyi. scikit-optimize, 2021. DOI: 10.5281/zenodo.5565057. URL: https://doi.org/10.5281/zenodo.5565057.

[106] P. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting and Exhibit*, number 1, pages 5–21, Reston, Virigina. American Institute of Aeronautics and Astronautics, 1992. DOI: 10.2514/6.1992-439. URL: http://arc.aiaa.org/doi/10.2514/6.1992-439.

[107] R. Maulik, H. Sharma, S. Patel, B. Lusch, and E. Jennings. A turbulent eddy-viscosity surrogate modeling framework for Reynolds-averaged Navier-Stokes simulations. *Computers and Fluids*, 227:104777, 2021. ISSN: 00457930. DOI: 10.1016/j.compfluid.2020.104777. URL: https://doi.org/10.1016/j.compfluid.2020.104777.

[108] J. Cai, P.-E. Angeli, J.-M. Martinez, G. Damblin, and D. Lucor. Reynolds Stress Anisotropy Tensor Predictions for Turbulent Channel Flow using Neural Networks, 2022. DOI: 10.48550/arXiv.2208.14301. URL: http://arxiv.org/abs/2208.14301.

[109] J. Cai, P.-E. Angeli, J.-M. Martinez, G. Damblin, and D. Lucor. Revisiting Tensor Basis Neural Networks for Reynolds stress modeling: application to plane channel and square duct flows, 2024. DOI: 10.1016/j.compfluid.2024.106246. URL: http://arxiv.org/abs/2403.11746.

[110] A. Man, M. Jadidi, A. Keshmiri, H. Yin, and Y. Mahmoudi. A divide-and-conquer machine learning approach for modeling turbulent flows. *Physics of Fluids*, 35(5), 2023. ISSN: 1070-6631. DOI: 10.1063/5.0149750. URL: https://pubs.aip.org/pof/article/35/5/055110/2887961/A-divide-and-conquer-machine-learning-approach-for.

[111] H. Mandler and B. Weigand. Feature importance in neural networks as a means of interpretation for data-driven turbulence models. *Computers and Fluids*, 265, 2023. ISSN: 00457930. DOI: 10.1016/j.compfluid.2023.105993.

[112] J. Wu, H. Xiao, R. Sun, and Q. Wang. Reynolds-averaged Navier-Stokes equations with explicit data-driven Reynolds stress closure can be ill-conditioned. *Journal of Fluid Mechanics*, 869:553–586, 2019. ISSN: 14697645. DOI: 10.1017/jfm.2019.205.

[113] A. Amarloo, P. Forooghi, and M. Abkar. Frozen propagation of the Reynolds force vector from high-fidelity data into the Reynolds-averaged simulations of secondary flows:1–16, 2022. DOI: 10.48550/arXiv.2209.03911. URL: http://arxiv.org/abs/2209.03911.

[114] P. Spalart. An Old-Fashioned Framework for Machine Learning in Turbulence Modeling. *ERCOFTAC Bulletin 134*, 2023. DOI: 10.48550/arXiv.2308.00837. URL: http://arxiv.org/abs/2308.00837.

[115] R. McConkey, E. Yee, and F.-S. Lien. On the Generalizability of Machine-Learning-Assisted Anisotropy Mappings for Predictive Turbulence Modelling. *International Journal of Computational Fluid Dynamics*, 36(7):555–577, 2022. ISSN: 1061-8562. DOI: 10.1080/10618562.2022.2113520. URL: https://www.tandfonline.com/doi/full/10.1080/10618562.2022.2113520.

[116] L. Riccius, A. Agrawal, and P.-S. Koutsourelakis. Physics-Informed Tensor Basis Neural Network for Turbulence Closure Modeling. *arXiv*, 2023. DOI: 10.48550/arXiv.2311.14576.

[117] D. C. Wilcox. Reassessment of the scale-determining equation for advanced turbulence models. *AIAA Journal*, 26(11):1299–1310, 1988. ISSN: 00011452. DOI: 10.2514/3.10041.

[118] F. R. Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32(8):1598–1605, 1994. ISSN: 00011452. DOI: 10.2514/3.12149.

[119] R. McConkey. tbnn, 2024. URL: https://github.com/rmcconke/tbnn.

[120] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, 2017. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[121] R. McConkey. integrity-basis-input-features, 2023. URL: https://github.com/rmcconke/integrity-basis-input-features.

[122] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for Activation Functions. *arXiv*, 2017. DOI: 10.48550/arXiv.1710.05941. URL: http://arxiv.org/abs/1710.05941.

[123] S. J. Reddi, S. Kale, and S. Kumar. On the Convergence of Adam and Beyond. *ICLR 2018*, 2018. DOI: 10.48550/arXiv.1904.09237. URL: http://arxiv.org/abs/1904.09237.

[124] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ICLR 2015*, 2015. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980.

[125] P. Schlatter and R. Örlü. Assessment of direct numerical simulation data of turbulent boundary layers. *Journal of Fluid Mechanics*, 659:116–126, 2010. ISSN: 0022-1120. DOI: 10.1017/S0022112010003113. URL: https://www.cambridge.org/core/product/identifier/S0022112010003113/type/journal_article.

[126] N. V. Nikitin, N. V. Popelenskaya, and A. Stroh. Prandtl's Secondary Flows of the Second Kind. Problems of Description, Prediction, and Simulation. *Fluid Dynamics*, 56(4):513–538, 2021. ISSN: 15738507. DOI: 10.1134/S0015462821040091.

[127] M. Emory and D. G. Iaccarino. Visualizing turbulence anisotropy in the spatial domain with componentality contours. *Center for Turbulence Research Annual Research Briefs*, 2014.

[128] F. Menter and R. Lechner. Best Practice: Generalized k-ω (GEKO) Two-Equation Turbulence Modeling in Ansys CFD. Technical report, ANSYS, 2021.

[129] C. Grabe, F. Jäckel, P. Khurana, and R. P. Dwight. Data-driven augmentation of a RANS turbulence model for transonic flow prediction. *International Journal of Numerical Methods for Heat & Fluid Flow*, 33(4):1544–1561, 2023. ISSN: 0961-5539. DOI: 10.1108/HFF-08-2022-0488. URL: https://www.emerald.com/insight/content/doi/10.1108/HFF-08-2022-0488/full/html.

[130] L. Sciacovelli, A. Cannici, D. Passiatore, and P. Cinnella. A priori tests of turbulence models for compressible flows. *International Journal of Numerical Methods for Heat & Fluid Flow*, 2023. ISSN: 0961-5539. DOI: 10.1108/HFF-09-2023-0551. URL: https://www.emerald.com/insight/content/doi/10.1108/HFF-09-2023-0551/full/html.

[131] J. M. Gimenez and F. Bre. Optimization of RANS turbulence models using genetic algorithms to improve the prediction of wind pressure coefficients on low-rise buildings. *Journal of Wind Engineering and Industrial Aerodynamics*, 193:103978, 2019. ISSN: 01676105. DOI: 10.1016/j.jweia.2019.103978. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167610519305781.

[132] S. Mauro, R. Lanzafame, M. Messina, and D. Pirrello. Transition turbulence model calibration for wind turbine airfoil characterization through the use of a Micro-Genetic Algorithm. *International Journal of Energy and Environmental Engineering*, 8(4):359–374, 2017. ISSN: 2008-9163. DOI: 10.1007/s40095-017-0248-2. URL: http://link.springer.com/10.1007/s40095-017-0248-2.

[133] D. A. Yoder and P. D. Orkwis. On the use of optimization techniques for turbulence model calibration. *Computers & Fluids*, 214(May):104752, 2021. ISSN: 00457930. DOI: 10.1016/j.compfluid.2020.104752. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045793020303224.

[134] Z. Gu, X. Song, Y. Jiang, and X. Gong. Optimization of the Realizable k - ϵ Turbulence Model Especially for the Simulation of Road Vehicle. In 2012. DOI: 10.4271/2012-01-0778.

[135] K. Barkalov, I. Lebedev, M. Usova, D. Romanova, D. Ryazanov, and S. Strijhak. Optimization of Turbulence Model Parameters Using the Global Search Method Combined with Machine Learning. *Mathematics*, 10(15):2708, 2022. ISSN: 2227-7390. DOI: 10.3390/math10152708. URL: https://www.mdpi.com/2227-7390/10/15/2708.

[136] B. Nadiga, C. Jiang, and D. Livescu. Leveraging Bayesian analysis to improve accuracy of approximate models. *Journal of Computational Physics*, 394:280–297, 2019. ISSN: 00219991. DOI: 10.1016/j.jcp.2019.05.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999119303407.

[137] M. Diessner, J. O'Connor, A. Wynn, S. Laizet, Y. Guan, K. Wilson, and R. D. Whalley. Investigating Bayesian optimization for expensive-to-evaluate black box functions: Application in fluid dynamics. *Frontiers in Applied Mathematics and Statistics*, 8, 2022. ISSN: 2297-4687. DOI: 10.3389/fams.2022.1076296. URL: https://www.frontiersin.org/articles/10.3389/fams.2022.1076296/full.

[138] Y. Morita, S. Rezaeiravesh, N. Tabatabaei, R. Vinuesa, K. Fukagata, and P. Schlatter. Applying Bayesian optimization with Gaussian process regression to computational fluid dynamics problems. *Journal of Computational Physics*, 449, 2022. ISSN: 10902716. DOI: 10.1016/j.jcp.2021.110788.

[139] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023. DOI: 10.1017/9781108348973.

[140] D. Maruyama, P. Bekemeyer, S. Görtz, S. Coggon, and S. Sharma. Data-driven Bayesian inference of turbulence model closure coefficients incorporating epistemic uncertainty. *Acta Mechanica Sinica/Lixue Xuebao*, 37(12):1812–1838, 2021. ISSN: 16143116. DOI: 10.1007/s10409-021-01152-5.

[141] A. Da Ronch, M. Panzeri, J. Drofelnik, and R. d'Ippolito. Sensitivity and calibration of turbulence model in the presence of epistemic uncertainties. *CEAS Aeronautical Journal*, 11(1):33–47, 2020. ISSN: 1869-5582. DOI: 10.1007/s13272-019-00389-y. URL: http://link.springer.com/10.1007/s13272-019-00389-y.

[142] J. Ray, L. Dechant, S. Lefantzi, J. Ling, and S. Arunajatesan. Robust Bayesian Calibration of a k-ϵ Model for Compressible Jet-in-Crossflow Simulations. *AIAA Journal*, 56(12):4893–4909, 2018. ISSN: 0001-1452. DOI: 10.2514/1.J057204.

[143] J. Li, S. Chen, F. Cai, S. Wang, and C. Yan. Bayesian uncertainty analysis of SA turbulence model for supersonic jet interaction simulations. *Chinese Journal of Aeronautics*, 35(4):185–201, 2022. ISSN: 10009361. DOI: 10.1016/j.cja.2021.07.039. URL: https://linkinghub.elsevier.com/retrieve/pii/S1000936121003332.

[144] W. Edeling, P. Cinnella, R. Dwight, and H. Bijl. Bayesian estimates of parameter variability in the k–ϵ turbulence model. *Journal of Computational Physics*, 258:73–94, 2014. ISSN: 00219991. DOI: 10.1016/j.jcp.2013.10.027. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999113007031.

[145] T. A. Oliver and R. D. Moser. Bayesian uncertainty quantification applied to RANS turbulence models. *Journal of Physics: Conference Series*, 318(4):042032, 2011. ISSN: 1742-6596. DOI: 10.1088/1742-6596/318/4/042032. URL: https://iopscience.iop.org/article/10.1088/1742-6596/318/4/042032.

[146] L. Fischer, D. James, S. Jeyaseelan, and M. Pfitzner. Optimizing the trench shaped film cooling design. *International Journal of Heat and Mass Transfer*, 214:124399, 2023. ISSN: 00179310. DOI: 10.1016/j.ijheatmasstransfer.2023.124399. URL: https://linkinghub.elsevier.com/retrieve/pii/S0017931023005458.

[147] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. DOI: 10.7551/mitpress/3206.001.0001.

[148] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014. URL: https://github.com/fmfn/BayesianOptimization.

