Symbolic Turbulence Modelling with Multi-Agent Reinforcement Learning

Zing Shawn Tan

Technische Universiteit Delft

TU Delft

Symbolic Turbulence Modelling with Multi-Agent Reinforcement Learning

by

Zing Shawn Tan

to obtain the degree of Master of Science Aerospace Engineering
at the Delft University of Technology,
to be defended publicly on Tuesday November 18, 2025 at 10:00 AM.

Student number:	5229286	
Project duration:	February 20, 2025 – October 31, 2025	
Thesis committee:	Dr. R.P. Dwight,	TU Delft, supervisor
	Dr. A. Eidi,	TU Delft, supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Cover image generated with Dall-E

TU Delft

Symbolic Turbulence Modelling with Multi- Agent Reinforcement Learning

Turbulence modelling remains a major challenge in computational fluid dynamics (CFD), as Reynolds-Averaged Navier-Stokes (RANS) closures rely on empirical relations introducing significant model-form uncertainty. Symbolic regression offers a path toward interpretable data-driven closures, but conventional Deep Symbolic Regression (DSR) struggles to efficiently explore the vast space of physically consistent expressions.

We propose a Multi-Agent Deep Symbolic Regression (MADSR) framework that reformulates turbulence model discovery as a cooperative multi-agent reinforcement learning (MARL) problem. Each agent discovers one scalar coefficient function in the tensor-basis expansion of the Reynolds-stress anisotropy tensor, sharing a common reward derived from frozen RANS evaluations. This cooperative setup promotes coordinated learning among model components.

In MADSR, the effectiveness of 2 MARL techniques, proximal policy optimization (PPO) and centralised training decentralised execution (CTDE), is investigated on symbolic turbulence modelling. Several MADSR variants are developed and tested, including a vanilla multi-agent DSR, a proximal policy optimization (PPO) based MADSR, an actor-critic MADSR, and a MAPPO-DSR inspired by multi-agent proximal policy optimization (MAPPO).

Applied to the Explicit Algebraic Reynolds-Stress Model (EARSM) and k-corrective RANS formulations, MADSR outperforms single-agent DSR in frozen RANS evaluations of the periodic-hill benchmark. The multi-agent structure enhances exploration efficiency and enables discovery of more consistent and interpretable turbulence closures. MADSR thus represents a promising step toward fully end-to-end, reinforcement-learning-based symbolic turbulence modelling.

Preface

After 5 years in Delft, I have finally completed both my Bachelor and Master's degrees in Aerospace Engineering. This thesis marks the end of my academic journey, and I am grateful for the knowledge and experiences I have gained along the way. These 5 years in Delft have been filled with many experiences, in which I have grown both academically and personally. Good or bad, these experiences have shaped me into the person I am today, and I will always cherish the memories I have made here.

Firstly, I would like to thank my parents, for their support allowing me to pursue my studies in the Netherlands. I would also like to thank my friends in Delft and back home for their companionship and support throughout these years. The great times we have shared together have made my time in Delft truly memorable.

I would like to express my gratitude to my main supervisor, Dr. Richard Dwight, for introducing me to this thesis topic and providing invaluable guidance throughout the research process. I would also like to thank my daily supervisor, Dr. Ali Eidi, for his support and time that he has dedicated to help me complete this thesis. The discussions and feedback with him have been instrumental in shaping the final outcome of this work. I would like to thank Tyler Buchanan for his assistance with providing the datasets I needed, and for his help in understanding certain aspects of the data.

Zing Shawn Tan
Delft, October 2025

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

List of Symbols

Symbol	Name	Description
u	Velocity	Velocity of the fluid at a point in space and time
$\bar{u}$	Mean velocity	Enseble-averaged velocity of the fluid
u'	Velocity fluctuations	Fluctuating component of velocity due to turbulence
p	Pressure	Pressure of the fluid at a point in space and time
$\bar{p}$	Mean pressure	Ensemble-averaged pressure of the fluid
p'	Pressure fluctuations	Fluctuating component of pressure due to turbulence
ρ	Density	Mass per unit volume of the fluid
k	Turbulent kinetic energy	Represents the energy in turbulent eddies
ν	Kinematic viscosity	Represents the fluid's resistance to flow, defined as $\nu = \frac{\mu}{\rho}$ where μ is dynamic viscosity and ρ is density
ν_t	Eddy viscosity	Represents the effective viscosity in turbulent flows
S_{ij}	Strain rate tensor	Represents the deformation of fluid elements in turbulence
Ω_{ij}	Rotation rate tensor	Represents the rotation of fluid elements in turbulence
ε	Dissipation rate	Rate at which turbulent kinetic energy is dissipated
ω	specific dissipation rate	Specific turbulence dissipation rate $\omega = \varepsilon/k$ in the $k-\omega$ model
τ_{ij}	Reynolds stress tensor	Additional stress due to turbulence in RANS equations
b_{ij}	Reynold stress anisotropy	Anisotropoic part of the Reynolds stress tensor in RANS equations
Δb_{ij}	Reynolds stress anisotropy correction	Correction term for the Reynolds stress anisotropy tensor
δ_{ij}	Kronecker delta	Used to represent the identity tensor in tensor equations
D_k	Turbulent destruction term	Represents the destruction of turbulent kinetic energy in turbulence models
P_k	Turbulent production term	Represents the production of turbulent kinetic energy in turbulence models
C_k	Turbulent convection term	Represents the convection of turbulent kinetic energy in turbulence models
α_{PH}	Geometric parameter	Parameter defining the geometry of the periodic hill
L_x/H	Domain length	Length of the periodic hill domain
L_y/H	Domain height	Height of the full periodic hill domain

$T_{ij}^{(n)}$	Basis tensors	Set of tensors used in tensor basis expansions for turbulence modelling
$\alpha^{(n)}$	Coefficient functions	Scalar functions of invariant flow features used in turbulence modelling
A	Advantage	Advantage of taking this action at this state
r_t	Reward	Immediate scalar feedback signal from the environment
R	Return	Total amount of rewards obtained durind an episode
γ	Discount factor	Factor by which future rewards are multiplied ($0 \leq \gamma \leq 1$)
s_t	State	Representation of the environment at a given time
τ	Trajectory	Full rollout of a policy
a_t	Action	Decision or move taken by the agent at a given time
L	Episode length	Total number of time steps in an episode
$\mathcal{L}$	Loss	Objective to be minimized during training
J	Objective function	Function to be maximized (e.g., expected return)
$V(s)$	State value function	Expected return starting from state s following policy π
$Q(s,a)$	Action value function	Expected return starting from state s, taking action a, and thereafter following policy π
$\pi(a\|s)$	Policy probabilities	Probability of taking action a in state s under policy π
θ	Actor parameters	Parameters of the policy network (actor)
ϕ	Critic parameters	Parameters of the value function network (critic)
ϵ	risk-seeking quantile	Top ϵ-th quantile of returns used in risk-seeking policy gradient updates
R_ϵ	risk-seeking baseline	Baseline return value corresponding to the ϵ quantile
y	True value	Ground truth target valueof the dataset for regression
$\hat{y}$	Predicted value	Model's predicted value for regression

List of Abbreviations

Abbreviation	Name
RANS	Reynolds-Averaged Navier-Stokes
DNS	Direct Numerical Simulation
LES	Large-Eddy Simulation
EVM	Eddy-Viscosity Model
SST	Shear Stress Transport
RST	Reynolds Stress Tensor
EARSM	Explicit Algebraic Reynolds-Stress Model
MDP	Markov Decision Process
POMDP	Partially Observable MDP
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
RNN	Recurrent Neural Network
TD	Temporal-Difference Learning
GAE	Generalised Advantage Estimation
TRPO	Trust Region Policy Optimisation
PPO	Proximal Policy Optimisation
DSR	Deep Symbolic Regression
MARL	Multi-Agent Reinforcement Learning
GP	Genetic Programming
uDSR	Unified Deep Symbolic Regression
MAPPO	Multi-Agent PPO
CTDE	Centralised Training, Decentralised Execution
TBNN	Tensor-Basis Neural Network
TBRF	Tensor-Basis Random Forest
SpaRTA	Sparse Regression of Turbulent Stress Anisotropy
RITA	Relative Importance Term Analysis
M-GEP	Multidimensional Gene Expression Programming
TBDSR	Tensor-Basis Deep Symbolic Regression

1 Introduction

Turbulence modelling remains one of the central challenges in computational fluid dynamics (CFD). Although the Reynolds-averaged Navier-Stokes (RANS) equations provide a computationally affordable way to simulate turbulent flows, they require closure models that introduce significant empirical uncertainty. Most RANS closures are derived from physical intuition and experimental calibration, which limit their generality and accuracy in complex flows involving strong pressure gradients, separation, or curvature effects. Improving the predictive capability of turbulence models, therefore, remains a key problem in CFD.

Recent advances in data-driven modelling have introduced new opportunities to overcome these limitations. High-fidelity datasets from direct numerical simulation (DNS) and large-eddy simulation (LES) provide detailed turbulence information that can be used to correct deficiencies in conventional models (Duraisamy et al., 2019). Symbolic regression has emerged as a promising approach for this task, as it allows the discovery of analytical expressions that are both interpretable and generalizable. Unlike black-box machine learning methods, symbolic regression yields explicit mathematical relations, making it well-suited for scientific modelling where interpretability and physical consistency are essential.

Deep Symbolic Regression (DSR) combines symbolic regression with deep reinforcement learning to automatically discover equations that fit data. In DSR, a recurrent neural network is trained to generate symbolic expressions as sequences of tokens, which are evaluated and rewarded based on their performance. However, the standard DSR formulation is limited to single-output regression tasks. This becomes restrictive when applied to turbulence modelling problems such as the Explicit Algebraic Reynolds-Stress Model (EARSM), where multiple coupled scalar functions must be learned simultaneously. In such cases, learning each equation independently can lead to inconsistencies and reduce model accuracy.

This work proposes a new framework called Multi-Agent Deep Symbolic Regression (MADSR), which extends DSR using multi-agent reinforcement learning (MARL) principles. In this formulation, each scalar coefficient function in the EARSM is represented by an individual agent, and all agents cooperate to minimise the overall model-form error in the predicted Reynolds-stress anisotropy tensor. All agents share a common reward derived from frozen RANS evaluations, encouraging coordinated learning across functions. This multi-agent approach allows for consistent, physically meaningful symbolic expressions to be discovered across all terms of the model.

The thesis investigates several variants of the MADSR framework, including a vanilla multi-agent DSR, an actor-critic MADSR, and a Multi-Agent Proximal Policy Optimization (MAPPO) DSR. Each variant is evaluated within the frozen RANS environment on the periodic-hill dataset, comparing its ability to discover accurate and interpretable turbulence model corrections. Through this, the work aims to address two major limitations of conventional DSR: the use of REINFORCE and the lack of coordination across multiple symbolic functions.

The objective of this research is to answer the following question:

How can multi-agent reinforcement learning techniques be integrated with deep symbolic regression to perform turbulence modelling?

This research aims to develop a novel framework that integrates multi-agent reinforcement learning (MARL) with deep symbolic regression (DSR) to address the challenges of multi-equation turbulence modelling. Key objectives include identifying suitable MARL algorithms for cooperative symbolic regression, evaluating the impact of MARL integration on learning efficiency and expression quality, assessing the framework's ability to discover EARSM closures and k-corrective RANS models, and determining whether the approach can produce accurate and interpretable turbulence models.

Thesis structure

The thesis begins with a background chapter that introduces the fundamental ideas required to understand this work. It also presents related work that has been done. The RANS equations are presented along with the closure problem and the motivation for turbulence models. The formulation of the k-ω SST model is discussed to illustrate how traditional eddy-viscosity models are derived and the limitations they introduce. The concept of model-form error is then explained, followed by a discussion of the EARSM and its tensor-basis representation. Then, reinforcement learning is introduced, covering basic reinforcement learning algorithms. The chapter then discusses DSR in detail, explaining how it combines symbolic regression with reinforcement learning to discover equations from data. Finally, the chapter reviews the multi-agent reinforcement learning concepts that form the foundation of this work.

The methodology chapter presents the design of the MADSR framework. It begins by outlining the limitations of standard DSR and motivates the multi-agent extension. It then describes how the EARSM is represented as a multi-agent environment, where each agent learns one coefficient function and all agents share a common reward based on their collective performance in frozen RANS. Different algorithmic variants are introduced, including the vanilla, actor-critic, and MAPPO-based MADSR formulations, along with the general training procedure and evaluation setup. Finally, the chapter details the experimental setup used to train and test the MADSR algorithms developed.

The results chapter then analyses the performance of these algorithms in learning turbulence model corrections. The discovered symbolic models are compared in terms of their predictive accuracy, interpretability, and consistency across agents. Additional studies, such as ablation tests and flow-field analyses, are used to examine the influence of different components and to assess how the learned models reproduce key flow features.

2

Background and related work

This chapter lays the foundation for the methods developed in this thesis, and it also provides some related work in the field of turbulence modelling and data-driven model discovery. It begins with the Reynolds-Averaged Navier-Stokes equations and standard turbulence closures, explaining the limitations of linear eddy-viscosity models and the motivation for model-form error corrections such as EARSM and the k-corrective approach. The k-ω SST model is presented as a representative two-equation turbulence model. The chapter then introduces data-driven correction methods, including tensor-basis learning and sparse symbolic approaches such as SpaRTA and RITA. This is followed by an overview of symbolic regression and its deep learning extension, Deep Symbolic Regression, where reinforcement learning methods are used to discover analytical expressions. The chapter concludes with a discussion of multi-agent reinforcement learning and recent extensions of DSR, forming the basis for the multi-agent deep symbolic regression framework proposed in this work.

2.1. RANS Turbulence Modelling

The incompressible viscous Navier-Stokes equations are described by the following equations:

$$\begin{aligned} \frac{\partial u_i}{\partial x_i} &= 0 \\ \frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} &= -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} \end{aligned} \tag{2.1}$$

These equations fully describe the fluid motion and can, in theory, be solved exactly through Direct Numerical Simulation (DNS), resulting in a highly accurate representation of the real physical flow. However, due to the prohibitive computational cost associated with resolving all scales of fluid motion, DNS is typically infeasible for practical engineering applications. Therefore, approximation methods such as Large Eddy Simulation (LES) and Reynolds-averaged Navier-Stokes (RANS) equations have been developed and widely adopted as computationally efficient alternatives.

RANS is a method that averages the Navier-Stokes equations over ensembles, where the velocity u_i and pressure p are decomposed into a mean part $\bar{u}_i$ and a fluctuating part u_i'. This results in the Reynolds-averaged Navier-Stokes equations:

$$\begin{aligned} \frac{\partial \bar{u}_i}{\partial x_i} &= 0 \\ \frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} &= -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{u}_i}{\partial x_j^2} - \frac{\partial \tau_{ij}}{\partial x_j}. \end{aligned} \tag{2.2}$$

Ensemble averaging eliminates the effect of turbulent fluctuations, and yields governing equations for the mean flow that look almost identical to the full Navier-Stokes equations. However, Equation 2.2

introduces an additional term, the Reynolds stress tensor $\tau_{ij} = \overline{u_i' u_j'}$, which represents the effect of the turbulent fluctuations on the mean flow. The Reynolds stress tensor is unknown, making the RANS equations an unclosed system of equations. To close the system, a turbulence model is required to model the Reynolds stress tensor.

Commonly, the Reynolds stress tensor is separated into an isotropic contribution ($\frac{2}{3}k\delta_{ij}$) from an anisotropic contribution b_{ij}:

$$\tau_{ij} = 2k\left(b_{ij} - \frac{1}{3}\delta_{ij}\right). \tag{2.3}$$

Here k is the turbulent kinetic energy, and δ_{ij} is the Kronecker delta. The linear eddy viscosity model (LEVM) is commonly used to model the anisotropic term, where the anisotropy tensor b_{ij} is defined to be linear in the strain rate tensor S_{ij}. This is known as Boussinesq's hypothesis by Boussinesq, 1877, which states that the Reynolds stress tensor is proportional to the strain rate tensor. The anisotropy tensor b_{ij} is defined as:

$$b_{ij} = -\frac{\nu_t}{k}S_{ij} \quad where \quad S_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \tag{2.4}$$

Substituting the eddy-viscosity model into the RANS equations:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j\frac{\partial \bar{u}_i}{\partial x_j} = -\frac{\partial}{\partial x_i}\left(\frac{\bar{p}}{\rho} + \frac{2}{3}k\right) + \frac{\partial}{\partial x_j}\left(2(\nu + \nu_t)S_{ij}\right). \tag{2.5}$$

The eddy viscosity model suggests that the effect of turbulence can be modelled as an additional viscosity term ν_t, where ν_t is the eddy viscosity. The eddy viscosity represents the additional momentum transport due to turbulent fluctuations. Determining the eddy viscosity ν_t, therefore, becomes the central task of the closure. RANS solvers obtain the eddy viscosity from auxiliary transport equations that relate ν_t to characteristic turbulence time and length-scales.

2.2. Two equation models

To model the eddy viscosity ν_t, two-equation turbulence models solve two additional transport equations for two turbulence variables. Here, the most common two-equation turbulence models, the k-ε, k-ω, and k-ω SST models are presented. These models model the eddy viscosity based on the turbulence kinetic energy k (TKE) and a second variable that represents the turbulence dissipation.

2.2.1. Turbulence kinetic energy

The turbulence kinetic energy is the mean kinetic energy per unit mass in the turbulent fluctuations. It is defined as the trace of the Reynolds stress tensor divided by two $k = trace(\tau_{ij})/2$, which can be expressed as:

$$k = \frac{1}{2}\overline{u_i' u_i'} = \frac{1}{2}(u'^2 + v'^2 + w'^2). \tag{2.6}$$

The turbulence kinetic energy forms the basis of many turbulence models, as it represents the intensity of turbulence in a flow. Thus, equations to describe the behaviour of the turbulence kinetic energy are essential for turbulence modelling. A transport equation for the turbulence kinetic energy can be derived from the Reynolds stress transport (RST) equations. The Reynolds stress transport equations are obtained by multiplying the fluctuating velocity components u_i' and u_j' by the incompressible RANS equations. The resulting transport equation for the Reynolds stress tensor contains terms representing production, dissipation, pressure-strain, and diffusion of the Reynolds stresses. By setting $i = j$ and summing over all components, a transport equation for the turbulence kinetic energy can be derived:

$$\frac{Dk}{Dt} = \underbrace{\tau_{ij}\frac{\partial \bar{u}_i}{\partial x_j}}_{\text{Production}} + \underbrace{\frac{\partial}{\partial x_j}\left(-\frac{1}{2}\overline{u_i' u_i' u_j'} - \frac{1}{\rho}\overline{p' u_j'} + \frac{1}{Re}\frac{\partial k}{\partial x_j}\right)}_{\text{Diffusion}} + \underbrace{\frac{1}{Re}\overline{\frac{\partial u_i'}{\partial x_j}\frac{\partial u_i'}{\partial x_j}}}_{\text{Dissipation}}. \tag{2.7}$$

While the exact transport equation for the turbulence kinetic energy can be derived from the RST equations, it contains several unclosed terms that require modelling. The production term contains the Reynolds stress tensor, which has already been modelled using the eddy-viscosity model. The first two terms in the diffusion term are called the turbulent transport and pressure diffusion. These 2 terms contain higher-order correlations of the fluctuating velocity and pressure fields, and are modelled using another eddy viscosity assumption, where they are assumed to be proportional to the gradient of the turbulence kinetic energy. The model is shown:

$$-\frac{1}{2}\overline{u_i' u_i' u_j'} - \frac{1}{\rho}\overline{p' u_j'} \approx \frac{\nu_t}{Pr_k}\frac{\partial k}{\partial x_j}. \tag{2.8}$$

Where Pr_k is the turbulent Prandtl number for k. The dissipation term is where the 2 equation turbulence models differ, and is the final term that needs to be modelled. The dissipation term represents the rate at which the turbulence kinetic energy is converted into thermal internal energy due to viscous effects. So for now, the dissipation term is denoted as ε.

2.2.2. The k-ε model

The k-ε model by Jones and Launder, 1972 is one of the most widely used two-equation turbulence models. The $k-\varepsilon$ model works under the assumption that the dissipation and production of turbulence kinetic energy are in equilibrium, so TKE does not increase or decrease over time. This leads to the following definition of the eddy viscosity:

$$\nu_t = C_\mu \frac{k^2}{\varepsilon} \tag{2.9}$$

Where C_μ is a model constant with a typical value of 0.09. ε remains unknown, so a second transport equation, the ε transport equation, is derived to model the behaviour of ε. This is done by postulating ε onto the k transport equation. The resulting transport equation for ε is shown in Equation 2.10.

$$\frac{D\varepsilon}{Dt} = C_{\varepsilon 1}\frac{\varepsilon}{k}\tau_{ij}\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_k}\right)\frac{\partial \varepsilon}{\partial x_j}\right] - C_{\varepsilon 2}\frac{\varepsilon^2}{k} \tag{2.10}$$

Where $C_{\varepsilon 1}$, $C_{\varepsilon 2}$ are model constants, and Pr_ε is the turbulent Prandtl number for ε. The assumption that the dissipation and production of turbulence kinetic energy are in equilibrium is valid for fully developed turbulent flows. However, the assumption does not hold in flows with strong pressure gradients and flow separation, so the k-ε model tends to perform well for external flows without separation. This is the main limitation of the k-ε model.

2.2.3. The k-ω model

The k-ω model by Wilcox, 1988 uses the specific dissipation rate ω instead of ε as the second variable. The specific dissipation rate is defined as the dissipation rate per unit turbulence kinetic energy, and thus the parameters in the k-ω model are:

$$\omega = \frac{\varepsilon}{C_\mu k}, \quad \nu_t = \frac{k}{\omega}. \tag{2.11}$$

A transport equation for ω is derived similarly to the ε transport equation, by postulating ω onto the k transport equation. The resulting transport equation for ω is:

$$\frac{D\omega}{Dt} = \alpha_\omega \frac{\omega}{k}\tau_{ij}\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_\omega}\right)\frac{\partial \omega}{\partial x_j}\right] - \beta\omega^2. \tag{2.12}$$

Where α_ω, β are model constants, and Pr_ω is the turbulent Prandtl number for ω. The k-ω model, while very similar in method to the k-ε model, is quite different in performance. It performs well in near-wall regions and adverse pressure gradient flows where there is flow separation. However, the k-ω model is sensitive to the freestream value of ω, which is an arbitrary value that must be specified at the boundary. This sensitivity can lead to significant variations in the predicted flow field, especially in free shear flows and external aerodynamics applications.

2.2.4. The $k-\omega$ SST model

The k-ω shear-stress transport (SST) model was introduced by Menter, 1994 to improve the predictive accuracy of two-equation eddy-viscosity models for engineering applications, particularly for flows involving strong adverse pressure gradients and flow separation. The model was derived through a combination of the most robust features of existing turbulence models: the near-wall accuracy and numerical stability of the k-ω model, and the freestream independence and performance in shear flows of the standard k-ε model.

The first step in Menter's formulation was the construction of a baseline (k-ω BSL) model that smoothly blends the k-ω and k-ε models. The k-ω formulation is retained in the inner and logarithmic regions of the boundary layer, while the model transitions to a k-ε formulation in the wake region and in free shear flows. This blending avoids the strong freestream sensitivity of the original k-ω model, which depends heavily on the arbitrary freestream value of ω. The transport equations for the k-ω SST model are:

$$\begin{aligned}\frac{Dk}{Dt} &= \hat{P_k} + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_k}\right)\frac{\partial k}{\partial x_j}\right] - \beta^* k\omega \\ \frac{D\omega}{Dt} &= \alpha\frac{\omega}{k}\hat{P_k} + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_\omega}\right)\frac{\partial \omega}{\partial x_j}\right] - \underbrace{\beta\omega^2 + 2(1-F_1)\frac{\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}}_{\text{Cross Diffusion Term}}.\end{aligned} \tag{2.13}$$

Here, $\hat{P_k} = \min(P_k, 10\beta^* k\omega)$ is a modified production term that limits the production of turbulence kinetic energy in stagnation regions, where P_k is the standard production term found in the other models. The model coefficients α, β, β^*, $\sigma_{\omega 2}$, and Pr_ω are blended between their k-ω and k-ε values using the function F_1. Both k-ω and k-ε model coefficients share the same set of coefficients, but have different values for them. Consider that the set of model coefficients is denoted as ϕ, then the blended coefficient is defined as:

$$\phi = F_1\phi_{k-\omega} + (1-F_1)\phi_{k-\varepsilon} \tag{2.14}$$

The k-ω SST model blends the two models using the cross diffusion term, which is the last term in the ω transport equation of Equation 2.13. When $F_1 = 1$, the term is inactive, and the ω transport equation reduces to the standard k-ϵ formulation. When $F_1 = 0$, the term becomes active, introducing a cross-diffusion effect that is characteristic of the k-ϵ model. F_1 is defined based on the distance to the nearest wall and the local flow variables, ensuring that it transitions smoothly from one model to the other based on the flow conditions. F_1 is equal to one in the viscous sublayer and logarithmic region of the boundary layer, where the k-ω model is most accurate, and zero in the wake region and free shear flows, where the k-ε model performs better. This controls the effect of the blending and ensures that the model behaves appropriately in different flow regimes.

Bradshaw's relation (Huang et al., 1992) suggests that the principal turbulent shear stress $\tau_{ij} = -\overline{\rho u'v'}$ is proportional to the turbulent kinetic energy k in the wake region of boundary layers. Standard two-equation eddy-viscosity models violate this relationship because they define the shear stress as $\tau_t = \rho\nu_t \partial U/\partial y$, which scales with the local strain rate rather than k. In adverse pressure gradient flows, this causes an overprediction of the turbulent shear stress.

To correct this, Menter introduced the Shear Stress Transport (SST) formulation (Menter et al., 2003), which modifies the eddy-viscosity definition to account for the effects of strong adverse pressure gradients. Menter modified the eddy-viscosity definition to limit its value in regions where production exceeds dissipation, thereby enforcing Bradshaw's relation:

$$\nu_t = \frac{a_1 k}{\max(a_1\omega, \Omega F_2)}, \tag{2.15}$$

Where Ω is the vorticity magnitude and F_2 is another blending function that activates the correction only within boundary layers ($F_2 = 1$ near walls and $F_2 = 0$ in free shear layers). This modification reduces the turbulent viscosity in adverse pressure gradient regions, improving the model's ability to predict flow separation.

k-ω SST Model

Turbulent Kinetic Energy Equation:

$$\frac{Dk}{Dt} = \hat{P_k} + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_k}\right)\frac{\partial k}{\partial x_j}\right] - \beta^* k\omega \tag{2.16}$$

Dissipation Rate Equation:

$$\frac{D\omega}{Dt} = \alpha\frac{\omega}{k}\hat{P_k} - \beta\omega^2 + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_\omega}\right)\frac{\partial \omega}{\partial x_j}\right] + 2(1-F_1)\frac{\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}. \tag{2.17}$$

Eddy Viscosity Definition:

$$\nu_t = \frac{a_1 k}{\max(a_1\omega, \Omega F_2)}, \tag{2.18}$$

Model Coefficients:
The coefficients of the k-ω and k-ε models are denoted as $\phi_{k-\omega}$ and $\phi_{k-\varepsilon}$ and are defined as:

$$\phi_{k-\varepsilon}: \quad \alpha = 0.44, \quad \beta = 0.0828, \quad \beta^* = 0.09, \quad \sigma_{\omega 2} = 1.0, \quad Pr_\omega = 0.856 \tag{2.19}$$

$$\phi_{k-\omega}: \quad \alpha = 5/9, \quad \beta = 3/40, \quad \beta^* = 0.09, \quad \sigma_{\omega 2} = 0.856, \quad Pr_\omega = 0.5 \tag{2.20}$$

Then the blended coefficient ϕ is defined as:

$$\phi = F_1\phi_{k-\omega} + (1-F_1)\phi_{k-\varepsilon} \tag{2.21}$$

Auxiliary Equations:

$$\hat{P_k} = \min(P_k, 10\beta^* k\omega) \tag{2.22}$$

$$F_1 = \tanh\left[\left(\min\left(\max\left(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\sigma_{\omega 2}k}{CD_{k\omega}y^2}\right)\right)^4\right] \tag{2.23}$$

$$CD_{k\omega} = \max\left[2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 10^{-10}\right] \tag{2.24}$$

$$F_2 = \tanh\left[\left(\max\left(\frac{2\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right)\right)^2\right] \tag{2.25}$$

2.3. Model-form error correction

High-fidelity DNS and LES studies reveal that the linear stress-strain assumption does not reproduce the true anisotropy tensor b_{ij} of turbulent flows. Pope, 1975 quantified the resulting model-form error as the difference between the DNS/LES tensor and its LEVM counterpart. The error, Δb_{ij}, is defined as:

$$\Delta b_{ij} = b_{ij}^{\text{DNS/LES}} - b_{ij}^{\text{LEVM}} = b_{ij}^{\text{DNS/LES}} + \frac{\nu_t}{k}S_{ij} \tag{2.26}$$

Where $b_{ij}^{\text{LEVM}} = -\nu_t S_{ij}/k$ has been substituted in the final expression for clarity.

In Pope, 1975, a non-linear correction term is proposed to correct the model-form error in the Reynolds-stress tensor. It is based on the assumption that the Reynolds stress tensor depends not only on the strain rate tensor S_{ij} but also on the rotation rate tensor Ω_{ij}, which is defined as:

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \quad \Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right). \tag{2.27}$$

Based on the Cayley-Hamilton theorem, any objective, symmetric, traceless tensor function of S_{ij} and Ω_{ij} can be expressed as a linear combination of ten invariant basis tensors $T_{ij}^{(n)}$. Hence, the model-form error reduces to the compact expansion shown in Equation 2.28.

$$\Delta b_{ij} = \sum_{n=1}^{10} \alpha^{(n)}(I_1, \ldots I_5) {T_{ij}}^{(n)} \tag{2.28}$$

This provided a way to correct the model-form error in the Reynolds-stress tensor, where the coefficients $\alpha_{(n)}$ are functions of the invariants $I_1, .., I_5$ of the strain rate and rotation rate tensors. Thus, the objective is to find the functions $\alpha^{(n)}$ that best fit the data. This is called an explicit algebraic Reynolds-stress model (EARSM), as it provides an explicit expression for the Reynolds-stress tensor in terms of the strain rate and rotation rate tensors.

Data-driven approaches to find an EARSM have been proposed, where the goal is to find the functions $\alpha^{(n)}$ that best fit the data. These approaches typically use frozen RANS, where already available high-fidelity data is used to train the model. The data is typically obtained from DNS or LES simulations, where the Reynolds-stress tensor is computed from the velocity field. The invariants $I_1, ..., I_5$ are computed from the strain rate and rotation rate tensors, and the functions $\alpha^{(n)}$ are then fitted to the data.

2.3.1. Data-driven EARSM

Several methods have been proposed to find the functions $\alpha^{(n)}$. Data-driven approaches have grown in popularity (Duraisamy et al., 2019), where high-fidelity data is obtained from DNS or LES simulations, and used to train machine learning models to predict the functions $\alpha^{(n)}$. These approaches typically use frozen RANS, where the input features are computed using the mean flow features from the high-fidelity data, and used to train the model. This avoids the need to run RANS simulations during training, which can be computationally expensive. The target outputs are the correction terms Δb_{ij}, which are computed from the difference between the high-fidelity Reynolds-stress tensor and the LEVM Reynolds-stress tensor. Many machine learning models have been proposed to find the functions $\alpha^{(n)}$, including neural networks, random forests, sparse regression, and symbolic regression.

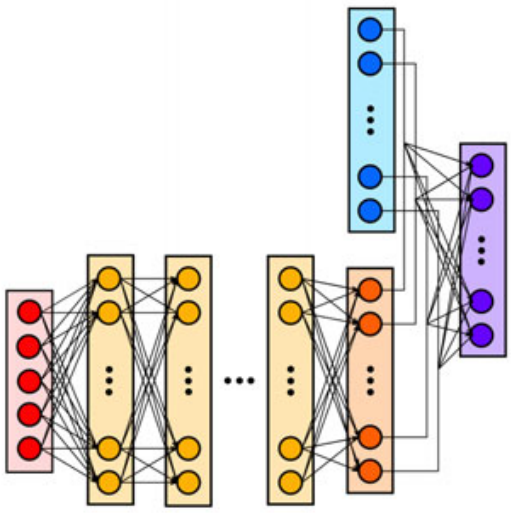


Figure 2.1: The architecture of TBNN. The input invariants are passed through several hidden layers to obtain the coefficients $\alpha^{(n)}$, which are then multiplied by the basis tensors to obtain the correction term Δb_{ij}. Image taken from Ling et al., 2016.

Ling et al., 2016 proposed a tensor-based neural network (TBNN) to find the functions $\alpha^{(n)}$. The authors used a neural network that takes the invariants $I_1, ..., I_5$ as input and outputs the coefficients $\alpha^{(n)}$. Then,

a final layer multiplies the coefficients with the basis tensors $T_{ij}^{(n)}$ to obtain the correction term Δb_{ij}. The TBNN is trained using a supervised learning approach, where the loss is the error between the predicted correction term and the true correction term obtained from high-fidelity data. Figure 2.1 shows a schematic of the TBNN architecture.

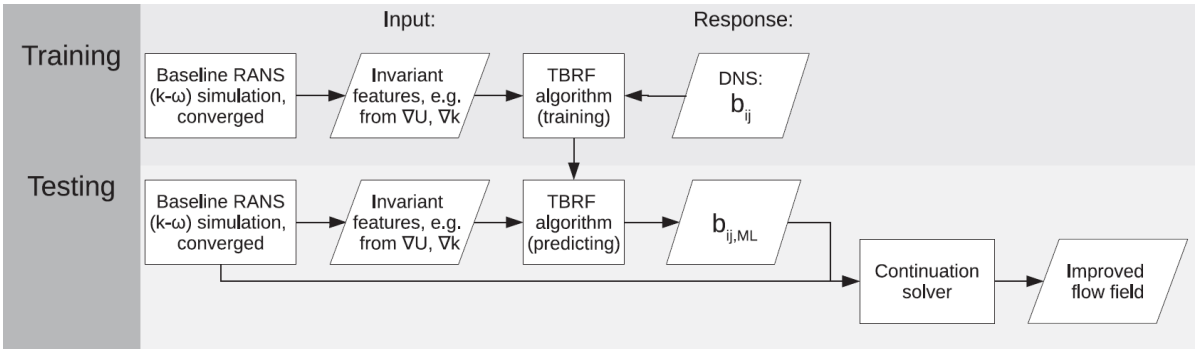


Figure 2.2: Schematic of how TBRF is used to predict the model-form error in the Reynolds-stress tensor. Image taken from Kaandorp and Dwight, 2020.

Kaandorp and Dwight, 2020 proposed the Tensor-Basis Random Forest (TBRF). Each decision tree in the forest partitions the invariant space and solves a local weighted least-squares problem to obtain optimum $\alpha^{(n)}$ values, while the tensor basis itself remains fixed, ensuring Galilean invariance. Compared with neural networks, TBRF involves fewer hyperparameters, offers out-of-bag error estimates for built-in validation, and supplies prediction-spread information via ensemble variance. Figure 2.2 shows a schematic of how TBRF is used to predict the model-form error in the Reynolds-stress tensor.

2.3.2. SpaRTA

Schmelzer et al., 2020 introduced the Sparse Regression of Turbulent Stress Anisotropy (SpaRTA) framework, a deterministic symbolic regression method for discovering algebraic stress models directly from high-fidelity data. Unlike stochastic approaches such as Gene Expression Programming, SpaRTA formulates model discovery as a sparse regression problem, identifying the smallest possible set of physically meaningful terms from a large predefined library. The method is based on elastic net regularisation, which blends LASSO and Ridge regression to promote sparsity while maintaining numerical robustness. In this way, SpaRTA yields interpretable algebraic models that can be efficiently implemented in RANS solvers.

A large candidate library of basis tensors and invariant functions is constructed, and the sparse regression selects the subset of terms that best describe the anisotropy tensor from DNS or LES data. This formulation makes the model deterministic and computationally inexpensive compared to evolutionary methods, while ensuring reproducibility across runs.

In addition to identifying corrections for the Reynolds-stress tensor, SpaRTA also introduced the k-corrective frozen RANS approach, which extends the frozen RANS framework. The motivation for this extension arises from the observation that modifying the Reynolds-stress tensor in RANS, particularly in k-ω models, changes the production term P_k in the turbulent kinetic energy equation. If this additional production is not accounted for, the modified Reynolds stresses can create residual errors in the TKE transport equation. The k-corrective approach explicitly models this residual by introducing an additive correction term R, resulting in a coupled correction of both the Reynolds-stress tensor and the turbulent kinetic energy equations.

The modified k-ω SST model with this correction can be written as:

$$\begin{aligned}\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} &= (P_k + R) - \beta^* \omega k + \frac{\partial}{\partial x_j}\left[(\nu + \sigma_k \nu_t)\frac{\partial k}{\partial x_j}\right], \\ \frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} &= \gamma \frac{\nu_t}{k}(P_k + R) - \beta \omega^2 + \frac{\partial}{\partial x_j}\left[(\nu + \sigma_\omega \nu_t)\frac{\partial \omega}{\partial x_j}\right] + CD_{k\omega}.\end{aligned} \tag{2.29}$$

Here, the residual term R introduces additional production in the TKE equation to compensate for the effects of the corrected Reynolds stresses. It is defined analogously to the anisotropy correction, as shown below:

$$R = 2k\, b_{ij}^{R} \frac{\partial U_i}{\partial x_j}, \tag{2.30}$$

Where b_{ij}^{R} is the residual anisotropy tensor, modelled as a linear combination of the basis tensors $T_{ij}^{(n)}$ with coefficients $\alpha_R^{(n)}(I_1, \dots, I_5)$. This formulation mirrors the structure of the Reynolds-stress correction term Δb_{ij} but applies it to the TKE residual, enabling both terms to be discovered within the same symbolic regression framework.

The combined correction terms Δb_{ij} and R are extracted from high-fidelity simulations using the k-corrective frozen RANS procedure, which iteratively solves the ω-equation while keeping the velocity and k fields frozen. This allows the residuals of the RANS equations to be computed directly from data, without the need for an expensive inverse optimisation. These residuals form the training targets for the SpaRTA regression. By regressing both Δb_{ij} and R, the framework can identify algebraic models that simultaneously correct the stress-strain relationship and the TKE production.

The resulting models are sparse, interpretable, and computationally robust, providing physical insight into which terms in the tensor polynomial basis contribute most significantly to the model-form error. Cross-validation across canonical separated flows, such as the periodic hill, converging-diverging channel, and curved backwards-facing step, showed that the SpaRTA-discovered models consistently improved RANS predictions of velocity and turbulence quantities. SpaRTA achieves these improvements with minimal model complexity, demonstrating that sparse algebraic corrections are sufficient to capture key deficiencies in the baseline k-ω SST model.

Overall, SpaRTA represents a systematic, deterministic alternative to evolutionary symbolic regression. By combining sparse regularisation with physically constrained tensor algebra. It also introduces the k-corrective frozen RANS approach, enabling simultaneous correction of both Reynolds stresses and TKE production. It was found that the turbulence production correction proved to be more significant than

2.3.3. Relative Importance Term Analysis (RITA)

Buchanan et al., 2025 introduced the Relative Importance Term Analysis (RITA) framework as a physics-based methodology for identifying and targeting regions in which RANS models require correction. Rather than modifying closure coefficients globally, RITA provides a systematic way to localise corrections by quantifying the relative magnitudes of physical processes in the turbulence kinetic energy equation. The approach enables selective application of data-driven corrections in separated shear layers, where RANS models typically fail, while preserving baseline performance in regions where the model is already accurate, such as attached boundary layers.

The foundation of RITA lies in analysing the balance of transport terms in the TKE equation of a standard $k-\omega$ SST model:

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = P_k - D_k + d_k, \tag{2.31}$$

where P_k is production, D_k is destruction (or dissipation), and d_k represents diffusive transport. Instead of focusing on the absolute magnitudes of these terms-which vary strongly across different flows, RITA uses non-dimensional ratios that describe the relative importance of each mechanism. These ratios reveal consistent, physical patterns across flow types, distinguishing boundary layers, shear layers, and recirculation regions through their characteristic energy balances.

The key idea is that the structure of separated shear layers can be identified from the balance between production and destruction in the TKE equation. In attached boundary layers, dissipation typically dominates ($|D_k| > |P_k|$), while in shear layers the two terms are nearly balanced ($|D_k| \approx |P_k|$), and in

the freestream their magnitudes are both small. By exploiting these systematic trends, RITA formulates a set of simple, interpretable indicators:

$$\phi_{D_k/P_k} = \frac{|D_k|}{|D_k| + |P_k|}, \qquad \phi_k = \frac{k}{k + 0.5|U|^2}, \qquad Re_\Omega = \frac{d_w^2 \Omega}{\nu}. \tag{2.32}$$

Here, ϕ_{D_k/P_k} quantifies the local production-destruction balance, ϕ_k represents the turbulent-to-total kinetic energy ratio, and Re_Ω is a vorticity-based Reynolds number that captures rotational intensity scaled by wall distance d_w. These dimensionless quantities are robust across different geometries and Reynolds numbers and directly encode the dominant flow mechanisms in a given region.

