Masterarbeit Nr. 487

Design and implementation of a data-driven wall function for the velocity in RANS simulations

Jihoo Kang

Issued by: Prof. Dr.-Ing. R. Radespiel
Institut für Strömungsmechanik
Head of Institute Prof: Prof. Dr.-Ing. R. Radespiel
Technische Universität Braunschweig

Supervisor: Dr.-Ing. Andre Weiner (TU Braunschweig)

Publication: 06.05.2022

Statutory Declaration in Lieu of an Oath

Hereby I, Jihoo Kang, declare that I have authored this Masterarbeit independently and without any external help and that I have not used any aids other than those indicated.

Braunschweig, 04.05.2022

Abstract

A wall function approach makes simulations at high Reynolds numbers possible to yield reliable results even in the case of coarser meshes. However, it is difficult for the approach to obtain good results if boundary conditions are not valid for the law of the wall. This is why a data-driven approach is needed. A data-driven wall function can be applied to the boundary layer conditions where the law of the wall is not applicable such as adverse pressure gradient conditions, walls with porous media, etc.

This project aims to create a machine learning (ML) wall function that can be used for linear and non-linear flow conditions from a simple 1D geometry, which does not need any direct numerical simulation (DNS) data. The data is obtained from a 1D channel at $Re_\delta = 1 \cdot 10^7$, and three ML models that have the labels of slopes at wall, slopes at cell faces, and velocity at faces in wall normal direction are trained with the maximum relative error of 16%. The trained ML models are applied to flat plate cases at $Re_x = 1 \cdot 10^7$, $3 \cdot 10^6$, and $6 \cdot 10^6$. With the correction of diffusive fluxes, the skin friction of the scenarios for the ML models is compared to that of the standard wall function at $y^+ = 0.05$, 1, 2, 5, 10, 30, 50, 100. The data-driven wall function yields a smaller confidence interval that corresponds to approximately 65% of the confidence interval for the standard wall function scenario, which means that the skin friction of the data-driven wall function is less mesh-dependent than that of the standard wall function. On the other hand, an airfoil case with a chord length of $1m$ at $Re_c = 3 \cdot 10^6$ is also investigated. The skin friction and the pressure coefficient of the ML models compare to those of the standard wall function at $y^+ = 0.05$, 1, 2, 3.5, 5, 10, 50, 100 with the correction of diffusive and convective fluxes. For the skin friction except at $y^+ = 3.5$, the data-driven wall function yields slightly the longer confidence interval that is 108.4% of the confidence interval for the standard wall function scenario. This implies that the data-driven wall function is more mesh-dependent than the standard wall function for the airfoil case.

Contents

Nomenclature

Latin Symbols

t	Time
D	Diffusive coefficient
S_Φ	Arbitrary source term
p	Pressure
V	Control volume
o	Control surface
f	Each face in a cell
F_c	All the faces in a cell
$\mathbf{n}$	Normal vector
$\mathbf{U}$	Velocity vector
$\mathbf{U}_f$	Velocity vector at each face
$\mathbf{U}_W$	Velocity vector on west side
$\mathbf{U}_E$	Velocity vector on east side
$\mathbf{U}_N$	Velocity vector on north side
$\mathbf{U}_S$	Velocity vector on south side
S_f	Surface area at each face
$\mathbf{S}_f$	Surface vector at each face
$\mathbf{S}_W$	Surface vector on west side
$\mathbf{S}_E$	Surface vector on east side
$\mathbf{S}_N$	Surface vector on north side
$\mathbf{S}_S$	Surface vector on south side
U_∞	Ambient velocity
$\tilde{U}$	Normalized velocity (0-1 scale)
U_{avg}	Integral average velocity
x	x-axis in Cartesian coordinates
y	y-axis in Cartesian coordinates
u	Velocity in x-direction
v	Velocity in y-direction
u^+	Non-dimensionalized velocity in wall units
y^+	Non-dimensionalized height in wall units
u_τ	Friction velocity
E	Constant for the law of the wall
C	Cost function
L	Loss function
x_{d_f}	d_f-th feature
y_{d_l}	d_l-th label
$\hat{y}_i$	Prediction of i-th label

$\mathbf{x}$	Feature vector
$\mathbf{y}$	Label vector
w_k	Random weight with k-th epoch
m	The number of training data sets
f_m	Mapping function
$\mathbb{R}$	Real number
z	Sum of neurons
a	Activation function
$\mathbf{a}$	Operation of sum and activation function
w	Sigmoid function
RE	Relative error
d	Diameter
Re	Reynolds number
Re_δ	Reynolds number for channel flow
Re_x	Reynolds number for flat plate
Re_c	Reynolds number for airfoil
C_f	Skin friction
C_p	Pressure coefficient

Greek Symbols

∇	Nabla operator
Φ	Arbitrary tensor property
ϕ	Mass flux
ν	Kinematic viscosity
ν_{eff}	Effective kinematic viscosity
ν_t	Turbulent kinematic viscosity
ρ	Density
Ω	Continuous flow domain
Ω_D	Discretized flow domain
Ω_c	Flow domain in one cell
$\boldsymbol{\epsilon}$	Strain tensor
$\boldsymbol{\tau}$	Deviatoric stress tensor
τ_w	Wall shear stress
κ	*Von Kármán* constant
γ_w	Exponent for blending at a wall
γ_f	Exponent for blending at the first cell face normal to a wall for diffusive fluxes
γ_ϕ	Exponent for blending at the first cell face normal to a wall for convective fluxes
μ	Mean value
σ	Standard deviation

Indices

w	Wall in general
$wall$	Wall for correction and training
f	Face in general / friction for C_f
$face$	The first cell face normal to a wall
W	West