[149] I. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967. ISSN: 00415553. DOI: 10.1016/0041-5553(67)90144-9. URL: https://linkinghub.elsevier.com/retrieve/pii/0041555367901449.

[150] C. L. Ladson. Technical Memorandum 4074: Effects of Independent Variation of Mach and Reynolds Numbers on the Low-Speed Aerodynamic Characteristics of the NACA 0012 Airfoil Section. Technical report 4074/19880019495, NASA, 1988.

[151] M. Marquillie, U. Ehrenstein, and J. P. Laval. Instability of streaks in wall turbulence with adverse pressure gradient. *Journal of Fluid Mechanics*, 681:205–240, 2011. ISSN: 14697645. DOI: 10.1017/jfm.2011.193.

[152] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark. KAN: Kolmogorov-Arnold Networks, 2024. DOI: 10.48550/arXiv.2404.19756. URL: http://arxiv.org/abs/2404.19756.

APPENDICES

Appendix A

Neural network hyperparameter selection

The hyperparameters for the neural networks were selected using a mixture of random search and grid search. Due to computational costs, the goal of this procedure was limited to finding a suitable set of hyperparameters to demonstrate the model framework, rather than find an optimum among thousands of combinations. First, the Keras hyperparameter tuner [84] was used to randomly search within the hyperparameter space. The primary hyperparameters were the number of hidden layers, number of neurons per layer, and the regularization parameter. For the random hyperparameter search, performance based learning rate scheduling was used to automatically reduce the learning rate and stop the training after convergence. Therefore the number of training epochs for the automatic search varied.

The results from the random search were then used to instruct a coarse grid search, completed manually. It was found that the mapping to be constructed here required a significant amount of manual trial and error. While the automatic hyperparameter search was useful to generate a series of candidate hyperparameter selections, boosting the performance of these candidates was necessary through a manual coarse grid search. In particular, while many of the top candidates produced by the random search had similar number of layers and neurons per layer, the performance was highly sensitive to the L2 regularization parameter.

Table A.1 presents a summary of the hyperparameter search results for the EVNN, with a few selected hyperparameter combinations. In total, over one hundred hyperparameter combination experiments were manually conducted to find an optimal combination.

Table A.1: Selected EVNN hyperparameter combinations from the optimization procedure. The bolded hyperparameters were selected for use in testing.

Neurons per layer	Hidden layers	L2 reg.	Epochs	Learning rate	Validation MSE (-)
30	14	1×10^{-5}	3000	5×10^{-5}	8.73×10^{-4}
30	14	1×10^{-5}	2000	1×10^{-4}	5.97×10^{-3}
30	14	1×10^{-4}	2000	3×10^{-4}	2.19×10^{-3}
30	14	1×10^{-4}	1500	3×10^{-4}	8.42×10^{-4}
30	14	1×10^{-4}	800	3×10^{-4}	1.46×10^{-3}
60	4	1×10^{-5}	300	3×10^{-4}	5.05×10^{-3}
60	6	1×10^{-5}	300	3×10^{-4}	5.02×10^{-3}
30	**14**	$\mathbf{1 \times 10^{-5}}$	**2000**	$\mathbf{1 \times 10^{-4}}$	$\mathbf{5.58 \times 10^{-4}}$

A similar procedure was completed to find suitable hyperparameters for the apNN.

Appendix B

Invariants of the basis tensors for 2D flows

Table B.1: Basis tensors in the minimal integrity basis for S, R, A_k, and A_{v2}. For any tensor A, the first invariant is given by $I_1(A) = \mathrm{tr}(A)$ and the second invariant is given by $I_2(A) = \frac{1}{2}[(\mathrm{tr}(A))^2 - \mathrm{tr}(A^2)]$.

Basis tensor	Expression	Non-zero I_1	Non-zero I_2	Reason for zero I_1	Reason for zero I_2
B_1	S^2	✓	✓	—	—
B_2	S^3	—	✓	Continuity	—
B_3	R^2	✓	✓	—	—
B_4	A_{v2}^2	✓	✓	—	—
B_5	A_k^2	✓	✓	—	—
B_6	R^2S	—	✓	Continuity	—
B_7	R^2S^2	✓	✓	—	—
B_8	R^2SRS^2	—	✓	Directly	—
B_9	A_{v2}^2S	✓	—	—	Directly

(continued on next page)

Table B.1 (continued)

Basis tensor	Expression	Non-zero I_1	Non-zero I_2	Reason for zero I_1	Reason for zero I_2
B_{10}	$A_{v2}^2 S^2$	✓	—	—	Directly
B_{11}	$A_{v2}^2 S A_{v2} S^2$	—	—	Directly	Directly
B_{12}	$A_k^2 S$	✓	—	—	Directly
B_{13}	$A_k^2 S^2$	✓	—	—	Directly
B_{14}	$A_k^2 S A_k S^2$	—	—	Directly	Directly
B_{15}	$R A_{v2}$	—	—	Directly	Directly
B_{16}	$A_{v2} A_k$	✓	✓	—	—
B_{17}	$R A_k$	—	—	Directly	Directly
B_{18}	$R A_{v2} S$	—	—	Directly	Directly
B_{19}	$R A_{v2} S^2$	—	—	Directly	Directly
B_{20}	$R^2 A_{v2} S$	—	—	Directly	Directly
B_{21}	$A_{v2}^2 R S$	✓	—	—	Directly
B_{22}	$R^2 A_{v2} S^2$	—	—	Directly	Directly
B_{23}	$A_{v2}^2 R S^2$	—	—	Continuity	Directly
B_{24}	$R^2 S A_{v2} S^2$	—	—	Directly	Directly
B_{25}	$A_{v2}^2 S R S^2$	✓	—	—	Directly
B_{26}	$R A_k S$	—	—	Directly	Directly
B_{27}	$R A_k S^2$	—	—	Directly	Directly
B_{28}	$R^2 A_k S$	—	—	Directly	Directly
B_{29}	$A_k^2 R S$	✓	—	—	Directly
B_{30}	$R^2 A_k S^2$	—	—	Directly	Directly
B_{31}	$A_k^2 R S^2$	—	—	Continuity	Directly

(continued on next page)

Table B.1 (continued)

Basis tensor	Expression	Non-zero I_1	Non-zero I_2	Reason for zero I_1	Reason for zero I_2
B_{32}	$R^2SA_kS^2$	—	—	Directly	Directly
B_{33}	$A_k^2SRS^2$	✓	—	—	Directly
B_{34}	$A_{v2}A_kS$	✓	—	—	Directly
B_{35}	$A_{v2}A_kS^2$	✓	—	—	Directly
B_{36}	$A_{v2}^2A_kS$	—	—	Directly	Directly
B_{37}	$A_k^2A_{v2}S$	—	—	Directly	Directly
B_{38}	$A_{v2}^2A_kS^2$	—	—	Directly	Directly
B_{39}	$A_k^2A_{v2}S^2$	—	—	Directly	Directly
B_{40}	$A_{v2}^2SA_kS^2$	—	—	Directly	Directly
B_{41}	$A_k^2SA_{v2}S^2$	—	—	Directly	Directly
B_{42}	$RA_{v2}A_k$	✓	—	—	Directly
B_{43}	$RA_{v2}A_kS$	✓	—	—	Directly
B_{44}	$RA_kA_{v2}S$	✓	—	—	Directly
B_{45}	$RA_{v2}A_kS^2$	✓	—	—	Directly
B_{46}	$RA_kA_{v2}S^2$	✓	—	—	Directly
B_{47}	$RA_{v2}SA_kS^2$	—	—	Directly	Directly

The input features used in the present work primarily consist of a set of invariants for a minimal integrity basis constructed from four mean field gradient tensors. Note that the flow considered in the present work is two-dimensional, which means that both the z-velocity component and all z-direction gradients are zero. As a result, the mean velocity

gradient tensor is expressed as

$$\nabla\vec{U} = \begin{bmatrix} \dfrac{\partial U}{\partial x} & \dfrac{\partial U}{\partial y} & 0 \\ \dfrac{\partial V}{\partial x} & \dfrac{\partial V}{\partial y} & 0 \\ 0 & 0 & 0 \end{bmatrix} . \tag{B.1}$$

The mean strain-rate tensor S assumes the following form:

$$S = \frac{1}{2}\left(\nabla\vec{U} + \nabla\vec{U}^{\mathrm{T}}\right) = \frac{1}{2}\begin{bmatrix} 2\dfrac{\partial U}{\partial x} & \dfrac{\partial U}{\partial y} + \dfrac{\partial V}{\partial x} & 0 \\ \dfrac{\partial V}{\partial x} + \dfrac{\partial U}{\partial y} & 2\dfrac{\partial V}{\partial y} & 0 \\ 0 & 0 & 0 \end{bmatrix} . \tag{B.2}$$

The mean rotation-rate tensor R is given by

$$R = \frac{1}{2}\left(\nabla\vec{U} - \nabla\vec{U}^{\mathrm{T}}\right) = \frac{1}{2}\begin{bmatrix} 0 & \dfrac{\partial U}{\partial y} - \dfrac{\partial V}{\partial x} & 0 \\ \dfrac{\partial V}{\partial x} - \dfrac{\partial U}{\partial y} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} . \tag{B.3}$$

The TKE gradient vector can be recast as the following antisymmetric tensor:

$$A_k = \begin{bmatrix} 0 & 0 & \dfrac{\partial k}{\partial y} \\ 0 & 0 & -\dfrac{\partial k}{\partial x} \\ -\dfrac{\partial k}{\partial y} & \dfrac{\partial k}{\partial x} & 0 \end{bmatrix} . \tag{B.4}$$

Similarly, the gradient of v^2 (vector) can be re-expressed as the following antisymmetric

tensor:

$$A_{v2} = \begin{bmatrix} 0 & 0 & \dfrac{\partial v^2}{\partial y} \\ 0 & 0 & -\dfrac{\partial v^2}{\partial x} \\ -\dfrac{\partial v^2}{\partial y} & \dfrac{\partial v^2}{\partial x} & 0 \end{bmatrix} . \tag{B.5}$$

A minimal integrity basis was constructed from the elements of the four independent tensors S, R, A_k, and A_{v2} using the procedure developed by Spencer and Rivlin [60], and recently used by Wu *et al.* [33]. The resulting integrity basis has 47 tensors, summarized in Table B.1.

Maple, a symbolic math toolbox, was used to evaluate the first and second invariants for each of these basis tensors. Many of the invariants are identically zero, either directly (mathematically), or as a result of the incompressibility of the flow. The latter physical constraint is expressed through the continuity equation which, for a two-dimensional flow, reduces to

$$\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 . \tag{B.6}$$

The third tensor invariant of any of the basis tensors comprising the minimal integrity basis formed from S, R, A_k and A_{v2} (viz., $\det(A)$ for any basis tensor A) is necessarily (and, trivially) zero. The latter follows directly from the fact that the determinant of each of the base (independent) tensors S, R, A_k and A_{v2}, used to construct the minimal integrity basis, vanishes identically.

Table B.1 summarizes the results of the symbolic analysis, which was used to guide the input feature selection in the present work. A check mark in the table indicates that the associated invariant of the basis tensor (B_1–B_{47}) is non-zero and, as a consequence, can be used potentially as an non-trivial input feature for the prediction of $\nu_t^\dagger$ and $a^\perp$ in EVNN and apNN, respectively.

Appendix C

Hyperparameter search space for generalizability investigation

Table C.1 shows the search space and tuning results for the hyperparameter optimization.

Table C.1: Hyperparameter tuning search space and results for each model. Hyperparameter tuning used 100 iterations of a Bayesian optimizer [105]. Shown here are the optimal values on the FULL training set (see Table 4.3). However, this procedure was completed to search for an optimized set of hyperparameters for each training dataset. The supplementary information includes the results for each training dataset.