Using these indicators, RITA defines a binary classifier σ_{SL} that identifies whether a given location belongs to a separated shear layer. The classifier is defined as:

$$\sigma_{SL} = \begin{cases} 1, & \text{if } \phi_{D_k/P_k} < 0.55,\ \phi_k \geq 0.12,\ Re_\Omega \geq 0.02, \\ 0, & \text{otherwise.} \end{cases} . \tag{2.33}$$

Cells with $\sigma_{SL} = 1$ are considered part of the shear layer and receive model corrections, while all others retain the baseline model. This zonal classification ensures that corrections are applied only in physically justified regions, preserving baseline accuracy in wall-bounded flows and log-law regions. The thresholds are empirically chosen based on consistent behaviour observed across canonical separated flows such as the periodic hill and NASA hump cases.

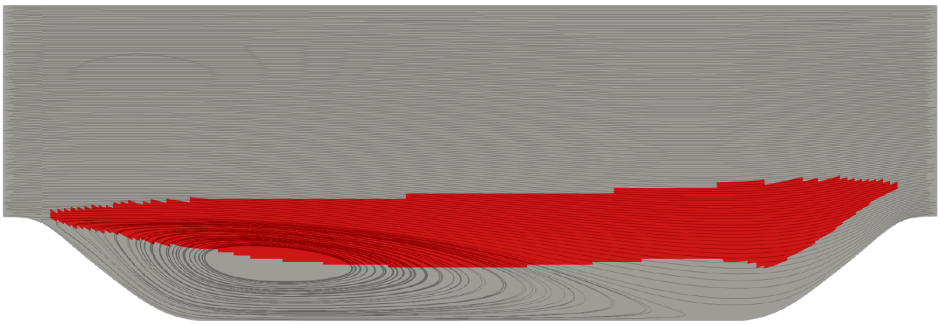


Figure 2.3: Visualization of the RITA shear-layer classifier σ_{SL} applied to the periodic hill flow at $Re = 10,595$. Regions where $\sigma_{SL} = 1$ (highlighted in red). Figure by Buchanan et al., 2025.

An example of the RITA classifier applied to the periodic hill flow is shown in Figure 2.3. Once the classifier identifies the shear-layer zones, correction terms derived through symbolic regression (e.g. SpaRTA) are applied only within those zones. This approach avoids global model modifications and introduces a physically interpretable form of zonal augmentation, where corrections to Reynolds stress anisotropy or TKE production are selectively activated by σ_{SL}. The final zonally augmented k-ω SST equations thus become:

RITA-augmented k-ω SST Model

Continuity and Momentum Equations:

$$\begin{aligned} \frac{\partial \bar{u}_i}{\partial x_i} &= 0 \\ \frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} &= -\frac{\partial}{\partial x_i}\left(\frac{\bar{p}}{\rho} + \frac{2}{3}k\right) + \frac{\partial}{\partial x_j}\left(2(\nu + \nu_t)S_{ij}\right) + \sigma_{SL}\frac{\partial}{\partial x_j}\left(2k\Delta b_{ij}\right) \end{aligned} \tag{2.34}$$

Turbulent Kinetic Energy Equation:

$$\frac{Dk}{Dt} = \hat{P_k} + \sigma_{SL} R + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_k}\right)\frac{\partial k}{\partial x_j}\right] - \beta^* k\omega \tag{2.35}$$

Specific Dissipation Rate Equation:

$$\frac{D\omega}{Dt} = \alpha\frac{\omega}{k}(\hat{P_k} + \sigma_{SL} R) - \beta\omega^2 + \frac{\partial}{\partial x_j}\left[\left(\frac{1}{Re} + \frac{\nu_t}{Pr_\omega}\right)\frac{\partial \omega}{\partial x_j}\right] + 2(1 - F_1)\frac{\sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}. \tag{2.36}$$

Eddy Viscosity Definition:

$$\nu_t = \frac{a_1 k}{\max(a_1\omega, \Omega F_2)}, \tag{2.37}$$

TKE production term:

$$\hat{P_k} = \min(-2(\nu_t S_{ij} + \sigma_{SL}\Delta b_{ij})\frac{\partial \bar{u}_i}{\partial x_j}, 10\beta^* k\omega) \tag{2.38}$$

where σ_{SL} is a binary classifier, determining whether the correction is applied based on the local flow conditions. Because σ_{SL} is binary and non-differentiated in the momentum, TKE, and specific dissipation rate equations, it does not affect the numerical stability of the transport equations or introduce spurious gradients at zonal interfaces.

Conceptually, RITA represents a shift from global model correction to targeted model enhancement. By characterising the flow through the internal dynamics of the RANS equations rather than externally defined features, it leverages the existing model structure to identify its own deficiencies. This makes the approach physically transparent, computationally efficient, and broadly applicable across different geometries. Moreover, since the classification criteria are based on invariant, non-dimensional quantities, RITA naturally generalises from two-dimensional to three-dimensional configurations, as demonstrated by its performance on flows such as the Faith Hill and Ahmed body cases.

2.4. Symbolic Regression

Traditional regression methods, such as linear regression, polynomial regression, and especially neural networks, typically assume a fixed model structure, with the parameters adjusted to best fit the data. While these approaches can provide an excellent fit to the data, they often do so at the expense of interpretability. In particular, models such as neural networks tend to act as black-box models, mapping inputs to outputs through complex, nontransparent internal mechanisms. As a result, it is often difficult to discern the underlying relationships or physical laws from such models, and to understand how predictions are made.

In contrast, symbolic regression is a form of regression analysis that searches over the space of mathematical expressions to identify the model that best fits a given dataset. Here, the model is typically represented as an explicit mathematical formula, such as:

$$y = a\, x_1^2 + b\, \sin(x_2) + c\, x_1 x_2 + d. \tag{2.39}$$

Where the objective is to discover both the functional form (the combination of operations and variables) and the optimal values for the constants. Unlike black-box models, symbolic regression aims to yield so-called white-box models: interpretable equations whose structure can often be related back to physical principles or domain knowledge. This interpretability is especially valuable in fields such as physics and engineering, where gaining insights into the underlying functional relationships is of fundamental importance.

A variety of methods have been developed for symbolic regression. Evolutionary algorithms, in particular, have been widely employed to evolve candidate mathematical expressions. It is based on natural selection, where the fittest individuals are selected to reproduce and create new offspring. Over successive generations, the population evolves towards better-fitting expressions.

Genetic programming (GP) (Koza, 1994, Schmidt and Lipson, 2009) is one of the most popular evolutionary approaches for symbolic regression. In GP, candidate solutions are represented as tree structures, where nodes correspond to mathematical operations (e.g. addition, multiplication, sine, cosine) and leaves represent input variables or constants. The algorithm evolves the population of trees through genetic operations such as selection, crossover, and mutation, guided by a fitness function that evaluates how well each expression fits the data.

More recently, deep learning techniques have been explored in the context of symbolic regression. One such approach is Deep Symbolic Regression, which combines the flexibility of deep learning with the interpretability of symbolic regression, enabling the discovery of complex mathematical expressions while preserving transparency.

Notably, recent works have leveraged large-scale pre-training and language models for symbolic regression. For example, Biggio et al., 2021 introduced a neural symbolic regression framework based on Transformers, which can be pre-trained on large, procedurally-generated equation datasets and then rapidly adapted to new tasks. Similarly, Valipour et al., 2021 proposed SymbolicGPT, a transformer-based language model that frames symbolic regression as a language modelling problem. By learning to "caption" datasets with the corresponding mathematical expressions, SymbolicGPT demonstrates improved speed and data efficiency, scaling well to higher dimensions and diverse functional forms.

2.4.1. Genetic Expression Programming for EARSM

Conventional symbolic regression approaches are effective for scalar regression tasks but are ill-suited for tensorial relationships, as they often produce expressions that violate dimensional consistency or tensor symmetry. The EARSM problem is inherently tensorial, requiring the regression of the anisotropy tensor Δb_{ij} as a function of tensorial inputs like the mean strain rate S_{ij} and rotation rate Ω_{ij}, multiplied by scalar coefficient functions of invariants, as explained in section 2.3. Standard symbolic regression methods, such as traditional genetic expression programming (GEP), lack the structural mechanisms to ensure that the generated expressions respect the mathematical properties of tensors.

To overcome this, Weatheritt and Sandberg, 2016 introduced a multidimensional extension of GEP (M-GEP), designed to regress tensor-valued quantities while maintaining valid tensor algebra. The key innovation of M-GEP is its host-plasmid architecture, inspired by biological systems. In this structure, the host chromosome encodes the tensorial form of the expression, while plasmids represent auxiliary scalar expressions that can be inserted into the host at specific positions. The host thus determines the overall tensor structure, whereas the plasmids encode scalar functions of invariants or other scalar quantities that modulate this structure. The concept is illustrated in Figure 2.4.

This modular design allows tensor expressions to be constructed hierarchically, preserving dimensional consistency while maintaining flexibility in form. The host chromosome operates on tensor-valued terminals (such as basis tensors $T_{ij}^{(n)}$ or the identity δ_{ij}) and tensor operations (e.g. addition, multiplication, or contraction), while the plasmids evolve independently to provide scalar coefficients that can be functions of invariants. When a host expression calls a plasmid symbol p, the plasmid is inserted and evaluated to yield a scalar multiplier. This creates a coupled co-evolutionary system in which hosts and plasmids evolve together, with their fitness jointly determined by the performance of the combined tensor expression.

The overall objective of the M-GEP framework is to regress an algebraic mapping between the anisotropy tensor a_{ij} and invariants derived from the mean strain rate S_{ij} and rotation rate Ω_{ij}.

$$a_{ij} = f(S_{ij}, \Omega_{ij}, k, \varepsilon), \tag{2.40}$$

This formulation mirrors the structure of traditional EARSMs, but crucially, both the tensor basis and the scalar functions are discovered autonomously through evolution, rather than being predefined by the modeller.

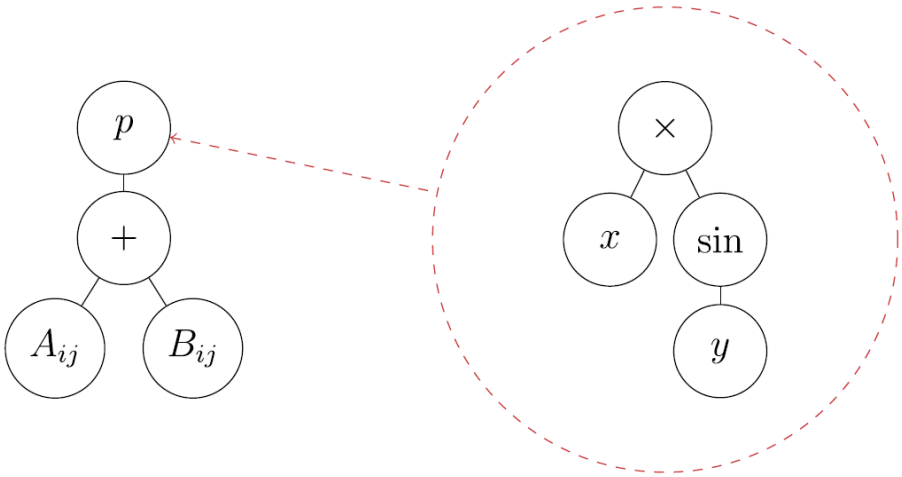


Figure 2.4: M-GEP framework. The host chromosome represents the tensorial structure, while plasmids encode scalar-valued expressions. The host can call plasmids at specific positions within its symbolic expression tree, allowing for flexible assembly of tensor expressions. Figure by Weatheritt and Sandberg, 2016.

The training process proceeds in two stages. In the a priori phase, M-GEP is trained on high-fidelity data extracted from DNS of turbulent flows, such as the backwards-facing step. The objective function measures the alignment between the predicted anisotropy tensor and the reference tensor from the DNS

$$\text{Fit}(a_{ij}^{\text{pred}}) = \sum_k \frac{a_{ij,k} a_{ij,k}^{\text{pred}}}{\sqrt{a_{mn,k} a_{mn,k}} \sqrt{a_{pq,k}^{\text{pred}} a_{pq,k}^{\text{pred}}}}, \tag{2.41}$$

which evaluates how closely the predicted tensor aligns with the true anisotropy tensor at each training point. The algorithm iteratively evolves both host and plasmid populations using genetic operators such as mutation, recombination, and transposition, while enforcing consistency between tensorial and scalar operations. The best-performing host-plasmid combinations are then validated a posteriori by embedding them directly into a RANS solver and testing their predictive performance on benchmark flows such as periodic hills and backwards-facing step geometries.

The results demonstrated that the M-GEP framework was able to autonomously discover algebraic Reynolds-stress models that achieved improved anisotropy alignment and flow-field predictions compared to both the baseline linear eddy-viscosity model and the established non-linear EARSM of comparable complexity. Importantly, multiple independent runs of the stochastic optimisation converged to functionally similar expressions, indicating that the algorithm was discovering consistent physical relationships rather than arbitrary curve fits. The discovered models showed systematic improvements in the prediction of separated flows, particularly in regions of recirculation where conventional RANS models typically fail.

2.5. Reinforcement Learning

Reinforcement learning (RL) (Sutton, Barto, et al., 1998) is a subfield of machine learning concerned with training agents to make sequential decisions through interaction with an environment. In this framework, an agent observes the environment, takes actions, and receives feedback in the form of rewards. The agent's objective is to learn a mapping from states to actions that maximises the expected cumulative reward over time.

A formal framework for RL is the Markov Decision Process (MDP) by Bellman, 1966, which describes the interaction between the agent and the environment in terms of:

- **States (s):** A representation of the current situation of the environment.

- **Actions (a):** The set of all possible decisions or moves the agent can make from a given state.
- **Transition probabilities ($P(s'|s,a)$):** The probability of moving from state s to state s' when action a is taken.
- **Reward ($r(s,a)$):** The immediate scalar feedback signal received after taking an action at some state.
- **Policy (π):** The agent's decision-making strategy, specifying the probability of taking each action in each state.
- **Discount factor (γ):** A parameter $0 \leq \gamma \leq 1$ that determines the importance of future rewards.

In an MDP, the agents' behaviour is defined by a stochastic policy $\pi(a|s)$. This policy defines a probability distribution over possible actions a given the current state s. The agent selects actions according to this distribution, allowing for exploration of different actions in the environment.

In some settings, the agent may not have access to the full state of the environment but only to partial observations, in which case the problem is modelled as a Partially Observable Markov Decision Process (POMDP). This introduces several new variables into the framework:

- **Observation** (o): The information the agent receives about the environment, which may be incomplete or noisy.
- **Observation probabilities** ($O(o|s)$): The probability of receiving observation o given the true state s.
- **Belief state** (b): A probability distribution over possible states, representing the agent's uncertainty about the true state.

2.5.1. Returns, Values, and Q-values

Returns, values, and Q-values (Bellman, 1966) are fundamental concepts in reinforcement learning that help quantify the long-term performance of an agent's policy. The objective of reinforcement learning is to learn an optimal policy $a \sim \pi(\cdot|s)$ that maximises the objective function J^π. The objective function is defined as the expected cumulative reward when starting from an initial state s_0 and following policy π. This is given by Equation 2.42.

$$J^\pi = \mathbb{E}[R(\tau)|s_0, \pi] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0, \pi\right] \tag{2.42}$$

Where R(τ) is the return of a trajectory τ, defined as the sum of discounted rewards along one episode. This is the total discounted reward the agent receives from time step $t = 0$ to the end of the episode. The discount factor γ determines the importance of future rewards, with values closer to 1 placing more emphasis on long-term rewards.

A value function V^π is the expected return when starting from a certain state s and following policy π thereafter. The value function is defined as:

$$V^\pi(s) = \mathbb{E}[R(\tau)|s_0 = s, \pi] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, \pi\right]. \tag{2.43}$$

Similarly, the action-value function, or Q-function, $Q^\pi(s,a)$, is defined as the expected return when starting from state s, taking some action a, and thereafter following policy π. The Q-function is defined as:

$$Q^\pi(s,a) = \mathbb{E}[R(\tau)|s_0 = s, a_0 = a, \pi] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a, \pi\right]. \tag{2.44}$$

These functions provide a way to evaluate the quality of states and actions under a given policy, and they are central to many reinforcement learning algorithms. They allow the agent to make informed decisions about which actions to take in order to maximise its long-term reward.

2.5.2. Policy

The policy $\pi(a|s)$ defines the agent's behaviour at a certain state, but there are many ways to represent and learn the policy. The most basic approach is to use a tabular representation, where the policy is stored in a table with one entry for each state-action pair. For example, Q-learning (Watkins et al., 1989) is a popular tabular method that learns the Q-values for each state-action pair and derives the policy by selecting the action with the highest Q-value in each state. However, tabular methods are limited to small state and action spaces, as the size of the table grows exponentially with the number of states and actions.

The advent of deep reinforcement learning (DRL) has further advanced the field by leveraging deep neural networks as powerful function approximators. In DRL, neural networks (Rosenblatt, 1958, Rumelhart et al., 1985) are used to approximate the policy, the value function, or both, enabling RL to scale to high-dimensional and complex state or action spaces. The usage of Recurrent neural networks (RNN) in the policy can even tackle POMDPs, as the RNN can maintain a belief state by encoding the history of observations and actions. This allows the agent to make decisions based on its internal state, which captures information about the environment that is based on past observations.

As DRL is now standard in most modern applications, throughout this work, ”reinforcement learning” refers specifically to deep reinforcement learning unless otherwise specified.

2.5.3. Policy Gradient Methods

Policy gradient methods are a class of reinforcement learning algorithms that directly optimise the policy by adjusting its parameters in the direction of the gradient of the expected cumulative reward. Unlike value-based methods, which learn a value function and derive the policy from it, policy gradient methods maintain a parameterised policy and update its parameters based on the observed rewards. It is a stochastic reinforcement learning method that uses a probability distribution $\pi(a|s)$ as the policy, and samples over it at every time step to select an action.

REINFORCE is a fundamental policy gradient algorithm introduced by Williams, 1992. It uses a neural network to output a probability distribution over actions, with the current state as input. The loss function for REINFORCE is:

$$L(\theta) = -\mathbb{E}\left[R(\tau)\log\pi_\theta(a|s)\right]. \tag{2.45}$$

Where $R(\tau)$ is the return of a trajectory τ. $\pi_\theta(a|s)$ is the policy of a neural network parameterised by θ. $\pi_\theta(a|s)$ is the probability of taking action a in state s under the current policy. The loss function encourages actions that lead to higher returns by weighting the log-probability of actions by their corresponding returns. For example, if some trajectory τ has a high return $R(\tau)$, the optimiser will increase the log-probability of the actions taken in that trajectory, making them more likely to be selected in the future. So whether a certain action is good or bad is determined by the return of the trajectory it was taken in, and the policy learns from experience by adjusting the probabilities of actions based on the returns they yield.

This approach has very high variance, as a full trajectory forms a tree with a^L combinations, where a is the number of possible actions, and L is the length of each trajectory. This means that the return will also have high variance as it is the sum of rewards over the trajectory. This can lead to slow convergence and instability in training, especially in environments with sparse or delayed rewards. So a baseline is usually introduced to reduce this variance. The new loss function is given by:

$$L(\theta) = -\mathbb{E}\left[(R(\tau) - b)\log\pi_\theta(a|s)\right] \tag{2.46}$$

The baseline b can be a constant, the average return, or a learned function. Using a baseline does not introduce bias but can significantly reduce the variance of the gradient estimates, leading to more stable and efficient training.

2.5.4. Actor-Critic Methods

Actor-critic methods build directly upon the basic REINFORCE algorithm. In REINFORCE, the policy parameters are updated using the full return from each episode as the learning signal. A user-defined baseline needs to be introduced to reduce the variance of the gradient estimates, which can be difficult to choose appropriately. Actor-critic methods by Mnih et al., 2016, Konda and Tsitsiklis, 1999, Sutton et al., 1999 address this by introducing a critic, which is a learned value function that estimates the expected return from each state.

In actor-critic methods, two neural networks are used: an actor network and a critic network. The actor is responsible for selecting actions according to the current policy, while the critic provides an estimate of the expected cumulative reward (the value) for each state. This leverages the knowledge of value functions to provide an appropriate baseline for the policy gradient algorithm. The simplest version of an actor-critic method is the temporal difference actor-critic (Sutton, 1988), where the critic estimates the value function $V_\phi(s)$ using a neural network with parameters ϕ. The loss functions for each are shown below:

$$\begin{aligned} L(\theta) &= -\mathbb{E}_\pi\left[\sum_{t=0}^{T-1}(r(s_t,a_t)+\gamma V_\phi(s_{t+1})-V_\phi(s_t))\log\pi_\theta(a_t|s_t)\right] \\ L(\phi) &= \mathrm{MSE}(r(s,a)+\gamma V_\phi(s')-V_\phi(s)). \end{aligned} \tag{2.47}$$

The term $r(s_t,a_t)+\gamma V_\phi(s_{t+1})$ acts as an estimate of the Q-value, where it is the expected return for taking the chosen action a_t in state s_t plus the value of being in the next state s_{t+1}. The term $V_\phi(s_t)$ is the baseline, which is the value of being in state s_t. The difference between these two terms is called the temporal difference (TD) error, which measures how much better or worse taking action a_t in state s_t is compared to the average action in the current state s_t.

This turns the trajectory-based policy gradient into a transition-based one. In actor-critic, every transition now has its own reward signal, indicating the quality of each action in each state. Thus, the policy learns by each transition, instead of weighting the entire trajectory with the same return. This significantly reduces the variance of the gradient estimates, which improves sample efficiency, and thus training time. A more general form for an actor-critic loss can be written by generalising the TD error into an advantage function $A(s,a)$, which measures the advantage of taking a certain action, when being in the current state. This results in:

$$\begin{aligned} L(\theta) &= -\mathbb{E}\left[A(s,a)\log\pi_\theta(a|s)\right] \\ L(\phi) &= \mathrm{MSE}(A(s,a)). \end{aligned} \tag{2.48}$$

The advantage function can be defined in several ways, but one approach is n-step bootstrapping, which considers the sum of rewards over several steps plus the value of future states. The n-step advantage function does not only consider the value of the next step like TD learning, but looks further ahead. The n-step advantage function is shown here:

$$A(s,a)=\sum_{k=0}^{n-1}\gamma^k r_{t+k}+\gamma^n V_\phi(s_{t+n}) \tag{2.49}$$

Another form of the advantage function is the Generalised Advantage Estimation (GAE) introduced by Schulman, Moritz, et al., 2015. GAE is a method for estimating the advantage function that balances bias and variance through a parameter λ. The GAE advantage function is shown below:

$$\begin{aligned} A(s,a) &= \sum_{t=0}^{\infty}(\gamma\lambda)^t\delta_t \\ \delta_t &= r_t+\gamma V_\phi(s_{t+1})-V_\phi(s_t) \end{aligned} \tag{2.50}$$

Here, δ_t is the TD error at time step t, and λ is a parameter that controls the trade-off between bias and variance. When $\lambda = 0$, GAE reduces to the one-step TD error, which has low variance but can be biased. When $\lambda = 1$, GAE becomes the Monte Carlo estimate of the advantage, which is unbiased but has high variance. By tuning λ, GAE can provide a more stable and efficient estimate of the advantage function, leading to improved learning performance in actor-critic methods.

2.5.5. Proximal Policy Optimization (PPO)

While actor-critic methods improve the sampling efficiency of policy gradient algorithms, they still lag behind other reinforcement learning algorithms. This is mainly due to the on-policy nature of policy gradient methods. Policy gradients are probabilistic and depend on Monte Carlo estimates to provide a good estimate of the gradient. This means that the policy can only be updated using data collected from the current policy, and not from previous policies. So every sample can only be used for one update step of the neural network, and thus, no epochs can be done, since the probability distribution shifts when the neural network changes.

The Trust Region Policy optimisation (TRPO) by Schulman, Levine, et al., 2015 circumvents this issue. It changes the original policy gradient algorithm into one where there is a shifting policy. The original actor loss can be rewritten as:

$$\begin{aligned} \nabla L(\theta) &= -\mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} A(s,a)\log\pi_\theta(a|s)\right] \\ &= -\sum_{t=0}^{T-1}\int\int A(s,a)\pi_\theta(a_t|s_t)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}da_t ds_t. \end{aligned} \tag{2.51}$$

If there are samples from a different distribution $\mu(a|s)$, the loss can be rewritten as:

$$\begin{aligned} \nabla L(\theta) &\approx -\sum_{t=0}^{T-1}\int\int A(s,a)\mu(a_t|s_t)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}da_t ds_t \\ &\approx -\mathbb{E}_{\mu}\left[\sum_{t=0}^{T-1} A(s,a)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}\right]. \end{aligned} \tag{2.52}$$

This approximations holds as long as the two distributions $\pi_\theta(a|s)$ and $\mu(a|s)$ are close. In TRPO, this is measured by the Kullback-Leibler (KL) divergence, which measures the difference between two probability distributions. TRPO is formulated as a constrained optimisation problem, where the objective is to minimise the actor loss while keeping the KL divergence between the two policies below a certain threshold. A general form of the TRPO optimisation problem is shown below:

$$\min L(\theta) \quad s.t. \quad D_{KL}(\pi_\theta(a|s)||\mu(a|s)) \leq \delta \tag{2.53}$$

Where δ is a hyperparameter that controls the maximum allowed divergence between the two policies. This ensures that the new policy $\pi_\theta(a|s)$ does not deviate too much from the old policy $\mu(a|s)$, which helps to stabilise training and prevent large, destabilising updates to the policy.

However, TRPO requires solving a complex constrained optimisation problem, making it difficult to implement in practice. The calculation of the KL divergence is also costly and inefficient. Thus, the Proximal Policy Optimisation (PPO) Schulman et al., 2017 was introduced as a simpler alternative to TRPO.

PPO achieves a similar effect using a simple clipped objective function for the actor, which prevents large updates to the policy that could lead to instability. The clipped objective function is shown in Equation 2.54. The main idea is to limit the ratio of the new policy $\pi_{\theta,i}(a|s)$ to the old policy $\mu(a|s)$ to

be within a certain range defined by the hyperparameter γ. This prevents large updates to the policy that could lead to instability.

$$L(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} \min\left(\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}A_t,\ \text{clip}(\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, 1-\gamma, 1+\gamma)A_t\right)\right] \tag{2.54}$$

PPO does not change the definition of the advantage function, so any of the previously mentioned methods can be used. In practice, PPO has become one of the most popular reinforcement learning algorithms due to its robustness, simplicity, and strong empirical performance across a range of complex control tasks.

2.6. Deep Symbolic Regression (DSR)

The inherently discrete structure of symbolic expressions presents a challenge for the direct application of conventional deep learning techniques. Deep Symbolic Regression (DSR), as introduced by Petersen et al., 2019, addresses this by combining symbolic regression with deep learning, specifically formulating symbolic regression as a reinforcement learning problem. The objective is to automatically discover mathematical expressions that accurately fit a given dataset, while retaining interpretability. DSR turns the symbolic regression into a POMDP, and it is defined by the following components:

- **State:** The hidden state of the RNN, which encodes the history of the generated sequence and serves as the POMDP state.
- **Action:** The selection of the next token from the RNN's output distribution.
- **Observation:** The context at each step, typically including information about the parent and sibling tokens and the number of dangling nodes in the tree.
- **Reward:** The quality of the generated expression, quantified by its performance on the dataset.

With this POMDP framework, reinforcement learning methods can be applied to symbolic regression. This allows the use of deep learning techniques for sequence modelling, namely RNNs, to be applied to the discrete and structured nature of symbolic expressions. The RNN is trained to generate expressions that maximise the expected reward, effectively learning to produce high-quality mathematical models that fit the data well.

2.6.1. DSR framework

In DSR, policy gradient methods are used as the main training method, namely the REINFORCE algorithm. But DSR introduces a novel variant of the REINFORCE algorithm, called the risk-seeking policy gradient. This variant focuses the learning signal on the top-performing samples in each batch, rather than optimising for the expected (average) reward. The general algorithm for DSR is:

1. Generate a batch of B expressions using the RNN.
2. Evaluate the reward for each expression based on its performance on the dataset.
3. Select the top ϵ fraction of expressions with the highest rewards.
4. Update the RNN parameters using the risk-seeking policy gradient, focusing on the selected top-performing expressions.
5. Repeat steps 1-4 until the exact expression is found, or the maximum iterations are met.

2.6.2. Expression generation

There are two main types of tokens in the expression: operators and terminals. Operators are functions that take one or more inputs (operands) and produce an output. Examples of operators include addition, multiplication, sine, and logarithm. Terminals are the basic building blocks of the expression, which have no inputs, like variables or constants. The RNN generates a sequence of these tokens to form a complete mathematical expression.

To enable reinforcement learning, symbolic expressions, naturally represented as trees, must be converted into a linear format suitable for sequential modelling. DSR accomplishes this by using a pre-order traversal of the expression tree. This linearization allows an RNN to generate expressions token by token, treating expression generation as a sequence modelling problem. A pre-order traversal visits the root node first, followed by recursively visiting the left and right subtrees. This method enables the RNN to generate expressions in a top-down manner, starting from the highest-level operator and progressively adding operands. The pre-order traversal can be illustrated with the following example expression:

$$traversal: -, *, x, log, y, *, 2, sin, y \Rightarrow (x \cdot \log(y)) - 2 \cdot \sin(y) \tag{2.55}$$

At each iteration of the full DSR process, the RNN generates a batch of size B expressions, each of maximum length L. The generation process is illustrated in Algorithm 1, and a schematic of the DSR framework is shown in Figure 2.5.

Algorithm 1 Expression Sampling Procedure

for $b = 1$ to B **do** — *Sample B expressions*
  $s \leftarrow s_0$
  for $t = 1$ to L **do**
    $\mathbf{p} \leftarrow \text{RNN}(o_t, s_t)$ — *Token probability distribution*
    $\mathbf{p} \leftarrow (\mathbf{p} + p_{priors})$ — *Apply priors*
    $a_t \sim \text{Softmax}(\mathbf{p})$ — *Sample next token*
    $(s_{t+1}, o_{t+1}) \leftarrow \text{EnvStep}(a_t, s_t, o_t)$ — *Next state & observation*
  end for
  if dangling nodes exist **then**
    Complete with random terminal tokens
  end if
end for

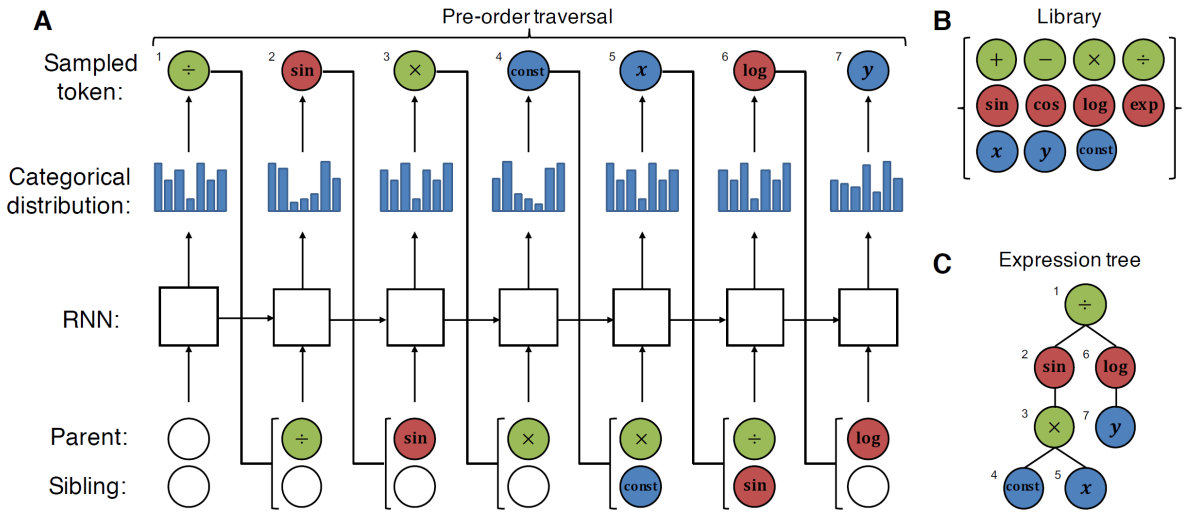


Figure 2.5: DSR framework. The RNN generates tokens of the symbolic expression in a pre-order traversal manner, and the generated expression is evaluated on the dataset to obtain a reward. Image by Petersen et al., 2019

Observations

The observations inform the policy in the context of the current token, and they also provide the RNN with information about the expression. The observation space consists of the following components:

- **Previous action:** The token selected in the previous step.

- **Parent token:** The token of the parent node in the expression tree.
- **Sibling token:** The token of the sibling that this token will be.
- **Dangling nodes:** The number of dangling nodes in the expression tree.

The previous action is straightforward, as it is just the previously selected action. The parent is the token type of the parent node for the current token, and it has to be an operator, as only operators can have children. The sibling is the token type of the sibling node for the current token, if the parent node has an arity larger than 1. For example, if the parent operator is a division, then the sibling is the numerator (the left or first child) of the division, and the current token would be the denominator. Thus, it is helpful to know what the current node would be dividing. Finally, the number of dangling nodes is the number of nodes in the expression tree that still need to be filled by terminal tokens.

The previous action, parent token, and sibling tokens are a discrete selection, and thus have to be represented as a vector so that the weights of the neural network do not become too large. This can be done with a learned encoding or a one-hot vector. The number of dangling nodes is a continuous value, and thus can be represented as a scalar. The full observation vector is then the concatenation of these components.

Priors
Before sampling the next token, priors are applied to the output distribution of the policy. These priors enforce syntactic constraints on the generated expressions, ensuring that they are valid and meaningful. When certain conditions are met when sampling expressions, the prior can prevent certain tokens from being the next token to be sampled. This is done by subtracting the token that should not be sampled by infinity. This effectively sets the probability of that token being sampled to zero after the softmax is applied. The priors used in DSR are:

- **Length prior:** This prior constrains the length of the expression. A minimum and maximum length can be set, and if the current length is below the minimum, only operators can be sampled. And if the current length is a certain amount from the maximum, and there are equal amounts of dangling nodes left, the prior will enforce that only terminal nodes can be sampled, thus controlling the length of expressions.
- **No inputs constraint:** This prior prevents the expression from having no inputs. An expression that is just a function of only constants would be meaningless. In the end, it would just evaluate to a single constant. For example, the expression $3.14 + sin(1.63)$ would just evaluate to 4.138, and thus would be pointless to have as a symbolic expression. This prior ensures that at least one input variable is present in the expression.
- **Inverse unary constraint:** This prior prevents the expression from having inverse unary operators, such as $log(exp(x))$ or $sqrt(x^2)$. These expressions are redundant, as the unary operators cancel each other out. This prior ensures that the expression does not contain such redundant structures.
- **Trigonometric constraint:** This prior prevents trigonometric functions from being a descendant of another trigonometric function. Although these are still valid functions, nested trigonometric functions like $sin(x_1 + cos(x_2))$ basically do not exist in any scientific domain.

Reward function
This process results in a batch of B candidate expressions, each represented as a sequence of tokens. These expressions are then evaluated to obtain their predicted values on the dataset $\hat{y}(x_1, ..., x_K)$. If constants are part of the expression, they are optimised to improve the fit. Then, the reward is evaluated using the reward function $R(\hat{y}, y)$, which quantifies how well the generated expression fits the data. The reward function can be defined in many ways, but commonly it is defined using the inversed normalised mean squared error, defined as:

$$R(\hat{y}, y) = \frac{1}{1 + (\hat{y} - y)^2 / \sigma(y)}. \tag{2.56}$$

Where $\sigma(y)$ is the standard deviation of the target values y. This reward function provides a bounded reward as $R \in [0, 1]$, where a perfect fit yields a reward of 1, and poor fits approach 0. Normalisation using the standard deviation also helps to stabilise training across datasets with different scales.

2.6.3. Risk-seeking policy gradient

Petersen et al., 2019 proposed a novel variant of the REINFORCE algorithm, referred to as the risk-seeking policy gradient. Unlike traditional policy gradient methods, which optimise for the expected (average) reward, the risk-seeking variant focuses the learning signal on the top-performing samples in each batch.

The risk-seeking objective is inspired by CVar, a risk-averse reinforcement learning algorithm proposed by Tamar et al., 2015. CVaR formalises robustness by optimising performance in the worst ϵ-fraction of outcomes rather than the mean. Let $R_\epsilon(\theta)$ denote the reward of the ϵth-quantile. The risk-averse objective function of CVaR is:

$$J_{\text{CVaR}}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\big|_{R(\tau)\leq R_\epsilon}\right]. \tag{2.57}$$

Equation 2.57 explicitly targets "guaranteed" performance under the worst trajectories. Basic policy gradient training spreads effort across the entire distribution, aiming to maximise the average return. In contrast, CVaR concentrates updates on under-performing samples to lift the worst-case behaviour. It does so by only updating with samples that fall below the ϵth-quantile, so the policy shifts its aim toward improving these worst-case outcomes.

The risk-seeking objective does the opposite of CVaR. It focuses on the best ϵ-fraction of outcomes rather than the worst ϵ-fraction. This changes the $\leq$ in Equation 2.57 to $\geq$, resulting in a risk-seeking objective function. Instead of using the full batch of B expressions to compute the policy gradient, only the top ϵ percentile of expressions are used to compute the gradient update. This causes the policy to only see the top epsilon expressions, and thus biases the policy toward only the best-performing expressions. This bias toward high-reward expressions is particularly suited to symbolic regression, where the primary aim is to find the best expression rather than to improve the population average. Formally, the risk-seeking policy gradient is defined as:

$$\nabla J(\theta) = \frac{1}{\epsilon N}\sum_{i=1}^{N}[R(\tau^{(i)}) - R_\epsilon]\cdot 1|_{R(\tau^{(i)})\geq R_\epsilon}\nabla_\theta \log \pi_\theta(\tau) \tag{2.58}$$

Here, R_ϵ is the $(1-\epsilon)$th-quantile of the rewards in the sampled batch. This introduces a baseline to the standard REINFORCE loss function, which helps to reduce the variance of the gradient estimates. This introduces a moving baseline for the algorithm. The returns are weighted based on how much better it performs compared to the R_ϵ baseline. This mechanism encourages the expressions generated to always do better than the ϵ-th percentile, constantly pushing the policy to improve. The indicator function $1|R(\tau^{(i)}) \geq R\epsilon$ ensures that only the top ϵ fraction of samples contribute to the gradient update.

2.7. DSR extensions

Many works have built upon DSR, trying to improve its performance in various ways. Some works have focused on improving the exploration of the search space, while others have integrated genetic programming into the process. Some works have even used DSR for turbulence modelling.

2.7.1. Improving exploration

While DSR leverages a risk-seeking policy gradient to bias learning toward high-reward expressions, its exploration dynamics can still be improved. Currently, it only uses an entropy term, which helps to maintain diversity in the policy. However, Landajuela et al., 2021 identified two key challenges that hamper effective exploration in DSR: (i) *early commitment*, where the policy collapses entropy on the first few tokens and repeatedly chooses the same initial branches; and (ii) *initialization bias*, where naive, uniform token priors skew the a priori length distribution toward overly long expressions.

Landajuela et al., 2021 proposes two complementary techniques to address these issues within policy-gradient search for symbolic optimisation (including SR): a hierarchical entropy regularizer and a soft length prior (SLP). Together, these methods improve recovery, sample efficiency, and solution compactness on the Nguyen benchmarks (Uy et al., 2011) when built on top of the DSR framework.

Hierarchical entropy regularizer.
Standard entropy bonuses sum the per-step entropies evenly,

$$R_H^{\text{std}}(\tau) = \sum_{i=1}^{|\tau|} H\big[\pi(\cdot \mid \tau_{1:(i-1)}; \theta)\big] \tag{2.59}$$

But in autoregressive generation, this concentrates exploration pressure on later tokens, allowing the policy to freeze early decisions (early commitment). To counteract this, the hierarchical entropy regularizer exponentially emphasises earlier actions:

$$R_H^{\text{hier}}(\tau) = \sum_{i=1}^{|\tau|} \gamma^{\,i-1} H\big[\pi(\cdot \mid \tau_{1:(i-1)}; \theta)\big], \qquad 0 < \gamma < 1. \tag{2.60}$$

Intuitively, Equation 2.60 weighs initial tokens with more entropy, causing the policy to have more entropy in initial tokens, encouraging diversity. This helps to prevent early commitment, as the policy is encouraged to explore different initial branches of the expression tree. The hyperparameter γ controls the rate of decay, with smaller values placing more emphasis on the earliest tokens. This hierarchical approach ensures that exploration is more evenly distributed across the entire sequence, leading to a more diverse set of generated expressions.