E	East
N	North
S	South
d_f	Number of features
d_l	Number of labels
i	i-th neuron in $(l-1)$-th layer
j	j-th neuron in l-th layer
l	l-th layer
n	Total number of layers
∞	Ambient
eff	Effective
$corr$	Corrected
num	Numerical
$model$	Model
$velBlend$	Velocity blending
$ypBlend$	y^+ blending
δ	Channel
avg	Average
x	x-axis in Cartesian coordinates
y	y-axis in Cartesian coordinates
c	Chord
p	Pressure
rep	Representative
ref	Reference

Abbreviations

CFD	Computational Fluid Dynamics
ML	Machine Learning
LES	Large Eddy Simulation
$RANS$	Reynolds-Averaged Navier-Stokes
$PINN$	Physics-Informed Neural Network
$TBNN$	Tensor Based Neural Network
NN	Neural Network
FNN	Feed-forward Neural Network
CNN	Convolutional Neural Network
$WMLES$	Wall-Modeled Large Eddy Simulation
$WRLES$	Wall-Resolved Large Eddy Simulation
DNS	Direct Numerical Simulation
$UIUC$	University of Illinois at Urbana-Champaign
SA	Spalart-Allmaras
FVM	Finite Volume Method
LHS	Left-Hand Side
GD	Gradient Descent
SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
MLP	Multilayer Perceptron
$ReLU$	Rectified Linear Unit

$SIMPLE$ Semi-Implicit Method for Pressure-Linked Equation

Chapter 1

Introduction

In fluid mechanics, computational approaches are broadly employed in order to save efforts and resources for experiments. One of these approaches is generally known as computational fluid dynamics (CFD) that applies to various engineering fields such as aeronautical engineering, chemical engineering, automotive fields, etc. However, in simulations, the treatment of specific wall areas such as turbine blades and pump impellers is mostly difficult to obtain reliable results since the flows in these areas generally have high Reynolds numbers that require finer meshes. When highly resolved mesh is used, then reliable results are to be expected. Nevertheless, a problem arises again because an aspect ratio of a cell will be large when the mesh is exceedingly resolved.

One approach to mitigate this problem is introducing wall functions. A wall function is a function from the asymptotic solutions for walls in canonical flows [12], which makes CFD simulations more reliable regardless of any mesh resolutions. One paper related wall functions by *Kalitzin et al.* [5] is introduced here. The authors implemented a new wall function that uses a shifted computational domain started from a certain distance from the wall [5]. For the several turbulence models such as Spalart-Allmaras (SA), $k-\omega$, $k-g$, and v^2-f, the performance of the new wall function was investigated in flat plate cases with zero pressure gradient and recirculating flow [5]. There are three regions in a turbulent boundary layer that correspond to a viscous sublayer, a logarithmic layer, and a buffer layer [5]. According to the results, when a first cell center normal to the wall is located in the buffer layer, the wall function that is analytically approximated cannot appropriately capture the behavior of the real parameter values, whereas the wall function can support the simulation to perform well when the first cell center is in the viscous sublayer or the logarithmic layer [5]. Thus, for the buffer layer, they employed a look-up table that is based on an accurate numerical wall solution in a flat plate case with zero pressure gradient, and they found that this approach was effective [5].

However, accurate solutions for a buffer layer are not available for every flow case. Hence, another approach that is referred to as data-driven or machine learning (ML) approach needs to be introduced so as to alleviate the problem of lack of exact solutions in the buffer layer. In addition, this approach not only mitigates the problem in the buffer layer, but it also makes the wall modeling applicable to more complex types of boundary layers that do not follow the law of the wall such as a boundary layer with pressure gradients. On the other hand, there are also disadvantages of the data-driven approach compared to the classical wall function approach. If the quality of data is too low or the data is not well distributed, the training will not be performed well, which leads to obtaining an improper mapping function. Thus, the data should be well distributed before training. In addition, the data-driven approach might lead to misinterpretation of results when no physical information is involved. The results should always be compared to the physical context after training. Over the past decades, this data-driven

approach for the CFD has been developed and improved in several ways. On the one hand, ML methods are introduced in large eddy simulations (LES). There are several types of usage of ML in LES, but only the literature related to wall treatments will be mentioned here [21, 23]. On the other hand, the methods are also used in Reynolds-Averaged Navier-Stokes (RANS) simulations, and the types of usage are mainly divided into three parts. A classification of certain areas in simulations [7], a usage of physics-informed neural networks (PINN) [8, 15, 18, 17], and a usage of physics-free neural networks [14, 11].

For LES, *Yang et al.* employed a PINN that is related to the vertically integrated thin-boundary-layer equations for directly usage of the velocity and the wall height, and the eddy population density scalings to approximate wall shear stresses at a viscous sublayer and a logarithmic layer, which become the actual inputs [21]. The trained neural network (NN) was applied to a wall-modeled LES (WMLES) in the channel flow situation at a small Reynolds number, and it could well capture the behavior of the law of the wall [21]. *Zhou et al.* created two feed-forward neural networks (FNN) by using the normalized features that correspond to the wall-normal distance, near wall velocities, and pressure gradients to predict wall shear stresses for the normal and the streamwise directions [23]. The model was trained from the wall-resolved LES (WRLES) data in a periodic hill case [23]. Consequently, the trained model is used to predict wall shear stresses for *a priori* test, and velocity and pressure fields for *a posteriori* test [23].

For the first type of usage of ML in RANS, *Ling and Templeton* trained several ML algorithms such as support vector machines, Adaboost decision trees, and random forests for classification based on the direct numerical simulation (DNS) and the LES data of several canonical flows in order to distinguish where the high uncertainty area is [7]. For a classifier, three eddy viscosity assumptions such as the isotropy of the eddy viscosity, the linearity of the Boussinesq hypothesis, and the non-negativity of the eddy viscosity were applied to train, which would be the output values from the non-dimensional inputs [7]. If each assumption cannot be applied or valid at one point after the simulation, the classifier classifies the point as uncertain [7]. The ML model could predict the high uncertainty area except the region near the wall, which implies that the wall treatment is difficult due to lack of the prediction accuracy for observation of uncertainty areas [7].