Model	Hyperparameter	Search space	Optimal value (FULL training set)
Neural network	Activation function	ReLU, SELU, or ELU	ReLU
	Number of hidden layers	[3, 20]	4
	Number of neurons per layer	[30, 800]	362
	L1 regularization parameter	$[10^{-8}, 10^{-1}]$	$2.32(10)^{-6}$
	L2 regularization parameter	$[10^{-8}, 10^{-1}]$	10^{-8}
	Learning rate	$[10^{-7}, 10^{-3}]$	$8.29(10)^{-4}$
Random forest	Number of estimators	[100, 1000]	889
	Max depth	[5, 50]	28
	Minimum samples to split internal node	$[10^{-5}, 10^{-1}]$	$1.0110)^{-5}$
	Minimum samples required at leaf node	$[10^{-5}, 10^{-1}]$	$4.04(10)^{-5}$
	Maximum fraction of features for estimator	[0.5, 1.0]	0.5
	Maximum number of samples for estimator	[0.5, 1.0]	0.520
XGBoost	Number of estimators	[100, 1000]	887
	Max depth	[5, 50]	37
	Learning rate	$[10^{-2}, 1]$	$1.30(10)^{-2}$
	Maximum fraction of features for estimator	[0.5, 1.0]	0.562
	Maximum number of samples for estimator	[0.5, 1.0]	0.788

Appendix D

Additional results for generalizability investigation

This material contains extra generalizability tests that could not be included in the main thesis due to space. While the thesis includes the generalizability test of the XGBoost model on a few selected flows, all tests for all models and all training datasets are presented here.

For each training set and for each model type, a hyperparameter optimization procedure was completed (see the original paper for more details). Then, these optimized models were tested on the hold-out test flows. This material is organized by test case. For a given test case, optimized models on various training sets are compared.

In total, there are 60 generalizability tests consisting of 3 model types $\times$ 4 training sets $\times$ 5 test cases. All test cases were held out of all training sets (see Table 3 in the paper). Each generalizability test is represented by a figure in this appendix showing the true and predicted values of the tensor basis coefficients $g^{(1)}$, $g^{(2)}$, and $g^{(3)}$.

For all figures, the following sub-figure numbering holds: (a) true $g^{(1)}$ field; (b) predicted $g^{(1)}$ field; (c) true $g^{(2)}$ field; (d) predicted $g^{(2)}$ field; (e) true $g^{(3)}$ field; and, (f) predicted $g^{(3)}$ field.

D.1 Periodic hills test set

D.1.1 PHLL4 training set

Neural network

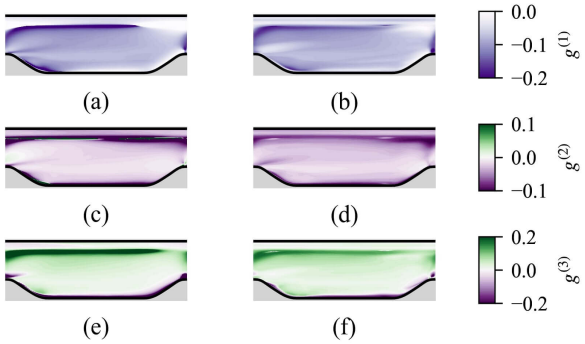


Figure D.1: Generalizability test on the periodic hills test case for the neural network model trained on the periodic hills PHLL4 set.

Random forest

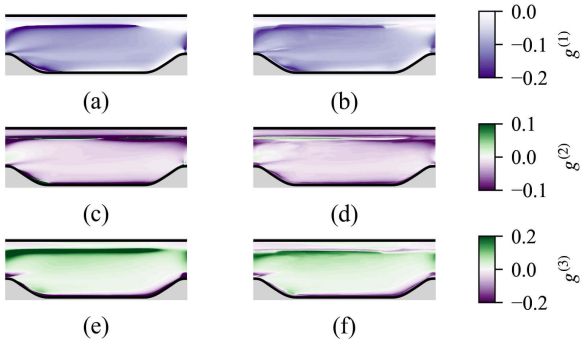


Figure D.2: Generalizability test on the periodic hills test case for the random forest model trained on the periodic hills PHLL4 set.

XGBoost

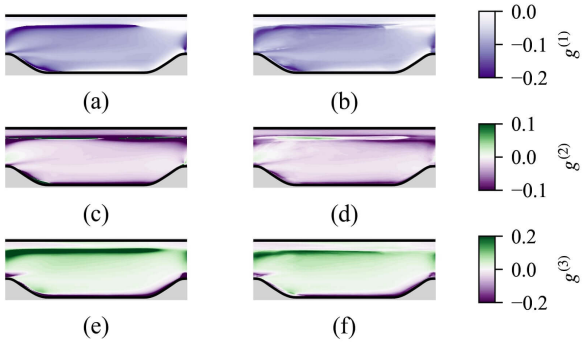


Figure D.3: Generalizability test on the periodic hills test case for the XGBoost model trained on the periodic hills PHLL4 set.

D.1.2 BUMP4 training set

Neural network

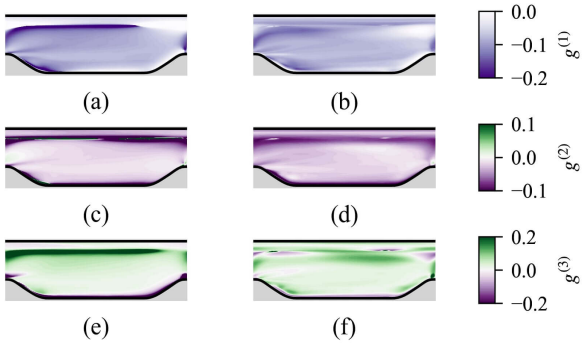


Figure D.4: Generalizability test on the periodic hills test case for the neural network model trained on the parametric bump BUMP4 set.

Random forest

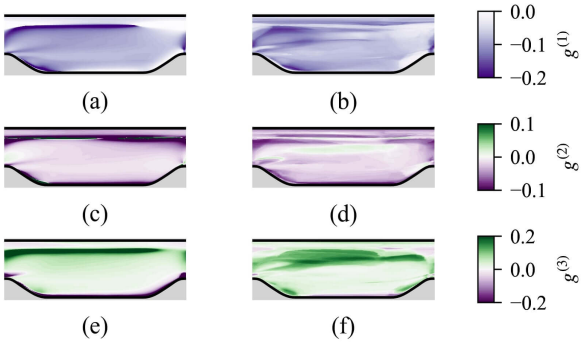


Figure D.5: Generalizability test on the periodic hills test case for the random forest model trained on the parametric bump BUMP4 set.

XGBoost

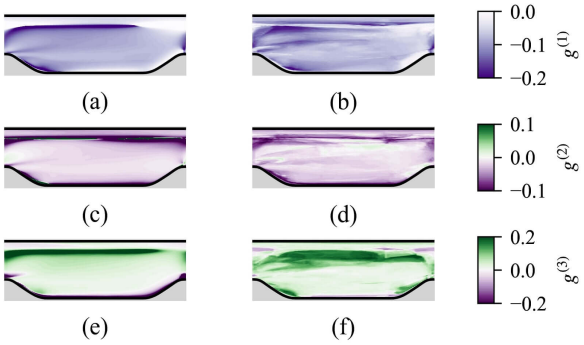


Figure D.6: Generalizability test on the periodic hills test case for the XGBoost model trained on the parametric bump BUMP4 set.

D.1.3 DUCT15 training set

Neural network

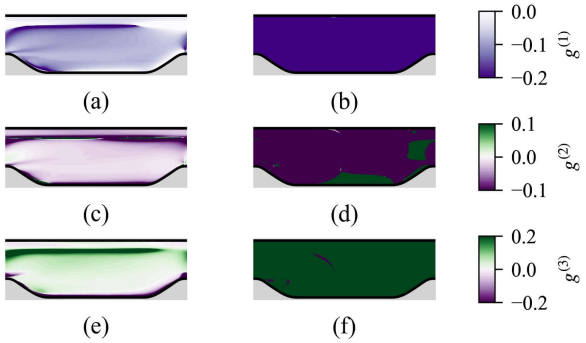


Figure D.7: Generalizability test on the periodic hills test case for the neural network model trained on the square duct DUCT15 set.

Random forest

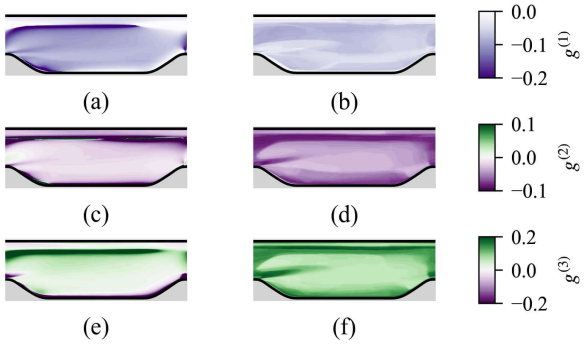


Figure D.8: Generalizability test on the periodic hills test case for the random forest model trained on the square duct DUCT15 set.

XGBoost

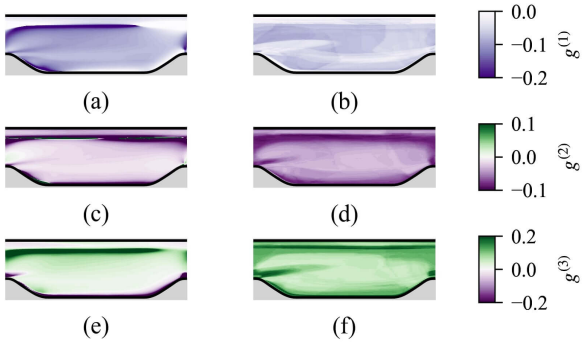


Figure D.9: Generalizability test on the periodic hills test case for the XGBoost model trained on the square duct DUCT15 set.

D.1.4 FULL training set

Neural network

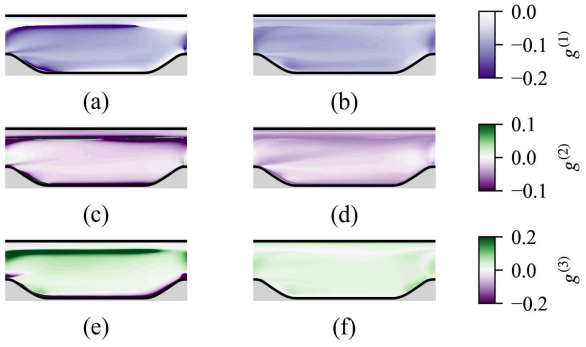


Figure D.10: Generalizability test on the periodic hills test case for the neural network model trained on the FULL set.

Random forest

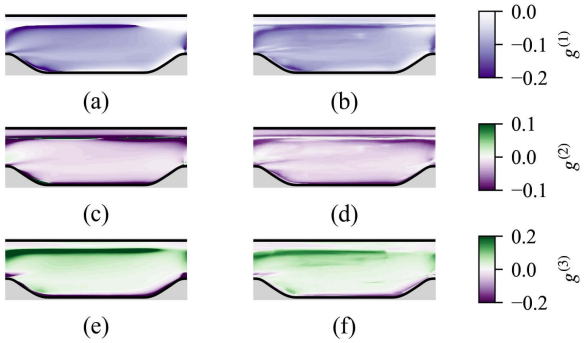


Figure D.11: Generalizability test on the periodic hills test case for the random forest model trained on the FULL set.

XGBoost

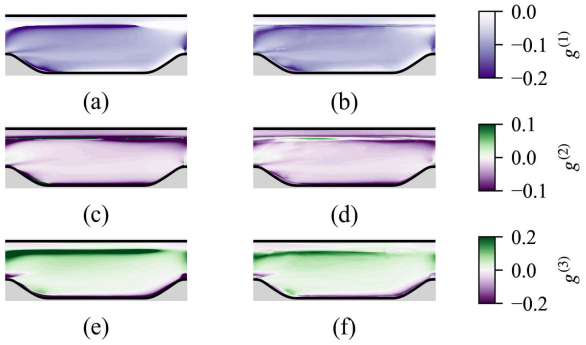


Figure D.12: Generalizability test on the periodic hills test case for the XGBoost model trained on the FULL set.