Soft length prior (SLP).
Even with a token-equalising logit prior that balances unary/binary/terminal categories, the sampling process places excessive mass on long sequences, and the expected length diverges. Rather than relying solely on hard min/max length constraints, used in DSR to ensure syntactic completeness Petersen et al., 2019, SLP injects a position-dependent logit bias that softly favours a target length band. Let i denote the current position in the sequence, the added logits are:

$$\psi_i^{\text{SLP}} = \underbrace{\Big(-\tfrac{(i-\lambda)^2}{2\sigma^2}\,\mathbf{1}_{i>\lambda}\Big)}_{\text{discourage too-long}}\mathbf{1}_{\text{binary}} \;\|\; \mathbf{0}_{\text{unary}} \;\|\; \underbrace{\Big(-\tfrac{(i-\lambda)^2}{2\sigma^2}\,\mathbf{1}_{i<\lambda}\Big)}_{\text{discourage too-short}}\mathbf{1}_{\text{terminal}}, \tag{2.61}$$

With hyperparameters λ (target length) and σ (width). In the probability space, Equation 2.61 behaves like a multiplicative Gaussian window: before the target length, it tempers terminal choices; after the target, it tempers binary expansions. The result is a smoother, learnable distribution over lengths that reduces reliance on hard truncation and balances the exploration of short and long expressions.

2.7.2. Neural-guided genetic programming

One limitation of DSR is that its exploration is restricted by the distribution learned by the RNN policy. Once the policy starts to focus its probability mass around a few promising regions, it becomes less likely to explore diverse expressions that might lead to higher rewards. To address this, Mundhenk et al., 2021 proposed combining the strengths of DSR and genetic programming (GP) through a hybrid approach called neural-guided GP population seeding.

In this method, the RNN policy from DSR is used to initialise the population of a GP algorithm. Instead of starting from a completely random set of expressions, the initial GP population is seeded with samples generated by the RNN, where the top expressions are used to train the RNN. This hybrid structure allows both methods to complement each other. The RNN provides informed initial populations based on its learned distribution, guiding GP toward promising regions of the search space. GP alone, on the other hand, can make large, non-local modifications to expressions through its stochastic operators, effectively exploring areas that the RNN is unlikely to reach on its own. The evolved GP individuals also introduce new high-reward samples that help the RNN avoid premature convergence and improve its exploration.

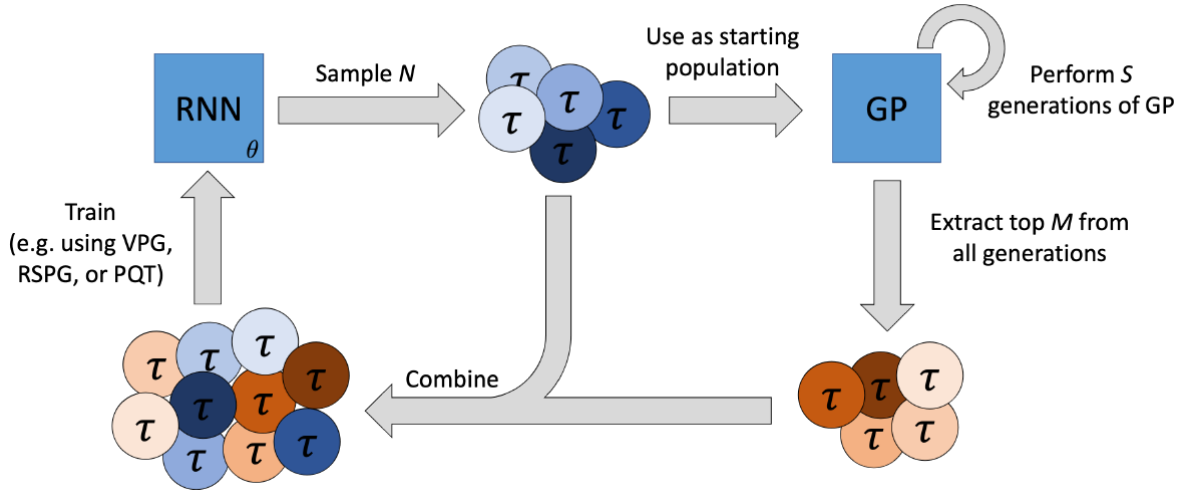


Figure 2.6: Neural-guided genetic programming population seeding framework. The RNN generates an initial population for GP, which evolves the expressions. The best GP individuals are then used to train the RNN. Image by Mundhenk et al., 2021

Figure 2.6 illustrates the overall framework of neural-guided GP. This weak coupling between the RNN and GP ensures that the system remains stable and avoids off-policy issues, while still allowing knowledge transfer between the two components. The RNN continuously improves its sampling distribution based on the best GP individuals, while GP benefits from increasingly better initial populations as training progresses. This interaction improves exploration efficiency and recovery rates on standard symbolic regression benchmarks compared to using DSR or GP alone.

2.7.3. Unified Deep Symbolic Regression (uDSR)

Landajuela et al., 2022 introduced the unified Deep Symbolic Regression (uDSR) framework, which combines several complementary symbolic regression strategies into a single modular system. The system builds upon DSR as its base and integrates additional modules that perform recursive simplification, large-scale pre-training, genetic optimisation, and linear subcomponent fitting. The uDSR framework consists of five main components:

1. **Recursive problem simplification (AIF):** Based on AI Feynman Udrescu et al., 2020, this module decomposes the main symbolic regression problem into smaller, lower-dimensional subproblems by detecting separable relationships in the dataset.

2. **Neural-guided search (DSR):** Serves as the core of the framework and provides a neural controller that generates symbolic expressions through policy gradient reinforcement learning.

3. **Large-scale pre-training (LSPT):** Inspired by the NeSymReS framework by Biggio et al., 2021, this component pre-trains the RNN controller using a transformer-based encoder-decoder model on a large dataset of synthetic symbolic regression problems. The pre-training phase enables the network to learn general relationships between input-output data pairs (X, y) and symbolic expressions, improving generalisation and convergence speed when fine-tuned with reinforcement learning on new tasks.

4. **Genetic programming (GP-meld):** Incorporated as an inner-loop optimisation module that evolves the expressions sampled by the RNN. GP applies crossover and mutation to these initial populations to produce new candidate expressions, in which the top-performing expressions are used to train the RNN.

5. **Linear models (LM):** Introduces a new `LINEAR` token that represents a full linear model composed of a set of basis functions $\phi(x)$ and coefficients β. The coefficients are solved using sparse linear regression, such as LASSO, during expression evaluation. This allows uDSR to efficiently

represent and optimise linear subcomponents within nonlinear expressions, improving accuracy without sacrificing interpretability.

All components in uDSR are connected within a single modular framework that operates in both pre-training and fine-tuning stages. The process begins with AIF decomposing the original problem into simpler sub-problems. For each sub-problem, the pre-trained controller from LSPT is fine-tuned using the risk-seeking policy gradient from DSR, where DSR acts as the expression generator for the transformer. The RNN-generated expressions are then evolved through GP to enhance exploration and diversity, while the LINEAR token efficiently optimises any embedded linear subcomponents.

2.7.4. Deep Symbolic Regression for EARSM

Some works have used DSR to develop explicit algebraic Reynolds-stress models (EARSM). Hemmes, 2022 was the first to use DSR to develop explicit algebraic Reynolds-stress models. The tensor basis DSR (TBDSR) method extends the DSR framework to tensor-valued regression tasks. In contrast to previous methods such as SpaRTA, which construct models as sparse linear combinations of a predefined library of functions, the TBDSR approach is fundamentally different. DSR enables the direct discovery of true analytical expressions, closed-form mathematical formulas, whose structure is not constrained by any fixed function library or basis (as in GEP as well). This allows for potentially more physically meaningful functional forms to emerge directly from the data.

The author used a modified version of DSR in which the policy outputs an N times larger action space, where N is the number of basis tensors. This is equivalent to having N separate policies, one for each basis tensor. Thus, each policy can be trained independently to discover the best function for that basis tensor.

The TBDSR method was also applied to the k-corrective RANS framework. DSR found explicit analytical expressions–true closed-form mathematical formulas–whose structure and composition are not restricted by a predefined basis for both the Reynolds-stress tensor correction, and the additional source term for TKE production R.

Tang et al., 2023 also used DSR to develop an explicit algebraic Reynolds-stress model for the Reynolds-stress tensor. The author used a different approach than Hemmes, 2022, where the author used a single policy that output a sequence that was N times longer. This sequence was then split into N parts, where each part corresponds to a basis tensor. The author only applied this to the Reynolds-stress tensor, and also for the k-ε turbulence model instead of the k-ω SST.

2.8. Multi-agent reinforcement learning

Multi-agent reinforcement learning (MARL) is a subfield of reinforcement learning that focuses on scenarios where multiple agents interact in a shared environment. Multiple agents with their own policies, and thus able to make their own decisions, interact within the same environment, each aiming to maximise its own cumulative reward. The agents may cooperate, compete, or act independently, leading to complex dynamics that differ significantly from single-agent reinforcement learning.

A key difficulty in MARL is that the environment becomes non-stationary from the perspective of any individual agent. In single-agent RL, the environment's transition dynamics and reward structure are fixed. In MARL, each agent is simultaneously learning and updating its own policy. As a result, the environment observed by one agent is constantly changing as other agents adapt their behaviours. This makes the learning problem more challenging, as strategies that are effective at one point may become suboptimal as other agents change their policies and adapt to each other's strategies. The unpredictability of other agents' decisions adds an additional layer of complexity to the learning process.

Many methods have been proposed for MARL. MADDPG by Lowe et al., 2017 is one of the most popular algorithms for continuous action spaces, which extends the DDPG algorithm to multi-agent settings. COMA by Foerster et al., 2018 is a policy gradient method that uses a counterfactual baseline to address the credit assignment problem in cooperative multi-agent settings. QMIX by Rashid et al., 2020 is a value-based method that decomposes the joint action-value function into individual agent value functions using a mixing network, allowing for efficient learning in cooperative tasks.

2.8.1. Multi Agent Proximal Policy Optimisation (MAPPO)

PPO is one of the most widely used reinforcement learning algorithms in single-agent domains, but it has seen limited adoption in MARL. This is largely due to the perception that PPO, being an on-policy method, is significantly less sample-efficient than off-policy algorithms such as MADDPG or QMix. However, recent studies Yu et al., 2022 have shown that when configured appropriately, PPO can achieve state-of-the-art results in cooperative multi-agent environments while maintaining competitive sample efficiency. The simplicity and robustness of PPO make it a strong baseline for multi-agent tasks, provided certain implementation and hyperparameter practices are followed. They introduced Multi-Agent PPO (MAPPO), a straightforward extension of PPO to multi-agent settings that leverages centralised training with decentralised execution (CTDE).

In MAPPO, agents with identical observation and action spaces typically share the same policy network, and each agent is differentiated by its agent ID as an input. This is known as parameter sharing, which acts as a form of information sharing between agents. During execution, each agent acts independently using only its local observation. Whereas during training, the shared critic leverages the global state to compute accurate value estimates and advantages.

During training, the objective function is the standard clipped PPO objective, which allows multiple policy updates with the same batch. The loss function for each agent's policy is defined the same as in single-agent PPO, just applied individually to each agent. The actor loss for agent i is:

$$L_i(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{N-1} \min \left(\frac{\pi_\theta(a_t|s_{t,i})}{\mu_i(a_t|s_{t,i})} A_t,\ \text{clip}\left(\frac{\pi_{\theta,i}(a_t|s_t)}{\mu_i(a_t|s_t)}, 1-\gamma, 1+\gamma \right) A_t \right) \right]. \tag{2.62}$$

To compute the policy gradient, the advantage $\hat{A}_t$ is estimated using GAE, as described in Equation 2.50. Where multi-agent PPO differs from single-agent PPO is in the critic network used to estimate the value function. In the multi-agent case, all agents share a centralised critic $V_\phi(s)$ parameterised by ϕ. This critic takes as input the full state s of the environment, rather than the local observation o_i of each agent. This means that the value function used can access additional centralised information beyond the agent's local observation, which is used only during training. This approach follows the centralised training with decentralised execution (CTDE) paradigm, where agents have their own policy for execution, but the critic has access to global information during training. The centralised critic acts as a form of communication between agents during training, allowing for more accurate value estimates and better coordination among agents.

The effectiveness of MAPPO arises from the combination of PPO's stability with the richer training signal provided by the centralised critic. Value normalisation, limited sample reuse, and appropriately sized training batches further mitigate instability from non-stationarity. Empirical results across multiple benchmarks, including the Multi-Agent Particle Environment (Lowe et al., 2017), the StarCraft Multi-Agent Challenge(Samvelyan et al., 2019), Google Research Football (Kurach et al., 2020), and Hanabi (Bard et al., 2020), demonstrate that MAPPO achieves performance comparable or superior to leading off-policy methods. These results highlight that PPO's limitations in multi-agent learning are largely practical rather than fundamental, and that with suitable configuration, MAPPO provides a simple yet powerful framework for cooperative multi-agent reinforcement learning.

3

Multi-agent DSR and experimental setup

DSR is a powerful framework for symbolic regression, but its standard formulation is limited to single-equation discovery. This chapter outlines the methodology developed to extend DSR into a multi-agent reinforcement learning framework, and the multi-agent techniques integrated to improve learning stability and efficiency. It also details the experimental setup used to evaluate the proposed MADSR framework in the context of turbulence modelling.

The chapter first defines the limitations of standard DSR in the context of EARSM, motivating the need for a multi-agent formulation. It then presents the proposed MADSR framework, detailing the formulation and describing the four algorithms developed, ranging from the baseline multi-agent DSR to the novel actor-critic and PPO-based variants. It then details the neural network architectures, expression generation process, reward formulation, and training procedures. Finally, the chapter presents the experimental setup used for turbulence modelling, including the dataset, input features, and implementation of the EARSM and k-corrective RANS frameworks, concluding with an overview of the general MADSR workflow.

3.1. Limitations of DSR in EARSM

Deep Symbolic Regression (DSR) provides a reinforcement learning-based framework for discovering interpretable mathematical expressions from data. However, in its standard form, DSR is designed for single-output regression problems, learning one equation at a time. In contrast, explicit algebraic Reynolds-stress models (EARSM) require discovering multiple scalar functions simultaneously to construct a tensorial mapping. This fundamental mismatch between DSR's single-equation formulation and EARSM's multi-equation structure introduces key limitations when applying DSR to turbulence modelling.

Previous works such as Hemmes, 2022 and Tang et al., 2023 have applied DSR to the EARSM problem, demonstrating that DSR can discover physically interpretable turbulence models. However, both studies encountered methodological constraints inherent to the original DSR design. Two key limitations are identified: (i) the independent treatment of each scalar coefficient function, and (ii) the use of REINFORCE as the learning algorithm.

3.1.1. Independent learning

In the EARSM formulation, the Reynolds stress anisotropy correction Δb_{ij} is represented as a linear combination of basis tensors $T_{ij}^{(n)}$, weighted by scalar coefficient functions $\alpha^{(n)}(I_1, \dots, I_5)$:

$$\Delta b_{ij} = \sum_{n=1}^{10} \alpha^{(n)}(I_1, \dots, I_5) T_{ij}^{(n)}. \tag{3.1}$$

Each $\alpha^{(n)}$ function contributes to the same target quantity Δb_{ij}, meaning that the equations are intrinsically coupled and must be learned jointly. However, existing DSR-based EARSM studies treat each function as an independent regression task. This ignores interdependencies between coefficient functions, leading to potentially inconsistent or redundant expressions across different basis terms.

Because the contribution of each function affects the others, independent learning may produce models that fit well in isolation but perform poorly when combined. A cooperative learning framework, where multiple expressions are learned simultaneously under a shared objective, would allow information exchange between functions, promoting consistency and coordination. Such an approach could capture the coupled nature of the EARSM equations more faithfully.

3.1.2. Use of REINFORCE

DSR is trained using the REINFORCE algorithm, a Monte Carlo policy gradient method that updates the policy based on full trajectory rewards. Although simple and unbiased, REINFORCE is known to suffer from high variance in its gradient estimates because it assigns a single reward to an entire trajectory. This slows convergence and can lead to unstable learning, particularly when reward signals are sparse, as is the case in symbolic regression, where rewards are only computed after a complete expression is generated.

To mitigate this, the original DSR formulation introduces a risk-seeking baseline, which focuses the gradient update on the top-performing samples within each batch. This stabilises learning, but does not address the variance problem directly.

This becomes particularly relevant in a multi-agent setting, where multiple agents share the same environment, thus making the search space much larger. A centralised critic that learns the joint value of all agents' states and actions could provide better credit assignment and coordination, allowing agents to exploit the shared structure of the problem. Therefore, extending DSR with actor-critic methods represents a natural next step toward more stable and cooperative symbolic regression.

3.2. Multi-Agent Deep Symbolic Regression for turbulence modelling

The motivation of this work is to overcome the limitations of standard DSR in multi-equation modelling by formulating the problem as a cooperative multi-agent reinforcement learning task. The key observation is that the EARSM problem naturally maps to a multi-agent system: each function $\alpha^{(n)}$ can be interpreted as an individual agent whose goal is to jointly minimise the model-form error in the predicted Reynolds stress anisotropy tensor. All agents share a common reward signal derived from the global performance of Δb_{ij}, ensuring cooperation toward a unified objective.

Each agent corresponds to one coefficient function $\alpha^{(n)}$, and has its own identity given by the basis tensor $T_{ij}^{(n)}$. Together, the agents need to cooperatively predict the Reynolds stress anisotropy correction Δb_{ij}. This formulation allows modern MARL algorithms, such as that proposed by Yu et al., 2022, to be directly applied to symbolic regression, providing mechanisms for shared learning, variance reduction, and improved coordination.

By leveraging multi-agent reinforcement learning, DSR can be extended beyond independent single-equation discovery toward the cooperative discovery of structured, physically consistent systems of equations. This brings us back to the central research question of this thesis:

How can multi-agent reinforcement learning techniques be integrated with deep symbolic regression to perform symbolic turbulence modelling?

The research question encapsulates the goal of developing a novel framework that combines the strengths of MARL and DSR to address the unique challenges of multi-equation turbulence modelling. This invokes several sub-questions:

1. What MARL algorithms are best suited for cooperative symbolic regression tasks, and how can they be integrated with DSR?
2. How does integrating MARL techniques impact the efficiency of learning and the quality of discovered expressions compared to base DSR?
3. How does the multi-agent DSR framework perform in discovering EARSM closures and k-corrective RANS models?
4. Is a multi-agent DSR approach capable of discovering accurate and interpretable models for turbulence modelling?

3.3. EARSM as a POMDP

EARSM can be thought of as a cooperative multi-agent problem, where each agent is responsible for learning a function $\alpha^{(n)}(x_1, \ldots x_K)$, where n is the agent index, and K is the number of input features. The expressions are then multiplied by their respective basis tensors to get the final Reynolds stress anisotropy tensor. This model is then evaluated using frozen RANS and given a reward based on its performance. The agents are then trained using reinforcement learning to maximise the reward. Thus, the multi-agent POMDP can be defined as follows:

- **Agents:** Each agent is responsible for sampling a function $\alpha^{(n)}(x_1, \ldots x_K)$. Each agent has its own unique contribution to the final expression. In the case of EARSM, each agent is responsible for learning a function that is multiplied by each Pope basis tensor, explained in section 2.3. The final expression is then the sum of all the agents' contributions.
- **Observation space:** The observation space is the current state of the expression, including the current position in the tree and the number of dangling nodes left in the expression.
- **State space:** The state space is the current expression being built by the agents. This is learnt using an RNN, which learns a hidden state representation of the expression.
- **Action space:** The action space is the set of possible tokens that can be added to the expression. This includes operators, variables, and constants.
- **Reward function:** The reward function $R(y, \hat{y})$ is a measure of the performance of the generated expressions.

3.4. MARL techniques

Formulating EARSM as a multi-agent POMDP allows the application of MARL techniques to train the agents cooperatively. In this work, two MARL techniques are investigated for their effectiveness in symbolic turbulence modelling: PPO and CTDE. These techniques are integrated into the DSR framework to create multi-agent variants of DSR. Before describing the specific methodology, the theoretical basis of how these techniques improve upon the limitations of DSR is explained.

3.4.1. Proximal Policy Optimization in multi-agent DSR

The integration of PPO into multi-agent DSR aims to enhance the sample efficiency of DSR by allowing multiple policy updates per batch of sampled expressions. This is to address the second limitation of DSR, which is the use of REINFORCE. PPO clipping allows the policy to be updated multiple times per batch, without the risk of large policy updates that could destabilise learning. This would allow the agents to make better use of the sampled expressions, as they can learn more from each batch.

3.4.2. Centralised Training Decentralised Execution in multi-agent DSR

The CTDE paradigm aims to improve both the independent learning and REINFORCE limitations of DSR. The CTDE paradigm introduces a centralised learned critic that estimates the joint value of all agents' states and actions, to replace the risk-seeking baseline used in DSR. As explained in subsection 2.5.4, the critic is a network that estimates the value function of the current state $V_\phi(s)$. Having such an estimate allows the computation of the advantage function, which quantifies how much better

(or worse) a particular action is compared to the agent's expected performance under its current policy. This allows the agent to learn which actions are better than average, when it is in a certain state.

This provides a more meaningful learning signal for the agent, as it can learn to take actions that are better than average at each step. As a result, the learning process transitions from being return-based, where agents learn which overall trajectories yield higher returns, to being transition-based, where they learn which individual actions are advantageous at each step. Thus, agents can learn which actions are better than average at each step, instead of learning which trajectories are better than others. This would reduce the variance of the gradient estimates, leading to more stable and efficient learning.

In the case of MADSR, the critic is centralised, meaning that it receives the observations of all agents and outputs a single value function. With access to the full state of the environment, the centralised critic can evaluate the combined effect of all agents' actions. The inclusion of a centralised critic would provide a stronger and more stable learning baseline for each actor, as it captures the interactions between agents and how the actions of one agent influence the outcomes of others. It could also learn the structure of the multi-agent problem, allowing it to provide more accurate value estimates. Thus, the centralised critic acts as a form of communication between agents, allowing them to coordinate their learning more effectively.

3.5. Algorithms

Four different multi-agent DSR (MADSR) algorithms are implemented and tested. The base DSR algorithm is extended to the multi-agent setting as a baseline test case, as this was already done in the works of Hemmes, 2022 and Tang et al., 2023. A PPO version of the base DSR algorithm is also implemented as a comparison to the vanilla multi-agent DSR. The main novelty of this work is the implementation of an actor-critic version of multi-agent DSR, as well as a PPO version of the actor-critic multi-agent DSR. These two algorithms are inspired by Yu et al., 2022, where a centralised critic and decentralised actors, the CTDE paradigm, is used. The four algorithms are summarised below.

The first is **vanilla multi-agent DSR (vDSR)**. This is the simplest version of multi-agent DSR. This algorithm uses the base DSR algorithm, but with multiple agents. This algorithm is fairly similar to Hemmes, 2022, where the original DSR algorithm, with its risk-seeking baseline and risk-seeking policy gradient, is used. But now it is extended to multiple agents. Each agent learns independently of the other, while still being able to share parameters.

The second algorithm is a PPO version of **vanilla multi-agent PPO DSR (vPPODSR)**. This algorithm extends the risk-seeking policy gradient to include the PPO clipping function. The risk-seeking baseline is still used, but PPO clipping allows the policy to be updated multiple times per batch, which aims to improve sample efficiency. Vanilla multi-agent DSR and its PPO version both do not use the CTDE paradigm, and thus do not have a learned critic. The agents still learn independently of each other, only sharing parameters.

The third algorithm is an actor-critic version of **multi-agent AC DSR (ACDSR)**. In this algorithm, a centralised critic is used to estimate the value function. An advantage function replaces the risk-seeking baseline, and the agents are trained using the advantage function. This algorithm introduces CTDE to multi-agent DSR, allowing the agents to learn cooperatively using a shared critic. This algorithm is used to assess the effectiveness of CTDE in multi-agent DSR. But the base REINFORCE loss function is still used, and thus only one update per batch is done.

The final algorithm is a PPO version of ACDSR, the **multi-agent PPO DSR (PPODSR)**. This algorithm extends the actor-critic multi-agent DSR by including PPO clipping. This algorithm uses both a centralised critic and PPO clipping, allowing the agents to learn cooperatively using a shared critic, while also being able to update the policy multiple times per batch. This is the most advanced algorithm implemented in this work, as it combines both PPO and CTDE techniques. This algorithm is expected to perform the best out of the four algorithms, as it combines the strengths of both techniques.

Table 3.1: Summary of the four implemented multi-agent DSR algorithms and their main characteristics.

Algorithm	Risk-seeking baseline	Critic	PPO clipping
Vanilla MADSR (vDSR)	✓	×	×
Vanilla PPO MADSR (vPPODSR)	✓	×	✓
Actor-Critic MADSR (ACDSR)	×	✓	×
PPO Actor-Critic MADSR (PPODSR)	×	✓	✓

These four algorithms, and their differences, are summarised in Table 3.1. They mainly differ in the training step, where the differences are whether a critic is used, whether PPO clipping (and thus epochs and mini-batches are used), and whether the risk-seeking baseline is used.

These four algorithms are tested and compared against each other to assess their performance. The main focus of this work is to assess the performance of the actor-critic multi-agent DSR and its PPO version, as these are the most novel algorithms. The vanilla multi-agent DSR and its PPO version are used as baselines to compare against.

Algorithm 2 General Training Procedure for Multi-Agent Deep Symbolic Regression (MADSR)

Initialize actor(s) $\pi_{\theta,n}$ and (optional) critic V_ϕ
for each training iteration **do**
  for i=1 to B **do**
    $\tau^{(i)} \sim \pi_{\theta,n}$ for all agents n — *Sample B expressions*
  end for
  $\min\{c_j\} \in \tau$ — *Fit constants for each expression*
  $\hat{y} \leftarrow F(X_{\text{train}}, X_{\text{context}})$ — *Evaluate expressions*
  $r^{(n,i)} \leftarrow R(y, \hat{y})$ — *Compute rewards*
  $\mathcal{D}_{sub} \leftarrow \{(s,a) \mid r^{(n,i)} \geq R_\epsilon\}$ — *Select top ϵ subset*
  $A_t \leftarrow \text{AdvantageFunction}(\tau)$ — *Compute advantage (Depends on algorithm)*
  $L_{\text{actor}} \leftarrow \text{ActorLoss}(\theta, \tau, A_t)$ — *Compute actor loss (Depends on algorithm)*
  $L_{\text{entropy}} \leftarrow \text{EntropyLoss}(\theta, \tau)$ — *Compute entropy loss*
  $\theta \leftarrow \theta + \nabla_\theta L_{\text{actor}} + w_e \nabla_\theta L_{\text{entropy}}$ — *Actor update*
  if critic used (Depends on algorithm) **then**
    $L_{\text{critic}} \leftarrow \text{CriticLoss}(\phi, \tau)$ — *Compute critic loss*
    $\phi \leftarrow \phi - w_c \nabla_\phi L_{\text{critic}}$ — *Critic update (MSE)*
  end if
end for

Algorithm 2 shows a general algorithm for multi-agent DSR, which applies to all algorithms. It starts with initialising the actor and critic networks. Then, for each training iteration, a batch B expressions are sampled from each agent's policy. The constants in the expressions are then optimised. The expressions are then evaluated on the training dataset X_{train} and context variables $X_{context}$ to get the model predictions $\hat{y}$ using the evaluation function F. The rewards are then calculated using the reward function $R(y, \hat{y})$. The top ϵ of the rewards are then selected to form the sub batch $\mathcal{D}_{sub}$.

All steps that came before this are common to all four algorithms. The definition of the advantage function (`AdvantageFuntion`), loss function (`ActorLoss`), and critic updates are where the algorithms differ. The advantage function is defined differently if the risk-seeking baseline or a learned critic is used. The actor loss function also differs depending on whether PPO clipping is used or not. The critic is only updated if a learned critic is used. The specific details of these components will be explained in the following sections.

3.6. Expression generation

Each agent is independent, and thus has its own policy π_n, where n is the agent index. So each agent samples B expressions using its own respective policy, in the same way as single-agent DSR, explained in subsection 2.6.2, where there are priors, observations, and actions. The only difference is that an additional agent ID is added to the observation space, which will be explained in more detail in subsection 3.6.2. This results in $N \times B$ expressions, where N is the number of agents.

3.6.1. Function sets

The function set determines the operators that can be used to generate the expressions. A good function set is important, as an insufficient function set can limit the expressiveness of the generated expressions, while a function set that is too large can make the search space too large, making it difficult for the agents to learn. The following function set is used: add, subtract, multiply, divide, logarithm, exponential, hyperbolic tangent, and regularised division (R_div).

Basic arithmetic operators (add, subtract, multiply, and divide) are essential for any expression. Logarithms and exponentials have seen some use in turbulence modelling, such as in Hemmes, 2022, where logarithms appeared quite often in the discovered expressions. A combination of logarithm and exponential can also be used to represent power functions, which could be useful in turbulence modelling. Hyperbolic tangent is also used quite often in turbulence modelling, as it is a smooth approximation of the clipping function, which is commonly used in turbulence modelling to prevent unphysical values.

$$R\_div(x) = \frac{x}{1 + x^2} \tag{3.2}$$

The regularised division is defined in Equation 3.2, and is the ratio of x and $1 + x^2$. This function was used as it was found to be able to approximate EARSM models quite well in Buchanan et al., 2025.

3.6.2. Network architecture

For the actor network, a similar architecture to the original DSR is used. It uses a multi-layer long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997) to provide state estimation and record the history of the expression. The outputs of the LSTM are then fed into linear layers, where the final layer output has a size of the action space, which is the number of possible tokens that can be next in the expression.

At each step, the agent takes in observations of the current state of the expression. The observation space is made up of the following components:

- The previous token that was added to the expression. This is represented as a one-hot vector of size equal to the number of possible tokens.
- The parent of the current token. This is represented as a one-hot vector of size equal to the number of possible tokens.
- The left sibling of the current token. This is also represented as a one-hot vector of size equal to the number of possible tokens. This token can be null if the current token is the first child of its parent, or if the parent is a unary operator.
- The number of dangling nodes left in the expression

The framework also uses parameter sharing between the agents, so the agents are able to communicate and help each other. To achieve this, an extra input is required in the observation space, which is the agent ID. This is represented as a one-hot vector of size equal to the number of agents, which is concatenated to the observation vector. This agent ID allows the network to differentiate between the agents. This allows each agent to learn different policies while still being able to share parameters.

For the critic network, a similar architecture to the actor network is used, where there is an LSTM followed by linear layers. The final linear layer has an output size of one, which is the estimated value function. For the observations, as the critic is centralised, it takes in the observations of all the agents. Thus, the observation space for the critic would be the concatenation of the observation spaces of all the agents.

Table 3.2 summarises the architecture details for both the actor and critic networks. The RNN for the actor is set to be narrower, but deeper, as the actor needs to be able to process the inputs and generalise to a distribution over actions. The RNN for the critic was set to be wider, as the critic needs to be able to process the observations of all agents, and thus would need to encode more information.

Table 3.2: Neural network architecture details for actor and critic.

	Actor RNN	Actor linear layers	Critic RNN	Critic linear layers
Number of layers	3	2	2	2
Hidden layer size	48	32	64	32
Activation function	-	$tanh$	-	$tanh$

3.6.3. Priors

Similar to single-agent DSR, priors are used to impose constraints on what expressions can be generated. These priors were optimised to fit the multi-agent setting. All agents share the same priors, as most constraints apply to all agents. The base priors that are used in the base single-agent DSR are also used in multi-agent DSR, but some priors were modified to better suit the multi-agent setting. Only modified priors are explained here, while unmodified priors are the same as in subsection 2.6.2.

- **Length prior:** This prior constrains the length of the expression. No minimum length is set to allow for simple expressions to be generated. This allows for expressions such as x_1 or 3.14 to be generated. Setting no minimum length in multi-agent DSR is important, as one agent may learn a simple function, such as a constant, or even 0, which would suggest that the contribution of this agent is not important. A maximum length of 10 tokens is set, as this encourages the agents to learn simple expressions and prevents overfitting. Since there are multiple agents, each agent can learn a simple expression, and the final expression can still be complex. For example, if there are 3 agents, each agent can learn an expression of length 10, resulting in a final expression equivalent to an expression of length 30.
- **No inputs constraint:** This prior prevents the expression from having no inputs. While it prevents expressions such as exp or $log(3.56 + tanh(5.47))$, which are redundant constants, it was augmented to still allow constants to be the only input. This is to allow for expressions such as 3.14 or 2.71, which are valid expressions in the case of multi-agent DSR.

Aside from this, additional constraints were added to further improve the performance of multi-agent DSR. These priors are:

- **Repeat constraint:** This prior prevents the same token from being sampled more than the maximum repeat limit. The constraint was used on constants, as a model can overfit really easily by using a large number of constants. An example is that the model can learn an expression that is just a linear combination of simple functions, and fit really well, which does not really provide much meaningful insight. Thus, a maximum repeat limit of 3 was set for constants.
- **Relational constraint:** This prior prevents certain relations between tokens. It can prevent a certain token from being a child, descendant or sibling of another token. In MADSR, it was used to prevent nested operators from occurring by preventing tokens from being descendants of the same token. This was from an observation of initial tests where expressions like $\log(\log(\log(x)))$ or $\tanh(\tanh(x))$ were being generated, which are not very physical and do not occur in science. The tokens that were chosen for this prior were the unary operators $\{\log, \exp, R_{div}, \tanh\}$.
- **Soft length prior:** This prior was proposed by Landajuela et al., 2021, and is explained further in subsection 2.7.1. A target length of two and a width of two was used. This is to encourage expressions to be as short as possible. This also encourages shorter expressions like x_1 or 3.14 to be generated, which could be useful in multi-agent DSR.

3.7. Reward function

Constants are one of the tokens that can be added to the expression, and are always one until this point. Before calculating the reward of the expressions, the constants need to be optimised to fit the data. This is done using *SciPy*'s BFGS optimiser, which is a quasi-Newton method for optimisation. This optimiser is used to find the optimal constants that minimise the MSE between the predicted and true values. Now, N expressions are obtained, each with its own optimised constants, and an example expression is shown here:

$$\alpha^{(1)}(x_1, x_2) = 4.8 \cdot log(x_1) + 2.3, \quad \alpha^{(2)}(x_1, x_2) = tanh({x_2}^2 + 0.13), \quad \alpha^{(3)}(x_1, x_2) = e^{-3.1 \cdot x_1} + 0.7x_2 \quad (3.3)$$

Now that the expressions are obtained, with their respective constants optimised, they are evaluated to get the prediction $\hat{y}$. First, the variables $x_1, ...x_K$ are input into the preorder traversal, and the expressions are evaluated to obtain the output $\alpha^{(n)}$.

In a more general sense, these $\alpha^{(n)}$ values are then put into some function $F(\alpha^{(1)}, ..., \alpha^{(N)})$, which is defined depending on the multi agent regression problem, to get the final prediction $\hat{y}$. In the case of EARSM, the function F is simply a weighted sum of the basis tensors, where each $\alpha^{(n)}$ is multiplied by its respective basis tensor, and then summed together to get the final Reynolds stress anisotropy tensor. This results in the model prediction $\hat{y}$.

To get the reward R, the model prediction $\hat{y}$ is compared to the true values y using some reward function $R(y, \hat{y})$. The R^2 score was used as the reward function, as it was found to work the best compared to other methods tested. The R^2 score is also a commonly used metric to measure the performance of EARSM models (Lăcătuş, 2024, Siddiqui, 2025), as it takes into account the variance of the dataset. For invalid expressions, where the expression cannot be evaluated due to mathematical errors such as division by zero, logarithm of a negative number or having an expression that is too long, a reward of -10 is given to heavily penalise the agents.

The R^2 score is defined in Equation 3.4 in the case of an EARSM model, where i and j are the indices of the tensor, and k is the index of the data point. The R^2 score is bounded between $-\infty$ and 1, where 1 is a perfect prediction, and 0 means that the model is as good as predicting the mean of the dataset. A negative R^2 score means that the model is worse than predicting the mean of the dataset.

$$R(y, \hat{y}) = 1 - \sum_i \sum_j \frac{\sum_k (y_{ij}^k - \hat{y}_{ij}^k)^2}{var(y_{ij})} \quad (3.4)$$

Since the $\hat{y}_{ij}$ and y_{ij} values are tensors, the magnitudes of each element might be different, and thus each element's contribution to the R^2 score might vary. If an element has a very small magnitude throughout the dataset, then it will have a very small contribution to the MSE, and consequently the R^2 score. This might cause the agents to ignore that element, as it does not contribute much to the reward. To prevent this, each element of the tensor is normalised using min-max normalisation, where each element is scaled to be between -1 and 1. This ensures that each element has an equal contribution to the R^2 score, and thus the agents are encouraged to learn all elements of the tensor. Before evaluating the reward, Equation 3.5 is applied to each element of the tensor, where y_{ij}^{min} and y_{ij}^{max} are the minimum and maximum values of the element y_{ij} in the dataset.

$$y_{ij}^k = 2 \cdot \frac{y_{ij}^k - y_{ij}^{min}}{y_{ij}^{max} - y_{ij}^{min}} - 1 \quad (3.5)$$

3.7.1. Training batch organisation

Before moving on to training the agents, the data needs to first be organised into a training batch. It was found that using risk-seeking, like in subsection 2.6.3, improved the performance when extended to multi-agent DSR. In preliminary tests, all algorithms performed worse when risk-seeking was not used, Thus, a similar approach is used here for all algorithms.

The rewards of the whole batch of size B are sorted in descending order, and the reward of the ϵ-th percentile was calculated. This reward is called the baseline reward R_ϵ, which is used as a threshold where only the expressions with rewards greater than or equal to R_ϵ are used to train the actors.

3.8. Advantage function

The advantage function is a measure of the performance of the current action taken by the agent. Therefore, it is an important component of the loss function as it provides a learning signal for the agents. The advantage function is defined differently depending on whether a risk-seeking baseline or a learned critic is used. Two types of advantage functions are used, depending on whether a risk-seeking baseline or a learned critic is used.

3.8.1. Risk-seeking advantage function

For algorithms that use the risk-seeking baseline, the baseline reward R_ϵ is used as a baseline. Thus, the advantage function is defined as:

$$A_{risk} = R(\tau) - R_\epsilon \tag{3.6}$$

In this case, the advantage function A is simply the difference between the total return of the trajectory $R(\tau)$ and the baseline reward R_ϵ. It does not change over the steps of the trajectory, and is constant for the entire trajectory.

3.8.2. Learned critic advantage function

There are many ways to calculate the advantage function, such as temporal difference learning, n-step returns, Monte Carlo returns, and generalised advantage estimation (Schulman, Moritz, et al., 2015). Due to the sparse nature of the rewards (only getting a reward at the end of the expression), it is best to bootstrap the value function as quickly as possible, so that long expressions can still have a good estimate of the value function. While generalised advantage estimation is the most commonly used method in MARL, in this application, Monte Carlo returns would be the best option. The Monte Carlo target is defined as:

$$y_l = \sum_{k=l+1}^{L} \gamma^{k-l} r_k \tag{3.7}$$

Where L is the terminal time step, γ is the discount factor, and r_k is the reward at time step k. The values are then predicted using the critic network, with all agents' observations as input. The advantage function is then calculated using Equation 3.8.

$$A(s_l) = y_l - V_\phi(s_l) \tag{3.8}$$

3.9. Loss function

Equation 3.9 shows the total loss function used to train the agents. The total loss is a combination of three losses: the actor loss $L_{actor}(\theta)$, the critic loss $L_{critic}(\phi)$, and the entropy loss $L_{entropy}(\theta)$.

$$L(\theta, \phi) = L_{actor}(\theta) + w_c \cdot L_{critic}(\phi) + w_e \cdot L_{entropy}(\theta) \tag{3.9}$$

w_c and w_e are weights for the critic loss and entropy loss respectively, and are hyperparameters that need to be tuned for each algorithm. The Adam optimiser (Kingma, 2014) is used to train the networks, and it was found that a learning rate of 5×10^{-4} worked well for all algorithms. A small learning rate was used to prevent overshooting, which could cause oscillations during training.

In this section, general forms of the loss functions are presented. The algorithms mainly differ in the forms of the loss functions used, as well as whether a critic is used or not. The exact forms of the loss functions for each specific algorithm being used will be explained in their respective subsections later.

3.9.1. Actor loss

The actor loss is the component of the loss to train the actor networks. This loss differs depending on the algorithm used, and is explained in the respective sections for each algorithm. The general form of the actor loss is shown below:

$$L_{actor}(\theta) = -\frac{1}{NB}\sum_{n=1}^{N}\sum_{i=1}^{B}\sum_{l=1}^{L} A_{l,i} \cdot 1|_{R(\tau)\geq R_\epsilon} l(\pi_\theta(a_l|s_{l,i}^{(n)})) \tag{3.10}$$

Here, A_t is the advantage function, and is defined differently depending on the algorithm used. The advantage function should always just be a constant, and no gradients should flow through it. Thus, no backpropagation is done through the advantage function.

$l(\pi_\theta(a_l|s_l^{(n)}))$ represents the log probability of the action taken at step l given the state $s_l^{(n)}$ for agent n. The exact form of this term depends on the algorithm used. It provides a way to update the probability distribution of the policy weighted by the advantage function. A higher log probability means that the action taken is more likely to be taken again in the future, while a lower log probability means that the action taken is less likely to be taken again. This term is where the actor network parameters θ are updated.