The second type is about PINN where the information of physics is introduced as a part of input features for training ML in this case. *Ling et al.* trained a deep NN to predict the Reynolds stress anisotropy tensor from high-fidelity data [8]. The model is called tensor based neural network (TBNN) which consists of two networks, one for five tensor invariants that correspond to scalar functions of two tensors $\mathbf{S}$ (strain rate tensor) and $\mathbf{R}$ (rotation rate tensor), and another one for ten isotropic basis tensors that are linear combinations of $\mathbf{S}$ and $\mathbf{R}$ [8]. The inputs for the tensor part are these ten tensors, and for the scalar part are the five invariants that are the trace of those combinations [8]. The trained model performed significantly better than the typical RANS models without ML in order to predict the anisotropy tensor (*a priori*) and find the velocity field (*a posteriori*) [8]. *Tian et al.* made a similar PINN which also corresponds to TBNN to predict the non-local pressure Hessian term in a velocity gradient tensor [15]. The model was trained from DNS data, and it turned out that the performance of the model was better than the model without applying physics information [15]. The other physics-informed ML structure that is of random forests was used to predict the Reynolds stresses by formulation of the function of discrepancies of Reynolds stresses between DNS data and a RANS simulation by *Wang et al.* [18]. Here, the automatically invariant inputs of RANS were employed that were proposed in the other literature [8], and they were set by the domain knowledge related to physical equation or formulation of fluid mechanics such as turbulence intensity, wall distance based Reynolds number, pressure gradient along streamline, and so on [18]. In the test phase, the fixed discrepancy values from the training phase were used to modify the baseline RANS simulations to target the outputs of the baseline simulations to the trained terms [18]. *Volpiani*

et al. made a new volume forcing term in the Boussinesq hypothesis, and calculated this term by a data-assimilation method from high-fidelity data [17]. The trained model was used in the case of a periodic hill at different Reynolds numbers and geometries that always contains flow separation and vortices [17]. The model precisely predicted the mean velocity fields in terms of interpolation as well as extrapolation compared to the typical RANS model [17].

The third type is the ML models that do not use any domain knowledge of fluid mechanics to form input features. *Thuerey et al.* trained a convolutional neural network (CNN) model to find velocity and pressure field in airfoil flow situations from the data obtained from RANS simulations with the airfoil shapes from University of Illinois at Urbana-Champaign (UIUC) [14]. They made inputs in three groups as directly used input, normalized input, and mean pressure subtracted input, and then compared results for various network sizes [14]. The results from the trained CNN were very similar to the original RANS solutions [14]. Meanwhile, *Maulik et al.* created an ML model that can predict eddy viscosity by the surrogate modeling in the backward-facing step case [11]. The inputs are the location and the velocity field from the data of a simulation with the SA turbulence model, and then the eddy viscosity is the output [11]. The trained model showed similar results to the original SA model for the test cases in the paper to predict the velocity, the eddy viscosity, and the skin friction [11]. They trained the model for the two equation models as well, and the model well predicted the related parameters [11].

As seen above, the trend of literature is as follows. First of all, there are more papers about wall treatment for LES, while the specific terms in equations are considered for RANS instead of wall boundary areas. Secondly, plenty of papers deal with PINNs that mostly improve the Reynolds stress tensor and seldom any terms in turbulence models, whereas few papers have mentioned physics-free NN models. Thirdly, it is not obvious if the trained models were iteratively used in the actual CFD solver to improve the simulation results. If the outputs from a trained model are fixed and inserted once into a solver, the simulation can mimic the performance of the generic solvers, but might not be able to improve the results.

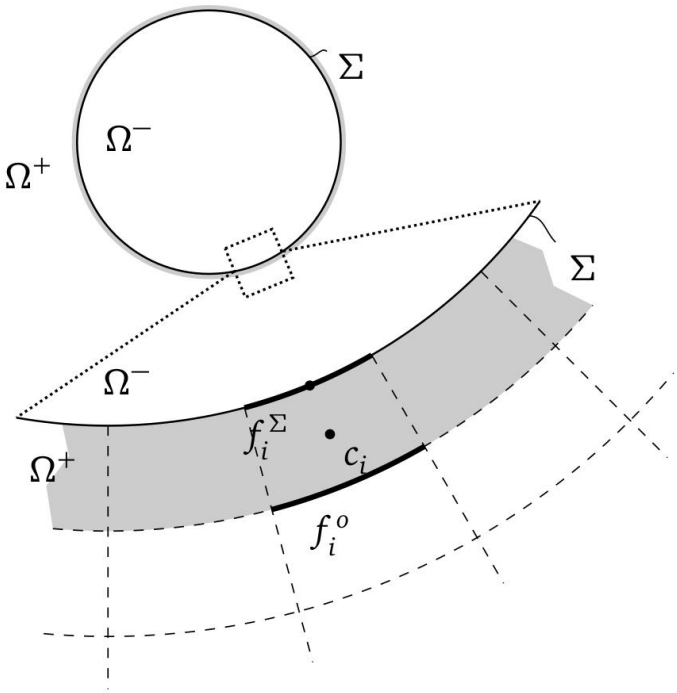

Figure 1.1: Sketch of a bubble interface [19]

In this work, the above three aspects will be considered. Since the literature related to RANS rarely focuses on the wall area itself, a wall treatment method in RANS, particularly for the SA model, will be investigated here. In addition, a physics-free NN will be introduced in which the output labels are for the later use of the solver correction. Instead of using PINNs, the NN simply maps the slopes at a wall and faces, and the velocity at faces in order to correct the diffusive and the convective fluxes in the Navier-Stokes equation. Furthermore, the trained ML model in this work will be applied to a CFD solver in OpenFOAM, which not only give the labels to the solver, but also receive the information of the features from the solver iteratively. Similar works can be found in two papers by *Weiner et al.* [20] and *Weiner* [19]. For both works, a combined ML model was applied to investigate the phenomena in bubble interfaces [19, 20]. As shown in Figure 1.1, a sketch of a bubble interface is depicted [19]. The profile of concentration boundary layers of a bubble was investigated, then the diffusive and the convective fluxes of the concentration field were corrected by the ratio of the ML model slope to the numerical slope at an interface f_i^{Σ} and the first cell face normal to the interface f_i^{o} [19, 20]. Similarly, a velocity profile of a certain fluid will be corrected in the current work instead of the concentration profile. A detailed explanation will be demonstrated at a later stage.