D.2 Parametric bump test set

D.2.1 PHLL4 training set

Neural network

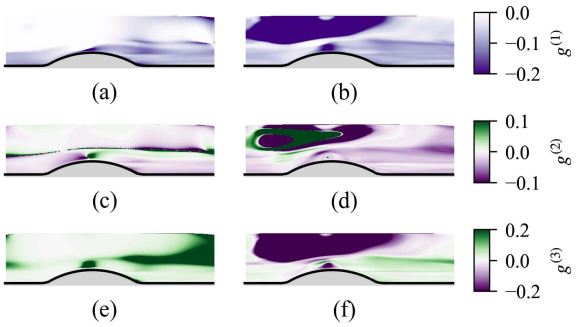


Figure D.13: Generalizability test on the parametric bumps test case for the neural network model trained on the periodic hills PHLL4 set.

Random forest

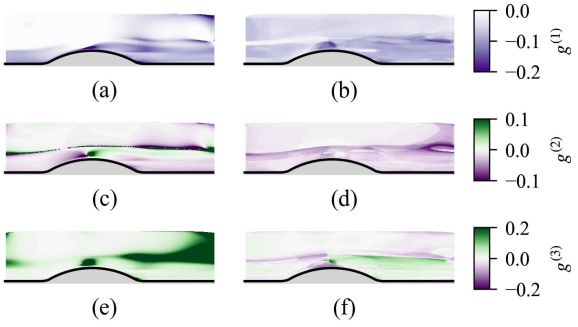


Figure D.14: Generalizability test on the parametric bumps test case for the random forest model trained on the periodic hills PHLL4 set.

XGBoost

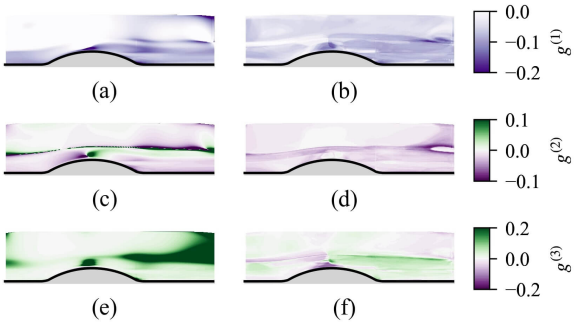


Figure D.15: Generalizability test on the parametric bumps test case for the XGBoost model trained on the periodic hills PHLL4 set.

D.2.2 BUMP4 training set

Neural network

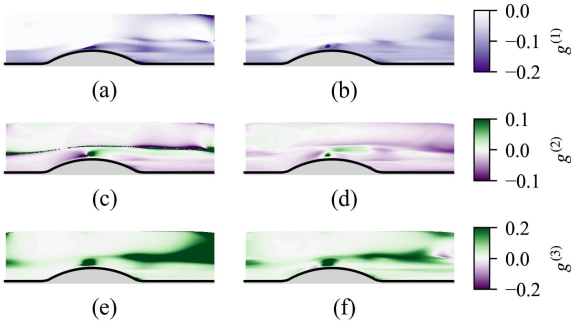


Figure D.16: Generalizability test on the parametric bumps test case for the neural network model trained on the parametric bump BUMP4 set.

Random forest

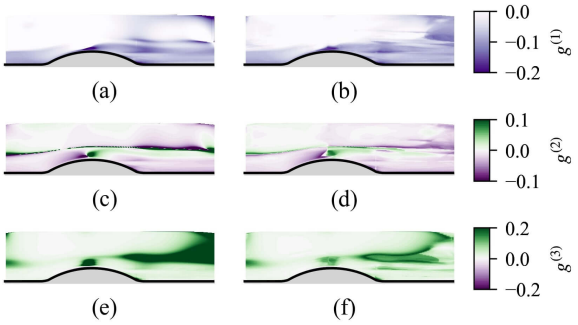


Figure D.17: Generalizability test on the parametric bumps test case for the random forest model trained on the parametric bump BUMP4 set.

XGBoost

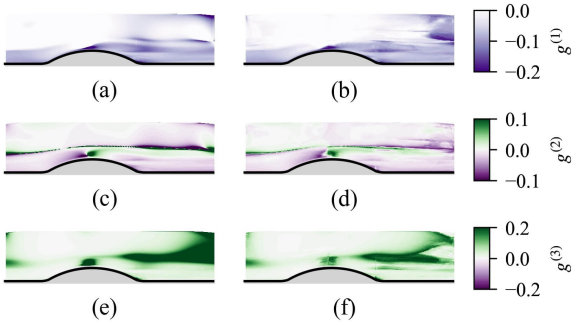


Figure D.18: Generalizability test on the parametric bumps test case for the XGBoost model trained on the parametric bump BUMP4 set.

D.2.3 DUCT15 training set

Neural network

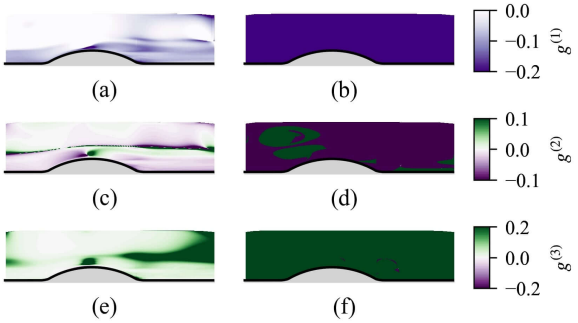


Figure D.19: Generalizability test on the parametric bumps test case for the neural network model trained on the square duct DUCT15 set.

Random forest

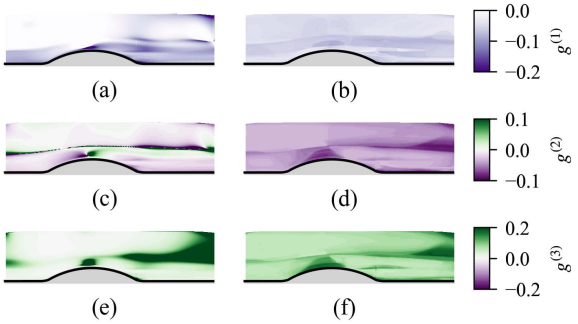


Figure D.20: Generalizability test on the parametric bumps test case for the random forest model trained on the square duct DUCT15 set.

XGBoost

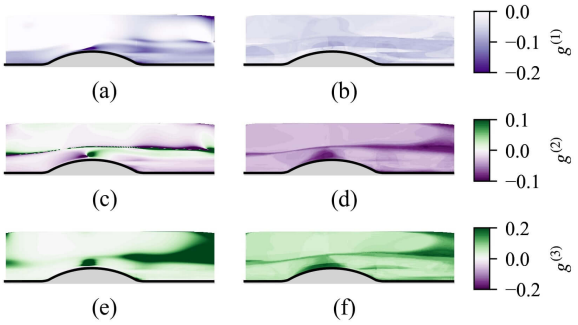


Figure D.21: Generalizability test on the parametric bumps test case for the XGBoost model trained on the square duct DUCT15 set.

D.2.4 FULL training set

Neural network

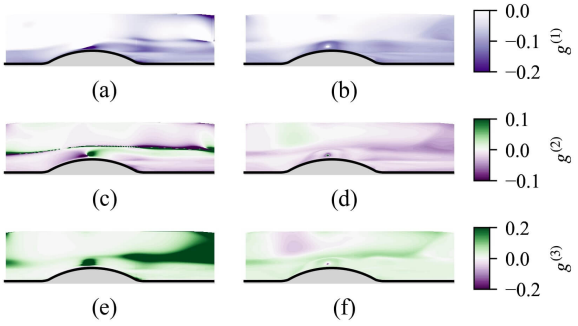


Figure D.22: Generalizability test on the parametric bumps test case for the neural network model trained on the FULL set.

Random forest

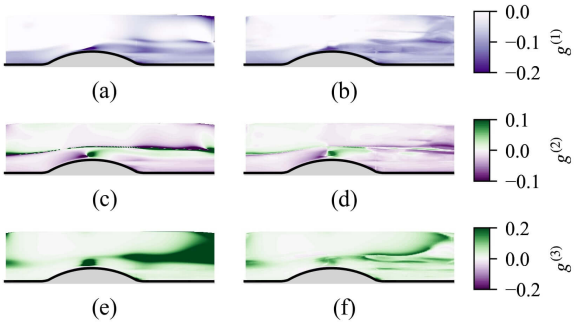


Figure D.23: Generalizability test on the parametric bumps test case for the random forest model trained on the FULL set.

XGBoost

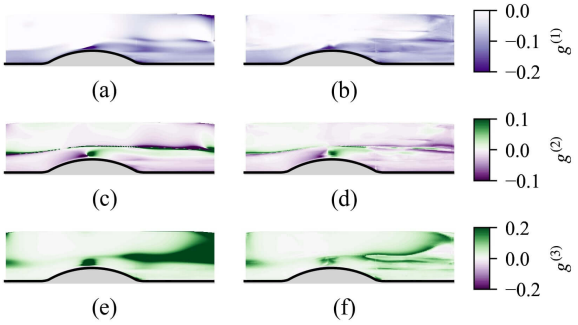


Figure D.24: Generalizability test on the parametric bumps test case for the XGBoost model trained on the FULL set.

D.3 Square duct test set

D.3.1 PHLL4 training set

Neural network

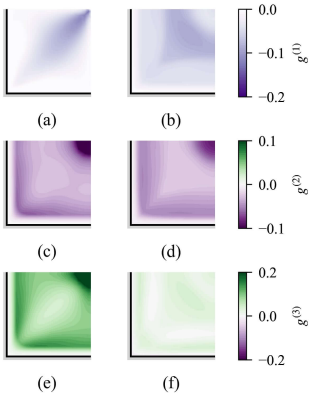


Figure D.25: Generalizability test on the square duct test case for the neural network model trained on the periodic hills PHLL4 set.

Random forest

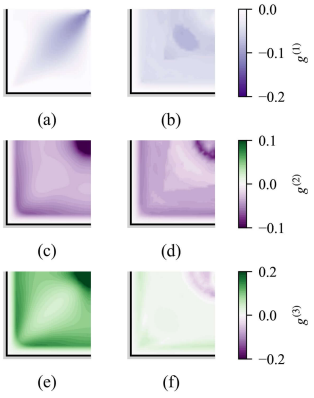


Figure D.26: Generalizability test on the square duct test case for the random forest model trained on the periodic hills PHLL4 set.

XGBoost

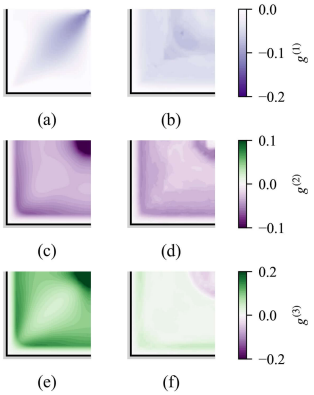


Figure D.27: Generalizability test on the square duct test case for the XGBoost model trained on the periodic hills PHLL4 set.

D.3.2 BUMP4 training set

Neural network

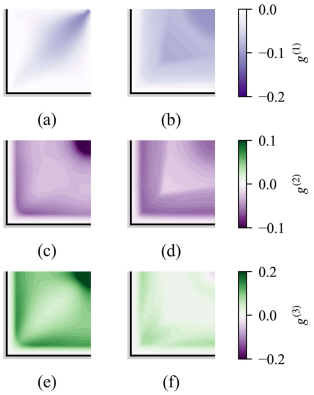


Figure D.28: Generalizability test on the square duct test case for the neural network model trained on the parametric bump BUMP4 set.

Random forest

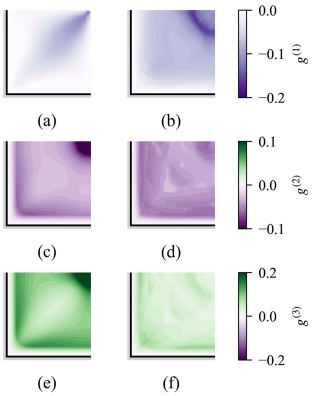


Figure D.29: Generalizability test on the square duct test case for the random forest model trained on the parametric bump BUMP4 set.

XGBoost

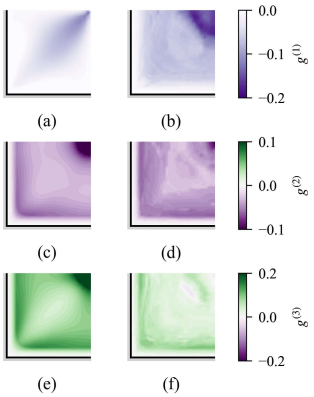


Figure D.30: Generalizability test on the square duct test case for the XGBoost model trained on the parametric bump BUMP4 set.