The indicator function $1|_{R(\tau^{(n)})\geq R_\epsilon}$ is the risk-seeking indicator, which represents use of the risk-seeking method. The indicator is 1 if the reward of the expression generated is greater than or equal to the baseline reward R_ϵ, and 0 otherwise.

$$L_{actor}(\theta) = -\mathbb{E}_{\pi,n}\left[A_l \cdot 1|_{R(\tau)\geq R_\epsilon} l(\pi_\theta(a_l|s_l^{(n)}))\right] \tag{3.11}$$

Since parameter sharing is used, the identity of the agent n is included in the state $s_l^{(n)}$. Thus, the policy network π_θ does not depend on the agent index n, and only the state $s_l^{(n)}$ depends on it. Since the agent index n is just a state, the training of the different agents can just be done by concatenating all the agents' data together, resulting in a full batch of size $N \times B$. In further sections, the averaging over n agents and batch size B will be written as in Equation 3.11 for simplicity and clarity.

3.9.2. Critic loss

The critic loss is the component of the loss used to train the critic. It is just the mean squared error of the advantage function, shown in Equation 3.12. This loss is only used for actor-critic algorithms.

$$L_{critic}(\phi) = \frac{1}{B}\sum_{i=1}^{B}\sum_{l=0}^{L-1} A(s_{l,i}, \phi)^2 \tag{3.12}$$

Here, $A(s_l, \phi)$ is the advantage function, s_l is the state of all agents at time step l, and ϕ is the parameters of the critic network. The state s_l here does not depend on the agent n, since the critic has access to the observations of all agents.

For the critic loss, it is important that the full batch is used to train the critic, and not just the top ϵ percentile of rewards. This is because the critic is supposed to learn the expected return of the policy, and not the expected return of the top ϵ percentile of rewards.

The critic learns the expected returns through Monte Carlo sampling, and thus needs to see the full distribution of rewards to learn an accurate estimate. If the critic only saw the top ϵ percentile of rewards, it would learn a biased estimate of the value function, leading to overestimation of the value function. Thus, the critic loss does not have the risk-seeking indicator function.

3.9.3. Entropy loss

The entropy loss is used to encourage exploration, and is the same for all algorithms. The hierarchical entropy by Landajuela et al., 2021 is used, which was explained in subsection 2.7.1. The entropy loss is shown in Equation 3.13, where $\pi_\theta(s_l^{(n)})$ is the probability distribution of the policy at state $s_l^{(n)}$, and $H(\pi_{\theta,n})$ is the entropy of that probability distribution.

$$L_{entropy}(\theta) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{l=0}^{L-1}\lambda^l H(\pi_\theta(s_l^{(n)})) \tag{3.13}$$

The search space is very large, and thus exploration is crucial to find good expressions. The entropy loss penalises low entropy policies, thus encouraging the actors to have a more uniform distribution over actions. This makes the entropy loss very important for MADSR, as without this loss, the agents would quickly converge to suboptimal policies and get stuck in local minima. The entropy loss encourages the agents to explore more, and thus find better expressions.

3.10. Loss function for each algorithm

The main difference between the four algorithms lies in their loss functions. In this section, the specific loss functions for each algorithm are explained in detail.

3.10.1. vanilla MADSR

This is just the base version of DSR, extended to the multi-agent setting. Each agent n maintains its own policy network $\pi_{\theta,n}$, and is trained using the risk-seeking policy gradient:

$$L_{actor}(\theta) = \mathbb{E}_{\pi,n}\left[(R(\tau) - R_\epsilon)\sum_{l=0}^{L-1}\cdot 1|_{R(\tau)\geq R_\epsilon}\log\pi_\theta(a_l|s_l^{(n)})\right]. \tag{3.14}$$

Here, the risk-seeking baseline R_ϵ is retained, and thus the risk-seeking advantage function is used. The original log probabilities of REINFORCE are used, and the critic is not used in this algorithm.

For this algorithm, an entropy weight of $w_e = 0.03$ was used, as the resulting advantages had small magnitudes, and thus a smaller entropy weight is needed so that the entropy loss does not dominate the total loss. As there is no critic in this algorithm, the critic weight w_c is set to 0, and the critic loss is not calculated.

3.10.2. vanilla MAPPO DSR

This algorithm extends the vanilla multi-agent DSR by including PPO in the multi-agent context. This allows for multiple epochs and mini-batch updates, which should lead to more data-efficient training, since the same data is used multiple times. The loss function for the PPO actor in this framework is:

$$L_{actor}(\theta) = \mathbb{E}_{\pi,n}\left[A_{risk}\sum_{l=0}^{L-1}\min\left(\frac{\pi_\theta(a_l|s_l^{(n)})}{\mu(a_l|s_l^{(n)})},\ \text{clip}\left(\frac{\pi_\theta(a_l|s_l^{(n)})}{\mu(a_l|s_l^{(n)})}, 1-\gamma, 1+\gamma\right)\right)1\Big|_{R(\tau)\geq R_\epsilon}\right]. \tag{3.15}$$

For this algorithm, it is mostly the same as vDSR, but with PPO clipping added to the loss function. Four epochs and a division into four mini-batches per iteration were used for this algorithm. A PPO clipping value of $\gamma = 0.2$ was used, as this is a commonly used value in PPO.

3.10.3. Actor-critic MADSR

This algorithm is where the main novelty of this work lies. It introduces CTDE into multi-agent DSR, turning it into an actor-critic method. In this framework, the critic loss is non-zero, and the critic network will be trained. The actor loss for this algorithm is:

$$L_{actor}(\theta) = \mathbb{E}_{\pi,n}\left[\sum_{l=0}^{L-1}A(s_l)\cdot 1\Big|_{R(\tau)\geq R_\epsilon}\log\pi_\theta(a_l|s_l^{(n)})\right]. \tag{3.16}$$

For the advantage function, the learned critic advantage function is used, as explained in section 3.8. In this case, the critic network also needs to be trained, and thus the critic loss is non-zero. The critic weight w_c was found to be quite an important parameter. A critic weight that is too large causes too much bias, and can cause overshoots of critic estimations. This causes oscillations in the critic estimation, which in turn causes oscillations in the advantage function, and finally causes oscillations in the actor loss. This results in unstable training. A critic weight that is too small is also detrimental, as the policy will change too fast compared to the critic, causing the critic to not be able to keep up with the policy changes. This causes the critic to provide poor value estimates, which in turn causes poor advantage estimates, and finally, poor actor updates. After hyperparameter tuning, a critic weight of $w_c = 0.7$ was found to be the optimal critic weight.

In actor-critic methods, the advantage function values have a larger magnitude compared to vanilla DSR in general. Thus, a larger entropy weight of $w_e = 1.5$ was used to prevent the actor loss from dominating the entropy loss. This prevents the actors from converging too quickly to a suboptimal policy.

3.10.4. MAPPO DSR

Similar to vanilla MADSR, the natural next step is to extend the method to include PPO in actor-critic DSR. The loss function for the PPO actor in this framework is:

$$L_{actor}(\theta) = \mathbb{E}_{\pi,n}\left[\sum_{l=0}^{L-1} A(s_l)\min\left(\frac{\pi_\theta(a_l|s_l^{(n)})}{\mu_i(a_l|s_l^{(n)})},\ \mathrm{clip}\left(\frac{\pi_\theta(a_l|s_l^{(n)})}{\mu(a_l|s_l^{(n)})}, 1-\gamma, 1+\gamma\right)\right)\cdot 1|_{R(\tau)\geq R_\epsilon}\right]. \quad (3.17)$$

Four epochs and a division into four mini-batches were also used for this algorithm, and a PPO clipping value of $\gamma = 0.2$ was also used. The same advantage function, and thus the same critic loss, as in ACDSR was used. The critic weight was also kept the same at $w_c = 0.7$. The entropy weight had to be slightly increased to $w_e = 2.0$, as the multiple updates per batch caused the policy to converge faster, and thus a larger entropy weight was needed to prevent premature convergence and constant exploration.

3.11. Experimental setup for turbulence modelling

In this section, the experimental setup for turbulence modelling is explained. Two different turbulence modelling frameworks are tested using multi-agent DSR: EARSM and k-corrective RANS. The flow case used is also illustrated here, and how the training data is used to train MADSR.

3.11.1. EARSM

In EARSM, the Reynolds stress anisotropy tensor Δb_{ij} is modelled as a linear combination of basis tensors $T_{ij}^{(n)}$, weighted by scalar functions $\alpha^{(n)}$ of invariant flow features, as in Equation 2.28. In this work, it was decided that only the first four basis tensors would be used. According to Pope, 1975, three basis tensors are sufficient to represent a two-dimensional flow. But using the 4th basis tensor allows for more complex expressions to be learned, and thus potentially better performance. Both the three- and four-basis tensor models were tested, and the four basis tensors are defined below:

$$T_{ij}^{(1)} = S_{ij} \qquad T_{ij}^{(2)} = S_{ij}\Omega_{ij} - \Omega_{ij}S_{ij} \quad (3.18)$$

$$T_{ij}^{(3)} = {S_{ij}}^2 - \frac{1}{3}I\,\mathrm{trace}({S_{ij}}^2) \qquad T_{ij}^{(4)} = {\Omega_{ij}}^2 - \frac{1}{3}I\,\mathrm{trace}({\Omega_{ij}}^2). \quad (3.19)$$

In a two-dimensional flow, only the first two basis tensors are non-zero. Thus, the first two invariants are used as inputs to the expressions. These invariants are defined as:

$$I_1 = trace({S_{ij}}^2) = S_{nm}S_{mn}, \quad I_2 = trace({\Omega_{ij}}^2) = \Omega_{nm}\Omega_{mn}. \quad (3.20)$$

Additionally, 2 other scalar inputs are used as well. These are:

- **$\mathbf{D_k/C_k}$ ratio:** This is the ratio between the turbulence destruction rate D_k, and the convection rate C_k of turbulent kinetic energy k. This is defined in Equation 3.21

$$D_k/Ck = \frac{|D_k|}{|D_k| + |C_k|} \tag{3.21}$$

- **$\mathbf{D_k/P_k}$ ratio:** This is the ratio between the turbulence destruction rate D_k, and the production rate P_k of turbulent kinetic energy k. This is defined in Equation 3.22

$$D_k/P_k = \frac{|D_k|}{|D_k| + |P_k|} \tag{3.22}$$

These variables were introduced in Buchanan et al., 2025, and were found to be useful in turbulence modelling. Thus, the final expressions generated by multi-agent DSR will have the form of $\alpha^{(n)}(I_1, I_2, D_k/Ck, D_k/P_k)$, where n is the agent index. The final correction model is then given by:

$$\Delta b_{ij} = \sum_{n=1}^{3,4} \alpha^{(n)}(I_1, I_2, D_k/Ck, D_k/P_k) T_{ij}^{(n)}. \tag{3.23}$$

3.11.2. k-corrective RANS

The k-corrective RANS provides a correction to the production deficit term R, which is added to the production term in the turbulent kinetic energy equation. It follows the framework by Schmelzer et al., 2020, and was explained further in subsection 2.3.2. The k-corrective RANS is slightly different from EARSM, as the production deficit term R is a scalar instead of a tensor like Δb_{ij}.

k-corrective RANS is still multi-agent in nature, as each agent generates an expression for a scalar function $\beta^{(n)}(I_1, I_2, D_k/Ck, D_k/P_k)$, but the evaluation function $F(\cdot)$ turns the tensor inputs into a scalar. An intermediary term b_{ij}^R is first computed using Equation 3.24, which has the same form as EARSM. It uses the same inputs as EARSM as well, but only the first 3 basis tensors are used.

$$b_{ij}^R = \sum_{n=1}^{3} \beta^{(n)}(I_1, I_2, D_k/Ck, D_k/P_k) T_{ij}^{(n)} \tag{3.24}$$

Currently, b_{ij}^R is still a tensor, but it needs to be turned into a scalar production correction term R. This is done using Equation 3.25, where P_k is the production rate of turbulent kinetic energy k.

$$R = 2k b_{ij}^R \frac{\partial u_i}{\partial x_j} + \beta^{(\varepsilon)}(I_1, I_2, D_k/C_k, D_k/P_k) \cdot \varepsilon \tag{3.25}$$

Here, an additional expression $\beta^{(\varepsilon)}$ is generated by another agent, where it is multiplied with the turbulence dissipation rate ε. This term was shown to be effective in Buchanan et al., 2025, and thus is included in this work as well. Thus, in total, four agents are used for the k-corrective frozen RANS framework, with three agents generating expressions for b_{ij}^R using 3 basis tensors, and one agent generating an expression for $\beta^{(\varepsilon)}$. The dissipation rate was calculated using Equation 3.26, where $C_\mu = 0.09$ is a model constant.

$$\varepsilon = C_\mu k \omega \tag{3.26}$$

3.11.3. Flow cases

The Flows Over Periodic Hills of Parameterized Geometries dataset by Xiao et al., 2019 is used as the dataset for symbolic regression in this work. It consists of DNS data of turbulent flow over a family of periodic hills with systematically varied geometries. The variation is defined by a single non-dimensional geometric parameter, α, which scales the hill width while keeping the hill height constant. Increasing

α produces a wider hill and a longer streamwise period, thereby changing the size and position of the separation bubble. The streamwise domain length is defined as a function of α according to:

$$L_x/H = 3.858\,\alpha + 5.142. \tag{3.27}$$

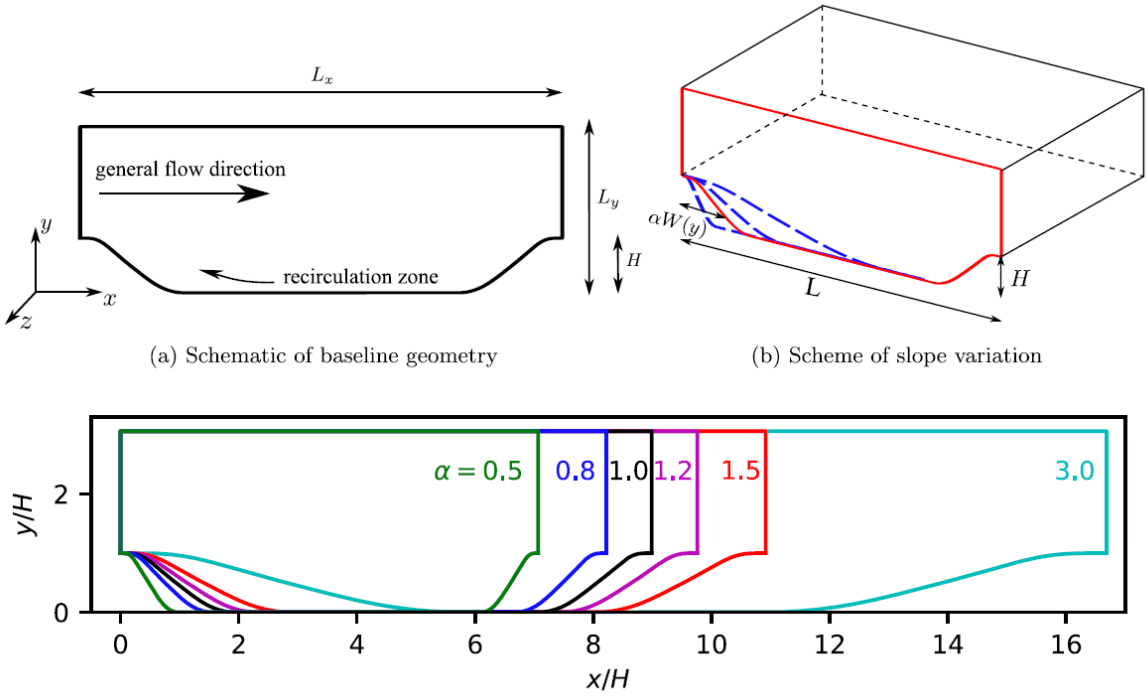


Figure 3.1: Parametrisation of the Periodic Hills geometry with respect to the geometric scaling parameter α. Image by Xiao et al., 2019.

where $\alpha \in \{0.5, 0.8, 1.0, 1.2, 1.5, 3.0\}$. The baseline configuration corresponds to $\alpha = 1.0$, giving $L_x/H = 9.0$. All simulations were performed at a Reynolds number of $Re = 5600$ using the high-order Incompact3d solver, with grid resolutions typically around $768 \times 385 \times 128$ depending on the geometry. The dataset provides consistent mean fields and turbulence statistics across different hill shapes, allowing systematic analysis of attached and separated flow regimes.

An extended database was added by Sylvain Laizet. This addition includes 29 simulations, also at $Re = 5600$, created by systematically varying three geometric and domain parameters: (i) *the hill width*, represented by five α values; (ii) *the length of the upstream and downstream flat sections*, L_f, with three settings; and (iii) *the domain height*, L_y, with three settings. To modify the domain length, Equation 3.27 was updated to:

$$\frac{L_x}{H} = 3.858\,\alpha + C_f, \qquad C_f \in \{2.142,\ 5.142,\ 8.142\}. \tag{3.28}$$

This updated domain-length relation was applied only to three of the α values, specifically $\alpha \in \{0.50,\ 1.00,\ 1.50\}$. The remaining two cases, $\alpha = 0.75$ and $\alpha = 1.25$, retained the baseline configuration with $C_f = 5.142$. Combined with the three domain heights $L_y/H \in \{2.024,\ 3.036,\ 4.048\}$, this results in a total of $3 \times 3 \times 3 + 2 = 29$ simulated flow cases. Since the same symbol α is used for both hill width scaling and the coefficient functions in MADSR, the geometric scaling parameter will be denoted as α_{PH} in the remainder of this work to avoid confusion.

This extended version of the dataset is used in the present work. Only the cases with $\alpha_{PH} = \{1.0, 1.5\}$ are considered, as the $\alpha_{PH} = 0.5$ case proved too challenging for initial model evaluations. This selection yields 18 flow cases in total for training and testing.

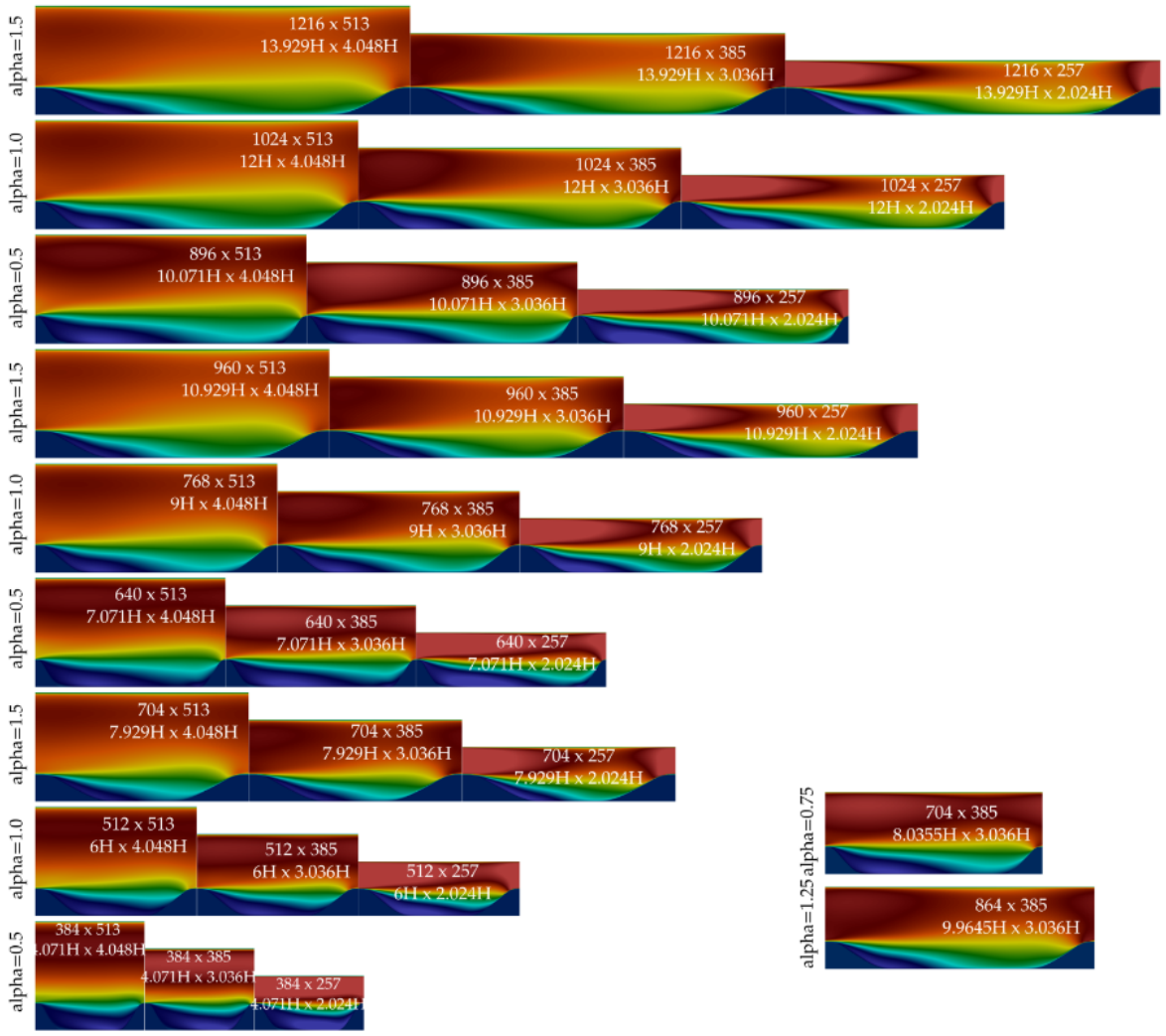


Figure 3.2: Extension of the Periodic Hills database by Laizet (2021), including additional variations in domain length and height. Image by Xiao et al., 2019.

The original study by Xiao et al., 2019 focused solely on developing and evaluating EARSM-based models and did not include the k-corrective RANS framework. In this work, a corresponding k-corrective frozen RANS dataset is generated for all 18 flow cases to model the production deficit term R. This was done in the same manner as in subsection 2.3.2, where an ordinary differential equation in terms of ω is solved, while keeping the TKE k frozen to the DNS values.

3.11.4. Training setup

A subset of this dataset was used for training MADSR. The training set consists of 2 flow cases, specifically those with $\alpha_{PH} = 1.0, L_x/H = 9, L_y/H = 3.036$ and $\alpha_{PH} = 1.5, L_x/H = 10.929, L_y/H = 3.036$. The remaining 16 flow cases were reserved for testing the generalisability of the discovered turbulence models. The 2 training cases were chosen to enable the models to be able to generalise to different geometries (α_{PH} values). The $\alpha_{PH} = 1.0$ case has a much larger separation compared to the $\alpha_{PH} = 1.5$ case, and thus the models have to learn to be able to predict both cases well.

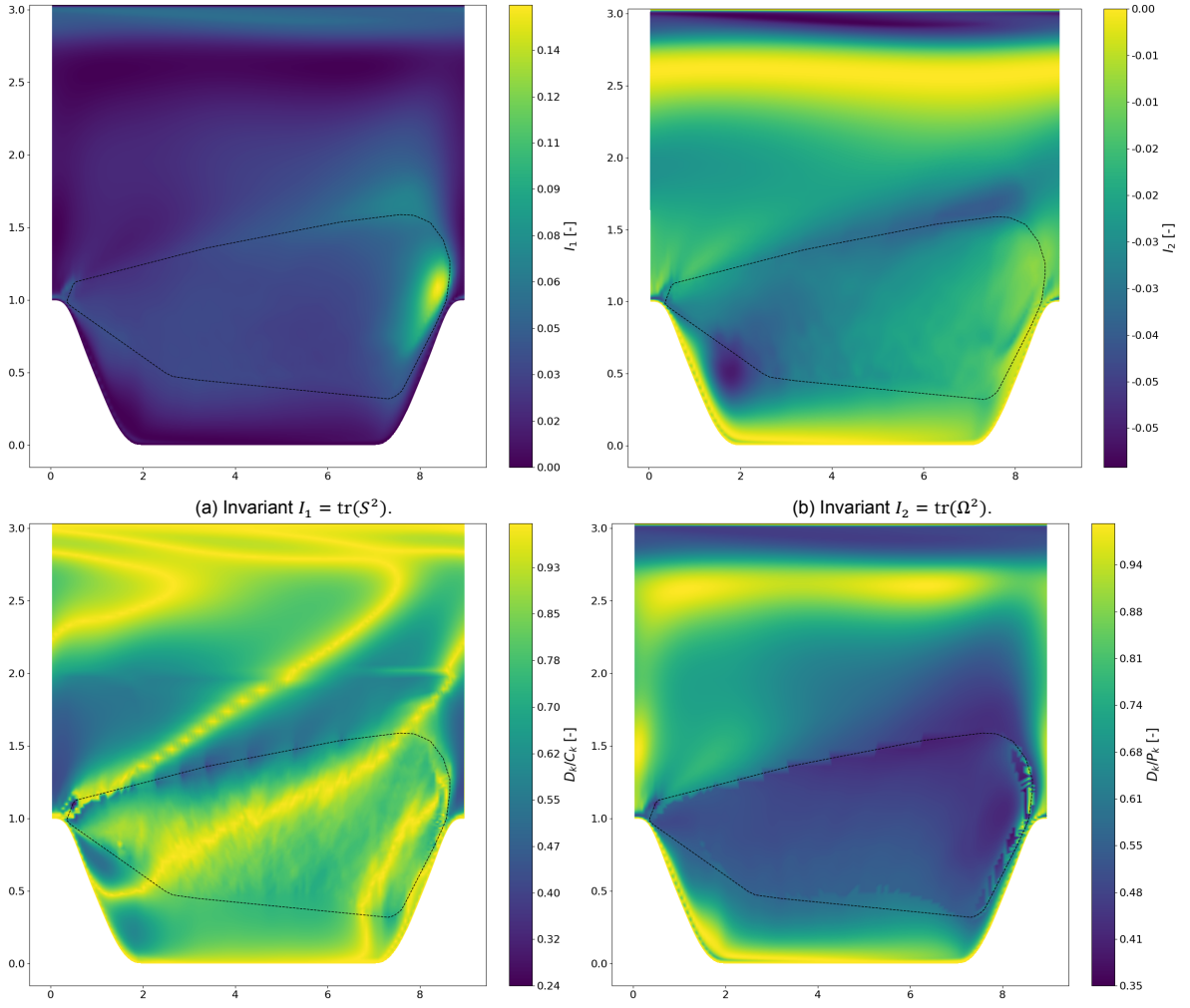
The training dataset combined has a total of around 30,000 data points, which is the whole field of the 2 training cases. This is very large, and thus needs to be reduced for training. Thus, the RITA classifier by Buchanan et al., 2025 was used to select only the most important data points for training. The classifier allows the model to focus on the most important regions of the flow, where the turbulence modelling errors are the largest, since most of the flow is already well modelled by RANS. This reduced

the dataset to around 5000 data points, which is more manageable for training.

A thousand training steps were used for training MADSR for both EARSM and k-corrective frozen RANS frameworks. Each training step consists of generating a batch of 1000 expressions per agent. This results in a total of 1 million reward function evaluations during training. This is quite a large number, but it is necessary to explore the large search space of possible expressions.

Input features

The field for the input features used for training MADSR for the $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$ case is shown in Figure 3.3. These features will be the scalar variables that can be used by the agents to generate expressions for the coefficient functions $\alpha^{(n)}$ or $\beta^{(n)}$. The features that are actually used are within the dotted black line, which is the region selected by the RITA classifier for training.



(a) Invariant $I_1 = \mathrm{tr}(S^2)$.

(b) Invariant $I_2 = \mathrm{tr}(\Omega^2)$.

(c) Ratio $D_k/(|D_k| + |C_k|)$ (denoted D_k/C_k).

(d) Ratio $D_k/(|D_k| + |P_k|)$ (denoted D_k/P_k).

Figure 3.3: Input scalar features used by the MADSR agents for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$: (a) first invariant I_1, (b) second invariant I_2, (c) D_k/C_k ratio, and (d) D_k/P_k ratio.

Basis tensors

For the basis tensors, only the first basis tensor $T_{ij}^{(1)}$ for the $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$ case is shown here as an example. The other basis tensors can be found in Appendix C, and are not shown here for brevity. Only the xx, xy, yy and zz components are visualised. The xz and yz components are not shown as they are all zero due to the 2D nature of the flow. The yx component is also not shown, as all basis tensors, as well as Δb_{ij} itself, are symmetric tensors, meaning that $b_{ij} = b_{ji}$. So the yx component is equal to the xy component, and does not provide any additional information.

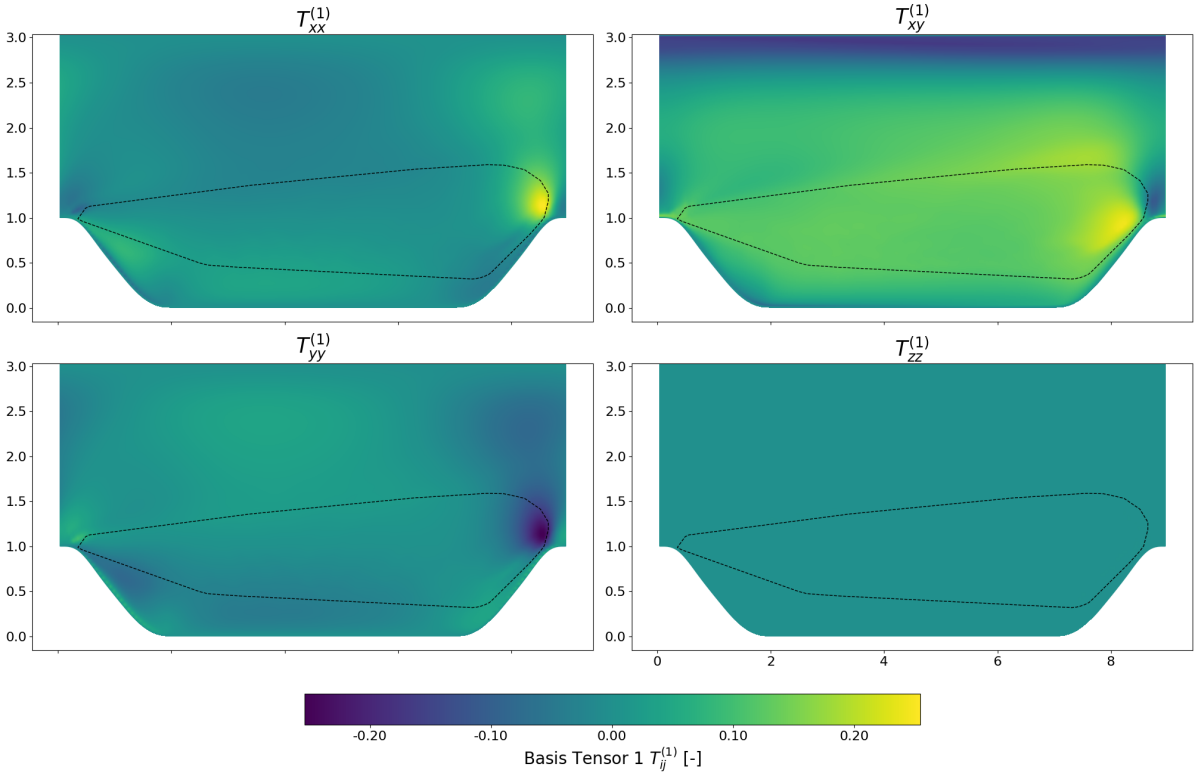


Figure 3.4: First Pope basis tensor for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$. This tensor is the input feature to the discovered models and is used to construct the Reynolds stress anisotropy tensor correction.

The field for the turbulent dissipation rate ε used for training MADSR for the $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$ case is shown in Figure 3.5. This variable is used in the k-corrective frozen RANS framework as an additional term to model the production correction term R.

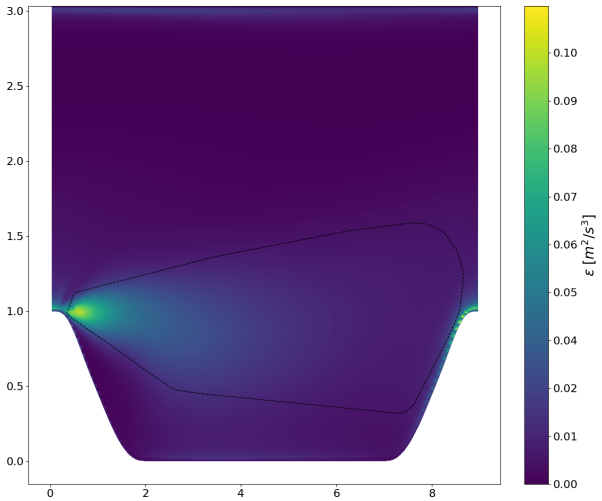
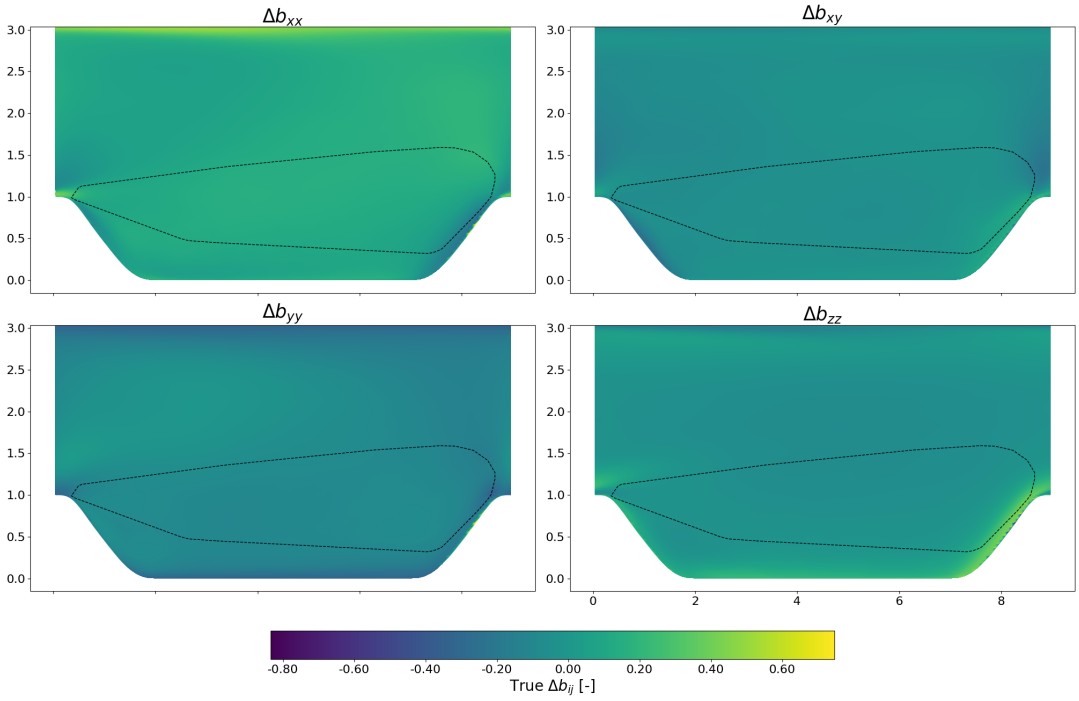


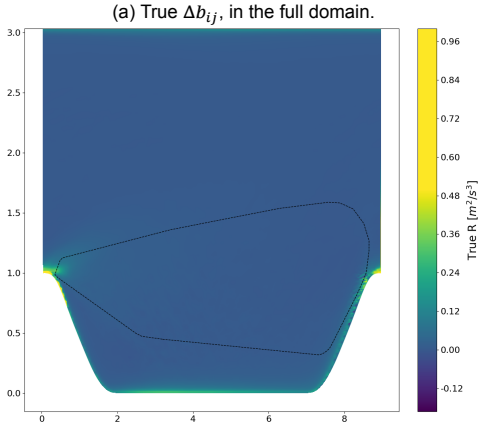
Figure 3.5: Turbulent dissipation rate ε for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$. This is used to model the production correction term.

Target fields

The target fields used for training MADSR for the $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$ case are shown in Figure 3.6. These fields are the ground truth values that the discovered models will try to predict. The Reynolds stress anisotropy correction Δb_{ij} is used for the EARSM framework, and is a tensor field.



(a) True Δb_{ij}, in the full domain.



(b) True production deficit R, in the full domain.

Figure 3.6: Target fields for MADSR, $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$: (a) Reynolds stress anisotropy correction Δb_{ij}, and (b) production correction term R. These fields are used to compute the reward signal during training.

The production correction term R is used for the k-corrective frozen RANS framework, and is a scalar field.

3.12. General framework

To summarise the methodology, the overall workflow of the MADSR framework is illustrated in Figure 3.7. The framework combines symbolic regression with reinforcement learning and the frozen RANS evaluation procedure to enable data-driven turbulence model discovery.

At each training iteration, the process begins with all agents sampling symbolic expressions from their respective policies. Each agent represents a coefficient function $\alpha^{(n)}$ that corresponds to a basis tensor $T_{ij}^{(n)}$ in the EARSM formulation.

The sampled expressions are then evaluated on the frozen RANS dataset. The generated expressions

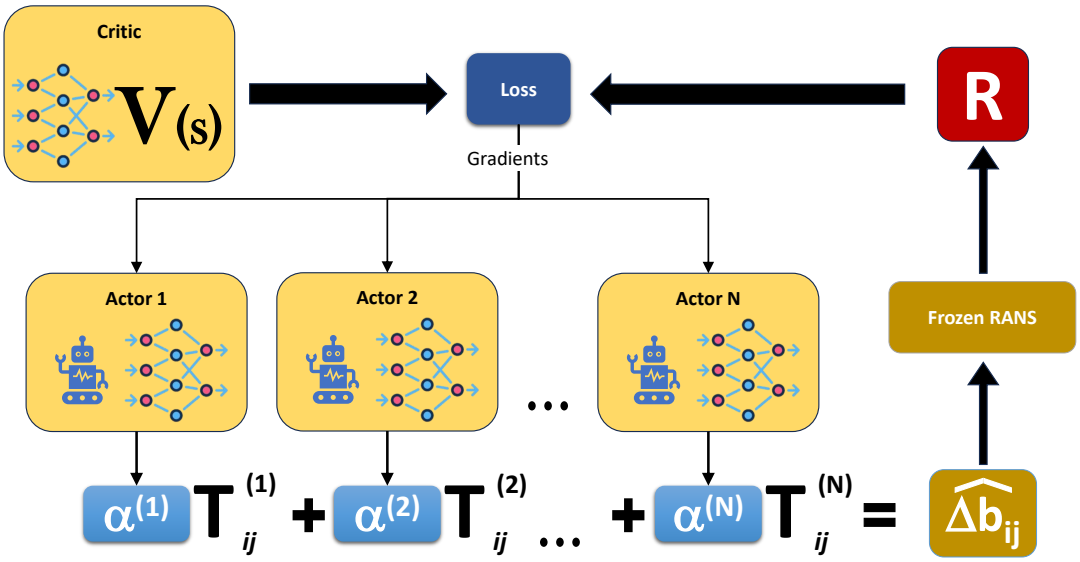


Figure 3.7: Overview of the proposed MADSR framework. Each policy network generates symbolic expressions for the coefficient functions $\alpha^{(n)}(I_1, \ldots, I_K)$, which are multiplied by the corresponding basis tensors $T^{(n)}ij$ to obtain the anisotropy correction Δbij. The resulting stress correction is evaluated using frozen RANS simulations to compute the reward signal. Multi-agent reinforcement learning methods are then used to train the networks cooperatively.

are combined to predict the Reynolds-stress anisotropy tensor (or production correction term in the k-corrective RANS case), which is compared to the reference data from high-fidelity simulations.

A reward is then computed based on the accuracy of the predictions, measured using the R^2 score. The reward serves as the learning signal for all agents. The top-performing expressions, according to the risk-seeking method, are selected and used to update the actor networks. If a critic is used, it is trained in parallel to estimate the value function, which is then used to compute the advantage function for actor updates.

These steps, sampling expressions, evaluating them with frozen RANS, computing rewards, and updating the networks, are repeated iteratively. Until a set number of training iterations is reached.

4

MADSR results and analysis of discovered models

This chapter presents the results obtained from applying the proposed MADSR framework to turbulence model discovery. The performance of the four developed algorithms: vDSR, vPPODSR, MADSR, and PPODSR is first compared across three correction types: the EARSM with three and four-tensor bases, and the scalar R correction. The best-performing algorithm is then analysed in detail, and the discovered models are evaluated for their interpretability, generalisability, and physical consistency. Subsequent sections present further optimisation of the identified models, followed by an ablation study to assess the contribution of individual tensor terms. Finally, flow-field visualisations are provided to examine the spatial accuracy of the predicted corrections and to highlight regions where the models perform well or exhibit deficiencies.

4.1. Algorithm comparison

In this section, the performance of the four different algorithms proposed in section 3.5 is compared on the three different turbulence modelling corrections: EARSM with three tensors, EARSM with four tensors and the R correction. A table showing the best R^2 values achieved by each algorithm for each correction is presented, along with training curves showing the performance of each algorithm over the course of training. These R^2 values correspond to the R^2 of only the training case, which is the parametrised periodic hill case with $\alpha_{PH} = 1.0, L_x/H = 9, L_y/H = 3.036$ and $\alpha_{PH} = 1.5, L_x/H = 10.929, L_y/H = 3.036$.