In Section 2, discretization methods for the momentum equation, rudimentary theories of wall functions, and the data-driven approach will be illustrated. In Section 3, modification of solver and flux correction methods will be explained. Afterward, extraction of data and mapping of labels by the NNs in a 1D channel case will be demonstrated in Section 4. Subsequently, the trained NNs will be applied in a 2D flat plate case to improve the capturing ability of the reference skin friction values at the wall in Section 5. The case is convection-dominated, and thus only diffusive fluxes will be corrected at a wall and the first cell face normal to the wall. In Section 6, the trained NNs will be applied in a NACA-0012 profile setting for the identical purpose. Convective fluxes will be involved in the section this time. In Section 7, the NN models will be tested at various Reynolds number settings for the model to be investigated if it can comprehensively capture the behavior of the actual turbulent phenomena near the wall. Finally, discussion and summary of this work will be mentioned in Section 8 and Section 9.

Chapter 2

Theoritical Background

2.1 Discretization of Momentum Equation

In this section, discretization methods of a momentum equation will be investigated. For a starting point, a general transport equation [10] is given by

$$\frac{\partial \Phi}{\partial t} + \nabla \cdot (\mathbf{U}\Phi) - \nabla \cdot (D\nabla\Phi) = S_\Phi, \tag{2.1}$$

where $\mathbf{U}$ is the given velocity, Φ is a scalar, vector, or tensor property, D is the diffusion coefficient, and S_Φ is a source term. If Φ is a vector or tensor value, a dyadic operator is added between $\mathbf{U}$ and Φ. For the momentum equation of incompressible flows to be formed, Φ is substituted for the velocity vector field $\mathbf{U}$ as follows

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{U} \otimes \mathbf{U}) - \nabla \cdot (\nu\nabla\mathbf{U}) = -\frac{\nabla p}{\rho}, \tag{2.2}$$

where ν is the kinematic viscosity, and the source term is replaced with the pressure gradient term without volume force. In this work, the main simulations will use steady-state solvers, and thus the temporal term disappears as follows

$$\nabla \cdot (\mathbf{U} \otimes \mathbf{U}) - \nabla \cdot (\nu\nabla\mathbf{U}) = -\frac{\nabla p}{\rho}. \tag{2.3}$$

Figure 2.1 shows the conformation of a general flow domain that is discretized [10]. A continuous flow domain Ω can be approximated as a discretized domain Ω_D. Ω_D consists of discretized cells, and one of the cells is referred to as Ω_c where the cell centered value in Ω_c corresponds to $\mathbf{x}_c$. With this notation, a finite volume method (FVM) is introduced by integrating over the cell Ω_c as follows

$$\int_{\Omega_c} \nabla \cdot (\mathbf{U} \otimes \mathbf{U}) dV - \int_{\Omega_c} \nabla \cdot (\nu\nabla\mathbf{U}) dV = -\int_{\Omega_c} \frac{\nabla p}{\rho} dV. \tag{2.4}$$

The terms on the left-hand side (LHS) of Equation (2.4) can be changed to surface integrals by

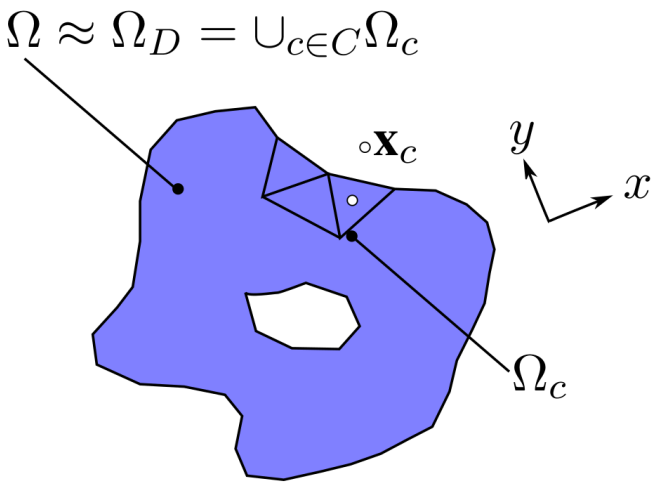

Figure 2.1: General discretized flow domain [10]

the divergence theorem as follows

$$\int_{\partial\Omega_c} (\mathbf{U} \otimes \mathbf{U}) \cdot \mathbf{n} do - \int_{\partial\Omega_c} (\nu \nabla \mathbf{U}) \cdot \mathbf{n} do = - \int_{\Omega_c} \frac{\nabla p}{\rho} dV. \tag{2.5}$$

The first term on the LHS of Equation (2.5) is called a convective term of the momentum equation, whereas the second term on the LHS corresponds to a diffusive term. The convective term and the diffusive term can be discretized as follows

$$\sum_{f \in F_c} (\mathbf{U}_f \otimes \mathbf{U}_f) \cdot \mathbf{S}_f - \sum_{f \in F_c} (\nu \nabla \mathbf{U}_f) \cdot \mathbf{S}_f = - \int_{\Omega_c} \frac{\nabla p}{\rho} dV, \tag{2.6}$$

where the subscript f means each face in the cell Ω_c, F_c corresponds to all the faces in the cell, and $\mathbf{S}_f$ is the surface vector that is made by multiplication of the normal vector $\mathbf{n}$ and the surface area at the face S_f.