D.3.3 DUCT15 training set

Neural network

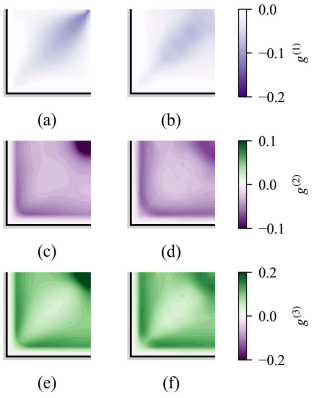


Figure D.31: Generalizability test on the square duct test case for the neural network model trained on the square duct DUCT15 set.

Random forest

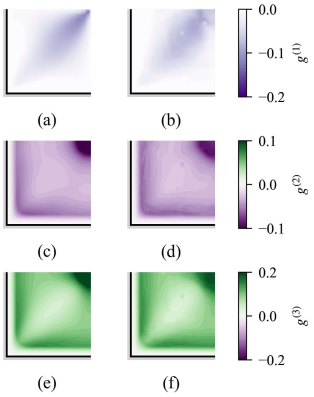


Figure D.32: Generalizability test on the square duct test case for the random forest model trained on the square duct DUCT15 set.

XGBoost

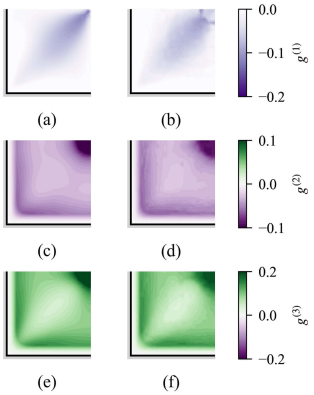


Figure D.33: Generalizability test on the square duct test case for the XGBoost model trained on the square duct DUCT15 set.

D.3.4 FULL training set

Neural network

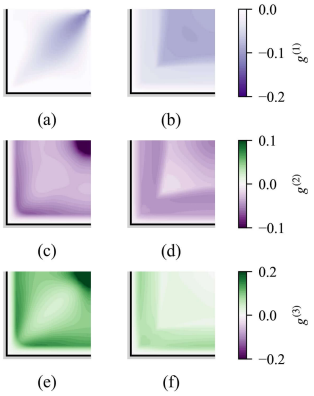


Figure D.34: Generalizability test on the square duct test case for the neural network model trained on the FULL set.

Random forest

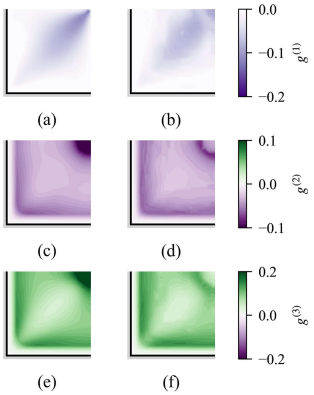


Figure D.35: Generalizability test on the square duct test case for the random forest model trained on the FULL set.

XGBoost

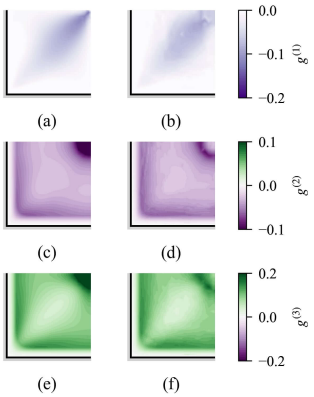


Figure D.36: Generalizability test on the square duct test case for the XGBoost model trained on the FULL set.

D.4 Converging-diverging channel test set

D.4.1 PHLL4 training set

Neural network

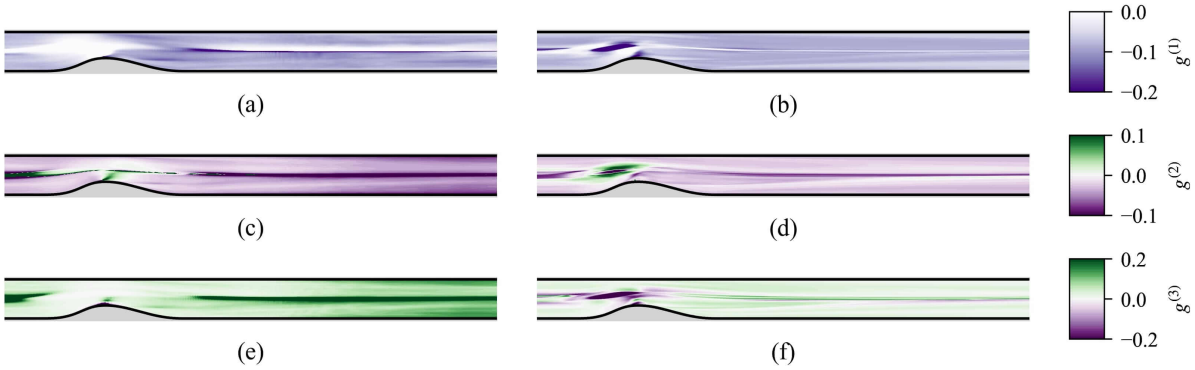


Figure D.37: Generalizability test on the converging-diverging channel test case for the neural network model trained on the periodic hills PHLL4 set.

Random forest

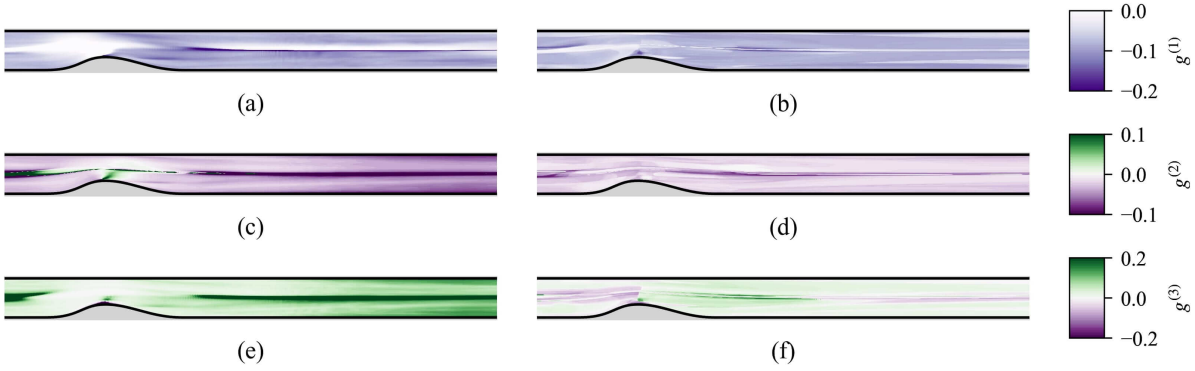


Figure D.38: Generalizability test on the converging-diverging channel test case for the random forest model trained on the periodic hills PHLL4 set.

XGBoost

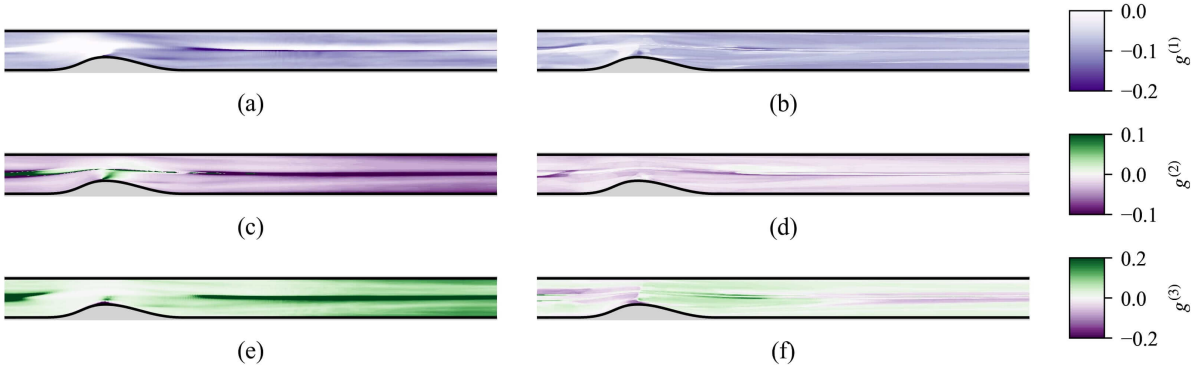


Figure D.39: Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the periodic hills PHLL4 set.

D.4.2 BUMP4 training set

Neural network

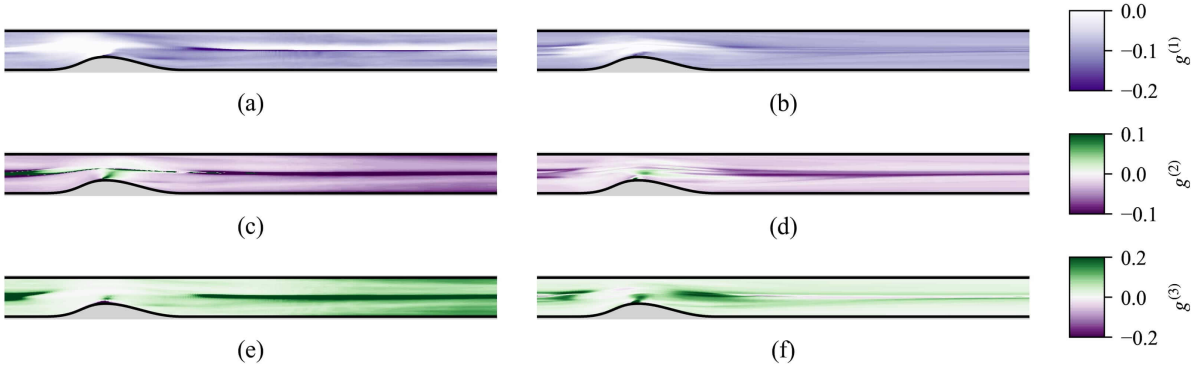


Figure D.40: Generalizability test on the converging-diverging channel test case for the neural network model trained on the parametric bump BUMP4 set.

Random forest

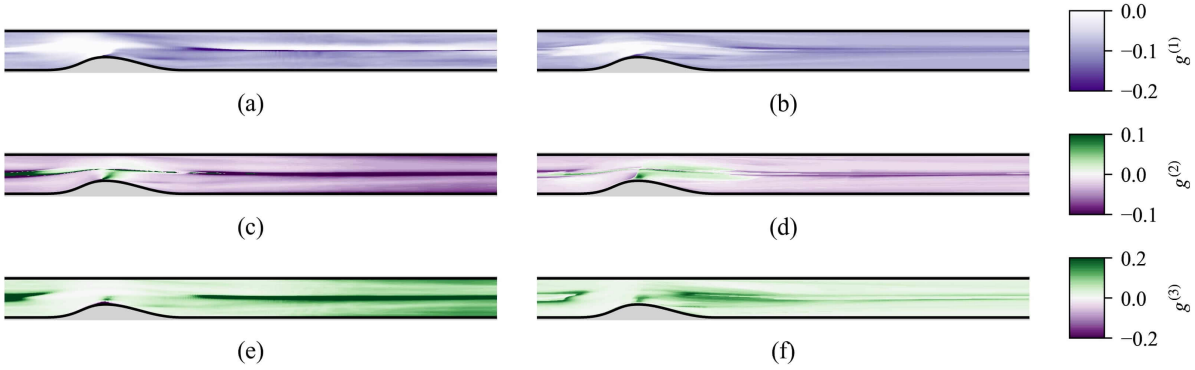


Figure D.41: Generalizability test on the converging-diverging channel test case for the random forest model trained on the parametric bump BUMP4 set.