In the training curves, three metrics are shown to evaluate the performance of each algorithm during training:

- **Best model reward:** This shows the reward of the best model found so far during training. This gives an indication of the algorithm's ability to find expressions, and what the best model it can find is. This is a very important metric, as the best model found is what will be used in practice. However, this metric alone does not give a complete picture of the algorithm's performance, as it could do this just by chance, and not be able to improve further.
- **Average sub batch reward:** This shows the average reward of the top ϵ models sampled in each training step. This indicates how consistent the algorithm is at finding good models, as reproducing good models on average consistently can lead to even better ones.
- **Maximum reward:** This shows the best reward achieved by a model in each training step. This metric is quite similar to the average sub batch reward, as it generally follows the same trend. However, it shows the variance in the models sampled, as a high maximum reward but low average reward indicates that the models sampled are quite varied in performance. Or when the maximum reward and average reward are close together, it indicates that the models sampled have converged to generating similar models.

4.1.1. EARSM with three tensors

Table 4.1 shows the best R^2 values achieved for each algorithm for the Δb_{ij} correction with three tensors (agents). Predicting Δb_{ij} proved to be a difficult task for all algorithms. All algorithms struggled to achieve a high R^2 value, with most algorithms achieving an R^2 value of around 0.4. Only with PPODSR was it possible to achieve an R^2 value above 0.5. Even then, the best model found only achieved an R^2 value of 0.518, which is still quite low.

Table 4.1: Best R^2 values achieved by each algorithm for EARSM with three tensors.

	vDSR	vPPODSR	ACDSR	PPODSR
Reward	0.402	0.448	0.405	0.518

This low R^2 value indicates that the models are not able to capture the behaviour of the Reynolds stress anisotropy tensor very well. This could be due to the complexity of the problem, as the Reynolds stress anisotropy tensor is a tensor quantity. The models discovered need to predict the four independent components of the Reynolds stress anisotropy tensor simultaneously, which is a challenging task. The complexity of the problem could also cause the algorithms to struggle to find good models, as the search space is likely very large and complex.

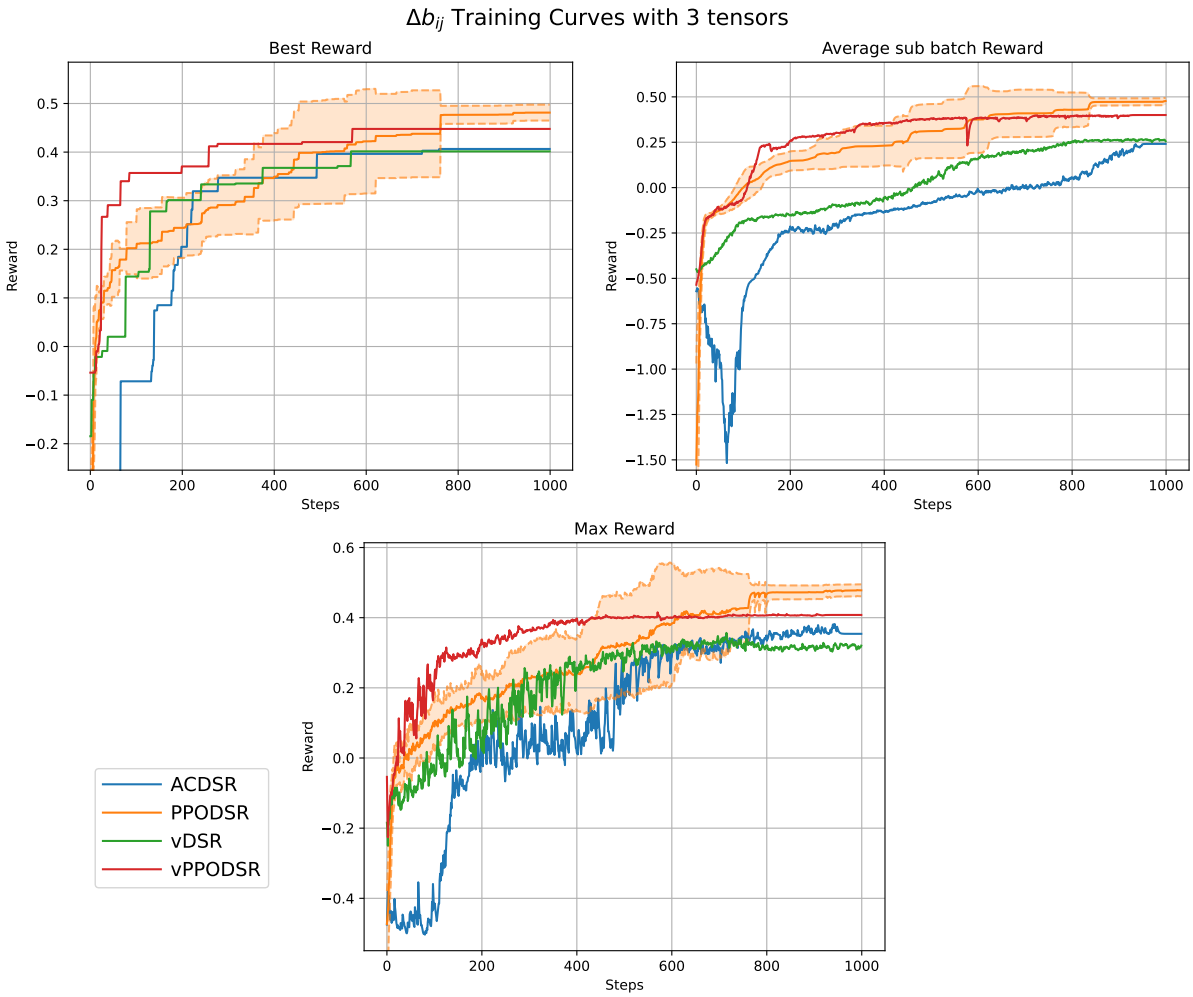


Figure 4.1: Comparison of training curves for different algorithms on EARSM with three tensors. Shaded areas represent the standard deviation across ten runs with different random seeds.

Figure 4.1 shows the training curves for the correction with three tensors. Both algorithms with PPO

perform much better than the other algorithms without. Since PPO is able to use training epochs, it can learn from good performing models more effectively. So whenever a good model is sampled, PPO can learn more from this sample, compared to the other algorithms that only learn from each sample once.

PPODSR outperformed vPPODSR here. vPPODSR learned very quickly in the beginning, exceeding PPODSR in the first half of the 1000 training steps. However, after this, it struggles to improve further. It reaches an average sub-batch R^2 value of around 0.4 at around 400 training steps, and then stagnates there for the rest of the training. This means that vPPODSR has converged to a local minimum and is not exploring the search space further. PPODSR, on the other hand, continued searching the space. It continued to learn from good models it finds, allowing it to improve further. Eventually, most runs in PPODSR discovered better models than any run with vPPODSR.

The three-tensor case proved to be quite stochastic, with large variations in performance between different runs, shown by the large error bars in Figure 4.1. Some runs were able to find good models quickly and learn very quickly from them, while other runs really struggled to find good models. This indicates that the search space is quite complex, and very specific models are needed to achieve good performance. If the algorithm is not able to find these models, it struggles to improve further. So EARSM with three tensors is very seed-dependent and requires good exploration. But with PPODSR, all runs were eventually able to find good models within a thousand training steps, shown by the smaller error bars towards the end of training, indicating that the runs are converging to similar performance. This shows that PPODSR is robust and is able to learn from any good models it finds, allowing it to eventually find good models in all runs.

One interesting thing to note is that there is a sharp dip in performance before ACDSR starts learning. This is a known phenomenon in actor-critic algorithms called critic bias. During initialisation, the critic outputs values close to zero for all states. Since the rewards are negative, this leads to a large negative advantage, which suggests that all actions taken were bad. This leads to the actor updating its policy to avoid all actions taken, which leads to a sharp drop in performance. As the critic learns to predict the rewards more accurately, this effect diminishes, and the actor is able to learn better policies.

While PPODSR also uses advantages, the sharp drop is not seen there. Since it uses multiple training epochs, the critic can learn much more quickly, and thus the actor does not suffer from this effect as much. But critic bias is still present in PPODSR, since the initial performance of PPODSR compared to vPPODSR is lower, where vPPODSR shoots up very quickly in the beginning, while PPODSR improves more slowly. This period of slow growth is likely due to critic bias, so while there is no sharp drop, PPODSR still suffers from critic bias to some extent.

4.1.2. EARSM with four tensors

Using four tensors adds an additional degree of freedom to the model, which should allow it to capture the behaviour of the Reynolds stress anisotropy tensor better. But this also increases the search space significantly, making it more difficult for the algorithms to find good models. However, PPODSR was still able to find a good model with an R^2 value of 0.551, shown in Table 4.2. This is an improvement over the best model found with three tensors, indicating that the additional tensor does help to improve the performance of the models.

Table 4.2: Best R^2 value obtained for each algorithm for the correction with four tensors.

	vDSR	vPPODSR	ACDSR	PPODSR
Reward	0.384	0.426	0.376	0.55

But the additional degree of freedom needs to be properly exploited by the algorithms, as the other algorithms performed worse in the four-tensor case compared to the three-tensor case. This indicates that the additional complexity of the search space makes it more difficult for the algorithms to find good models, and only PPODSR was able to effectively explore the search space and find good models.

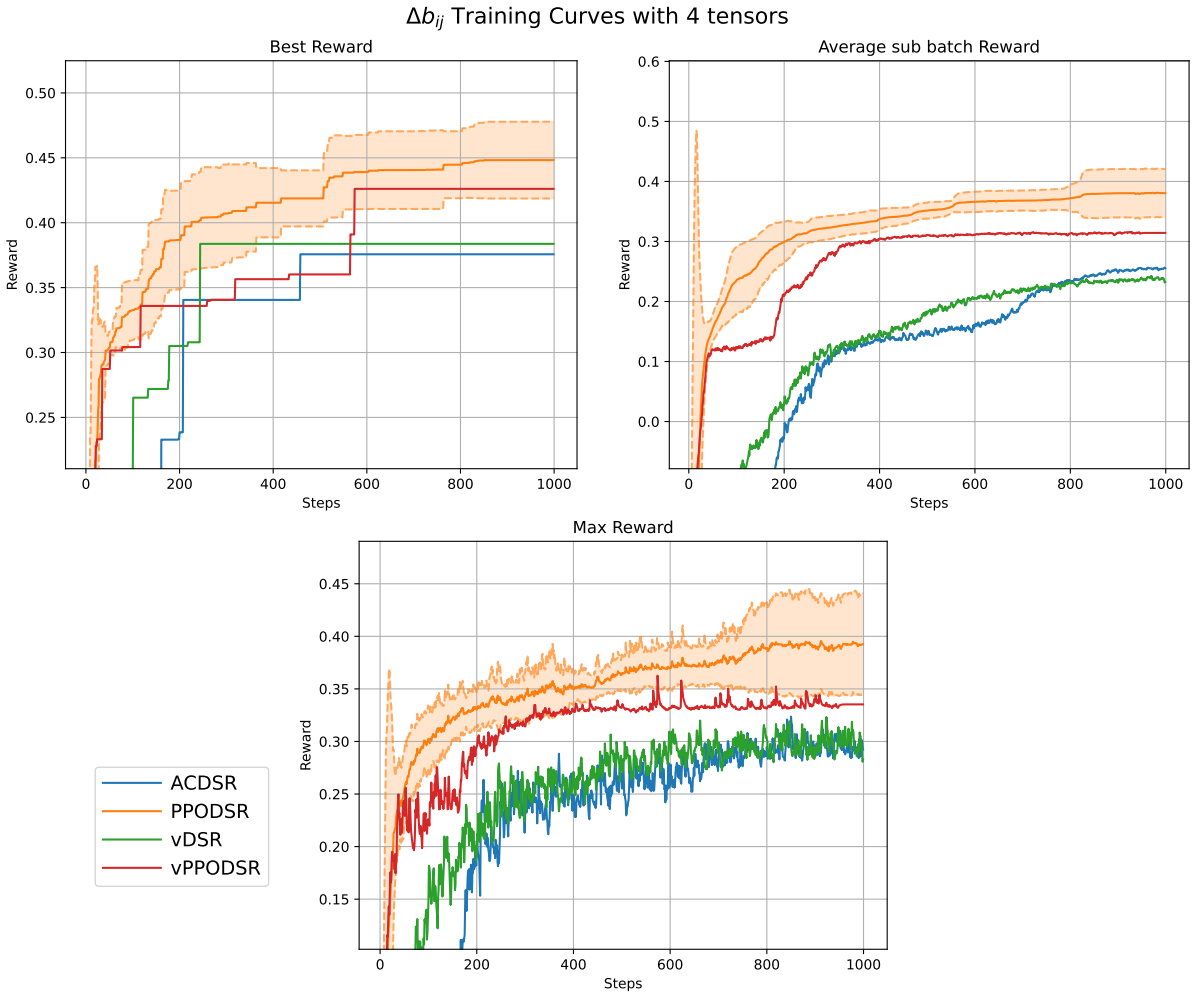


Figure 4.2: Comparison of training curves for different algorithms on EARSM with four tensors. Shaded areas represent the standard deviation across ten runs with different random seeds.

The algorithms in the four-tensor case show similar trends to the three-tensor case, as shown in Figure 4.2. The PPO-based algorithms outperform the other algorithms significantly, with PPODSR being the best performing algorithm. vPPODSR again shows quick initial improvement, but then stagnates after around 300 training steps. PPODSR, on the other hand, is able to continue improving throughout the training process, eventually finding better models than vPPODSR.

The greater degree of freedom in the four-tensor case allowed the algorithms to find well performing models more easily. This is shown by the smaller variance of PPODSR at the start. The four-tensor case is more methodical, where the improvements are steadier and consistent, compared to the three-tensor case, where the improvements were more sporadic and depended heavily on finding specific models. But towards the end of training, the variance increases again, as the algorithms start to explore more and find different models, due to the larger search space. This shows that the four-tensor case is overall more stable and easier to learn, but still has a complex search space that requires good exploration.

4.1.3. k-corrective RANS

The k-corrective RANS provides a correction to the production deficit term R, which is a scalar quantity, as explained in subsection 3.11.2, and Figure 3.6b shows a visualisation of the target field. In further sections, this correction will be referred to as the R correction for brevity. In the case of the R correction, all algorithms were able to achieve much higher R^2 values compared to the Reynolds stress anisotropy tensor corrections. This is likely because the R correction is a scalar quantity, making it easier to predict

compared to the tensor quantities. The best model found achieved an R^2 value of 0.822, shown in Table 4.3, which is significantly higher than the best models found for the other two corrections.

Table 4.3: Best R^2 value obtained for each algorithm for the R correction.

	vDSR	vPPODSR	ACDSR	PPODSR
Reward	0.808	0.806	0.816	0.822

In Figure 4.3, the R^2 each algorithm achieves at the start (a random model) is approximately 0.65, much higher than the starting R^2 values for the other two corrections, which had negative starting R^2 values. This is once again because predicting a scalar quantity is easier, and even a random model is able to achieve a reasonable R^2 value.

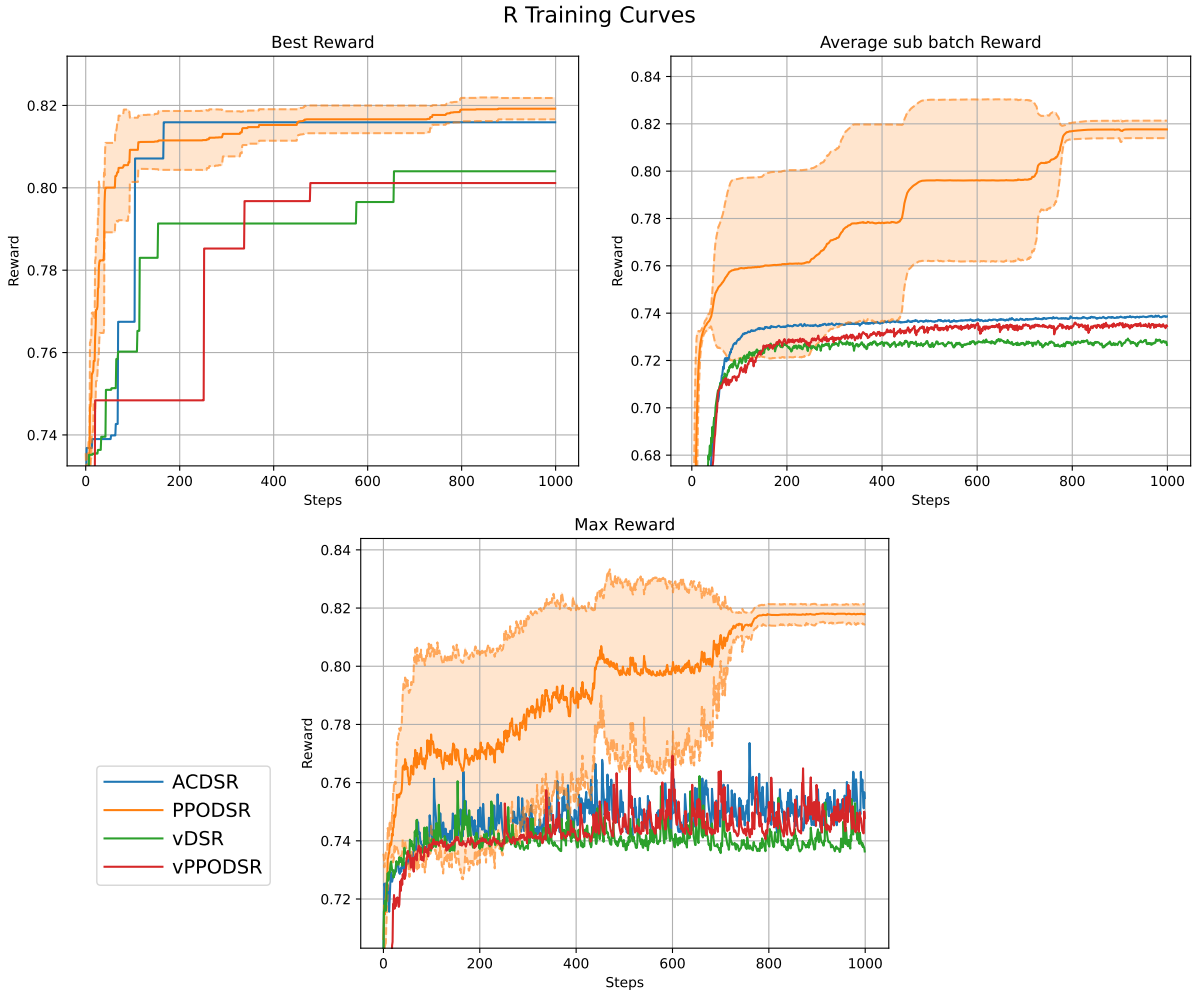


Figure 4.3: Comparison of training curves for different algorithms on the R correction. Shaded areas represent the standard deviation across ten runs with different random seeds.

But all algorithms other than PPODSR struggle to improve on the R corrections. This is mainly seen in the maximum reward and the average sub batch reward curves. For the other algorithms, it stagnates at an R^2 of approximately 0.73, at around 200 steps, and does not improve further. While the scalar form of the R correction makes it easier to predict, it also means that more information is lost when calculating the rewards. Differences in models that would lead to significant differences in the tensor corrections may only lead to small differences in the scalar R correction. This makes it more difficult for the algorithms to distinguish between good and bad models.

PPODSR, on the other hand, does manage to escape this local minimum and continues to improve, reaching an average sub-batch R^2 of approximately 0.80. Since the average of the top-ϵ models is performing well in PPODSR, the algorithm is able to explore in that area of the search space more thoroughly. As shown in the best model curve, the best model found by PPODSR constantly improves, getting better and better models as training progresses, while the other algorithms do not manage to find better models after the initial stagnation.

Other algorithms do manage to find at least one good model eventually, and only ever find a few models that perform better than 0.8. This is likely due to probabilistic luck, as expression generation is stochastic. The difference between PPODSR and the other algorithms is that when PPODSR gets lucky and finds a good model, it is able to exploit that and learn from it. PPODSR learnt the structure of the expressions generated by the agents, and identified which parts of the expressions are beneficial and which are not, allowing it to further improve on the models it finds.

This is where CTDE proves to be very useful, as it allows the critic to learn the structure of why certain expressions perform well. Using the value estimates from the critic, the actor can learn which parts of the expressions are beneficial and which are not. This allows the actor to learn what structures of expressions are good and to exploit that knowledge to generate better expressions. This is especially useful in the R correction case, where the reward signal is weak, and it is difficult to distinguish between good and bad models.

The R correction does not suffer from the critic bias problem seen in the other two corrections. This is likely because the starting R^2 value is much higher, meaning that most of the initial models sampled do not lead to large negative advantages. Therefore, the actor manages to learn good policies from the start and does not suffer from the sharp drop in performance seen in the other two corrections.

4.1.4. Summary of algorithm comparison

EARSM proved to be a difficult problem to model, with low R^2 values achieved for both the three-tensor and four-tensor corrections. The difficulty of the EARSM problem can be seen from the inputs and target in section 3.11. The tensor nature of the target makes it a complex problem to model, as the model needs to capture the behaviour of multiple components of the tensor simultaneously. Each component of the target also has very different distributions and magnitudes, and the same is true for the basis tensors. The complex problem thus makes most models have a low R^2 value on average, making it difficult for the algorithms to learn from good models.

PPODSR was found to be the best performing algorithm across all three corrections models. It outperforms specifically on the R correction, where it is able to escape local minima that the other algorithms get stuck in. Inclusion of PPO improves the performance of the algorithms significantly, as both PPO-based algorithms outperformed their non-PPO counterparts in all three corrections. While CTDE did not lead to significant improvements in performance on its own, when combined with PPO, it did lead to significant improvements, as seen by the performance of PPODSR compared to vPPODSR. A combination of both PPO and CTDE is thus the most effective approach for turbulence model discovery using MADSR. Therefore, PPODSR is used for all further experiments, and all references to MADSR refer to PPODSR. And all models found in further sections are trained using PPODSR.

4.2. Analysis of found models

In this section, the best models found by PPODSR for each of the three corrections are analysed in detail. The models are evaluated based on their interpretability, generalisability, complexity, and accuracy. All discovered models can be found in Appendix A. An analysis of the best models found during training is first presented, followed by an analysis of the most generalisable models found.

4.2.1. Best models in training

Equation 4.1, Equation 4.2, and Equation 4.3 show the best performing models found for the correction with three tensors, four tensors and the R correction, respectively. These were the best models achieved in training, meaning that they achieved the highest R^2 value only on the training case.

$$\Delta b_{ij} = \left[e^{\frac{0.069 I_2}{I_1^2}} - 0.562 \right] T_{ij}^{(1)} + e^{\frac{D_k/P_k}{I_1 + 0.339}} T_{ij}^{(2)} + \left[\frac{1.357}{I_1} + \frac{1 - e^{I_2}}{I_1 I_2} \right] T_{ij}^{(3)} \tag{4.1}$$

The best model found for the correction with three tensors is shown in Equation 4.1. This model achieved an R^2 value of 0.518 on the training case. It can be seen that all terms in the expression contain an exponential function, indicating that the model relies heavily on exponentials to capture the behaviour of the Reynolds stress anisotropy tensor. It also uses mostly the 2 invariants I_1 and I_2, with only one term using D_k/P_k.

$$\begin{aligned} \Delta b_{ij} = &\left[\tanh\left(\frac{1.85 I_2}{I_1} \right) + 0.545 \right] T_{ij}^{(1)} + \frac{0.411}{-I_2 + I_1 + 0.045} T_{ij}^{(2)} \\ &+ \frac{0.146 - 0.702 I_1}{I_2} T_{ij}^{(3)} + \frac{0.5 - 3.05 {D_k/P_k}^4}{I_2} T_{ij}^{(4)} \end{aligned} \tag{4.2}$$

This model achieved an R^2 value of 0.551 on the training case. The second best model found for the four-tensor correction achieved an R^2 value of only 0.50, showing a significant gap between the best model and the rest. This indicates that this model is unique in its structure, allowing it to capture the behaviour of the Reynolds stress anisotropy tensor better than any other model found.

The model is also relatively simple and quite interpretable. All other models found are generally very complex, with many nested functions and operations. This model, on the other hand, is relatively straightforward, using modelling terms with simple expressions such as division, addition and hyperbolic tangent.

The model does not use any complex functions such as exponentials or logarithms, and only the hyperbolic tangent function is used in the first tensor. These functions are readily available for the algorithm to use, so the fact that the model does not use them indicates that they are not necessary to capture the behaviour of the Reynolds stress anisotropy tensor in this case. The use of constants in the model is also lower than the allowed number, with every function only using two constants, while up to three were allowed.

$$\begin{aligned} b_{ij}^R &= 92.4 \log\left(-0.982 + \frac{0.0003474}{I_2} \right) T_{ij}^{(1)} + D_k/P_k \, T_{ij}^{(2)} + \log\left(-0.00238 D_k/P_k \right) T_{ij}^{(3)} \\ R &= 2k b_{ij}^R \frac{\partial u_i}{\partial x_j} + \left[\frac{9.75 \left(0.00418 - D_k/C_k \right)}{\left(0.00418 - D_k/C_k \right)^2 + 1.0} + 5.4 \right] \varepsilon \end{aligned} \tag{4.3}$$

The best model found for the R correction is shown in Equation 4.3. This model achieved an R^2 value of 0.822 on the training case. It can be seen that the model relies heavily on logarithmic functions to capture the behaviour of the Reynolds stress anisotropy tensor. But there are a lot of spurious constants in the model, where many constants are close to zero, indicating that they and their corresponding terms do not contribute much to the model. This indicates that the model is likely overfitting to the training data, and may not generalise well to other cases.

4.2.2. Best performing models

In section 4.1, the model with the highest R^2 value during training is presented. This means that the model performed best on the training set, which is the case $\alpha_{PH} = 1.0, L_x/H = 9, L_y/H = 3.036$ and $\alpha_{PH} = 1.5, L_x/H = 10.929, L_y/H = 3.036$. However, this does not necessarily mean that this model will perform best on the other test cases. Therefore, the models need to be evaluated on all test cases to determine which model generalises best. The definition of best performing model is also ambiguous, as a model might perform best on one case but poorly on another. So, different definitions of best performing model are considered here:

- **Training:** The model that performs best on the training case
- α_{PH}**=1.0:** The model that performs best on all α_{PH} = 1.0 cases. This is done by concatenating the data of all α_{PH} = 1.0 cases and evaluating the model on that.

- α_{PH}**=1.5:**The model that performs best on all α_{PH} = 1.5 cases. Similarly, all data from α_{PH} = 1.5 cases is concatenated, and the model is evaluated on that.
- **All:** The model that performs best on all cases. All data from both α_{PH} = 1.0 and α_{PH} = 1.5 cases is concatenated, and the model is evaluated on that.
- **Mean:** The model with the highest mean R^2 across all cases. The data is not concatenated here; instead, the model is evaluated on each case individually, and the mean performance is taken.

The performance metric used here is the R^2 value, which is the same metric used during training. The performance of the best models will be shown in a table like Table 4.4.

Three-tensor Δb_{ij} correction
The performance of the best models, and which category they were best at, is shown in the table Table 4.4. The performance of each model in all metrics is also shown.

Table 4.4: Best performing models for the correction with three tensors.

Model	Best in	Training	α_{PH}=1.0	α_{PH}=1.5	All	Mean
M_1	Training, α_{PH}=1.5, all, mean	0.518	0.282	0.443	0.368	0.347
M_2	α_{PH}=1.0	0.498	0.285	0.424	0.359	0.344

From Table 4.4 it can be seen that the model that performed best on the training case, also performed best on most other metrics. This indicates that the model generalises well to other cases and is not overfitting to the training data. The only exception is that the model does not perform best on the α_{PH} = 1.0 cases, where a different model M_2 performs slightly better. However, the difference in performance between M_1 and M_2 in the α_{PH} = 1.0 cases is only 0.003, which is very small compared to the overall performance difference between M_1 and M_2.

Four-tensor Δb_{ij} correction
A similar trend can be seen here as well in Table 4.5. The model that performed best on the training case also performed best on most other metrics. The only exception is the model that performed best on the α_{PH} = 1.0 cases, which is a different model M_2. Unlike the previous case, M_2 had a significantly better performance in the α_{PH} = 1.0 case, with a difference of 0.012 compared to M_1. But M_2 performed very poorly in all other cases, indicating that it has overfit heavily to the α_{PH} = 1.0 cases. Thus, M_1 is still the most generalisable model found here, as it performs well across all cases without overfitting to any specific case.

Table 4.5: Best performing models for the correction with four tensors.

Model	Best in	training	α_{PH}=1.0	α_{PH}=1.5	All	Mean
M_1	Training, α_{PH}=1.5, all, mean	0.551	0.358	0.459	0.418	0.389
M_2	α_{PH}=1.0	0.37	0.37	0.294	0.37	0.324

R correction
In the R correction, four models were identified that each had their own strengths. This is quite different from the Reynolds stress corrections, where one model was able to perform well across most cases. Here, there is more variety with different models excelling in different areas. Model M_1 that did best in the training case performed poorly in most other cases, except the case where α_{PH} = 1.5. This suggests that the model overfitted to α_{PH} = 1.5, and is not able to generalise well to other cases. This shows that the training set is quite biased towards α_{PH} = 1.5 in the R correction, and the model has learned to perform better in that case, at the expense of performance in other cases.

Table 4.6: Best performing models for the R correction.

Model	Best in	Training	α_{PH}=1.0	α_{PH}=1.5	All	Mean
M_1	Training	0.822	0.376	0.851	0.481	0.769
M_2	α_{PH}=1.0	0.816	0.394	0.828	0.49	0.76
M_3	α_{PH}=1.5	0.814	0.376	0.858	0.483	0.764
M_4	All, mean	0.82	0.39	0.854	0.493	0.78

The models M_2 and M_3 overfit to α_{PH} = 1.0 and 1.5, respectively, achieving the best performance in those cases. The overfitting is more obvious when the performance of the models in their non-favoured α_{PH} is observed. M_2 performs poorly in α_{PH} = 1.5, achieving an R^2 value of only 0.828, while M_3 performs poorly in α_{PH} = 1.0, achieving an R^2 value of only 0.376. But an interesting thing to note about M_2 is that the performance in the full dataset is quite good, achieving an R^2 value of 0.49, which is comparable to the best model M_4. This indicates that overfitting to α_{PH} = 1.0 is more beneficial for generalisability to the full dataset, compared to overfitting to α_{PH} = 1.5.

M_4 achieved the best performance in the full dataset, as well as the best mean performance across all cases. M_4 was quite different from the other models, as it did not overfit to any specific case. It achieved an R^2 value of around 0.82 in the training case, and also good performance in both α_{PH} = 1.0 and 1.5 cases, having only a 0.004 difference from the best models in those cases. This indicates that M_4 is the most generalisable model found, as it is able to perform well across all cases without overfitting to any specific case.

4.3. Further optimised models

To achieve more generalisability, further optimisation was performed on all models found during training. These models have many coefficients in them, and they can be further optimised to find better coefficients that generalise better across all cases. The equations of all models were taken, and the coefficients were further optimised using the full dataset, instead of just the training case. The optimisation was done using the same optimisation method as used during training, which is the BFGS algorithm from scipy. The optimisation only changes the coefficients in the models, while keeping the structure of the models the same.

Three-tensor Δb_{ij} correction

The performance of M_1 across all metrics is shown in the Table 4.7. After optimisation, it was found that M_1 outperformed all other models across all metrics. The R^2 value increases by quite a lot in the full cases, improving from 0.368 to 0.393. The biggest improvement was seen in the α_{PH} = 1.0 cases, where the R^2 value improved from 0.282 to 0.352, a significant improvement. But performance in the α_{PH} = 1.5 cases decreased quite significantly, from 0.443 to 0.397. This indicates that the optimisation has focused more on improving performance in the α_{PH} = 1.0 cases, at the expense of performance in the α_{PH} = 1.5 cases. The final optimised model is shown in Equation 4.4, where this model is the final model used for evaluation.

Table 4.7: Best performing optimised model for the correction with three tensors.

Model	Training	α_{PH}=1.0	α_{PH}=1.5	all	Mean
M_1	0.497	0.352	0.397	0.393	0.368

$$\Delta b_{ij} = \left[e^{\frac{0.0464 I_2}{I_1^2}} - 0.63 \right] T_{ij}^{(1)} + e^{\frac{D_k/P_k}{I_1 + 0.346}} T_{ij}^{(2)} + \left[\frac{1.304}{I_1} + \frac{1.0 - e^{I_2}}{I_1 I_2} \right] T_{ij}^{(3)} \tag{4.4}$$

Four-tensor Δb_{ij} correction
The performance of M_1 across all metrics is shown in the Table 4.8. Similarly to the three-tensor case, it was found that M_1 outperformed all other models across all metrics after optimisation. The biggest improvement was seen in the α_{PH} = 1.0 cases, where the R^2 value improved from 0.358 to 0.413, a significant improvement. This improvement in performance on the α_{PH} = 1.0 cases also led to a large improvement in the mean performance, which improved from 0.389 to 0.423. Unlike the three-tensor case, performance in the α_{PH} = 1.5 cases also improved, increasing from 0.459 to 0.463.

Table 4.8: Best performing optimised model for the correction with four tensors.

Model	Training	α_{PH}=1.0	α_{PH}=1.5	all	Mean
M_1	0.525	0.413	0.463	0.449	0.423

While the R^2 value of the training case decreased, it is more important for the model to generalise well across all cases, rather than just performing well in the training case. Therefore, optimising the coefficients of the model was beneficial in improving its overall performance. The optimised model is shown in Equation 4.5, and this is the final model used for evaluation.

$$\Delta b_{ij} = \left[\tanh\left(\frac{1.467 I_2}{I_1}\right) + 0.52\right] T_{ij}^{(1)} + \frac{0.33}{-I_2 + I_1 + 0.0233} T_{ij}^{(2)} + \frac{0.121 - 0.484 I_1}{I_2} T_{ij}^{(3)} + \frac{0.405 - 2.16\, D_k/{P_k}^4}{I_2} T_{ij}^{(4)} \tag{4.5}$$

R correction
Unlike the Δb_{ij} corrections, optimising the coefficients of the R correction models did not lead to significant improvements in performance. As shown in Table 4.9, the performance of all models only increased slightly with the optimisation, and the relative performance between the models remained the same, where every model still retained its position as the best model for its respective metric. Thus, optimisation of the coefficients did not lead to significant improvements in performance for the R correction models. Thus, further analysis needs to be done on the models found during training.

Table 4.9: Best performing optimised models for the R correction.

Model	Best in	Training	α_{PH}=1.0	α_{PH}=1.5	All	Mean
M_1	Training	0.814	0.385	0.861	0.49	0.775
M_2	α_{PH}=1.0	0.8	0.396	0.849	0.496	0.748
M_3	α_{PH}=1.5	0.802	0.383	0.868	0.49	0.77
M_4	All, Mean	0.811	0.394	0.855	0.496	0.776

4.4. Ablation study

To further understand the impact and contribution of each tensor term in the discovered models, an ablation study was performed on the best-performing model obtained during training. In this study, each coefficient function $\alpha^{(n)}$ was systematically set to zero, thereby removing the corresponding tensor basis term $T_{ij}^{(n)}$ from the model.

To assess the influence of individual and combined terms, all possible combinations of tensor removals were evaluated. This includes removing one tensor at a time, pairs of tensors, triplets, and so on, up to removing all but one tensor. For each ablation configuration, the coefficients were re-optimised using the BFGS algorithm on the full dataset to ensure that the remaining terms were adjusted optimally in

the absence of the removed tensors. The modified model was then re-evaluated on the dataset, and its performance was compared to that of the original full model. This procedure was applied to all corrections considered in this work, including the three-tensor, four-tensor, and R-correction models.

4.4.1. Δb_{ij} correction

For the Δb_{ij} correction with three and four tensors, no meaningful patterns were observed in the ablation results. The removal of any individual tensor term led to a significant drop in model performance, indicating that each tensor contributed substantially to the overall predictive capability of the model, and is very co-dependent on each other. Even with the re-optimisation of coefficients, none of the ablated models were able to approach the performance of the full model. This suggests that the discovered models rely on a complex interplay between all tensor terms to accurately capture the underlying physics of the Reynolds stress anisotropy tensor.

4.4.2. R correction

The ablation study for the R correction model revealed more nuanced insights into the contributions of individual tensor terms. As shown in Figure 4.4, the removal of certain tensor terms had a more pronounced effect on model performance than others. Specifically, the removal of the epsilon term led to the most significant decrease in the R^2 value. Any ablation that does not include epsilon sees a huge drop in performance. This indicates that the epsilon term plays a crucial role in capturing the Production deficit in the flow and is essential for accurate predictions.

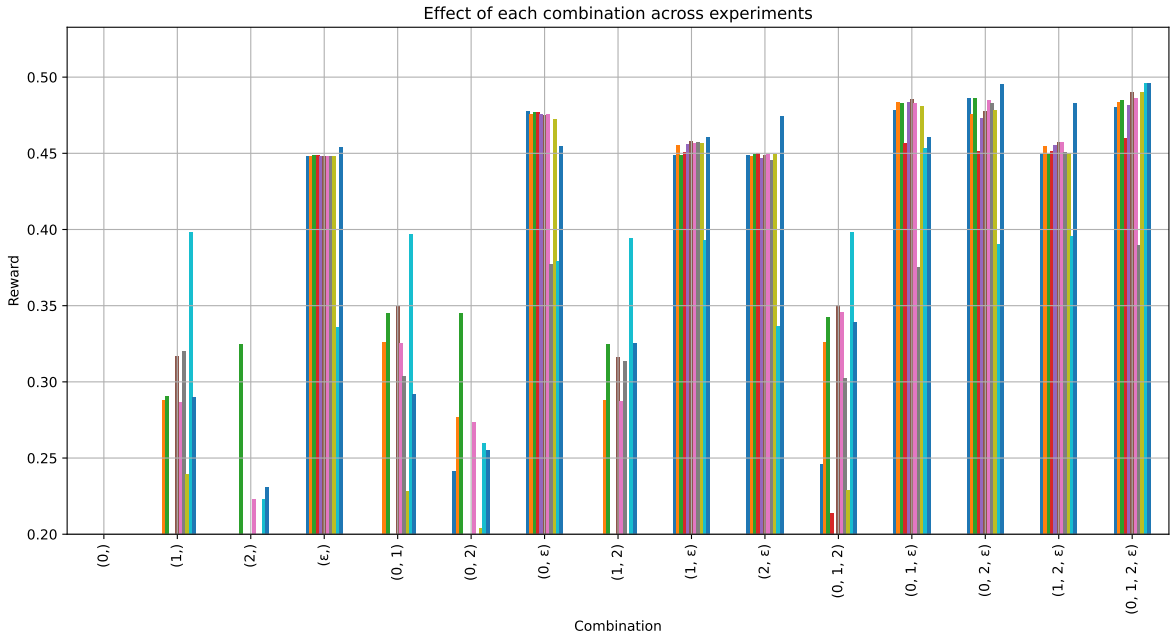


Figure 4.4: R^2 values for different ablation combinations of the R correction model. Each bar represents a different combination of input features, with the full model on the far right. Each colour represents a different model from the best performing models.

The removal of the first tensor term also resulted in a noticeable decline in performance, albeit less severe than that observed with the epsilon term. It can be seen that for the ablations with only 2 terms, the ones that include the first tensor term performed significantly better than those that do not. This suggests that the first tensor term also contributes meaningfully to the model's predictive capability.

The second and third tensor terms appeared to have a very minor impact on model performance when removed individually. Ablations excluding these terms did not exhibit substantial drops in R^2 value, as long as the epsilon and first tensor terms were retained. In certain models, some ablations of terms two or three, after coefficient optimisation, even led to improvements in performance compared to the

full model. This indicates that while these terms may provide some additional information, they mainly act as components that refine the model, rather than being critical for its overall accuracy.

The redundancy of the 2nd and 3rd tensor terms is further highlighted by an observation that was made during training. During training, it was observed that all good models found tended to have very similar structures for the first tensor term and the epsilon term, while the 2nd and 3rd tensor terms varied significantly between different good models. This is reflected in the best models found during training as well, which can be found in Appendix A, where the 2nd and 3rd tensor terms have very different structures in the best models, while the first tensor term and epsilon term are quite similar.

For the first tensor term, the equations generally have some form of $\log(I_2)$. This occurred in many runs, but in very different forms, such as $\log(I_2 + c_1)/I_2$, $(\log(I_2 + c_2))^2$, or $\log(c_3 + \log\frac{c_4}{I_2})$. This indicates that the model relies on some logarithmic relationship with I_2 to capture the behaviour of the R term.