2.2 Wall Functions

As mentioned in the introduction part, a wall function is a fixed profile of parameters from the asymptotic solutions [12] to capture the proper behavior in simulations regardless of mesh resolution. In this section, several basic theories related to wall functions will be investigated.

2.2.1 Law of the Wall

When the velocity and the height of cells are non-dimensionalized, turbulent boundary layers without pressure gradient consist of a viscous sublayer and a logarithmic layer. The non-dimensionalized units are known as wall units. In the viscous sublayer, the function [13] is given by

$$u^+ = y^+, \tag{2.7}$$

where

$$y^+ = \frac{yu_\tau}{\nu} \tag{2.8}$$
$$u^+ = \frac{u}{u_\tau} \tag{2.9}$$
$$u_\tau = \sqrt{\frac{\tau_w}{\rho}}, \tag{2.10}$$

for $y^+ \leq 5$.

On the other hand, the solution [3] in the logarithmic layer is given by

$$u^+ = \frac{1}{\kappa} \ln (Ey^+), \tag{2.11}$$

for $y^+ \geq 30$, where $E = 5.0$ and $\kappa = 0.41$.

The region between the viscous sublayer and the logarithmic layer is called a buffer layer located at $5 < y^+ < 30$, but there is no particular law for this area, which might lead to unexpected behavior in simulations. Therefore, several mitigation methods are available, and one of those methods is creating a function that can capture all the regions.

2.2.2 Spalding's Function

Spalding's function is a function that can well fit the empirical velocity distribution and capture the law of the wall in both of a viscous sublayer and a logarithmic layer with only one equation [13] as follows

$$y^+ = u^+ + 0.1108[e^{0.4u^+} - 1 - 0.4u^+ - \frac{1}{2}(0.4u^+)^2 - \frac{1}{6}(0.4u^+)^3 - \frac{1}{24}(0.4u^+)^4], \tag{2.12}$$

where the fourth order term can be omitted. This original version can be modified as a new equation[1] by using κ and E as follows

$$y^+ = u^+ + \frac{1}{E}[e^{\kappa u^+} - 1 - \kappa u^+ - \frac{1}{2}(\kappa u^+)^2 - \frac{1}{6}(\kappa u^+)^3]. \tag{2.13}$$

In addition, this function mitigates unstable behavior in the buffer layer as well because the value in that area is determined by the function. Figure 2.2 demonstrates how Spalding's function shapes in wall units. When this function is applied as a wall function in OpenFOAM, velocity fields for coarser meshes can also be calculated. In this work, Spalding's function will be the reference function for correcting convective and diffusive fluxes to calculate the effective viscosity ν_{eff}. A detailed explanation of implementation methods will be discussed in Section 3.

2.2.3 Wall Shear Stresses

A method to investigate turbulent flow behavior near a wall is finding shear stresses at the wall. Therefore, in this subsection, a derivation of wall shear stresses will be demonstrated. As a starting point, Equation (2.3) is used here. This equation can be compared to the conservation

[1] `https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-wall-turbulence-nutUSpaldingWallFunction.html`

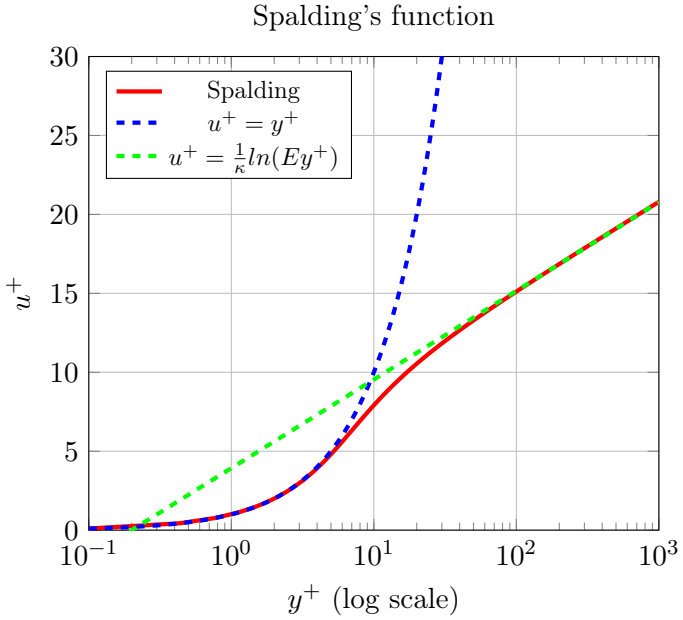

Figure 2.2: Spalding's function plot

form of the Cauchy momentum equation[2] for incompressible flows without time derivative and volume force, which is given by

$$\nabla \cdot (\mathbf{U} \otimes \mathbf{U}) - \nabla \cdot \boldsymbol{\tau} = -\frac{\nabla p}{\rho}. \tag{2.14}$$

Assuming a Newtonian fluid, the following relation

$$\nabla \cdot \boldsymbol{\tau} = \nabla \cdot (\nu \nabla \mathbf{U}) \tag{2.15}$$

holds. The term $\nabla \cdot \boldsymbol{\tau}$ that is a divergence of the deviatoric tensor [2] should be investigated, since shear stresses are caused by momentum diffusion. For this term to be investigated, the following constitutive relation for incompressible flows is to be used

$$\boldsymbol{\tau} = 2\nu\boldsymbol{\epsilon}, \tag{2.16}$$

where

$$\boldsymbol{\epsilon} = \frac{1}{2}(\nabla \mathbf{U} + \nabla \mathbf{U}^T). \tag{2.17}$$