XGBoost

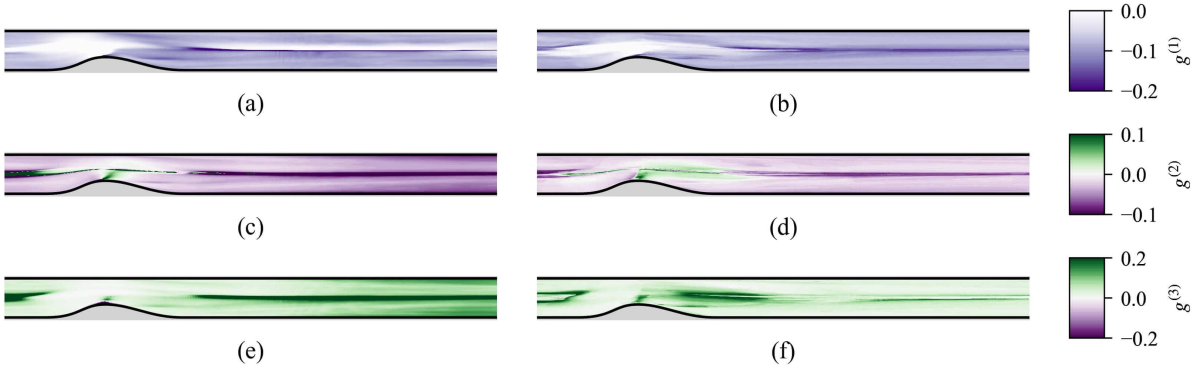


Figure D.42: Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the parametric bump BUMP4 set.

D.4.3 DUCT15 training set

Neural network

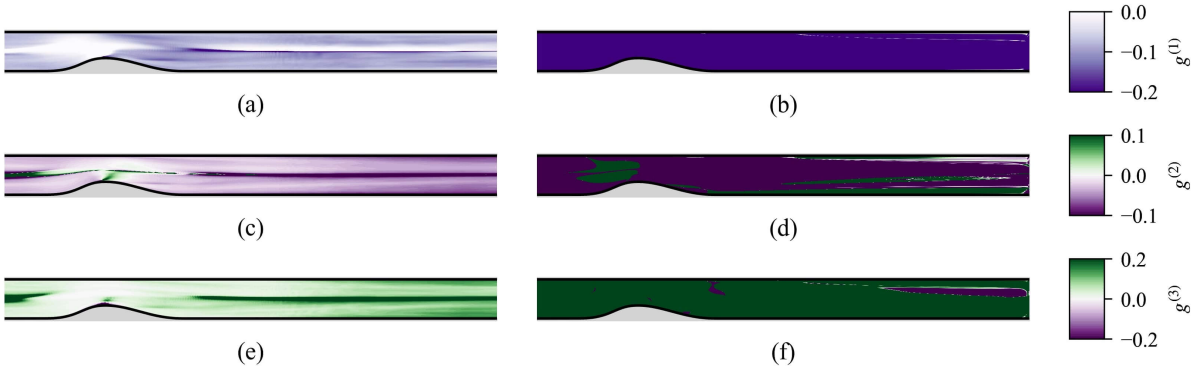


Figure D.43: Generalizability test on the converging-diverging channel test case for the neural network model trained on the square duct DUCT15 set.

Random forest

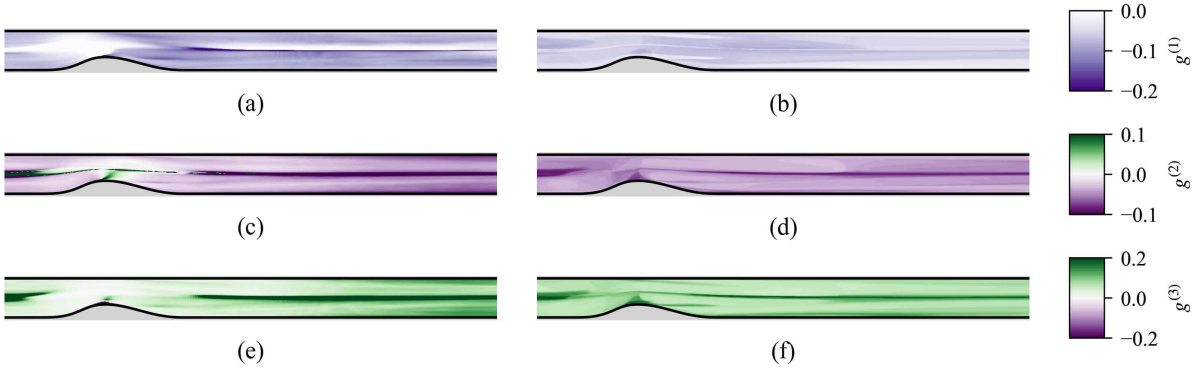


Figure D.44: Generalizability test on the converging-diverging channel test case for the random forest model trained on the square duct DUCT15 set.

XGBoost

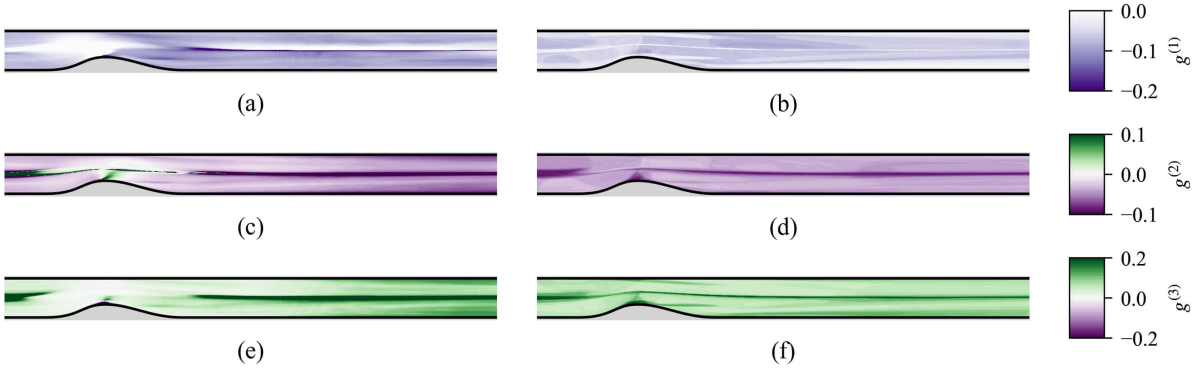


Figure D.45: Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the square duct DUCT15 set.

D.4.4 FULL training set

Neural network

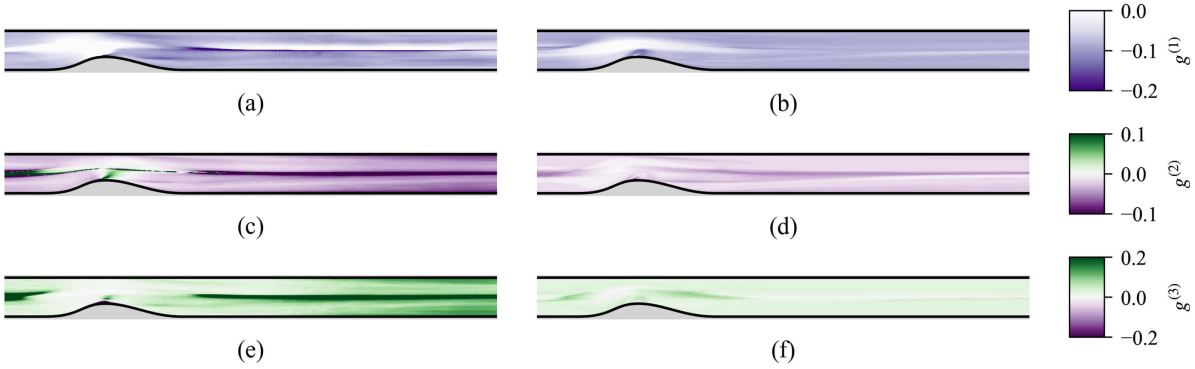


Figure D.46: Generalizability test on the converging-diverging channel test case for the neural network model trained on the FULL set.

Random forest

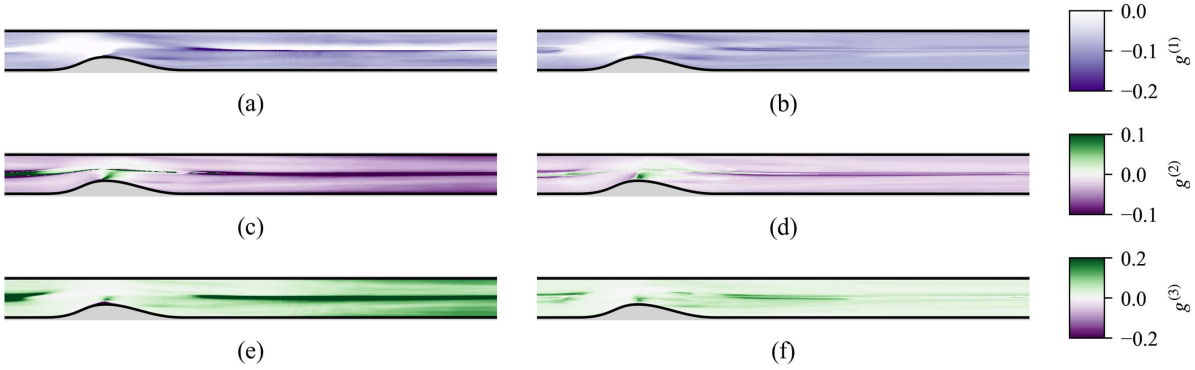


Figure D.47: Generalizability test on the converging-diverging channel test case for the random forest model trained on the FULL set.

XGBoost

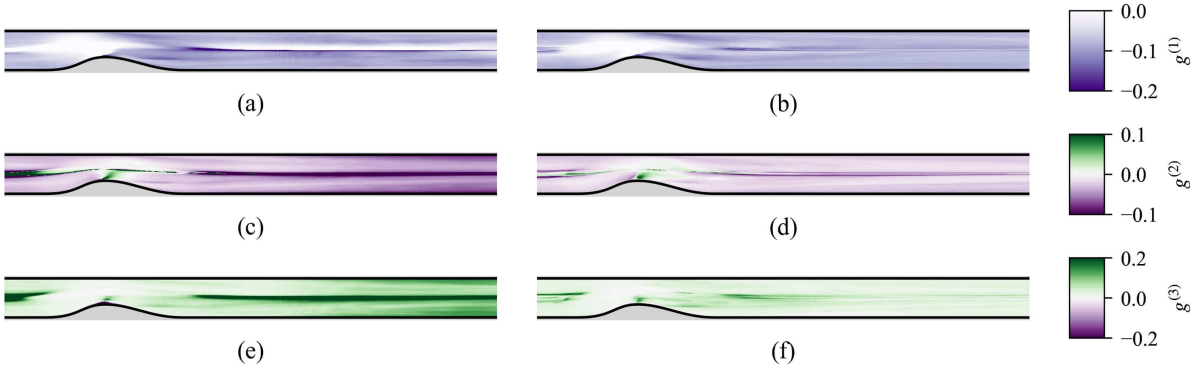


Figure D.48: Generalizability test on the converging-diverging channel test case for the XGBoost model trained on the FULL set.

D.5 Curved backward-facing step test set

D.5.1 PHLL4 training set

Neural network

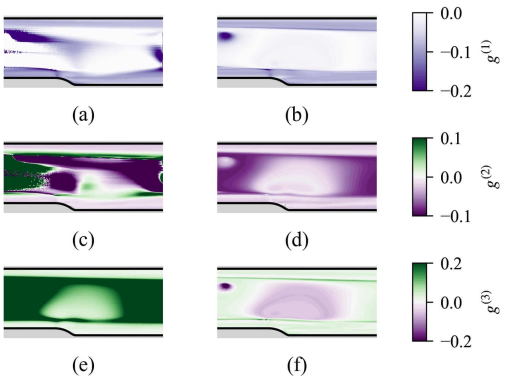


Figure D.49: Generalizability test on the curved backward-facing step test case for the neural network model trained on the periodic hills PHLL4 set.