For the epsilon term, the equations generally have some form of $c_1 \cdot R\_div(D_k/C_k) + c_2$, where R_div is the regularised division operation defined in subsection 3.6.1. This indicates that R_div and D_k/C_k are important in capturing the behaviour of the R term. In Buchanan et al., 2025, one of the best models found for the R correction using sparse regression was $c_1 \cdot R\_div((D_k/C_k)/c_2)$, which is very similar to the forms found here. Hoefnagel, 2023 also found that a good model for the R correction can be just a constant multiplied by epsilon, without any other terms. Thus, a combination of these two observations have been found by MADSR, resulting in the final model for the R correction is shown in Equation 4.6.

$$R = 2k\left[-2.66 + \frac{\log(I_2 + 0.972)}{I_2}\right]T_{ij}^{(1)}\frac{\partial u_i}{\partial x_j} + \left[-10.1 \cdot \frac{D_k/C_k}{{D_k/C_k}^2 + 1} + 5.3\right]\varepsilon \tag{4.6}$$

Compared to the coefficients found in Buchanan et al., 2025, the coefficients found here are a magnitude higher. This is likely due to the different way ϵ is defined here. In Buchanan et al., 2025, ϵ is defined as $k\omega$, while it is defined as $0.09k\omega$ here. This coefficient of 0.09 comes from the standard $k-\omega$ SST turbulence model formulation, but it was not applied in RITA, as there are coefficients associated with each model term that can absorb this scaling factor. This scaling factor of 0.09 leads to a difference in the coefficients found here compared to Buchanan et al., 2025.

4.5. Flow field analysis

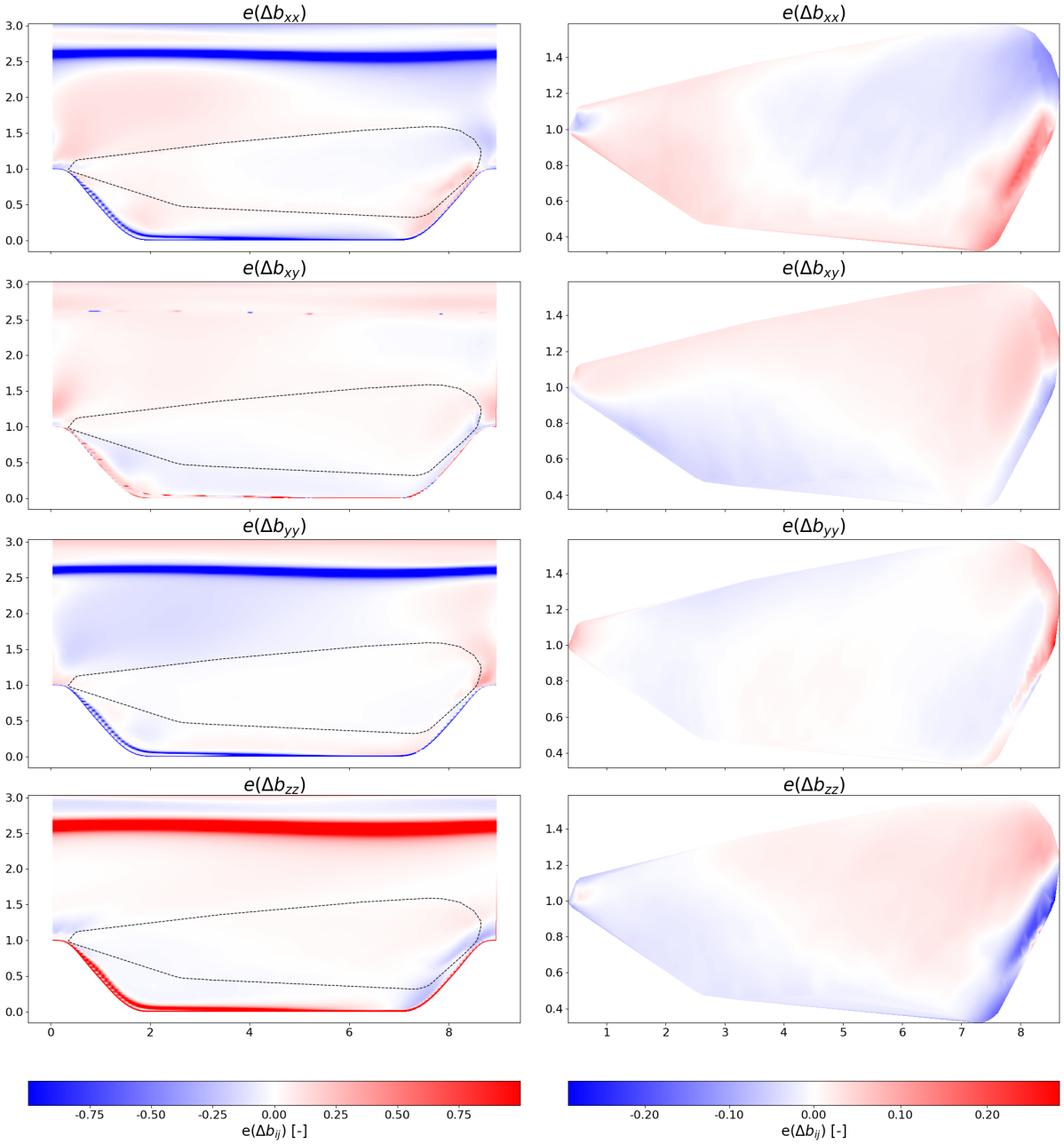
Now that the best models have been identified and optimised, it is important to visualise the predicted fields to analyse where the models perform well and where they struggle. This section presents visual comparisons between the true and predicted fields for the best-performing models in each correction category. All plots are clipped between -1 and 1 for better visualisation, as some areas in the domain have very high errors that would skew the colour map otherwise.

4.5.1. α_{PH} = 1.0

The following analysis focuses on the case with α_{PH} = 1.0, Lx/H = 9 and Ly/H = 3.036. This case is chosen as it is one of the training cases, so the models should perform well here.

Three tensor Δb_{ij} correction

Figure 4.5 shows the difference between the true and predicted Reynolds stress anisotropy tensor $e(\Delta b_{ij})$ correction using the three-tensor model for the case with α_{PH} = 1.0, Lx/H = 9 and Ly/H = 3.036. The left plot shows the difference in the full domain, while the right plot shows the difference only in the shear region classified by RITA. From Figure 4.5b, it can be seen that the model performs quite well in the shear region, with only small errors seen in most areas, for all components of the tensor. The model struggles the most in the reattached flow region, where most of the errors are concentrated. This indicates that the model is not able to capture the complex flow behaviour in that region well, leading to larger errors.



(a) Difference between the true and predicted Δb_{ij} using a three-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using a three-tensor model, in the RITA classified shear region.

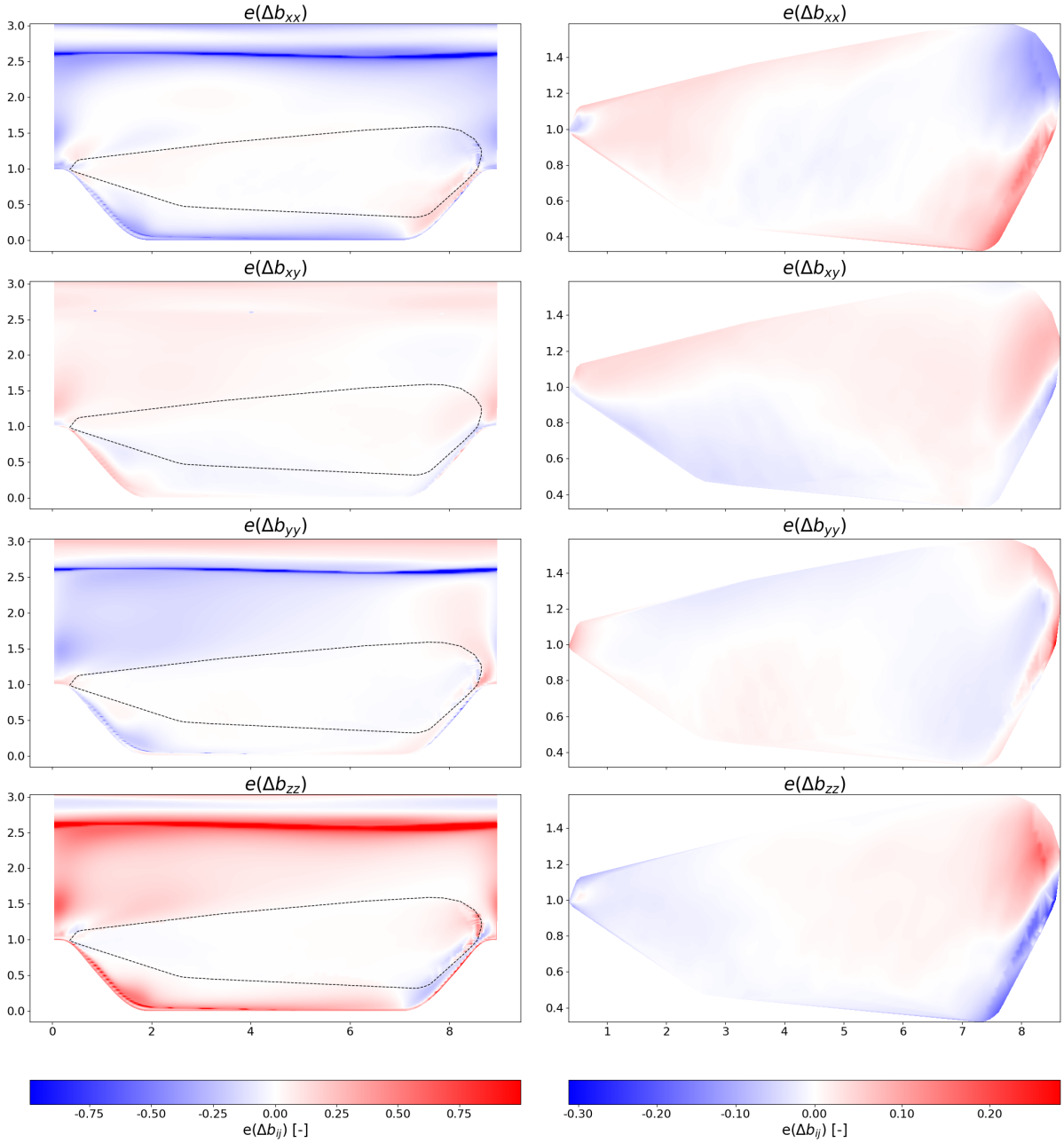
Figure 4.5: Difference in predicted vs true Δb_{ij}, $e(\Delta b_{ij}) = \Delta \hat{b}_{ij} - \Delta b_{ij}$, for α_{PH}=1.0, Lx/H=9, Ly/H=3.036, using the three-tensor model. Red indicates overprediction, where the model predicts a higher anisotropy than the true values, while blue indicates underprediction. White regions indicate no difference between the predicted and true Δb_{ij}.

The model also predicts the xy and yy components quite well in the shear region, with only small errors seen throughout. The xx and zz components have larger errors compared to the other two components, indicating that the model is not able to capture those components as well.

Figure 4.5a shows the performance of the model in the full domain. There is a large line seen at the top of the domain for all components. This feature mainly comes from the input features and basis tensors. From Figure 3.3, it can be seen that in this region, the second invariant has values that are close to zero. The flow in this region is uniform, and so the velocity gradients are zero. Thus, in the model, there are divisions by zero or very small numbers, leading to large spikes in the output. The models are trained only on the shear region, so they did not learn how to handle the zeros in the input

features and basis tensors.

The model also struggles a lot near the wall. Large errors are seen along the wall for all components, where the errors have a value of 1 or -1. As the values in the field are clipped between -1 and 1, this means that the model is making very large errors that are even larger than 1 in magnitude. Similarly to the region at the top of the domain, there are also some values of zero in the input features near the wall, as seen in Figure 3.3. This indicates that the model is very inaccurate near the wall, and is not able to generalise to the near wall at all.



(a) Difference between the true and predicted Δb_{ij} using a four-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using a four-tensor model, in the RITA classified shear region.

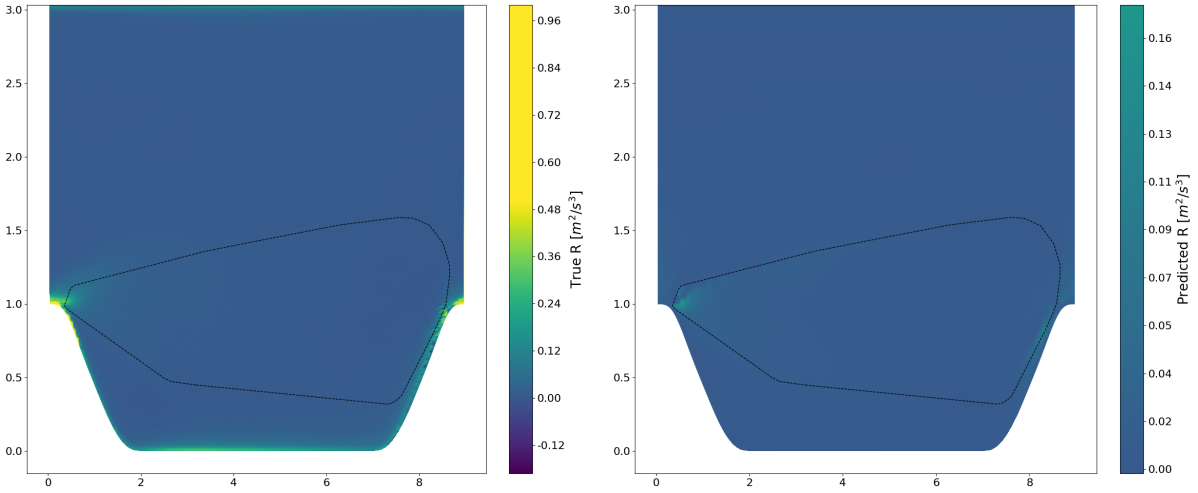
Figure 4.6: Difference in predicted vs true Δb_{ij}, $e(\Delta b_{ij}) = \Delta \hat{b}_{ij} - \Delta b_{ij}$, for α_{PH}=1.0, Lx/H=9, Ly/H=3.036, using the four-tensor model. Red indicates overprediction, where the model predicts a higher anisotropy than the true value, while blue indicates underprediction. White regions indicate no difference between the predicted and true Δb_{ij}.

Four tensor Δb_{ij} correction
Figure 4.6 shows the difference between the true and predicted Reynolds stress anisotropy tensor corrections. The four-tensor correction has lower errors overall compared to the three-tensor correction. This is expected, since the four-tensor model has more degrees of freedom, allowing it to fit the data better. Similar to the three-tensor model, the largest errors are seen in the reattached flow region.

In the full domain, the four-tensor model has more error throughout the domain, outside the shear region, in the region that should have simpler flow behaviour. The error is quite uniform throughout the domain, indicating that the model is not able to generalise towards the regions with simpler flow behaviour.

But while the four-tensor model has more error outside the shear region, it has lower error at the near-wall region compared to the three-tensor model. The large errors seen along the wall in the three-tensor model are not present in the four-tensor model in all components. In the xy and yy components, the errors along the wall are even small, indicating that the four-tensor model is able to generalise to near-wall effects better.

R correction
Figure 4.7 shows the true and predicted turbulence production deficit using the R correction model. The true R is mostly zero everywhere, except in the separation region at the start of the drop, reattached region, and on the walls at the bottom of the domain. The model was able to capture the structure at the separation point quite well and was also able to predict, to an extent, the reattached region.



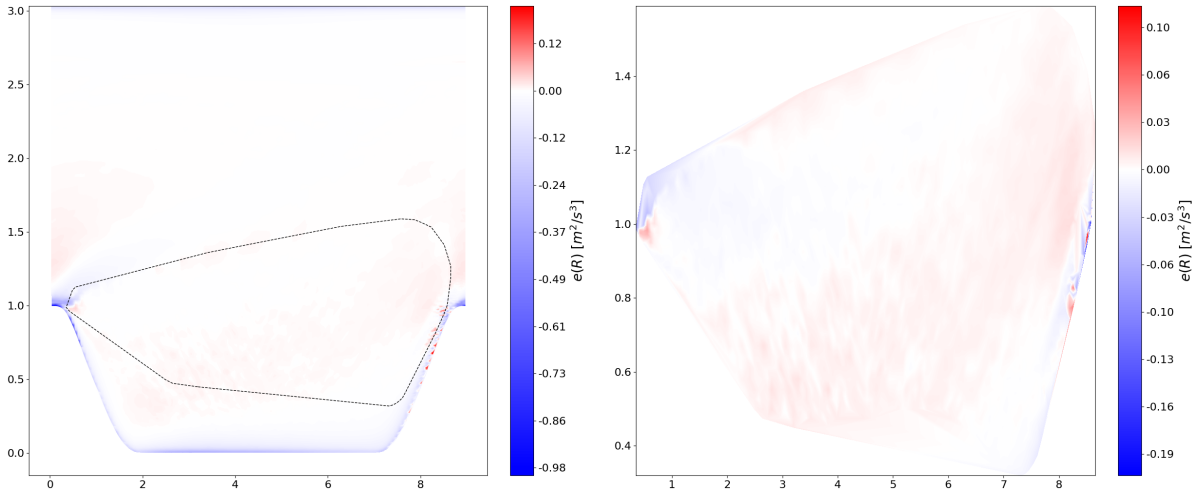
(a) True R. (b) Predicted R using the R correction model.

Figure 4.7: Plots of the true and predicted R for α_{PH}=1.0, Lx/H=9, Ly/H=3.036.

Figure 4.8 shows the prediction capabilities of the R correction model in more detail. The model, while capable of predicting the separation region at the start, underpredicts the magnitude of the production deficit there. Looking at Figure 4.8b, specifically at the separation region, the model is seen to underpredict in general, while also overpredicting the production deficit slightly at the bottom part of the separation region.

At the reattached region, the true field seems to be quite noisy, with many small regions of high or low production. But the model is able to capture an overall trend of a slight positive production deficit in that region, although it misses many of the smaller structures seen in the true field. In the middle, where the production deficit is mostly zero, the model predicted that well, with only small errors seen.

The performance outside the shear region is quite good in general, since most of the field is zero there. However, the slight non-zero production deficit at the bottom wall is not predicted by the model at all. This means that the model is once again not able to capture near-wall effects well, similar to the Reynolds stress anisotropy tensor corrections.



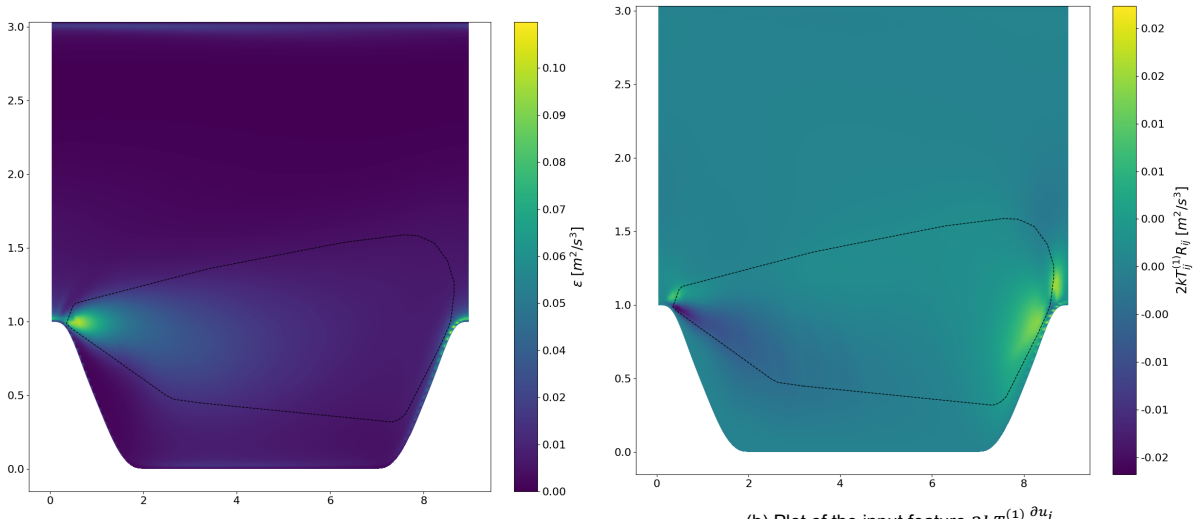
(a) Difference between the true and predicted R using the final R correction model, in the full domain.

(b) Difference between the true and predicted R using the final R correction model, in the RITA classified shear region.

Figure 4.8: Difference in predicted vs true R, $e(R) = \hat{R} - R$, for α_{PH}=1.0, Lx/H=9, Ly/H=3.036. Red indicates overprediction, where the model predicts a higher production deficit than the true R, while blue indicates underprediction. White regions indicate no difference between the predicted and true R.

A plot of the input feature ε is also visualised in Figure 4.9a. This analysis is to visualise the input features used in the R correction model. It can be seen that epsilon has a very similar structure to the true R field, with high values at the separation region, but it does not have any values at the reattachment region. Since the values at the separation region are the highest, epsilon is able to capture most of the production deficit there. This explains the high importance of epsilon seen in the ablation study.

The other input feature $2kT_{ij}^{(1)} \frac{\partial u_i}{\partial x_j}$, is visualised in Figure 4.9b. This feature was seen to be less important than epsilon in the ablation study, but it still contributed significantly to the model performance. It can be seen that this feature has non-zero values at both the separation and reattached regions. While the structure at the separation region is not as similar to the true R field, the reattached region is captured quite well. This indicates that this feature is mainly responsible for predicting the production deficit at the reattached region, while epsilon is mainly responsible for predicting the production deficit at the separation region. But neither feature has non-zero values at the wall region, explaining why the model is not able to predict the production deficit there.



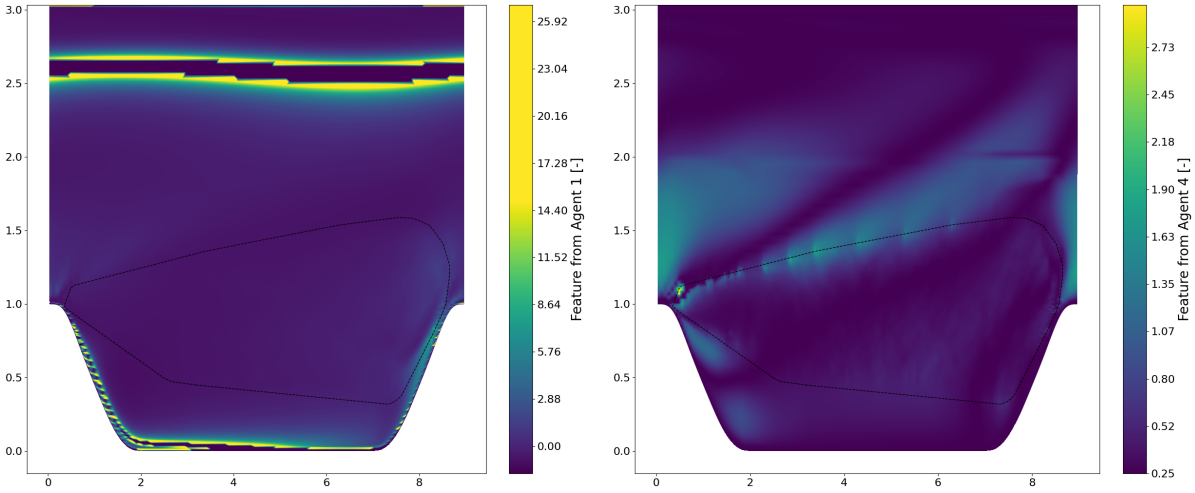
(a) Plot of the input feature ε.

(b) Plot of the input feature $2kT_{ij}^{(1)} \frac{\partial u_i}{\partial x_j}$.

Figure 4.9: Plots of the 2 input features used in the R correction model for α_{PH}=1.0, Lx/H=9, Ly/H=3.036.

This effect is more obvious when looking at the corresponding plots of the models found by MADSR. In Figure 4.10a, the model found by MADSR for the first tensor term is visualised. It can be seen that within the shear region, there are only values at the reattachment region. This filters out the positive values of the first tensor term at the separation region, and only retains the values at the reattachment region. Outside the shear region, the model also has non-zero values along the wall and along the top of the domain. These values mainly come from the $1/I_2$ term in the model, as I_2 approaches zero in these regions, leading to high values of the model. But in the first tensor term, these values are filtered out by the basis tensor itself, which is zero in these regions.

In Figure 4.10b, the model found by MADSR has a uniform value throughout the domain of around 0.25. This means that the model is mainly scaling epsilon by a constant factor to predict the production deficit. The contribution of the R_div term comes from the beginning of the shear region, where the values of R are slightly higher than the rest of the domain. This indicates that the model is mainly relying on epsilon to predict the production deficit, with some small adjustments made by the R_div term.



(a) Plot of the model for the first tensor term.

(b) Plot of the model for the ϵ term.

Figure 4.10: Plots of the 2 input features used in the R correction model for α_{PH}=1.0, Lx/H=9, Ly/H=3.036.

4.5.2. α_{PH} = 1.5

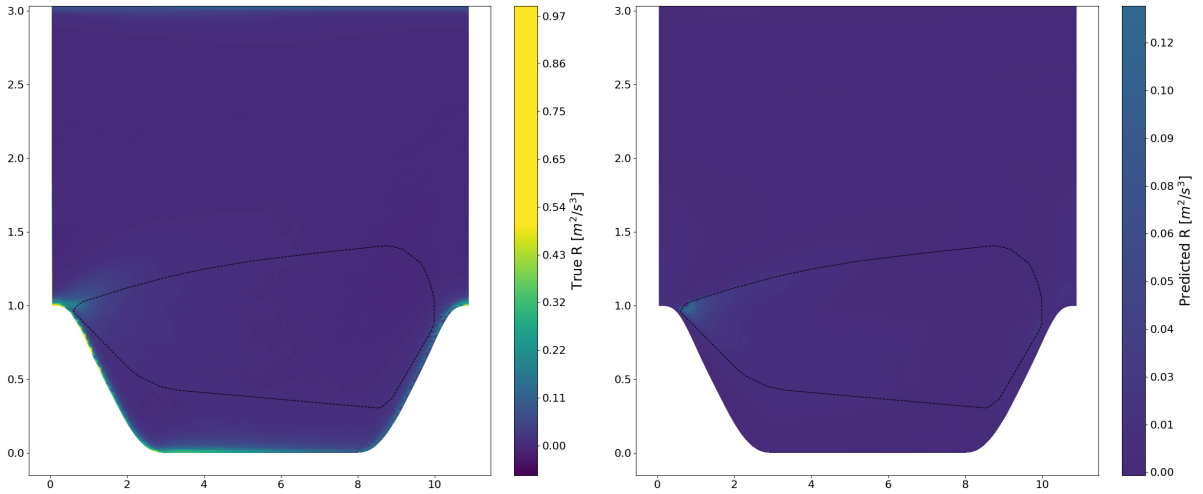
Δb_{ij} correction

For the α_{PH} = 1.5 case, the visualisations of the Reynolds stress anisotropy tensor corrections are similar to the α_{PH} = 1.0 case for both the three and four-tensor models, so they are not shown here for brevity. The only difference is that the errors are generally lower overall, since the flow is less complex in this case. The model is able to predict the shear region well, with only small errors seen throughout. The visualisations can be found in Appendix B.

R correction

Figure 4.11 shows the true and predicted turbulence production deficit using the R correction model for the case with α_{PH} = 1.5, Lx/H = 10.929 and Ly/H = 3.036. The true field is again mostly zero everywhere, except in the separation region at the start of the drop, but the reattachment region has almost no production deficit at all, unlike the α_{PH} = 1.0 case. Since the separation in this case is milder, the values of the production deficit are much lower in magnitude as well. In this case, the model is able to capture the structure at the separation point quite well and does not predict any production deficit at the reattachment region, which is correct.

In Figure 4.12, the model can predict the production deficit much better than in the α_{PH} = 1.0 case. The error is much smaller overall, with only small regions of under- and over-prediction seen. The model still underpredicts the magnitude of the production deficit, and then again overpredicts slightly at the

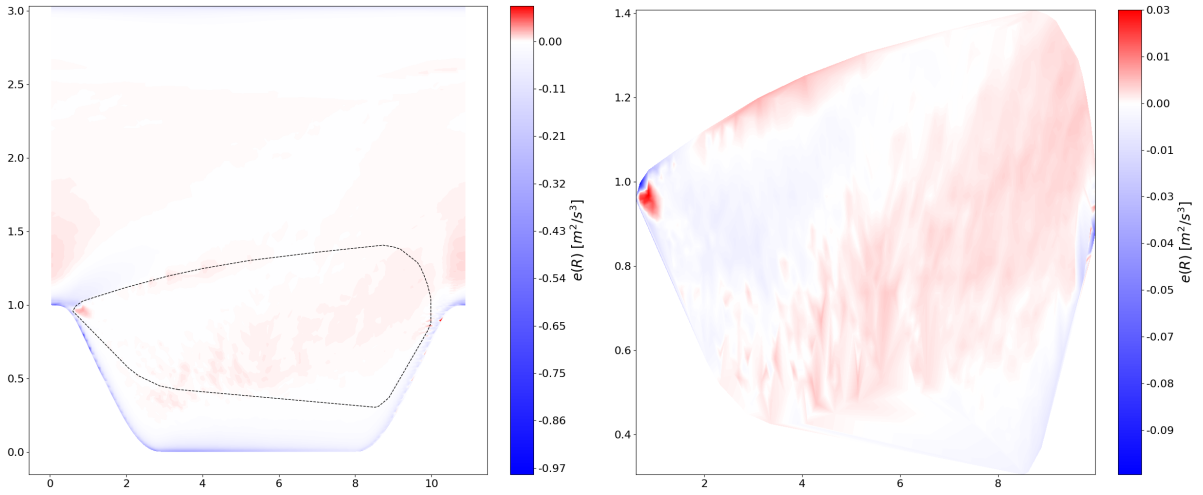


(a) True R.

(b) Predicted R using the R correction model.

Figure 4.11: Plots of the true and predicted R field for α_{PH}=1.5, Lx/H=10.929, Ly/H=3.036.

bottom part of the separation region. However, similar to the α_{PH} = 1.0 case, the model still does not predict the near-wall effects seen at the bottom wall of the domain.



(a) Difference between the true and predicted R using the final R correction model, in the full domain.

(b) Difference between the true and predicted R using the final R correction model, in the RITA classified shear region.

Figure 4.12: Difference in predicted vs true R, $e(R) = \hat{R} - R$, for α_{PH}=1.5, Lx/H=10.929, Ly/H=3.036. Red indicates overprediction, where the model predicts a higher production deficit than the true R, while blue indicates underprediction. White regions indicate no difference between the predicted and true R.

4.5.3. Summary of the flow field analysis

In general, all models were able to predict the shear region well for both Reynolds stress anisotropy tensor corrections and the R correction. Most of the errors were concentrated in the reattached flow region at the leeward side of the domain. All models also fail to predict near-wall effects well. Especially for the Δb_{ij} corrections, where large errors were seen along the wall in many cases.

Additionally, observation of the structure of the input features and models used in the R correction provided insights into the importance of each term in predicting different regions of the production deficit. Epsilon was found to be crucial for capturing the separation region, while the first tensor term was more important for predicting the reattachment region.

5

Conclusions

The Reynolds-Averaged Navier-Stokes (RANS) equations contain closure terms that must be modelled to represent the effects of turbulence on the mean flow. In practice, these are commonly closed with eddy-viscosity models under the Boussinesq approximation, which imposes a linear relation between the Reynolds stress anisotropy and the mean strain rate. While effective for many attached flows, such linear closures struggle in shear layers of separated flows. The explicit algebraic Reynolds-stress models (EARSM) seek to encode these effects via tensor-basis expansions with scalar coefficient functions. Data-driven approaches have been increasingly used to perform EARSM by learning from high-fidelity simulation data. Among these methods, symbolic regression is particularly valuable because it discovers analytical expressions that remain interpretable while improving model accuracy. This balance between data-driven correction and physical interpretability makes symbolic regression a promising tool for advancing turbulence modelling and for producing corrections that can be analysed.

Deep symbolic regression (DSR) has recently emerged as a symbolic regression technique that leverages deep learning to efficiently explore the space of mathematical expressions. By training neural networks to generate candidate expressions and using reinforcement learning to guide the search, DSR can discover compact and accurate models for complex systems. But applying DSR to turbulence modelling presents challenges, as DSR does regression for single equations, while EARSM requires learning multiple coefficient functions simultaneously.

In this work, it was observed that the coefficient functions in EARSM form a cooperative multi-agent problem, where each coefficient function contributes to the overall model performance. DSR also proposes a reinforcement learning framework for symbolic regression. Therefore, we propose to extend DSR to a multi-agent setting, where each agent learns to generate one coefficient function, and the agents cooperate to optimise the overall EARSM performance. By reformulating the EARSM symbolic regression problem as a multi-agent reinforcement learning task and using DSR as the underlying symbolic regressor, modern multi-agent reinforcement learning algorithms can be employed to train the agents to cooperatively discover accurate and interpretable EARSM closures. Thus, the research question addressed in this work is:

How can multi-agent reinforcement learning techniques be integrated with deep symbolic regression to perform symbolic turbulence modelling?

To answer the research question, several sub-questions were investigated:

1. **What MARL algorithms are best suited for cooperative symbolic regression tasks, and how can they be integrated with DSR?**

   Throughout this work, several algorithmic variants of the Multi-Agent Deep Symbolic Regression (MADSR) framework were developed to investigate how different reinforcement learning strategies affect symbolic model discovery. The baseline formulation, the vanilla MADSR, extended the Deep Symbolic Regression (DSR) algorithm into a multi-agent setting, where each agent

independently learned one scalar coefficient function while sharing a common reward. This simple extension already demonstrated the advantages of cooperative learning over the standard single-agent DSR. Building upon it, the PPO-based variant (vPPODSR) introduced the clipping mechanism of proximal policy optimisation(PPO) to stabilise training and allow multiple gradient updates per batch, improving convergence speed and robustness.

To further enhance coordination between agents and reduce the high-variance gradient estimates inherent to REINFORCE, an actor-critic variant (ACDSR) was developed. This was inspired based on the multi-agent proximal policy optimization (MAPPO) algorithm by Yu et al., 2022, which uses a centralised critic and decentralised execution (CTDE) paradigm to provide more informative feedback to each agent. Finally, a PPO version of ACDSR was implemented, which combined the sample efficiency of PPO with the CTDE framework, resulting in the PPODSR algorithm.

2. **How does integrating MARL techniques impact the efficiency of learning and the quality of discovered expressions compared to base DSR?**

   All MADSR variants were tested on the challenging EARSM symbolic regression task. The implementation of PPO, instead of REINFORCE or actor-critic, was found to be the most effective in improving the learning efficiency of the algorithms. Both vPPODSR and PPODSR significantly outperformed the vanilla MADSR and ACDSR in terms of average and best reward in 1000 training steps. This indicates that the PPO clipping mechanism effectively stabilised training and allowed for more efficient policy updates, leading to faster convergence towards high-quality solutions.

   The CTDE framework used in ACDSR and PPODSR also contributed to improved performance by enabling better coordination among agents. However, the benefits of CTDE only manifested when combined with PPO, as ACDSR did not outperform the vanilla MADSR. This is due to critic bias, which misleads policy updates when the critic is not well-trained, in which case the multiple epochs of PPO help to mitigate this issue. PPODSR achieved the highest average and best rewards, demonstrating that the combination of PPO and CTDE provides the most effective framework for cooperative symbolic regression. CTDE was able to allow the agents to better understand their joint impact on the overall EARSM performance, and learn the underlying structure of the target expressions more effectively, allowing PPODSR to discover much better expressions than the other algorithms.

3. **How does the multi-agent DSR framework perform in discovering EARSM closures and k-corrective RANS models?**

   The EARSM correction to regress a Δb_{ij} was found to be too challenging for all MADSR variants, as none were able to discover symbolic expressions that had an R^2 score above 0.55 on the frozen RANS evaluation. Both a three-tensor and four-tensor basis EARSM were considered, where both formulations proved too complex for the MADSR algorithms to learn effectively. The high complexity of a tensor regression, where multiple tensor basis components interact to form the final Reynolds stress anisotropy, made the search space too large for the MADSR algorithms to explore.

   However, in the case of the k-corrective frozen RANS evaluation, where only a single scalar correction to the turbulent kinetic energy production deficit was learned, the MADSR algorithms were able to discover accurate and interpretable EARSM closures. An R^2 score of 0.8 was achieved by every algorithm, with PPODSR again achieving the best performance.

4. **Is a multi-agent DSR approach capable of discovering accurate and interpretable EARSM closures?**

   The models discovered during training for the Δb_{ij} problem were found to be quite generalisable toward the test set, with only a small drop in R^2 performance from training to testing. A further optimisation of the coefficients in the discovered expressions further improved the test R^2 scores. The R correction models discovered were much less generalisable, where different models achieved very different test performances for different metrics. Even with coefficient optimisation, there were almost no changes in test performance, indicating that the discovered expressions were overfitted to the training data.

An ablation study of the basis tensors was also done to observe the impact of each tensor. All basis tensors were found to be equally important for the Δb_{ij}, where the R^2 value decreased a very large amount when any tensors were removed. The epsilon term was found to be the most critical, as removing it caused a significant drop in performance. The first basis tensor also contributed meaningfully, where removing it led to a moderate decrease in accuracy. The second and third tensors were found to be completely redundant for this task, as removing either had a negligible impact, or even improved performance. This insight suggests that for the k-corrective task, a simplified EARSM formulation using only the epsilon and first basis tensor may be sufficient, reducing model complexity while maintaining accuracy.

An analysis of the models discovered also revealed insightful patterns. The algorithms consistently identified certain mathematical structures, where all runs that achieved high accuracy contained similar sub-expressions. For the ε coefficient function, $c_1 \cdot R_div(D_k/P_k) + c_2$ terms were commonly found, reproducing a result found by Buchanan et al., 2025. Additionally, the $\log(I_2)/I_2$ structure was frequently discovered for the first tensor. These recurring patterns highlight the ability of the MADSR framework to uncover meaningful and interpretable expressions that align with established turbulence modelling concepts.

In conclusion, the integration of advanced MARL methods like PPO and CTDE in PPODSR significantly enhanced learning efficiency and model quality compared to base DSR. It was able to identify structures in the k-corrective task, demonstrating the potential of multi-agent DSR for symbolic turbulence modelling. However, the complexity of full tensor-based EARSM remains a challenge, indicating avenues for future research.

5.1. Limitations

One key limitation of MADSR is the computational cost of coefficient optimisation. This is the bottleneck of the entire MADSR framework, as coefficients are really powerful and are able to help the discovered expressions fit the data much better. However, optimising coefficients requires multiple evaluations of the frozen RANS solver, which is computationally expensive. The agents learn that coefficients are important for maximising reward, and thus frequently propose new expressions that require coefficient optimisation. This leads to a large number of coefficients to be optimised, which significantly slows down training. Coefficient optimisation made up around 90% of the total training time for all MADSR variants. Future work could explore more efficient coefficient optimisation methods or surrogate models to approximate the frozen RANS evaluations, to reduce this computational burden.

One of the main constraints of this work was the time available to implement and test the various MADSR algorithms. Due to the complexity of integrating MARL techniques with DSR, significant effort was required to develop, debug and optimise the algorithms. As a result, only a limited number of MARL variants could be explored, and extensive hyperparameter tuning was not feasible. Only MAPPO was able to be developed, tested and analysed in depth, while other MARL algorithms, such as counterfactual multi-agent policy gradients (COMA) by Foerster et al., 2018, were proposed but did not reach the implementation stage.

There was also limited time for inference of the discovered models in a RANS simulation. While frozen RANS evaluations provided a useful proxy for assessing model performance, a full RANS simulation would have offered a more comprehensive evaluation of the discovered EARSM closures. However, due to time constraints, only frozen RANS tests were conducted, leaving the true performance of the models in practical simulations unverified.

5.2. Recommendations for future work

GP-meld, and uDSR by Mundhenk et al., 2021 and Landajuela et al., 2022 respectively, are two recent symbolic regression algorithms that were built upon DSR and have shown promising results in symbolic regression. Future work could explore integrating these other symbolic regression techniques into the MADSR framework to see if they can further enhance the performance of multi-agent symbolic turbulence modelling.

COMA by Foerster et al., 2018 is another MARL algorithm that further improves upon the CTDE framework by using a counterfactual baseline to learn more effective policies. Implementing a COMA-based MADSR variant could provide additional insights into how different MARL strategies impact cooperative symbolic regression.