When Equation (2.17) is inserted into Equation (2.16), the result is given by

$$\boldsymbol{\tau} = \nu(\nabla \mathbf{U} + \nabla \mathbf{U}^T). \tag{2.18}$$

Subsequently, this shear stress term can be expressed as a matrix form in a 2D space as follows

$$\boldsymbol{\tau} = \nu \begin{bmatrix} 2\partial_x u & \partial_x v + \partial_y u \\ \partial_x v + \partial_y u & 2\partial_y v \end{bmatrix}, \tag{2.19}$$

[2] `https://en.wikipedia.org/wiki/Navier-Stokes_equations`

where

$$\mathbf{U} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.20}$$

$$\mathbf{n} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -n_y \tag{2.21}$$

$$\nabla \mathbf{U} = \begin{bmatrix} \partial_x u & \partial_y u \\ \partial_x v & \partial_y v \end{bmatrix} \tag{2.22}$$

$$\nabla \mathbf{U}^T = \begin{bmatrix} \partial_x u & \partial_x v \\ \partial_y u & \partial_y v \end{bmatrix}. \tag{2.23}$$

The surface normal vector at a wall is depicted in Figure 2.3.

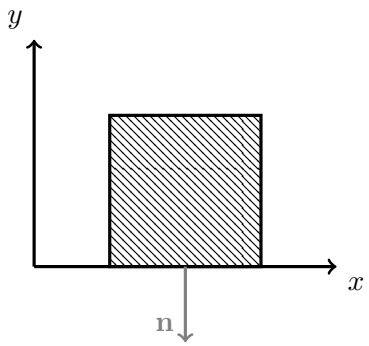

Figure 2.3: Surface normal vector at a wall

With respect to FVM, an integral is added to all terms of the Navier-Stokes equation. Particularly, the diffusive term can be transformed by the divergence theorem as follows

$$\int_V \nabla \cdot \boldsymbol{\tau} dv = \oint_o \boldsymbol{\tau} \cdot \mathbf{n} dA. \tag{2.24}$$

Afterward, this surface integral can be changed to the sum of each face, and the shear stress term at the wall is given by

$$\boldsymbol{\tau} \cdot \mathbf{n} = \nu \begin{bmatrix} 2\partial_x u & \partial_x v + \partial_y u \\ \partial_x v + \partial_y u & 2\partial_y v \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \nu \begin{bmatrix} -\partial_x v - \partial_y u \\ -2\partial_y v \end{bmatrix}. \tag{2.25}$$

Since the velocity in y-direction v is much slower than the velocity in x-direction u, all the terms related to v is negligible as follows

$$\boldsymbol{\tau} \cdot \mathbf{n} = \nu \begin{bmatrix} -\partial_y u \\ 0 \end{bmatrix}. \tag{2.26}$$

According to Equation (2.26), the wall shear stress exerts on the surface with the opposite direction of the flow, but the sign of the term can be neglected for calculating only magnitude. Consequently, the wall shear stress τ_w is given by

$$\tau_w = \nu \cdot \frac{\partial u}{\partial y}|_{wall}. \tag{2.27}$$

In turbulent flows, ν_{eff} is the effective viscosity that consists of the molecular viscosity and the turbulent eddy viscosity, and thus Equation (2.27) is changed to

$$\tau_w = \nu_{eff}\frac{\partial u}{\partial y}|_{wall} = (\nu + \nu_t)\frac{\partial u}{\partial y}|_{wall}. \tag{2.28}$$

As shown in Equation (2.28), the wall shear stress term is formed by multiplication of the molecular viscosity and the velocity gradient at the wall in a simulation. In the case of an inclined or declined wall in 2D, Equation (2.28) can be calculated as

$$\tau_w = \sqrt{\tau_{w,x}^2 + \tau_{w,y}^2}, \tag{2.29}$$

in Cartesian coordinates.

2.3 Data-driven Approach

2.3.1 Supervised Learning

Supervised learning is one of the ML types such as regression, classification, and deep learning, whose goal is to find a mapping function between inputs and outputs. Inputs in the supervised learning are known as features, whereas outputs are called labels. Features mean the arguments of a mapping function, and labels are the corresponding function values [19]. A machine learning technique is commonly used when the mapping function is not found yet, but only the data of features and labels is available.

When it comes to a mathematical sense, a feature vector is introduced as follows

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & ... & x_{d_f} \end{bmatrix}^T, \tag{2.30}$$

where d_f is the number of features [19]. Analogously, a label vector is also introduced as

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & y_3 & ... & y_{d_l} \end{bmatrix}^T, \tag{2.31}$$

where d_l is the number of labels [19]. A mapping function f_m [19] is defined as

$$f_m : \mathbb{R}^{d_f} \rightarrow \mathbb{R}^{d_l}. \tag{2.32}$$

A supervised learning process is to find the f_m based on the data with the number of training data sets m. During the process, the labels are iteratively predicted until the difference between the predicted labels and the true labels is sufficiently small. This difference is called a loss function (or a cost function), and there are several types of loss functions such as 0-1 loss function, mean squared error (MSE) loss function, and so on. Amongst them, the MSE loss function L [22] is commonly used. This MSE loss function is given by

$$L = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2, \tag{2.33}$$

where y_i is the true label and $\hat{y}_i$ is the prediction at i-th training set. The purpose of this training is to minimize the loss function, and the iterative process such as gradient descent is used to find the minimum point of the loss. In a regression problem that has a convex MSE loss function, it is not difficult to find the global minimum, and therefore the minimal loss can straightforwardly be found. However, in a deep learning problem using an NN, there are a number of local minima, which means that it is difficult to discover which one is the global minimum. This causes the training to yield different results, but these results can normally be trusted. The machine learning in this work is conducted with a basic NN model called multilayer perceptron (MLP). Thus, the MLP algorithm will be introduced in the next subsection.