Random forest

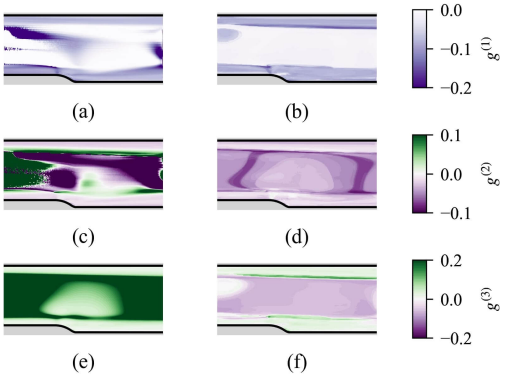


Figure D.50: Generalizability test on the curved backward-facing step test case for the random forest model trained on the periodic hills PHLL4 set.

XGBoost

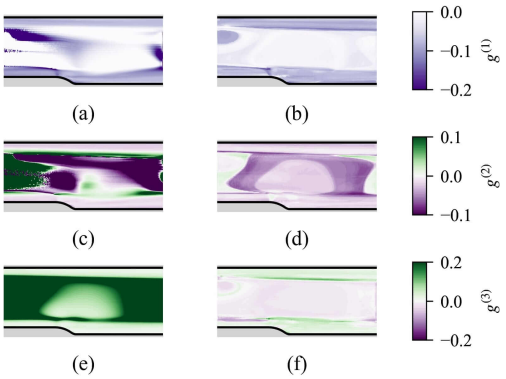


Figure D.51: Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the periodic hills PHLL4 set.

D.5.2 BUMP4 training set

Neural network

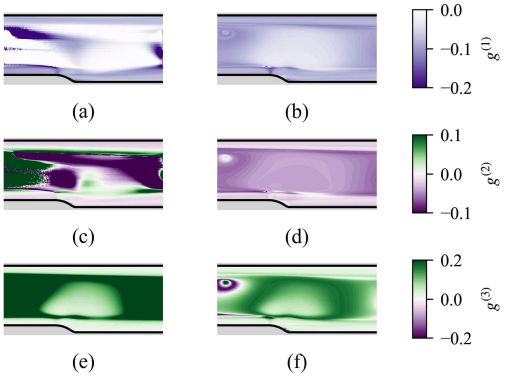


Figure D.52: Generalizability test on the curved backward-facing step test case for the neural network model trained on the parametric bump BUMP4 set.

Random forest

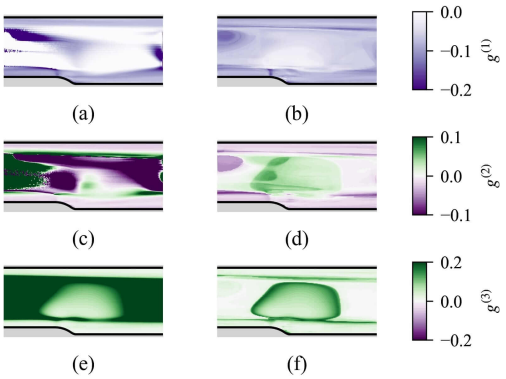


Figure D.53: Generalizability test on the curved backward-facing step test case for the random forest model trained on the parametric bump BUMP4 set.

XGBoost

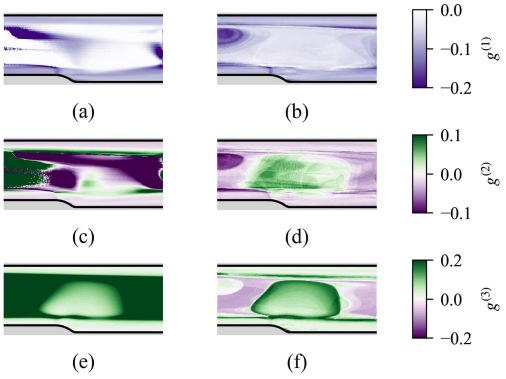


Figure D.54: Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the parametric bump BUMP4 set.

D.5.3 DUCT15 training set

Neural network

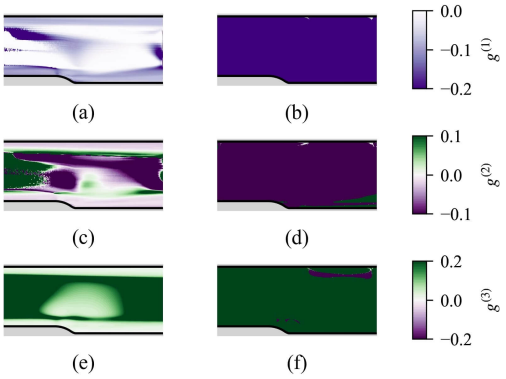


Figure D.55: Generalizability test on the curved backward-facing step test case for the neural network model trained on the square duct DUCT15 set.

Random forest

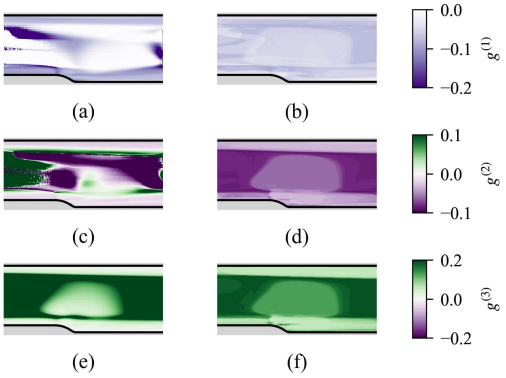


Figure D.56: Generalizability test on the curved backward-facing step test case for the random forest model trained on the square duct DUCT15 set.

XGBoost

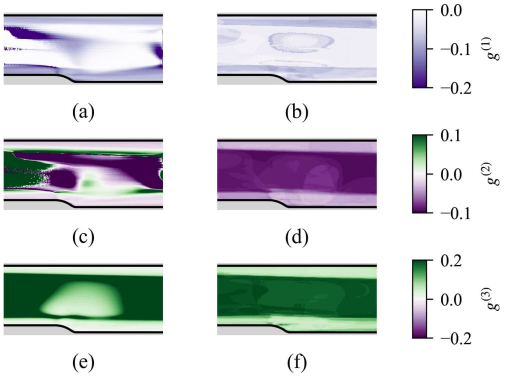


Figure D.57: Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the square duct DUCT15 set.

D.5.4 FULL training set

Neural network

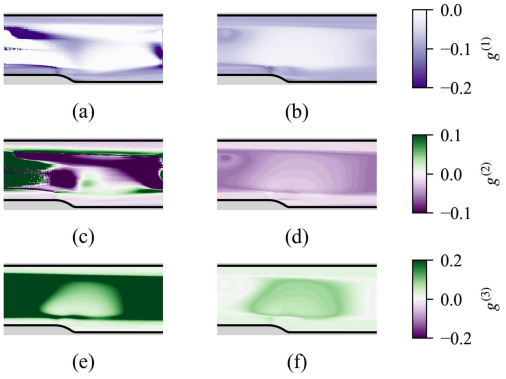


Figure D.58: Generalizability test on the curved backward-facing step test case for the neural network model trained on the FULL set.

Random forest

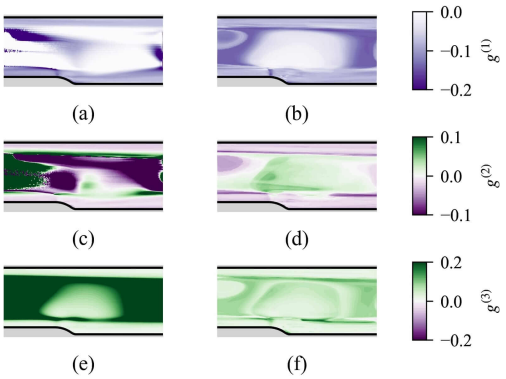


Figure D.59: Generalizability test on the curved backward-facing step test case for the random forest model trained on the FULL set.

XGBoost

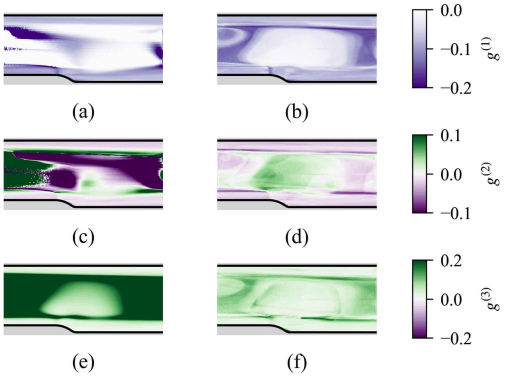


Figure D.60: Generalizability test on the curved backward-facing step test case for the XGBoost model trained on the FULL set.

Appendix E

Extended invariants of the basis tensors

Wu et al.'s integrity basis is derived from four gradient tensors: $\hat{S}$, $\hat{R}$, $\hat{A}_p$, and $\hat{A}_k$ [33]. These tensors are calculated as follows:

$$\hat{S}_{ij} = C_S \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) , \tag{E.1}$$

$$\hat{R}_{ij} = C_R \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i}\right) , \tag{E.2}$$

$$\hat{A}_p = C_{A_p} \epsilon_{ijl} \frac{\partial p}{\partial x_l} , \tag{E.3}$$

$$\hat{A}_k = C_{A_k} \epsilon_{ijl} \frac{\partial k}{\partial x_l} , \tag{E.4}$$

$$\tag{E.5}$$

where C_S, C_R, C_{A_p}, and C_{A_k} are scalars which non-dimensionalize their corresponding gradient tensors. For example, Ling et al. [23] chose $C_S = C_R = k/\varepsilon$, so that $\hat{S}$ is dimensionless.

Without loss of generality, the scalars p and k in the above can be swapped out for

other scalars, such as ε, ω, or ϕ [93], thereby replacing $\hat{A}_{p,ij}$ and $\hat{A}_{k,ij}$ with $\hat{C}_{ij}$ and $\hat{D}_{ij}$:

$$\hat{C}_{ij} = C_C \epsilon_{ijl} \frac{\partial s_1}{\partial x_l} \ , \tag{E.6}$$

$$\hat{D}_{ij} = C_D \epsilon_{ijl} \frac{\partial s_2}{\partial x_l} \ , \tag{E.7}$$

$$\tag{E.8}$$

where s_1 and s_2 are two scalar fields. For example, in the present study, $s_1 = \omega$, and $s_2 = k$. In three dimensional cartesian coordinates, the strain rate, rotation rate, and

antisymmetric tensors associated with the two scalar gradients are:

$$\hat{S}_{ij} = \frac{C_S}{2} \begin{bmatrix} 2\dfrac{\partial U_1}{\partial x_1} & \dfrac{\partial U_1}{\partial x_2} + \dfrac{\partial U_2}{\partial x_1} & \dfrac{\partial U_1}{\partial x_3} + \dfrac{\partial U_3}{\partial x_1} \\ \dfrac{\partial U_2}{\partial x_1} + \dfrac{\partial U_1}{\partial x_2} & 2\dfrac{\partial U_2}{\partial x_2} & \dfrac{\partial U_2}{\partial x_3} + \dfrac{\partial U_3}{\partial x_2} \\ \dfrac{\partial U_3}{\partial x_1} + \dfrac{\partial U_1}{\partial x_3} & \dfrac{\partial U_3}{\partial x_2} + \dfrac{\partial U_2}{\partial x_3} & 2\dfrac{\partial U_3}{\partial x_3} \end{bmatrix}, \tag{E.9}$$

$$\hat{R}_{ij} = \frac{C_R}{2} \begin{bmatrix} 0 & \dfrac{\partial U_1}{\partial x_2} - \dfrac{\partial U_2}{\partial x_1} & \dfrac{\partial U_1}{\partial x_3} - \dfrac{\partial U_3}{\partial x_1} \\ \dfrac{\partial U_2}{\partial x_1} - \dfrac{\partial U_1}{\partial x_2} & 0 & \dfrac{\partial U_2}{\partial x_3} - \dfrac{\partial U_3}{\partial x_2} \\ \dfrac{\partial U_3}{\partial x_1} - \dfrac{\partial U_1}{\partial x_3} & \dfrac{\partial U_3}{\partial x_2} - \dfrac{\partial U_2}{\partial x_3} & 0 \end{bmatrix}, \tag{E.10}$$

$$\hat{C}_{ij} = C_C \begin{bmatrix} 0 & -\dfrac{\partial s_1}{\partial x_3} & \dfrac{\partial s_1}{\partial x_2} \\ \dfrac{\partial s_1}{\partial x_3} & 0 & -\dfrac{\partial s_1}{\partial x_1} \\ -\dfrac{\partial s_1}{\partial x_2} & \dfrac{\partial s_1}{\partial x_1} & 0 \end{bmatrix}, \tag{E.11}$$

$$\hat{D}_{ij} = C_D \begin{bmatrix} 0 & -\dfrac{\partial s_2}{\partial x_3} & \dfrac{\partial s_2}{\partial x_2} \\ \dfrac{\partial s_2}{\partial x_3} & 0 & -\dfrac{\partial s_2}{\partial x_1} \\ -\dfrac{\partial s_2}{\partial x_2} & \dfrac{\partial s_2}{\partial x_1} & 0 \end{bmatrix}. \tag{E.12}$$

Under some conditions, input features derived from invariants of the minimal integrity basis derived by Wu et al. [33] can vanish. These conditions occur when there are zero-gradients in the flow, as all tensors in Wu's integrity basis are derived from gradient-based tensors.