A

Found models

A.1. Three tensor models

$$\begin{aligned}\Delta b_{ij} = {} & \frac{\log\left(-0.6073516 I_1 - 0.9701\right)}{I_1} T_{ij}^{(1)} \\ & + D_k/P_k \left(6.67375731 - 3.066198 I_1\right) + D_k/P_k \, T_{ij}^{(2)} \\ & + 2 I_1 + 8.0494 \log\left(I_2\right) + 32.97330 \, T_{ij}^{(3)}\end{aligned} \tag{A.1}$$

$$\begin{aligned}\Delta b_{ij} = {} & I_2 - 0.219538 + \frac{\log\left(I_1\right) + 3.33445}{11.11857859 \left(0.299899 \log\left(I_1\right) + 1\right)^2 + 1.0} T_{ij}^{(1)} \\ & + 7.088580 \left(0.3755954 I_2 + 0.3755954 \log\left(D_k/P_k\right) + 1\right)^2 \, T_{ij}^{(2)} \\ & + I_1 + I_2 + 7.9757805 \log\left(I_2\right) + 32.74024 \, T_{ij}^{(3)}\end{aligned} \tag{A.2}$$

$$\begin{aligned}\Delta b_{ij} = {} & -0.22994212 + \frac{\tanh\left(\log\left(I_1\right) + 3.32426210\right)}{\tanh^2\left(\log\left(I_1\right) + 3.324262\right) + 1.0} T_{ij}^{(1)} \\ & + 5.1438855 \tanh\left(D_k/P_k\right) + 1.3907924 \, T_{ij}^{(2)} \\ & + I_2 + D_k/P_k + 8.061203 \log\left(\tanh\left(I_2\right)\right) + 32.68499 \, T_{ij}^{(3)}\end{aligned} \tag{A.3}$$

$$\begin{aligned}\Delta b_{ij} = {} & D_k/P_k + \tanh\left(\log\left(I_1\right) + 3.5890425\right) - 0.9907184 \, T_{ij}^{(1)} \\ & + 8.68850295 D_k/P_k e^{-3.637422 I_1} \, T_{ij}^{(2)} \\ & + D_k/P_k + 8.1676853 \log\left(\tanh\left(\frac{I_2}{I_2^2 + 1.0}\right)\right) + 32.983259 \, T_{ij}^{(3)}\end{aligned} \tag{A.4}$$

$$\begin{aligned}\Delta b_{ij} = {} & D_k/P_k + \tanh\left(\log\left(I_1\right) + 3.589042\right) - 0.990718 \, T_{ij}^{(1)} \\ & + 8.6885029 D_k/P_k e^{-3.6374224 I_1} \, T_{ij}^{(2)} \\ & + D_k/P_k + 8.1676853 \log\left(\tanh\left(\frac{I_2}{I_2^2 + 1.0}\right)\right) + 32.983259 \, T_{ij}^{(3)}\end{aligned} \tag{A.5}$$

$$\begin{aligned}\Delta b_{ij} = {} & -0.254087766 + \frac{1.1004971 \left(\log\left(I_1\right) + 3.35048\right)}{11.22578 \left(0.298463 \log\left(I_1\right) + 1\right)^2 + 1.0} T_{ij}^{(1)} \\ & + 0.38621060 - \log\left(\frac{I_1}{I_1^2 + 1.0} - 0.003228117\right) T_{ij}^{(2)} \\ & + D_k/C_k + 8.4002369 \log\left(\frac{I_2}{I_2^2 + 1.0}\right) + 33.571316 \, T_{ij}^{(3)}\end{aligned} \tag{A.6}$$

$$\begin{aligned}\Delta b_{ij} = &-0.2573897 + \frac{1.103406\,(\log(I_1) + 3.3484447)}{11.21208\,(0.29864611\log(I_1) + 1.0)^2 + 1.0}\,T_{ij}^{(1)} \\ &+ 1.799611e^{-0.4428176D_k/P_k \log\left(\frac{I_1}{I_1^2+1.0}\right)}\,T_{ij}^{(2)} \\ &+ 7.863320 + \frac{\left(I_2^2 + 1.0\right)\log\left(\frac{I_2}{I_2^2+1.0} + 1.14202\right)}{I_2}\,T_{ij}^{(3)}\end{aligned} \tag{A.7}$$

$$\begin{aligned}\Delta b_{ij} = &\tanh\left(\frac{1.7746841I_2}{I_1}\right) + 0.5294934\,T_{ij}^{(1)} \\ &+ D_k/P_k\,(-31.931387I_1 + 31.931387\tanh(I_2) + 9.62890)\;T_{ij}^{(2)} \\ &+ D_k/C_k + \frac{7.9030252e^{\tanh(I_2)} - 7.77172}{I_2}\,T_{ij}^{(3)}\end{aligned} \tag{A.8}$$

$$\begin{aligned}\Delta b_{ij} = &-I_1 + \frac{29.856229I_1 - 1.3530607}{891.394458\,(I_1 - 0.045319)^2 + 1.0}\,T_{ij}^{(1)} \\ &+ 14.3204573\left(-\frac{0.26425I_1}{I_1^2 + 1.0} + D_k/P_k^4 + 0.458055\right)^2\,T_{ij}^{(2)} \\ &+ \frac{18.605215\,(-0.231836I_2 - 1)^2 - 18.47599}{I_2}\,T_{ij}^{(3)}\end{aligned} \tag{A.9}$$

$$\begin{aligned}\Delta b_{ij} = &\,I_1 + \frac{87.8321I_2 + 1.31935}{7714.482852276495\,(I_2 + 0.015021)^2 + 1.0}\,T_{ij}^{(1)} \\ &+ -9.306656I_1 + 8.29487D_k/P_k\,T_{ij}^{(2)} \\ &+ \frac{18.215\,(-0.234307I_2 - 1)^2 - 18.0867327}{I_2}\,T_{ij}^{(3)}\end{aligned} \tag{A.10}$$

$$\begin{aligned}\Delta b_{ij} = &\tanh(49.3514420I_1 - 1.078916) - \tanh\left(e^{I_1}\right)T_{ij}^{(1)} \\ &+ -D_k/C_k - 1.113158\log(I_1) + 0.8944105\,T_{ij}^{(2)} \\ &+ \frac{\log(\tanh(I_2) + 1.1473849)}{\tanh(I_2)} + 7.31\,T_{ij}^{(3)}\end{aligned} \tag{A.11}$$

$$\begin{aligned}\Delta b_{ij} = &\log(D_k/P_k + \tanh(26.5719756I_1 - 0.63223)) + \tanh(I_2)\,T_{ij}^{(1)} \\ &+ 1.8547810e^{-I_2}e^{3.001294I_2 + 3.001294D_k/P_k^2}\,T_{ij}^{(2)} \\ &+ \frac{\log(-I_2 + 1.366782\tanh(I_2) + 1.00615)}{I_2\tanh(I_1)}\,T_{ij}^{(3)}\end{aligned} \tag{A.12}$$

$$\begin{aligned}\Delta b_{ij} = &\tanh(D_k/P_k) + \frac{\log(\tanh(23.35388I_1 + D_k/P_k - 0.46824426))}{D_k/P_k}\,T_{ij}^{(1)} \\ &+ e^{\frac{\tanh\left(0.8425314D_k/P_k\right)}{I_1 + 0.26470568}}\,T_{ij}^{(2)} \\ &+ \frac{-\tanh(I_2) + \tanh(9.886245\log(I_2 + 1.01357))}{\tanh(I_2)}\,T_{ij}^{(3)}\end{aligned} \tag{A.13}$$

$$\begin{aligned}\Delta b_{ij} = &\tanh(\log(40.139816I_1)) - 0.5818273\,T_{ij}^{(1)} \\ &+ 5.21211\left(-0.438019D_k/P_k + 0.3134e^{4I_1} - 1\right)^2\,T_{ij}^{(2)} \\ &+ \frac{19.357267\,(0.227288\tanh(I_2) + 1)^2 - 19.22425}{\tanh(I_2)}\,T_{ij}^{(3)}\end{aligned} \tag{A.14}$$

$$\begin{aligned}\Delta b_{ij} = {} & e^{\frac{0.069013 I_2}{I_1^2}} - 0.562223\, T_{ij}^{(1)} \\ & + e^{\frac{D_k/P_k}{I_1 + 0.339138}}\, T_{ij}^{(2)} \\ & + \frac{1.3573065 + \frac{1.006193 - e^{I_2}}{I_2}}{I_1}\, T_{ij}^{(3)}\end{aligned} \tag{A.15}$$

A.2. Four tensor models

$$\begin{aligned}\Delta b_{ij} = {} & \log\left(\tanh\left(D_k/P_k\right) + \tanh\left(25.39547 I_1 - 0.5690660\right)\right) T_{ij}^{(1)} \\ & + D_k/P_k \left(-D_k/P_k + 16.71201 e^{-D_k/P_k - 5.837830 \tanh\left(I_1\right)}\right) T_{ij}^{(2)} \\ & + D_k/P_k + \frac{0.0181181 \log\left(\tanh\left(I_1\right) + 19.729477\right)}{I_2}\, T_{ij}^{(3)} \\ & + -D_k/P_k + \frac{D_k/P_k + \frac{0.56671415 - 3.259539 \tanh\left(D_k/P_k\right)}{D_k/P_k^2}}{D_k/P_k}\, T_{ij}^{(4)}\end{aligned} \tag{A.16}$$

$$\begin{aligned}\Delta b_{ij} = {} & \tanh\left(36.49104 I_1 - 0.8122938\right) - 0.694032\, T_{ij}^{(1)} \\ & + 5.18814151 D_k/P_k \log\left(\log\left(I_1\right)\right) + 0.7039421\, T_{ij}^{(2)} \\ & + \left(19.3401549 \log\left(I_2\right) + 90.41928\right) \log\left(\log\left(I_1\right)\right) T_{ij}^{(3)} \\ & + 161.39159 \left(0.1773484 \log\left(I_1\right) + 1\right)^2\, T_{ij}^{(4)}\end{aligned} \tag{A.17}$$

$$\begin{aligned}\Delta b_{ij} = {} & \tanh\left(32.473811 e^{I_1} - 33.153506\right) - 0.711702\, T_{ij}^{(1)} \\ & + 4.128068 e^{D_k/P_k} - 3.0427278\, T_{ij}^{(2)} \\ & + -52.04001 e^{236.38951 I_2}\, T_{ij}^{(3)} \\ & + -106.4722531 e^{-6.217191 D_k/P_k}\, T_{ij}^{(4)}\end{aligned} \tag{A.18}$$

$$\begin{aligned}\Delta b_{ij} = {} & \frac{87.288974 I_2 + 1.2584340}{7619.365 \left(I_2 + 0.01441687\right)^2 + 1.0} + 0.093267\, T_{ij}^{(1)} \\ & + -I_1 - 1.0607126 \tanh\left(I_1\right) + 3.8227338\, T_{ij}^{(2)} \\ & + \frac{0.24676941}{7.188260 I_2 + 0.01601041}\, T_{ij}^{(3)} \\ & + 18.7124708 - \frac{12.609462}{D_k/P_k}\, T_{ij}^{(4)}\end{aligned} \tag{A.19}$$

$$\begin{aligned}\Delta b_{ij} = {} & \tanh\left(35.83744 I_1 - 0.80982 m\right) - 0.6784039\, T_{ij}^{(1)} \\ & + \frac{D_k/P_k}{0.6110479 I_1 + 0.10882}\, T_{ij}^{(2)} \\ & + \frac{-0.26526776 \tanh\left(I_1\right) - 1.55433}{I_1}\, T_{ij}^{(3)} \\ & + 59.530775 \log\left(I_2\right) + 150.62068200\, T_{ij}^{(4)}\end{aligned} \tag{A.20}$$

$$\begin{aligned}\Delta b_{ij} = {} & 2.7912215 \tanh\left(33.402389098277372 I_1\right) - 2.517549\, T_{ij}^{(1)} \\ & + I_1 \left(-\tanh\left(D_k/P_k\right) - 19.18629\right) + 4.442680\, T_{ij}^{(2)} \\ & + \frac{0.07813 - \frac{0.2614064 \tanh\left(I_1\right)}{\tanh^2\left(I_1\right)+1.0}}{I_2} T_{ij}^{(3)} \\ & + \frac{5.2306414 - \frac{4.42701}{D_k/P_k}}{D_k/P_k} T_{ij}^{(4)}\end{aligned} \tag{A.21}$$

$$\begin{aligned}\Delta b_{ij} = {} & \tanh\left(402.0398 I_1^2\right) - 0.732917145\, T_{ij}^{(1)} \\ & + 22.32267494 \left(-I_1 + 0.211654 D_k/P_k + 0.3452129989\right)^2 T_{ij}^{(2)} \\ & + \frac{-\frac{0.0011114 D_k/C_k}{D_k/P_k\left(D_k/C_k^2+1.0\right)} + 0.0520341}{I_2} T_{ij}^{(3)} \\ & + \frac{-1.13851853 - \frac{0.343730}{D_k/P_k}}{D_k/P_k^2} T_{ij}^{(4)}\end{aligned} \tag{A.22}$$

$$\begin{aligned}\Delta b_{ij} = {} & \tanh\left(\frac{1.8506339 I_2}{I_1}\right) + 0.5448001\, T_{ij}^{(1)} \\ & + \frac{0.41127}{-I_2 + \tanh\left(I_1\right) + 0.04497598} T_{ij}^{(2)} \\ & + \frac{0.145761 - 0.7018434 I_1}{I_2} T_{ij}^{(3)} \\ & + \frac{0.4968954 - 3.0541985 D_k/P_k^4}{I_2} T_{ij}^{(4)}\end{aligned} \tag{A.23}$$

$$\begin{aligned}\Delta b_{ij} = {} & -0.2569117 + \frac{1.076682044 \left(\log\left(I_1\right) + 3.3448089\right)}{11.18774 \left(0.298970739 \log\left(I_1\right) + 1\right)^2 + 1.0} T_{ij}^{(1)} \\ & + e^{0.44589346\left(-1 + \frac{0.49605960}{\log\left(D_k/P_k\right)}\right)^2} T_{ij}^{(2)} \\ & + -137.585841 \left(0.24607960 \log\left(\frac{I_2}{I_2^2 + 1.0}\right) + 1\right)^2 T_{ij}^{(3)} \\ & + 178.31175420 \left(-0.2953203 \log\left(I_2\right) - 1\right)^2 - 8.524447\, T_{ij}^{(4)}\end{aligned} \tag{A.24}$$

A.3. R correction models

$$\begin{aligned}b_{ij}^R = {} & 8.18597 \left(0.3495141 \log\left(I_2\right) + 1\right)^2 - 1.3008349\, T_{ij}^{(1)} \\ & + D_k/P_k^2\, T_{ij}^{(2)} \\ & + \log\left(I_1\right) + \log\left(I_1 - 0.038195922\right) T_{ij}^{(3)} \\ R = {} & 2k b_{ij}^R \frac{\partial u_i}{\partial x_j} + -\frac{9.791443 D_k/C_k}{D_k/C_k^2 + 1.0} + 5.41922386\, \varepsilon\end{aligned} \tag{A.25}$$

$$
\begin{aligned}
b_{ij}^{R} &= -1.6865210 - \frac{0.0293670}{I_2}\, T_{ij}^{(1)} \\
&\quad + \frac{I_1\left(D_k/C_k^2 + 1.0\right)}{D_k/C_k} + 0.4954660\, T_{ij}^{(2)} \\
&\quad + I_2\left(I_1 + 1.0030402 D_k/P_k\right)\, T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + 5.37861\left(1 + \frac{0.08739}{D_k/C_k}\right)^2 - 6.13405869\,\varepsilon
\end{aligned}
\tag{A.26}
$$

$$
\begin{aligned}
b_{ij}^{R} &= -2.8482158 + \frac{\left(I_2 - 0.03460117\right)\left(I_2^2 + 1.0\right)}{I_2}\, T_{ij}^{(1)} \\
&\quad + \frac{\left(I_2^2 + 1.0\right)\left(-\frac{0.348073 D_k/P_k}{D_k/P_k^2 + 1.0} + 0.092201474\right)}{I_2}\, T_{ij}^{(2)} \\
&\quad + -\frac{1.3009507979179462}{I_1 + \frac{I_1}{I_1^2 + 1.0}}\, T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + 17.86783 \tanh^2\left(\frac{D_k/C_k}{D_k/C_k^2 + 1.0} - 0.7576128\right)\varepsilon
\end{aligned}
\tag{A.27}
$$

$$
\begin{aligned}
b_{ij}^{R} &= 92.361870 \log\left(-0.981 + \frac{0.00034737}{I_2}\right) T_{ij}^{(1)} \\
&\quad + D_k/P_k\, T_{ij}^{(2)} \\
&\quad + \log\left(-0.002383189 D_k/P_k\right) T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + \frac{9.7528611\left(0.00417560661 - D_k/C_k\right)}{\left(0.00417560 - D_k/C_k\right)^2 + 1.0} + 5.402693\,\varepsilon
\end{aligned}
\tag{A.28}
$$

$$
\begin{aligned}
b_{ij}^{R} &= -2.63576413 + \frac{\log\left(I_2 + 0.9727890\right)}{I_2}\, T_{ij}^{(1)} \\
&\quad + 1.738909\left(0.8688337 I_2 - 1\right)^4\, T_{ij}^{(2)} \\
&\quad + D_k/C_k\, T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + 30.81619\left(-\frac{0.1801401 D_k/C_k}{D_k/C_k^2 + 1.0} + 1\right)^2 - 25.00792\,\varepsilon
\end{aligned}
\tag{A.29}
$$

$$
\begin{aligned}
b_{ij}^{R} &= -2.560547939 + \frac{\log\left(I_2 + 0.9719790\right)}{I_2}\, T_{ij}^{(1)} \\
&\quad + e^{1.930099\left(1 - 0.71979 D_k/P_k\right)^2} - 0.62188\, T_{ij}^{(2)} \\
&\quad + I_1 + \tanh\left(D_k/P_k - e^{2 D_k/C_k}\right) T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + -\frac{10.0813 D_k/C_k}{D_k/C_k^2 + 1.0} + 5.553959\,\varepsilon
\end{aligned}
\tag{A.30}
$$

$$
\begin{aligned}
b_{ij}^{R} &= \frac{\log\left(2 I_2 - 0.9658\right)}{I_2}\, T_{ij}^{(1)} \\
&\quad + \log\left(\frac{3.83707850 D_k/P_k}{D_k/P_k^2 + 1.0} - 19.631618\right) - 1.624077\, T_{ij}^{(2)} \\
&\quad + -3.14528076\, T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + D_k/C_k - \frac{14.030364 D_k/C_k}{D_k/C_k^2 + 1.0} + 6.653005\,\varepsilon
\end{aligned}
\tag{A.31}
$$

$$
\begin{aligned}
b_{ij}^{R} &= -27.962971 \log\left(1.050019 + \frac{0.00100530}{I_2}\right) T_{ij}^{(1)} \\
&\quad + 2.1198627 e^{-0.9152880 D_k/P_k}\, T_{ij}^{(2)} \\
&\quad + \log\left(I_2\right) T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + -\frac{10.04101 D_k/C_k}{D_k/C_k^2 + 1.0} + 5.5901802\,\varepsilon
\end{aligned}
\tag{A.32}
$$

$$
\begin{aligned}
b_{ij}^{R} &= 15.5600 \left(0.29538510 \log\left(I_2\right) + 1\right)^2 T_{ij}^{(1)} \\
&\quad + 2.69537 \log\left(1.261896093 + \frac{0.12050144}{I_2}\right) T_{ij}^{(2)} \\
&\quad + \log\left(0.3882543 \tanh\left(I_1\right)\right) T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + -8.5788159 + 15.5501 e^{-\frac{D_k/C_k}{D_k/C_k^2 + 1.0}}\,\varepsilon
\end{aligned}
\tag{A.33}
$$

$$
\begin{aligned}
b_{ij}^{R} &= 0.4625229 + \frac{0.5157122}{D_k/C_k} T_{ij}^{(1)} \\
&\quad + D_k/C_k + 14.411944 - \frac{6.6492}{D_k/C_k} T_{ij}^{(2)} \\
&\quad + -4.795497669567968\, T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + \frac{476.64625 I_2}{3098.2498 I_2^2 + 1.0} + 5.513608\,\varepsilon
\end{aligned}
\tag{A.34}
$$

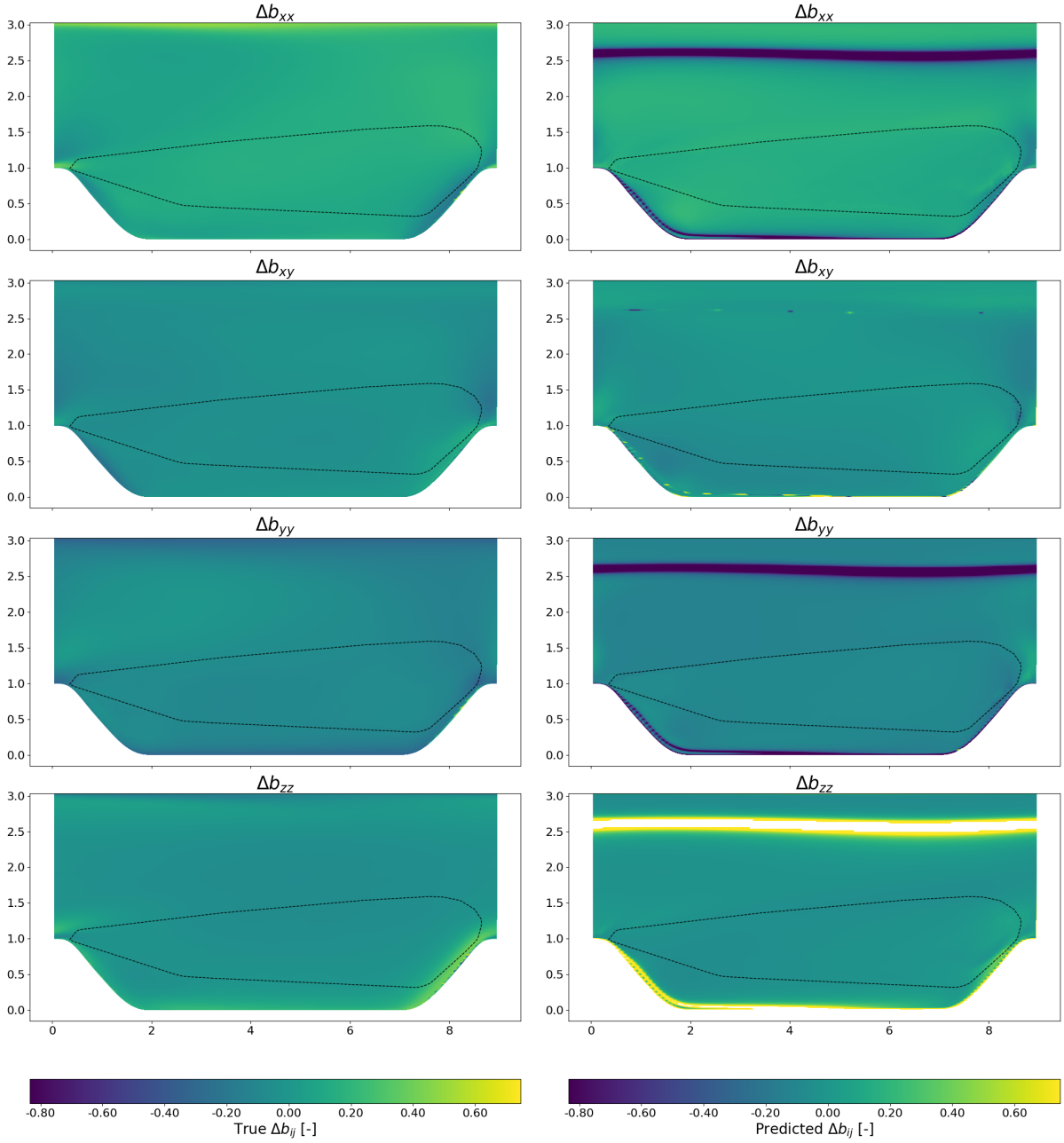
$$
\begin{aligned}
b_{ij}^{R} &= 11.967906 \tanh\left(\frac{144.86785 I_2}{I_2^2 + 1.0}\right) + 12.113802\, T_{ij}^{(1)} \\
&\quad + \frac{46.2661322 I_2 \left(\tanh^2\left(D_k/C_k\right) + 1.0\right)}{\tanh\left(D_k/C_k\right)} + 5.09341\, T_{ij}^{(2)} \\
&\quad + -2.761168212 - \frac{0.0191}{\tanh\left(\frac{I_2}{I_2^2 + 1.0}\right)} T_{ij}^{(3)} \\
R &= 2k b_{ij}^{R} \frac{\partial u_i}{\partial x_j} + D_k/P_k \left(-\frac{17.43371 D_k/C_k}{D_k/C_k^2 + 1.0} + 10.10268\right) \varepsilon
\end{aligned}
\tag{A.35}
$$

B

Predicted Flow fields

B.1. $\alpha_{PH} = 1.0$ case

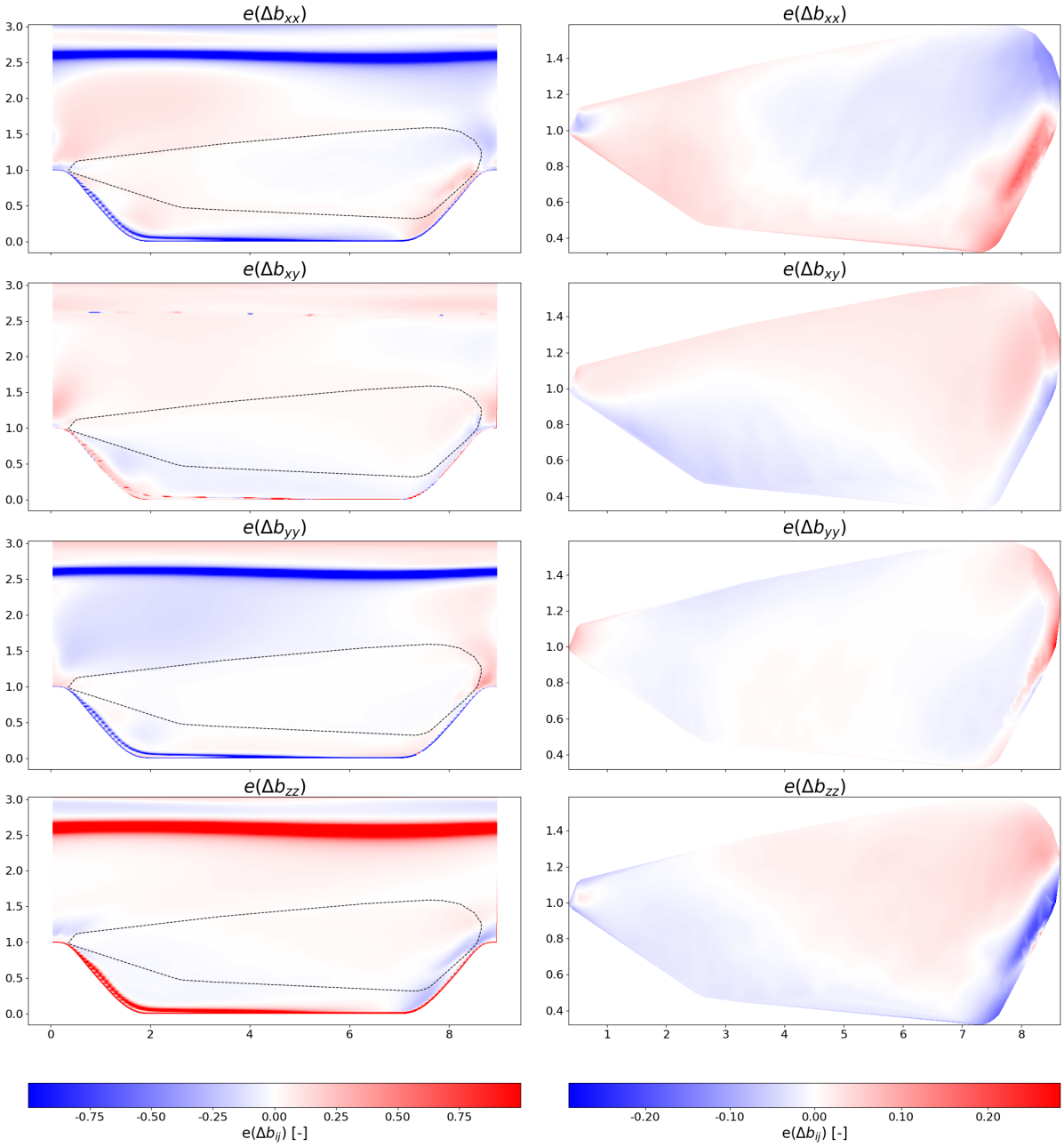
B.1.1. Δb_{ij} prediction results with three-tensor model



(a) True Δb_{ij} using the final three-tensor model.

(b) Predicted Δb_{ij} using the final three-tensor model.

Figure B.1: True vs predicted Δb_{ij} using the final three-tensor model for the $\alpha_{PH} = 1.0$ case.

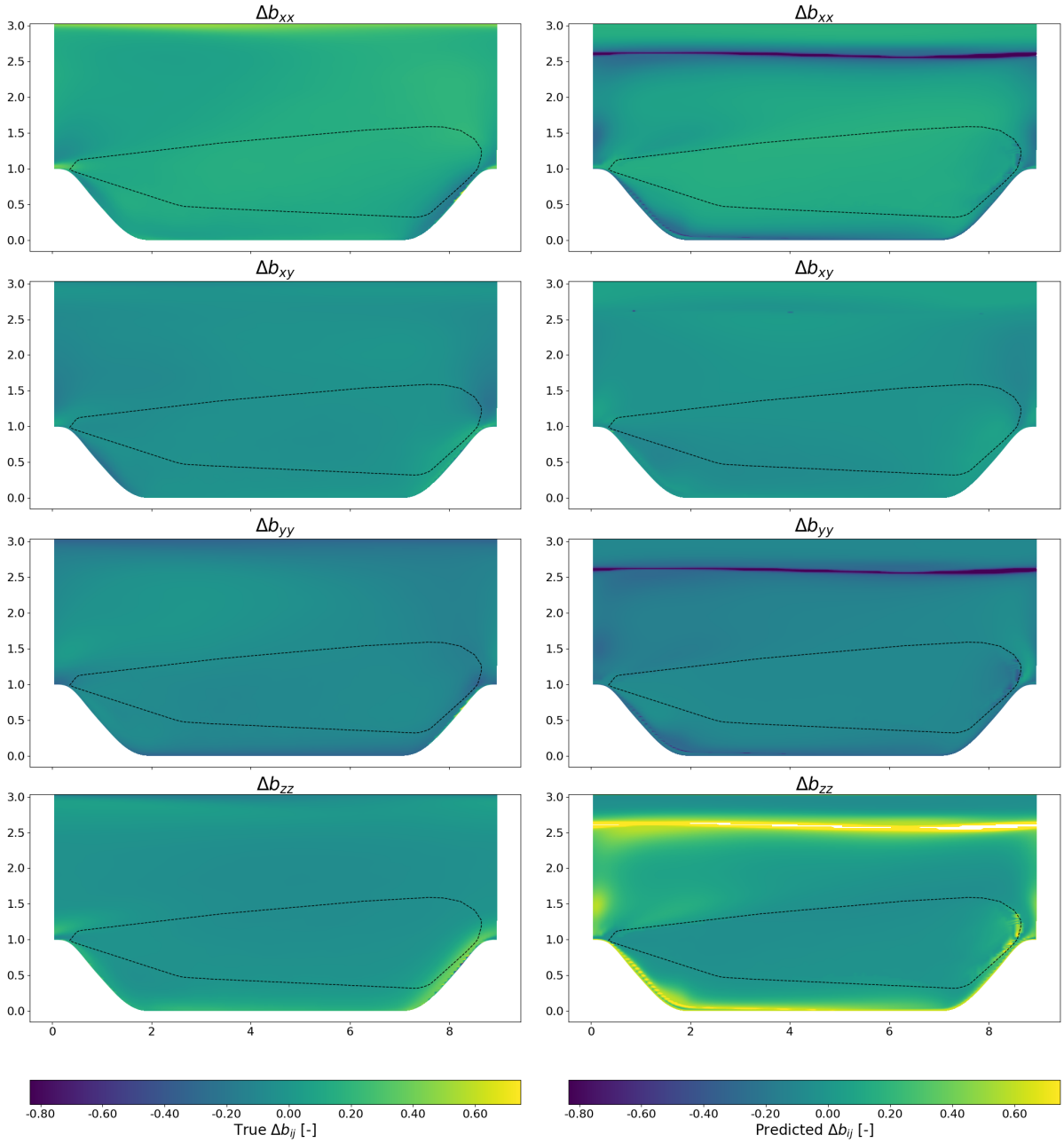


(a) Difference between the true and predicted Δb_{ij} using the final three-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using the final three-tensor model, in the RITA classified shear region.

Figure B.2: True vs predicted Δb_{ij} using the final three-tensor model for the $\alpha_{PH} = 1.0$ case.

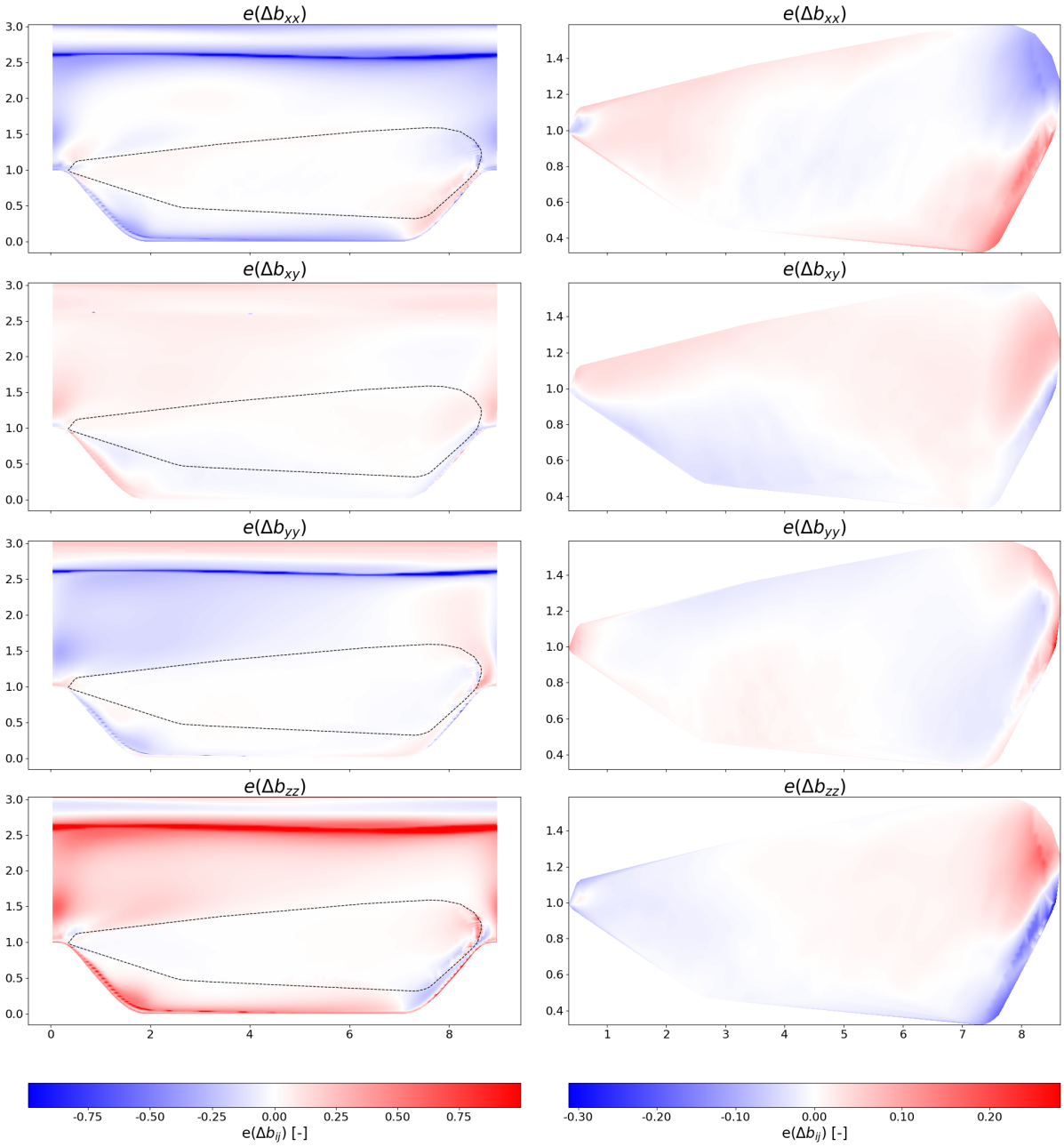
B.1.2. Δb_{ij} prediction results with four-tensor model



(a) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the RITA classified shear region.

Figure B.3: True vs predicted Δb_{ij} using the final four-tensor model for the $\alpha_{PH} = 1.0$ case.

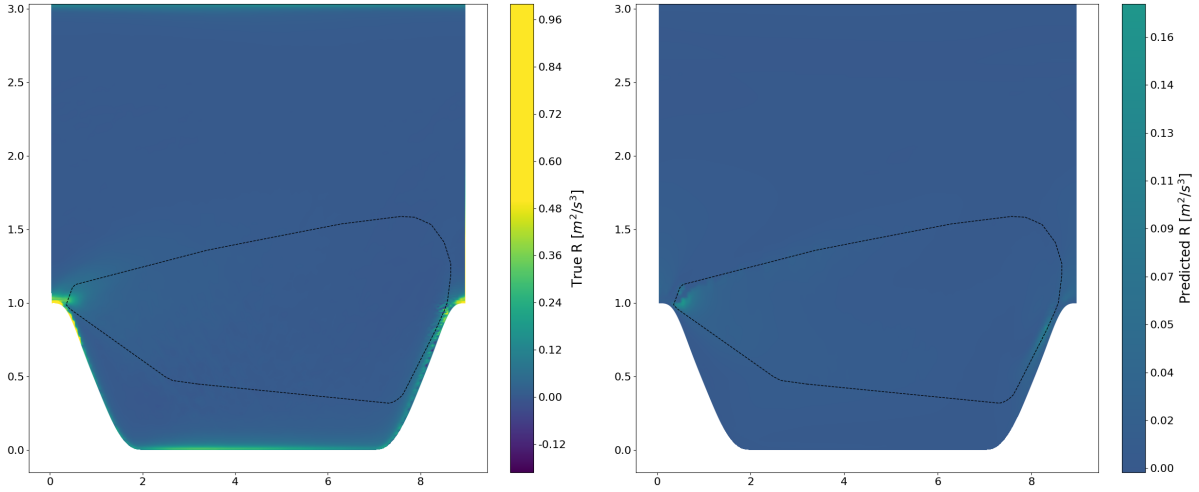


(a) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the RITA classified shear region.

Figure B.4: True vs predicted Δb_{ij} using the final four-tensor model for the $\alpha_{PH} = 1.0$ case.

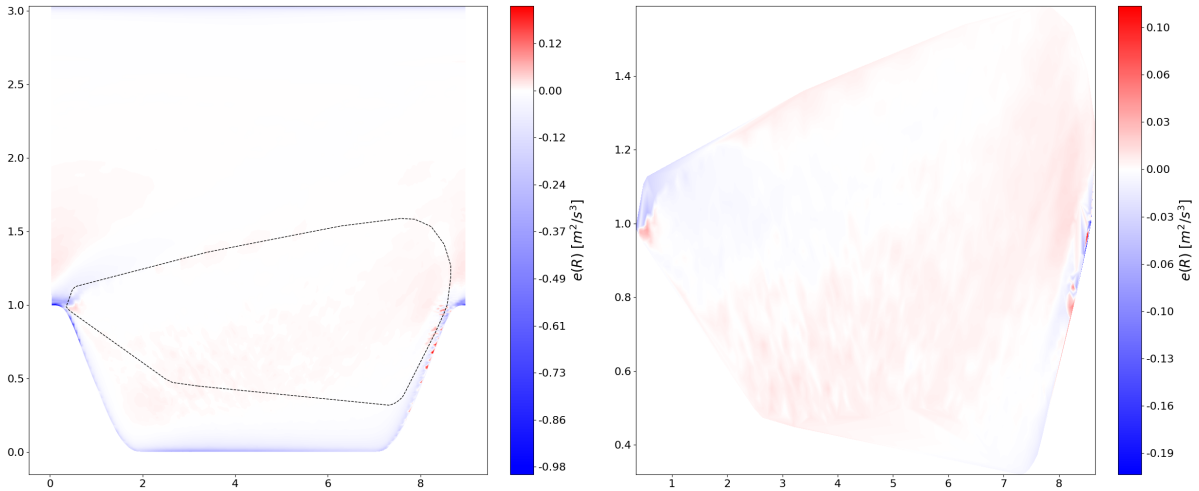
B.1.3. R correction prediction results



(a) True R field.

(b) Predicted R field using the R correction model.

Figure B.5: Plots of the true and predicted R field for α_{PH}=1.0, Lx/H=9, Ly/H=3.036.