2.3.2 Multilayer Perceptron

An MLP consists of layers and neurons as shown in Figure 2.4 [9]. There are an input layer and an output layer in one MLP, and several hidden layers are located between the input layer and the output layer. In each layer, a number of neurons exist, and the number of neurons per layer does not have to be identical each other. From the input layer, inputs are passed to each neuron, and the calculation by so-called activation functions such as rectified linear unit (ReLU) and sigmoid function is executed in each neuron. Subsequently, the information from all the neurons is transmitted to the output layer. This process corresponds to forward pass to optimize the model weights in the forward direction and evaluate the model outputs compared to the true values.

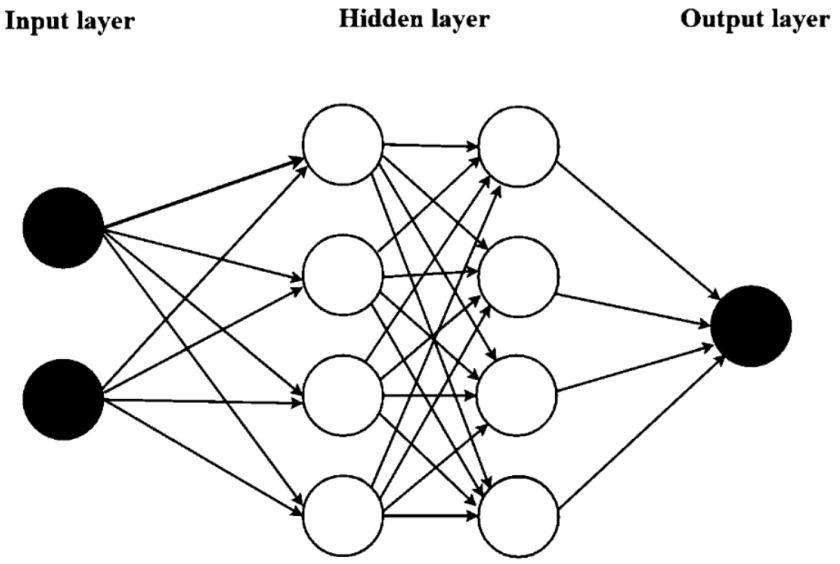

Figure 2.4: Basic structure of an MLP with 2 inputs, 1 output, and 4 neurons per hidden layer [9]

The calculation process of the forward pass consists of several stages. At one neuron location in layer l, the initial stage of calculation is given by

$$z_j^l(\mathbf{x}^{l-1}) = \sum_{i=1}^{N^{l-1}} w_{ji}^{l-1} x_i^{l-1} = (\mathbf{W}^T \cdot \mathbf{x})^{l-1}, \tag{2.34}$$

where the index i denotes each neuron in layer $(l-1)$, j denotes each neuron in layer l, $\mathbf{W}$ is a weight matrix, and N^{l-1} is the number of weights in layer $(l-1)$ [19]. z denotes the sum of the neurons in the previous layer, but this cannot be a new input for the new layer. A transformation that is non-linear is needed to map a complicated function. Therefore, an activation function a is introduced. By calculation with the activation function, a new j-th input neuron x_j^l at the layer l [19] is finally given by

$$x_j^l = a_j^l(\sum_{i=1}^{N^{l-1}} w_{ji}^{l-1} x_i^{l-1}) = a_j^l(z_j^l(\mathbf{x}^{l-1})). \tag{2.35}$$

The mapping function f_m is calculated by the same operation explained above [19] N_l times as follows

$$f_m(\mathbf{x}) = \mathbf{a}^{N_l} \circ \mathbf{a}^{N_{l-1}} \circ ... \circ \mathbf{a}^1(\mathbf{x}), \tag{2.36}$$

where $\circ$ is the operation that calculates z and a. Consequently, a prediction at the output layer [19] is calculated as follows

$$\hat{y}_i = f_m(\mathbf{x}_i). \tag{2.37}$$

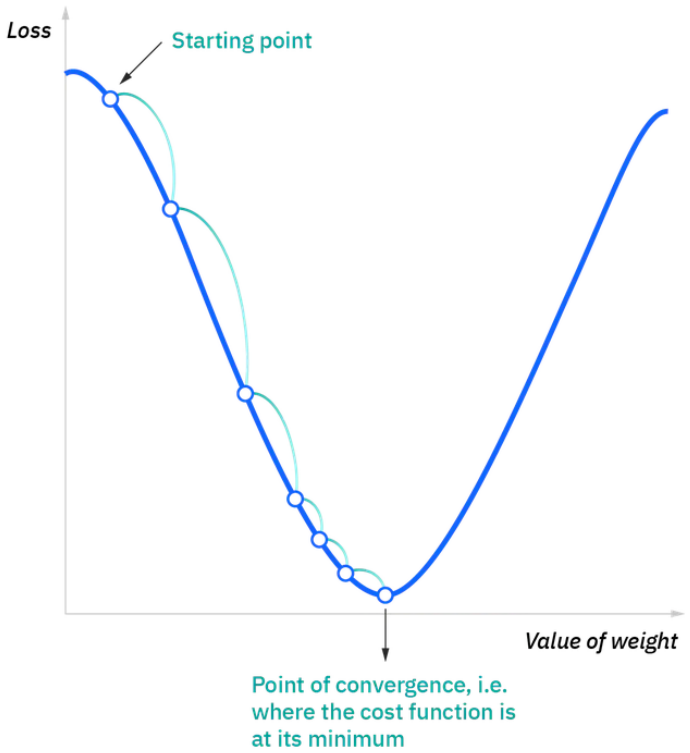

Figure 2.5: Sketch of the gradient descent method[3]