Here, we consider two cases:

(I) Two-dimensional flow.

The zero pressure gradient boundary layer and periodic hills cases in the present study fall into this category. With the coordinate system defined as it was in Section 5.3.1, the velocity vector is $U_j = (U_1, U_2, 0)$, and the gradient tensors take the following form in three dimensional space:

$$\hat{S}_{ij} = \frac{C_S}{2} \begin{bmatrix} 2\dfrac{\partial U_1}{\partial x_1} & \dfrac{\partial U_1}{\partial x_2} + \dfrac{\partial U_2}{\partial x_1} & 0 \\ \dfrac{\partial U_2}{\partial x_1} + \dfrac{\partial U_1}{\partial x_2} & 2\dfrac{\partial U_2}{\partial x_2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{E.13}$$

$$\hat{R}_{ij} = \frac{C_R}{2} \begin{bmatrix} 0 & \dfrac{\partial U_1}{\partial x_2} - \dfrac{\partial U_2}{\partial x_1} & 0 \\ \dfrac{\partial U_2}{\partial x_1} - \dfrac{\partial U_1}{\partial x_2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{E.14}$$

$$\hat{C}_{ij} = C_C \begin{bmatrix} 0 & 0 & \dfrac{\partial s_1}{\partial x_2} \\ 0 & 0 & -\dfrac{\partial s_1}{\partial x_1} \\ -\dfrac{\partial s_1}{\partial x_2} & \dfrac{\partial s_1}{\partial x_1} & 0 \end{bmatrix}, \tag{E.15}$$

$$\hat{D}_{ij} = C_D \begin{bmatrix} 0 & 0 & \dfrac{\partial s_2}{\partial x_2} \\ 0 & 0 & -\dfrac{\partial s_2}{\partial x_1} \\ -\dfrac{\partial s_2}{\partial x_2} & \dfrac{\partial s_2}{\partial x_1} & 0\ . \end{bmatrix} \tag{E.16}$$

(II) Three-dimensional flow with zero gradients in one direction..

The square duct case in the present study falls into this category. With the coordinate system defined as it was in Section 5.3.1, the velocity vector is $U_j = (U_1, U_2, U_3)$. All gradients in the x_1 direction are zero: $\partial()/\partial x_1 = 0$. In this case, the gradient tensors

take the following form in three dimensional space:

$$\hat{S}_{ij} = \frac{C_S}{2} \begin{bmatrix} 0 & \frac{\partial U_1}{\partial x_2} & \frac{\partial U_1}{\partial x_3} \\ \frac{\partial U_1}{\partial x_2} & 2\frac{\partial U_2}{\partial x_2} & \frac{\partial U_2}{\partial x_3} + \frac{\partial U_3}{\partial x_2} \\ \frac{\partial U_1}{\partial x_3} & \frac{\partial U_3}{\partial x_2} + \frac{\partial U_2}{\partial x_3} & 2\frac{\partial U_3}{\partial x_3} \end{bmatrix}, \tag{E.17}$$

$$\hat{R}_{ij} = \frac{C_R}{2} \begin{bmatrix} 0 & \frac{\partial U_1}{\partial x_2} & \frac{\partial U_1}{\partial x_3} \\ \frac{\partial U_1}{\partial x_2} & 0 & \frac{\partial U_2}{\partial x_3} - \frac{\partial U_3}{\partial x_2} \\ \frac{\partial U_1}{\partial x_3} & \frac{\partial U_3}{\partial x_2} - \frac{\partial U_2}{\partial x_3} & 0 \end{bmatrix}, \tag{E.18}$$

$$\hat{C}_{ij} = C_C \begin{bmatrix} 0 & -\frac{\partial s_1}{\partial x_3} & \frac{\partial s_1}{\partial x_2} \\ \frac{\partial s_1}{\partial x_3} & 0 & 0 \\ -\frac{\partial s_1}{\partial x_2} & 0 & 0 \end{bmatrix}, \tag{E.19}$$

$$\hat{D}_{ij} = C_D \begin{bmatrix} 0 & -\frac{\partial s_2}{\partial x_3} & \frac{\partial s_2}{\partial x_2} \\ \frac{\partial s_2}{\partial x_3} & 0 & 0 \\ -\frac{\partial s_2}{\partial x_2} & 0 & 0\ . \end{bmatrix} \tag{E.20}$$

Cases **(I)** and **(II)** were analyzed using sympy [120] to determine which integrity basis tensor invariants are non-zero, and therefore suitable as potential input features. The first and second invariants of a rank two tensor are scalar functions, defined by:

$$I_1(A_{ij}) = A_{ii} \tag{E.21}$$

$$I_2(A_{ij}) = \frac{1}{2}\left((A_{ii})^2 - A_{ij}A_{ji}\right) \tag{E.22}$$

The third invariant, $I_3 = \mathbf{det}(A_{ij})$ is zero for all of the integrity basis tensors, since they are either antisymmetric, or symmetric and zero trace. These two invariants Table E.1 shows the results of this analysis.

Table E.1: Non-zero invariants of the minimal integrity basis tensor formed by $\hat{S}_{ij}$, $\hat{R}_{ij}$, $\hat{C}_{ij}$, and $\hat{D}_{ij}$.

Tensor	**Expression**	**(I)** $I_1 \neq 0$	**(I)** $I_2 \neq 0$	**(II)** $I_1 \neq 0$	**(II)** $I_2 \neq 0$
$B^{(1)}_{ij}$	$\hat{S}_{ik}\hat{S}_{kj}$	✓	✓	✓	✓
$B^{(2)}_{ij}$	$\hat{S}_{ik}\hat{S}_{kl}\hat{S}_{lj}$	—	✓	—	✓
$B^{(3)}_{ij}$	$\hat{R}_{ik}\hat{R}_{kj}$	✓	✓	✓	✓
$B^{(4)}_{ij}$	$\hat{C}_{ik}\hat{C}_{kj}$	✓	✓	✓	✓
$B^{(5)}_{ij}$	$\hat{D}_{ik}\hat{D}_{kj}$	✓	✓	✓	✓
$B^{(6)}_{ij}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lj}$	—	✓	—	✓
$B^{(7)}_{ij}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{S}_{mj}$	✓	✓	✓	✓
$B^{(8)}_{ij}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{R}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	✓	—	✓
$B^{(9)}_{ij}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{S}_{lj}$	✓	—	—	✓
$B^{(10)}_{ij}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{S}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B^{(11)}_{ij}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{S}_{lm}\hat{C}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	✓	✓
$B^{(12)}_{ij}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{S}_{lj}$	✓	—	—	✓
$B^{(13)}_{ij}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{S}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B^{(14)}_{ij}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{S}_{lm}\hat{D}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	✓	✓
$B^{(15)}_{ij}$	$\hat{R}_{ik}\hat{C}_{kj}$	—	—	✓	✓
$B^{(16)}_{ij}$	$\hat{C}_{ik}\hat{D}_{kj}$	✓	✓	✓	✓
$B^{(17)}_{ij}$	$\hat{R}_{ik}\hat{D}_{kj}$	—	—	✓	✓
$B^{(18)}_{ij}$	$\hat{R}_{ik}\hat{C}_{kl}\hat{S}_{lj}$	—	—	—	✓
$B^{(19)}_{ij}$	$\hat{R}_{ik}\hat{C}_{kl}\hat{S}_{lm}\hat{S}_{mj}$	—	—	✓	✓

(continued on next page)

Table E.1 (continued)

Tensor	**Expression**	**(I)** $I_1 \neq 0$	**(I)** $I_2 \neq 0$	**(II)** $I_1 \neq 0$	**(II)** $I_2 \neq 0$
$B_{ij}^{(20)}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{C}_{lm}\hat{S}_{mj}$	—	—	—	✓
$B_{ij}^{(21)}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{R}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B_{ij}^{(22)}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{C}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	—	—	—	✓
$B_{ij}^{(23)}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{R}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	—	—	—	✓
$B_{ij}^{(24)}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{C}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	—	✓
$B_{ij}^{(25)}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{S}_{lm}\hat{R}_{mn}\hat{S}_{no}\hat{S}_{oj}$	✓	—	✓	✓
$B_{ij}^{(26)}$	$\hat{R}_{ik}\hat{D}_{kl}\hat{S}_{lj}$	—	—	—	✓
$B_{ij}^{(27)}$	$\hat{R}_{ik}\hat{D}_{kl}\hat{S}_{lm}\hat{S}_{mj}$	—	—	✓	✓
$B_{ij}^{(28)}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{D}_{lm}\hat{S}_{mj}$	—	—	—	✓
$B_{ij}^{(29)}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{R}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B_{ij}^{(30)}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{D}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	—	—	—	✓
$B_{ij}^{(31)}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{R}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	—	—	—	✓
$B_{ij}^{(32)}$	$\hat{R}_{ik}\hat{R}_{kl}\hat{S}_{lm}\hat{D}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	—	✓
$B_{ij}^{(33)}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{S}_{lm}\hat{R}_{mn}\hat{S}_{no}\hat{S}_{oj}$	✓	—	✓	✓
$B_{ij}^{(34)}$	$\hat{C}_{ik}\hat{D}_{kl}\hat{S}_{lj}$	✓	—	—	✓
$B_{ij}^{(35)}$	$\hat{C}_{ik}\hat{D}_{kl}\hat{S}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B_{ij}^{(36)}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{D}_{lm}\hat{S}_{mj}$	—	—	✓	✓
$B_{ij}^{(37)}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{C}_{lm}\hat{S}_{mj}$	—	—	✓	✓
$B_{ij}^{(38)}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{D}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	—	—	—	✓
$B_{ij}^{(39)}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{C}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	—	—	—	✓
$B_{ij}^{(40)}$	$\hat{C}_{ik}\hat{C}_{kl}\hat{S}_{lm}\hat{D}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	✓	✓
$B_{ij}^{(41)}$	$\hat{D}_{ik}\hat{D}_{kl}\hat{S}_{lm}\hat{C}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	✓	✓

(continued on next page)

Table E.1 (continued)

Tensor	**Expression**	**(I)** $I_1 \neq 0$	**(I)** $I_2 \neq 0$	**(II)** $I_1 \neq 0$	**(II)** $I_2 \neq 0$
$B_{ij}^{(42)}$	$\hat{R}_{ik}\hat{C}_{kl}\hat{D}_{lj}$	✓	—	—	✓
$B_{ij}^{(43)}$	$\hat{R}_{ik}\hat{C}_{kl}\hat{D}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B_{ij}^{(44)}$	$\hat{R}_{ik}\hat{D}_{kl}\hat{C}_{lm}\hat{S}_{mj}$	✓	—	✓	✓
$B_{ij}^{(45)}$	$\hat{R}_{ik}\hat{C}_{kl}\hat{D}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	✓	—	—	✓
$B_{ij}^{(46)}$	$\hat{R}_{ik}\hat{D}_{kl}\hat{C}_{lm}\hat{S}_{mn}\hat{S}_{nj}$	✓	—	—	✓
$B_{ij}^{(47)}$	$\hat{R}_{ik}\hat{C}_{kl}\hat{S}_{lm}\hat{D}_{mn}\hat{S}_{no}\hat{S}_{oj}$	—	—	—	✓

Source code which supports the analysis in this appendix and enables further investigation is available on Github [121].