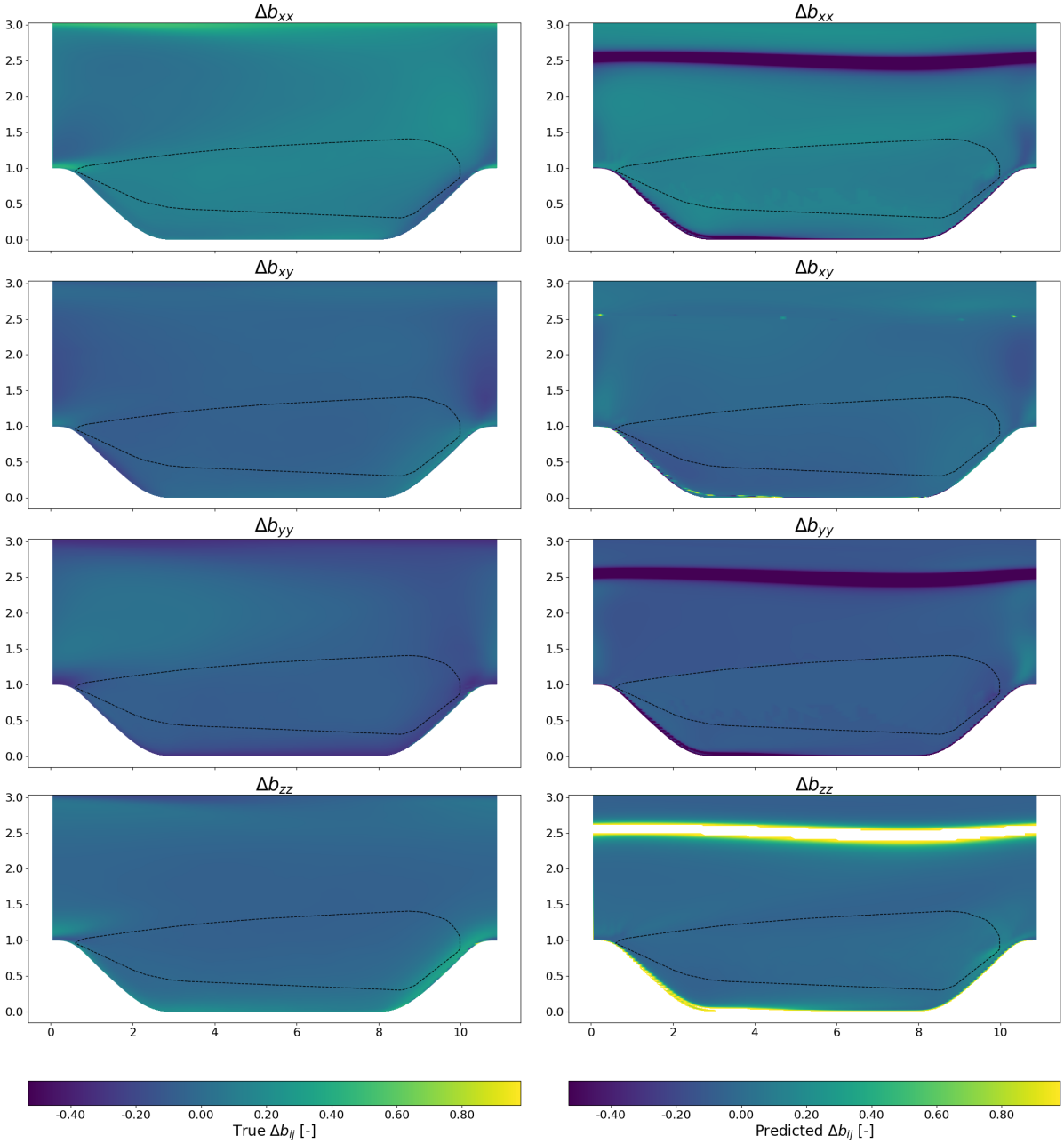
(a) Difference between the true and predicted R using the final R correction model, in the full domain.

(b) Difference between the true and predicted R using the final R correction model, in the RITA classified shear region.

Figure B.6: Difference in predicted vs true R field for α_{PH}=1.0, Lx/H=9, Ly/H=3.036. Red colours indicate overprediction, where the model predicts a higher production deficit than the true field, while blue colours indicate underprediction. White regions indicate no difference between the predicted and true fields.

B.2. $\alpha_{PH} = 1.5$ case

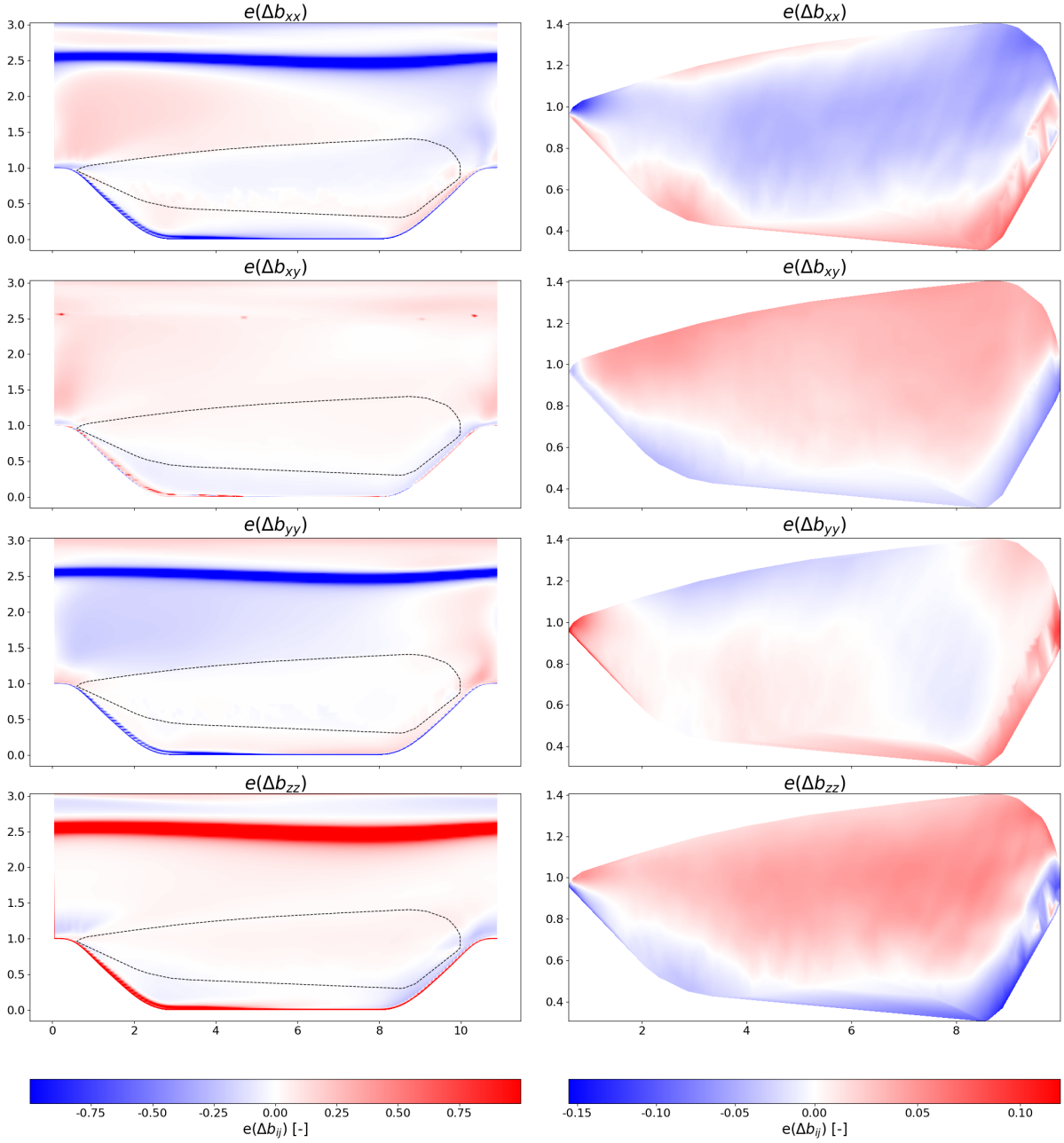
B.2.1. Δb_{ij} prediction results with three-tensor model



(a) True Δb_{ij} using the final three-tensor model.

(b) Predicted Δb_{ij} using the final three-tensor model.

Figure B.7: True vs predicted Δb_{ij} using the final three-tensor model for the $\alpha_{PH} = 1.5$ case.

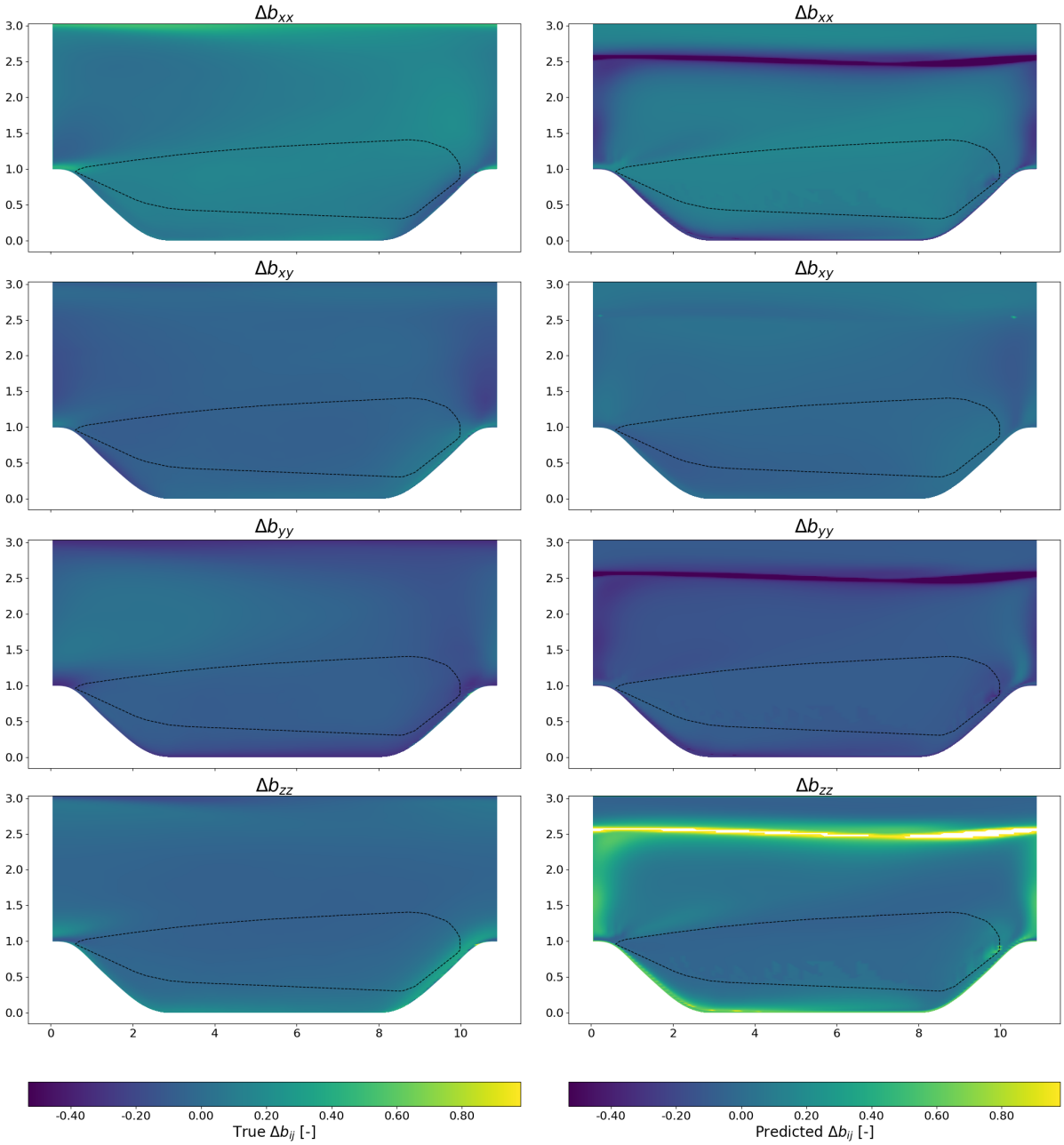


(a) Difference between the true and predicted Δb_{ij} using the final three-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using the final three-tensor model, in the RITA classified shear region.

Figure B.8: True vs predicted Δb_{ij} using the final three-tensor model for the $\alpha_{PH} = 1.5$ case.

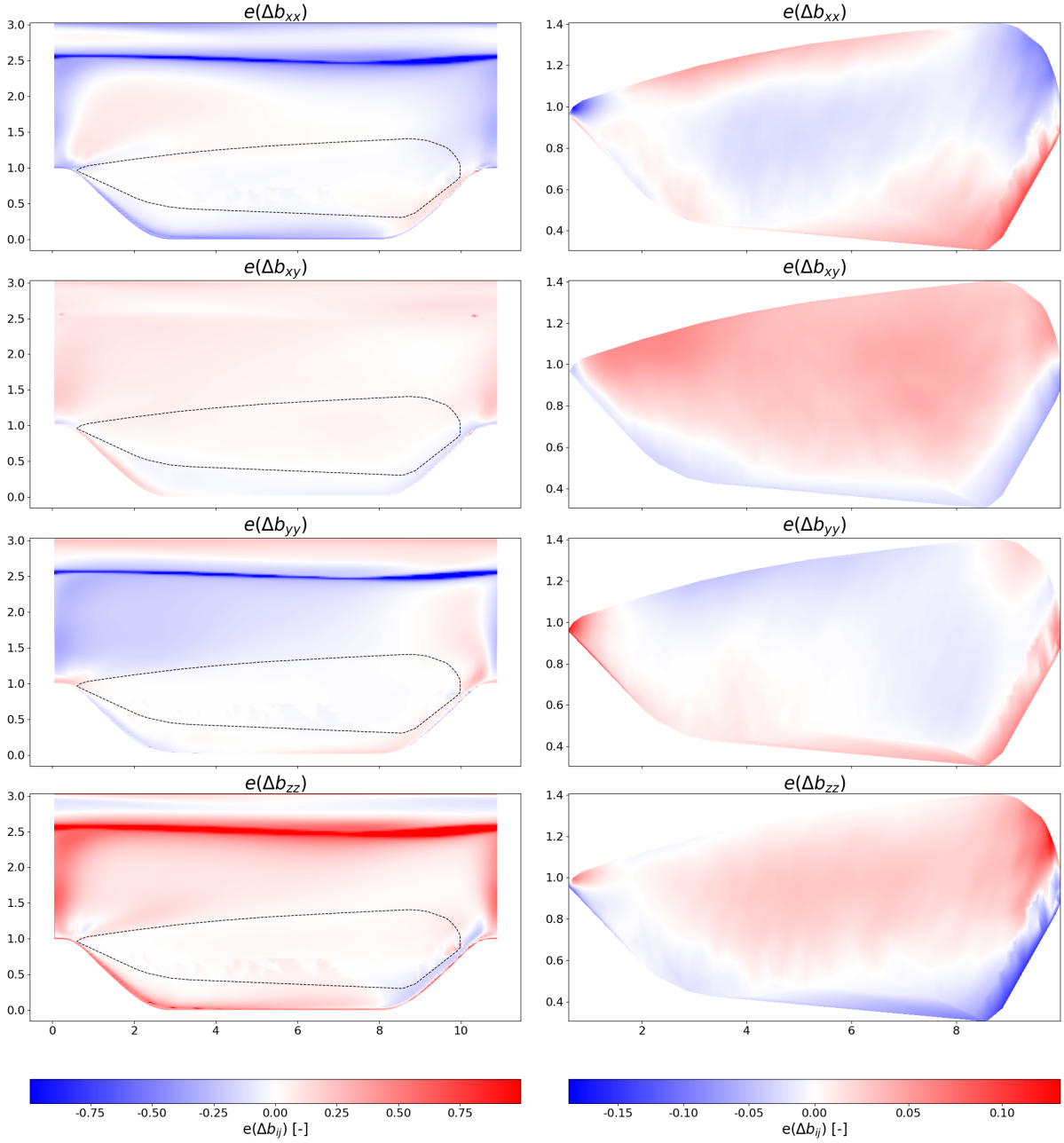
B.2.2. Δb_{ij} prediction results with four-tensor model



(a) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the RITA classified shear region.

Figure B.9: True vs predicted Δb_{ij} using the final four-tensor model for the $\alpha_{PH} = 1.5$ case.

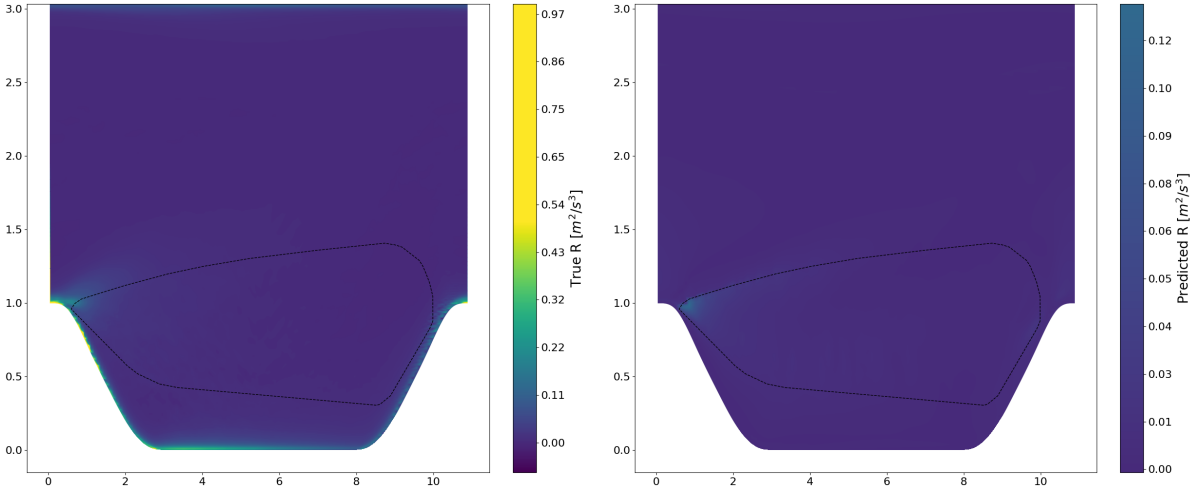


(a) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the full domain.

(b) Difference between the true and predicted Δb_{ij} using the final four-tensor model, in the RITA classified shear region.

Figure B.10: True vs predicted Δb_{ij} using the final four-tensor model for the $\alpha_{PH} = 1.5$ case.

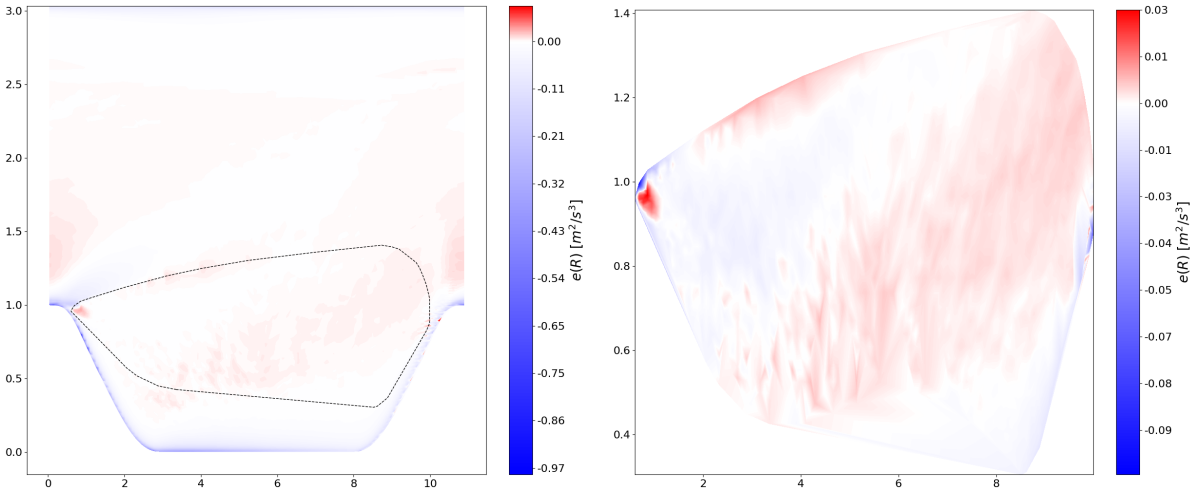
B.2.3. R correction prediction results



(a) True R field.

(b) Predicted R field using the R correction model.

Figure B.11: Plots of the true and predicted R field for α_{PH}=1.5, Lx/H=10.929, Ly/H=3.036.



(a) Difference between the true and predicted R using the final R correction model, in the full domain.

(b) Difference between the true and predicted R using the final R correction model, in the RITA classified shear region.

Figure B.12: Difference in predicted vs true R field for α_{PH}=1.5, Lx/H=10.929, Ly/H=3.036. Red colours indicate overprediction, where the model predicts a higher production deficit than the true field, while blue colours indicate underprediction. White regions indicate no difference between the predicted and true fields.

C

Basis tensors

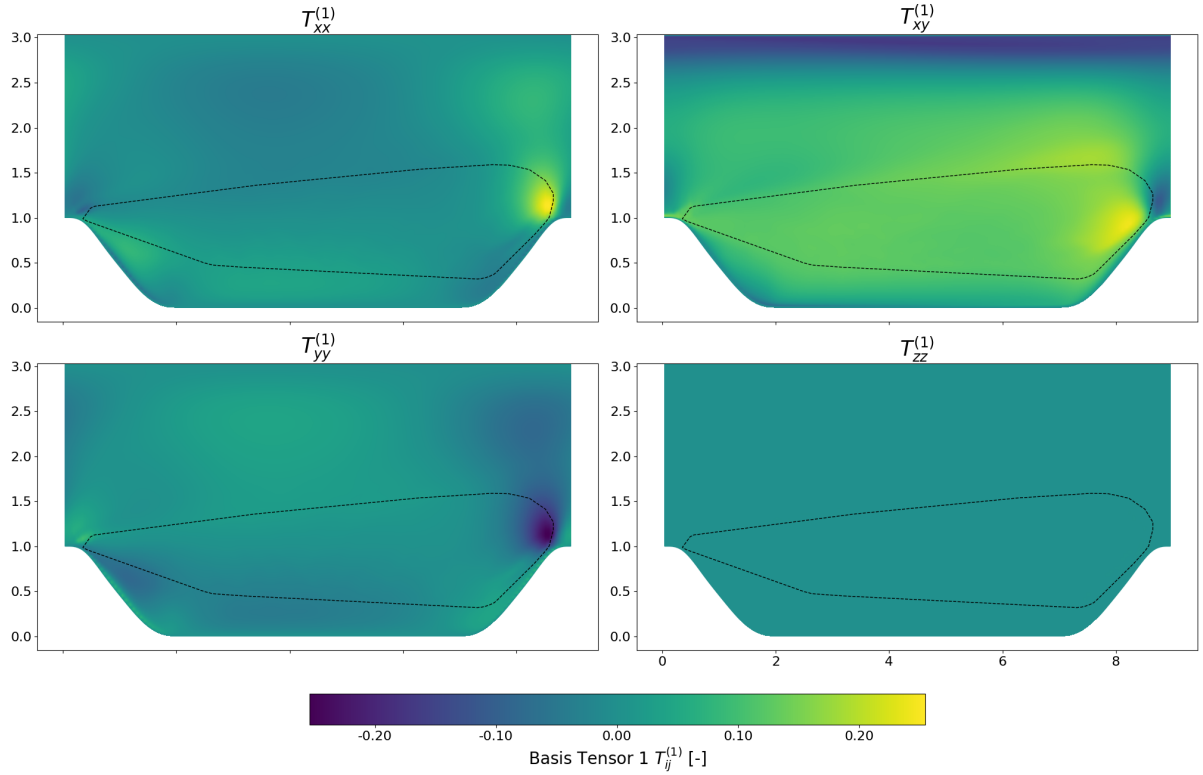


Figure C.1: First Pope basis tensor for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$.

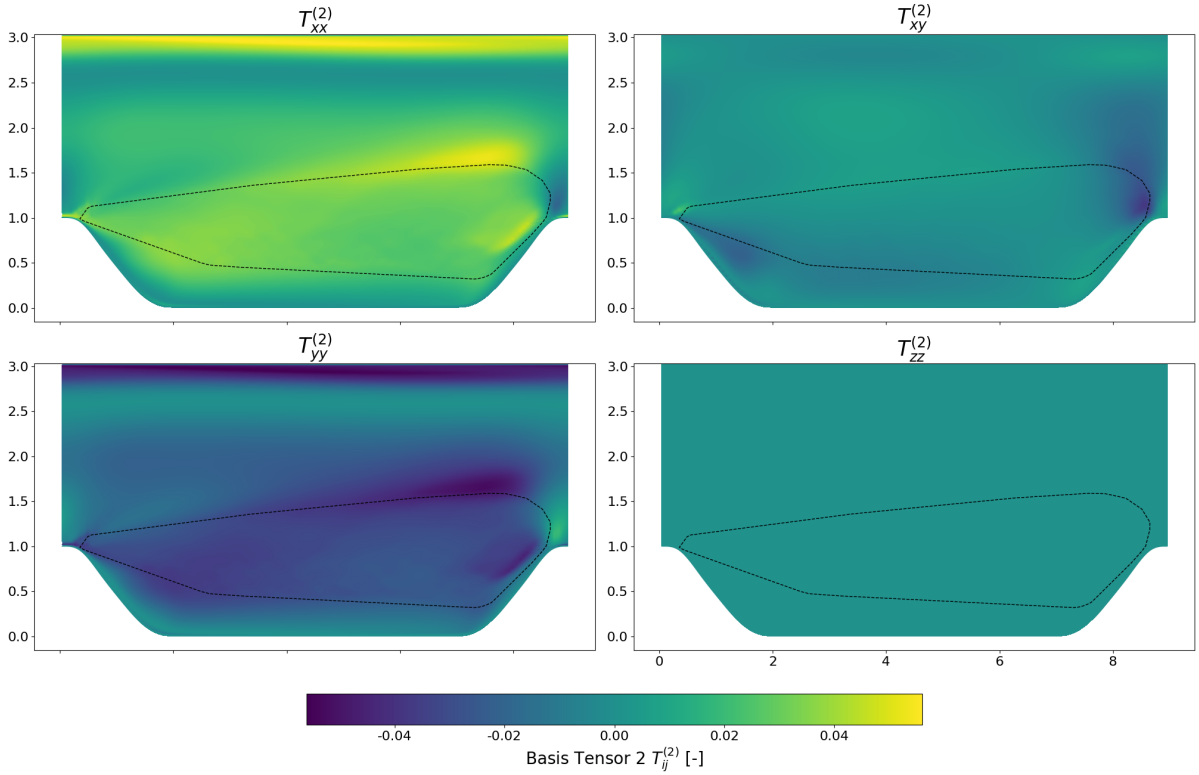


Figure C.2: Second Pope basis tensor for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$.

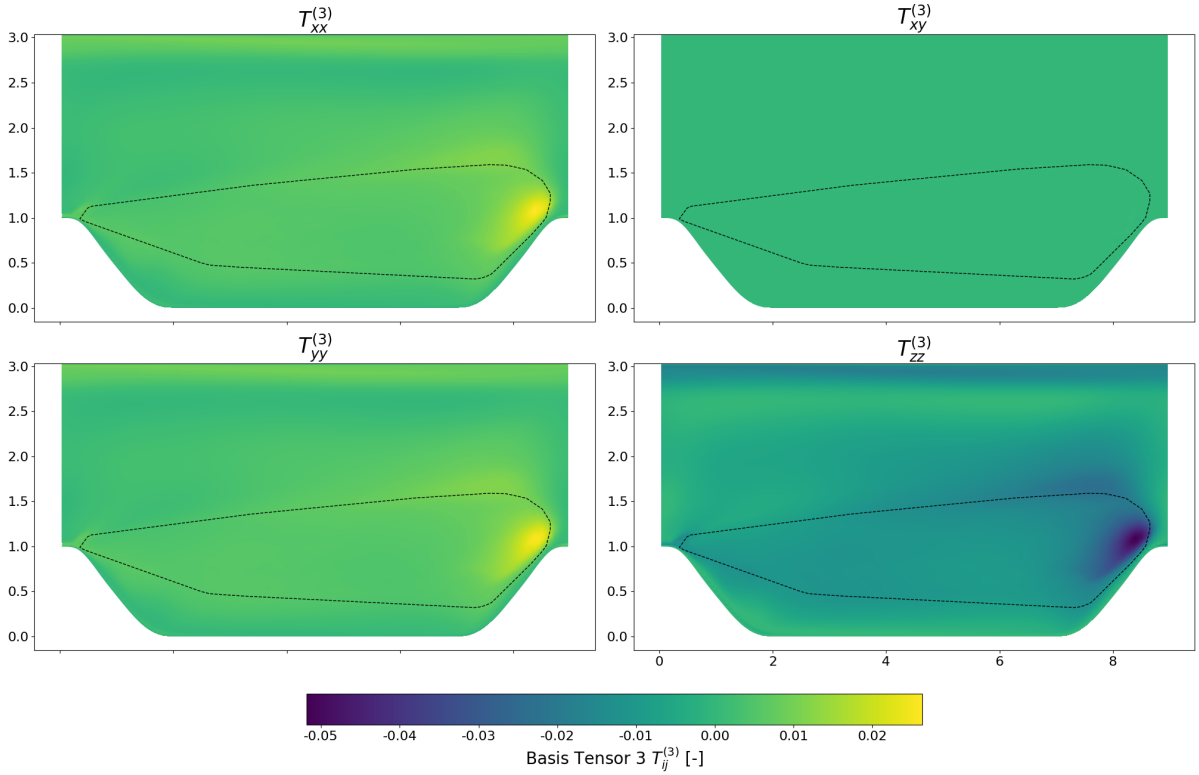


Figure C.3: Third Pope basis tensor for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$.

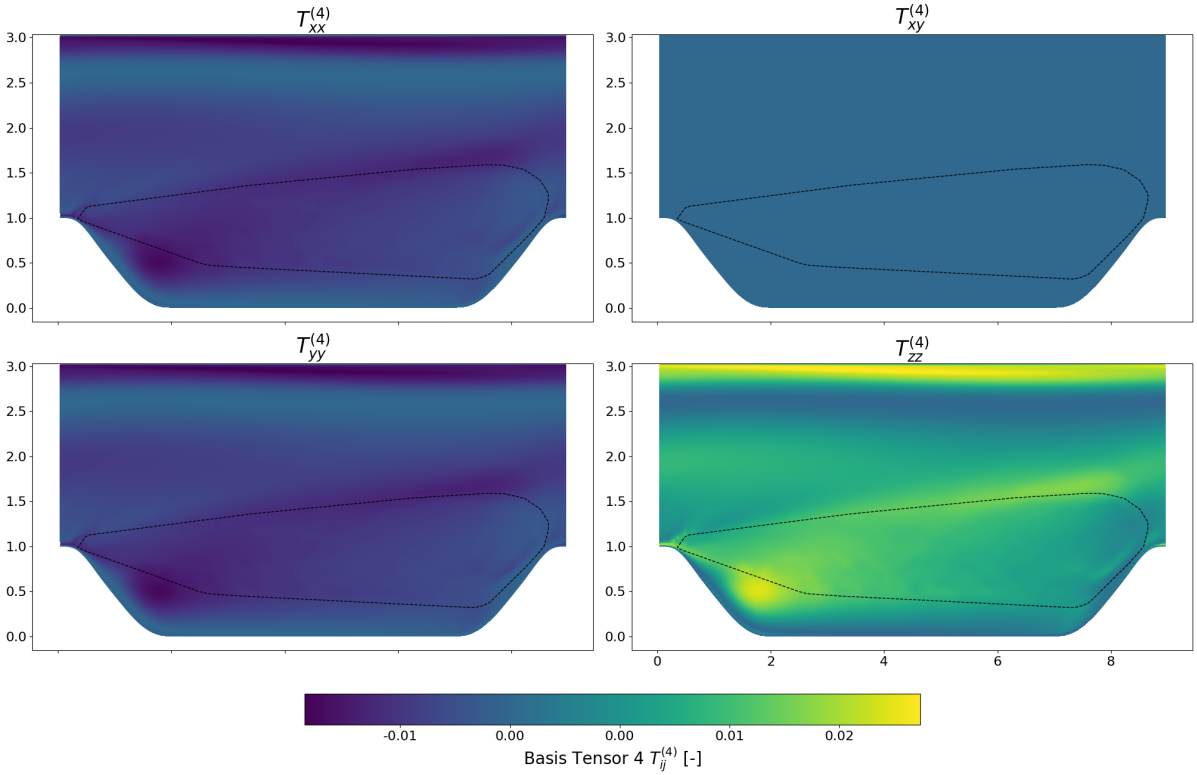


Figure C.4: Fourth Pope basis tensor for $\alpha_{PH} = 1.0$, $L_x/H = 9$, $L_y/H = 3.036$.

D

Counterfactual Multi-Agent Policy Gradients (COMA)

Counterfactual Multi-Agent Policy Gradients (COMA) is a multi-agent reinforcement learning algorithm that was researched, but there was not enough time to implement it in this thesis. This appendix provides a detailed explanation of the COMA algorithm, its motivation, and how it addresses the credit assignment problem in cooperative multi-agent settings.

In cooperative multi-agent reinforcement learning, all agents typically share a single global reward signal. This poses a fundamental challenge known as the credit assignment problem, where it becomes difficult for each agent to determine how its individual actions contribute to the overall team performance. When the joint reward depends on the combined actions of all agents, a single agent cannot easily infer whether a change in its behaviour would have improved the outcome. This results in high variance and unstable learning signals when applying standard policy gradient methods.

One classical approach to mitigate this problem is through difference rewards by Wolpert and Tumer, 2001. The main idea behind difference rewards is to provide each agent with a shaped reward that measures its individual contribution to the team's performance. For an agent a, the difference reward D_a is defined as

$$D_a = r(s,u) - r(s,(u_{-a}, c_a)), \tag{D.1}$$

where $r(s,u)$ is the global reward obtained from the joint action u, and $r(s,(u_{-a}, c_a))$ is the reward that would have been obtained if all other agents' actions u_{-a} remained the same, but agent a had instead taken a default action c_a. This difference measures how much agent a's actual action improved (or worsened) the team reward relative to a counterfactual scenario. In theory, optimising such shaped rewards leads each agent to maximise its true contribution to the team objective, encouraging cooperative and coordinated behaviour.

Although difference rewards provide a principled way to assign credit, they are rarely practical to compute in complex environments. Estimating the counterfactual reward $r(s,(u_{-a}, c_a))$ would require repeatedly simulating the environment under alternate actions for every agent at every step, which is computationally prohibitive. Moreover, the choice of the default action c_a is often arbitrary and task-dependent, making the method difficult to generalise.

The COMA algorithm Foerster et al., 2018 provides a practical, differentiable approximation to difference rewards using a centralised critic. COMA adopts a centralised training with decentralised execution (CTDE) setup. Each agent maintains a decentralised policy $\pi_a(u_a|\tau_a)$ that depends only on its local observation history τ_a, but during training, a centralised critic $Q(s,\tau,u)$ is used to evaluate the joint action given the full state s and the joint observation histories τ. The critic is trained to estimate the expected return of the joint action, acting as a learned model of the global reward function.

COMA replaces the explicit computation of difference rewards with a learned, counterfactual advantage that compares the Q-value of the current joint action with a baseline that marginalises out the individual agent's action. For agent a, this advantage is defined in Equation D.2.

$$A_a(s, \tau, u) = Q(s, \tau, u) - \sum_{u'_a} \pi_a(u'_a|\tau_a) Q(s, \tau, (u_{-a}, u'_a)). \tag{D.2}$$

This formulation can be interpreted as a learned version of the difference reward, where the counterfactual term $\sum_{u'_a} \pi_a(u'_a|\tau_a) Q(s, \tau, (u_{-a}, u'_a))$ serves as a baseline that estimates what the team's expected return would be if agent a's action were replaced with all possible alternatives, weighted by its policy. The result is a per-agent credit assignment signal that captures how much better or worse the joint outcome was compared to this counterfactual expectation. Crucially, this baseline depends only on the centralised critic and can be computed efficiently in a single forward pass of the network, avoiding the need for additional simulations or predefined default actions.

The counterfactual advantage provides a low-variance, unbiased gradient estimate for each agent's policy. Each agent's policy parameters θ_a are updated using the gradient, shown in Equation D.3.

$$\nabla_{\theta_a} J = \mathbb{E}_\pi \left[\nabla_{\theta_a} \log \pi_a(u_a|\tau_a) \, A_a(s, \tau, u) \right], \tag{D.3}$$

Which encourages the agent to increase the likelihood of actions that contribute positively to the global outcome relative to its counterfactual baseline. Since the baseline is independent of the sampled action u_a, its expected contribution to the gradient is zero, ensuring that the update remains unbiased.

Conceptually, COMA can be viewed as an actor-critic implementation of difference rewards, where the centralised critic acts as a differentiable model that estimates how the global return would change if an individual agent's action were altered. This transforms the discrete notion of difference rewards into a continuous, learnable advantage function that can be optimised end-to-end with gradient descent. In doing so, COMA provides a scalable solution to the credit assignment problem, enabling decentralised agents to learn coordinated behaviour through centralised training.

Bibliography

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., et al. (2020). The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, *280*, 103216.

Bellman, R. (1966). Dynamic programming. *science*, *153*(3731), 34–37.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., & Parascandolo, G. (2021). Neural symbolic regression that scales. *International Conference on Machine Learning*, 936–945.

Boussinesq, J. (1877). *Essay on the theory of running water*. National Printing.

Buchanan, T., Lăcătuş, M., West, A., & Dwight, R. P. (2025). Data-driven rans closures using a relative importance term analysis based classifier for 2d and 3d separated flows. *arXiv preprint arXiv:2504.06758*.

Duraisamy, K., Iaccarino, G., & Xiao, H. (2019). Turbulence modeling in the age of data. *Annual review of fluid mechanics*, *51*(1), 357–377.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. *Proceedings of the AAAI conference on artificial intelligence*, *32*(1).

Hemmes, J. (2022). Data-driven turbulence modelling of algebraic reynolds-stress models using deep symbolic regression.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hoefnagel, K. (2023). Multi-flow generalization in data-driven turbulence modeling: An exploratory study.

Huang, P., Bradshaw, P., & Coakley, T. J. (1992). *Assessment of closure coefficients for compressible-flow turbulence models* (tech. rep.).

Jones, W. P., & Launder, B. E. (1972). The prediction of laminarization with a two-equation model of turbulence. *International journal of heat and mass transfer*, *15*(2), 301–314.

Kaandorp, M. L., & Dwight, R. P. (2020). Data-driven modelling of the reynolds stress tensor using random forests with invariance. *Computers & Fluids*, *202*, 104497.

Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, *12*.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, *4*(2), 87–112.

Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., et al. (2020). Google research football: A novel reinforcement learning environment. *Proceedings of the AAAI conference on artificial intelligence*, *34*(04), 4501–4510.

Lăcătuş, M. (2024). Improving data-driven rans turbulence modelling for separated flow scenarios.

Landajuela, M., Lee, C. S., Yang, J., Glatt, R., Santiago, C. P., Aravena, I., Mundhenk, T., Mulcahy, G., & Petersen, B. K. (2022). A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, *35*, 33985–33998.

Landajuela, M., Petersen, B. K., Kim, S. K., Santiago, C. P., Glatt, R., Mundhenk, T. N., Pettit, J. F., & Faissol, D. M. (2021). Improving exploration in policy gradient search: Application to symbolic optimization. *arXiv preprint arXiv:2107.09158*.

Ling, J., Kurzawski, A., & Templeton, J. (2016). Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, *807*, 155–166.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, *30*.

Menter, F. R. (1994). Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA journal*, *32*(8), 1598–1605.

Menter, F. R., Kuntz, M., Langtry, R., et al. (2003). Ten years of industrial experience with the sst turbulence model. *Turbulence, heat and mass transfer*, *4*(1), 625–632.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, 1928–1937.

Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., & Petersen, B. K. (2021). Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*.

Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., & Kim, J. T. (2019). Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*.

Pope, S. B. (1975). A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, *72*(2), 331–340.

Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2020). Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, *21*(178), 1–51.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (tech. rep.).

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., & Whiteson, S. (2019). The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

Schmelzer, M., Dwight, R. P., & Cinnella, P. (2020). Discovery of algebraic reynolds-stress models using sparse symbolic regression. *Flow, Turbulence and Combustion*, *104*, 579–603.

Schmidt, M., & Lipson, H. (2009). Distilling free-form natural laws from experimental data. *science*, *324*(5923), 81–85.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning*, 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Siddiqui, A. A. (2025). Data-driven correction fields for turbulence modeling: A priori and a posteriori study.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*(1), 9–44.

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction* (Vol. 1). MIT press Cambridge.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, *12*.

Tamar, A., Glassner, Y., & Mannor, S. (2015). Policy gradients beyond expectations: Conditional value-at-risk.

Tang, H., Wang, Y., Wang, T., & Tian, L. (2023). Discovering explicit reynolds-averaged turbulence closures for turbulent separated flows through deep learning-based symbolic regression with non-linear corrections. *Physics of Fluids*, *35*(2).

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., & Tegmark, M. (2020). Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 4860–4871, Vol. 33). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/33a854e247155d590883b93bca53848a-Paper.pdf

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., & Galván-López, E. (2011). Semantically-based crossover in genetic programming: Application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, *12*(2), 91–119.

Valipour, M., You, B., Panju, M., & Ghodsi, A. (2021). Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*.

Watkins, C. J. C. H., et al. (1989). Learning from delayed rewards.

Weatheritt, J., & Sandberg, R. (2016). A novel evolutionary algorithm applied to algebraic modifications of the rans stress–strain relationship. *Journal of Computational Physics*, *325*, 22–37.

Wilcox, D. C. (1988). Reassessment of the scale-determining equation for advanced turbulence models. *AIAA journal*, *26*(11), 1299–1310.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3), 229–256.

Wolpert, D. H., & Tumer, K. (2001). Optimal payoff functions for members of collectives. *Advances in Complex Systems*, *4*(02n03), 265–279.

Xiao, H., Wu, J.-L., Laizet, S., & Duan, L. (2019). Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations. https://arxiv.org/abs/1910.01264

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., & Wu, Y. (2022). The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, *35*, 24611–24624.