In MLPs, the loss function should also be minimized by the method known as optimization for deep learning. One of the popular optimization methods is called gradient descent (GD)[3]. The GD method finds the weight that has the minimum loss value. Figure 2.5[3] explains how the method works. At every iteration, the point of the weight is calculated and moved to another location by using the derivative of the weight itself with a learning rate α [1, 4] as follows

$$w_{k+1}^l = w_k^l - \alpha \frac{\partial L}{\partial w_k^l}, \tag{2.38}$$

where w_k^l is a weight at l-th layer with k-th epoch. The subscript ji is excluded in Equation (2.38). In `PyTorch`, there are several advanced optimizers such as the Adam optimizer [6], stochastic gradient descent (SGD) [16], etc. In this iterative optimization process, derivatives should be calculated to search for the way to reach the minimum point of loss for an MLP, but the calculation is more complicated than Equation (2.38) because there are several hidden layers in a deep NN. For this purpose, a backpropagation [4] is to be introduced. The derivative of the MLP is calculated from the output layer to the input layer by usage of the chain rule, and thus the derivative at layer l [4] is calculated by the backpropagation as follows

$$\frac{\partial L}{\partial w^l} = -(y_i - \hat{y}_i)\frac{d\hat{y}_i}{dw^n}\frac{dw^n}{dw^{n-1}}\frac{dw^{n-1}}{dw^{n-2}}\cdots\frac{dw^{l+1}}{dw^l}, \tag{2.39}$$

where the superscript n means the total number of layers in the MLP. Subsequently, the weight

[3] `https://www.ibm.com/cloud/learn/gradient-descent`

is updated by the optimization method such as GD according to Equation (2.38). The forward propagation and the backpropagation execute back and forth in the loop, and consequently the mapping can be found.

Chapter 3

Implementation of Numerical Methods

3.1 Solver Modification

The cases in this work, which are 2D flat plate cases and a 2D airfoil case, are basically steady-state simulations. Therefore, the steady-state solver `simpleFoam` is used. However, this basic solver cannot be used as it is because the NN models that will be used in this project should also be involved in this simulation. For the NN models to be applied, the `simpleFoam` solver must be modified. The original version of algorithm SIMPLE[1] (Semi-Implicit Method for Pressure-Linked Equations) is briefly mentioned as follows.

1. Set the boundary conditions
2. Solve the momentum equation to compute the velocity field
3. Compute the mass fluxes at the cell faces
4. Solve the pressure equation and apply the relaxation factor
5. Correct the mass fluxes at the cell faces
6. Correct the velocities based on the new pressure field
7. Update the boundary conditions
8. Repeat 2 to 7

The basic idea of the data-driven wall modeling is correcting fluxes at a wall and the first cell face at each adjacent cell normal to the wall. This means that the momentum equation in `simpleFoam` should use cell face values instead of cell center values. Hence, the header file `UEqn.H` that contains the momentum equation is to be modified.

Listing 3.1: UEqn.H

```
surfaceScalarField nuEff = fvc::interpolate(turbulence->nuEff());
#include "nuEffCorrection.H"
#include "fluxCorrection.H"
tmp<fvVectorMatrix> tUEqn
(
    fvm::div(phi, U)
  + MRF.DDt(U)
  // - fvc::div(turbulence->nuEff() * dev2(T(fvc::grad(U))))
  - fvm::laplacian(nuEff, U)
```

[1] https://openfoamwiki.net/index.php/OpenFOAM_guide/The_SIMPLE_algorithm_in_OpenFOAM

```
 ==
    fvOptions(U)
);
```

Listing 3.1 shows the momentum equation part in `UEqn.H`. In the equation, the laplacian term needs cell face values of the effective viscosity ν_{eff}, but the term with `fvc` can be commented out because it is only used for stability and the value goes to zero when a simulation converges. Moreover, the cell face values of the effective viscosity are difficult to be applied to this term. Before the calculation of the momentum equation, the header files `nuEffCorrection.H` and `fluxCorrection.H` should be included to correct the effective viscosity for diffusive fluxes and the mass flux ϕ for convective fluxes. Thus, the original SIMPLE algorithm is changed as follows.

1. Set the boundary conditions
2. **Correct the diffusive fluxes at the wall and the first cell face**
3. **Correct the convective fluxes at the first cell face**
4. Solve the momentum equation to compute the velocity field
5. Compute the mass fluxes at the cell faces
6. Solve the pressure equation and apply the relaxation factor
7. Correct the mass fluxes at the cell faces
8. Correct the velocities based on the new pressure field
9. Update the boundary conditions
10. Repeat 2 to 9

With this new algorithm, the momentum equation can receive the corrected ν_{eff} and ϕ.

Every correction method is performed in these header files including the calculation by the NN models. A detailed explanation of this correction method will be discussed in Section 3.2. The file `datadriven_wmSimpleFoam.C` for the flat plate case is the main structure of the modified solver, and the file `ddwmSimpleFoam_airfoil.C` is for the airfoil case. In each main solver file, `torch/script.h`, `findCellFaceLabels.H`, and `initDatadrivenTurbulenceModel.H` are included for `PyTorch`, cell and face indices, and initialization of the NN models, respectively. `writeData.H` is the header file for writing out field values. The rest of the settings are identical to the original `simpleFoam` solver.

3.2 Flux Correction Methods

In this section, flux correction methods will be discussed. The idea of the flux correction methods is came from Figure 3.1. Figure 3.1 depicts the difference between the actual slope and the numerical slope at the wall and the first cell face in wall units with respect to mesh resolution. The blue graph corresponds to Spalding's function, whereas the orange graph is the numerically interpolated velocity. The slope at the wall always underestimates the actual slope. Meanwhile, the slope at the first cell face normal to the wall overestimates the real one for coarser meshes, but underestimates it for finer meshes. When it comes to the velocity value at the first cell face, it always underestimates the actual velocity. The discrepancies lead to the results that are largely mesh-dependent, and thus not only the wall correction but also the first cell face correction should be employed for mesh-independent behavior.

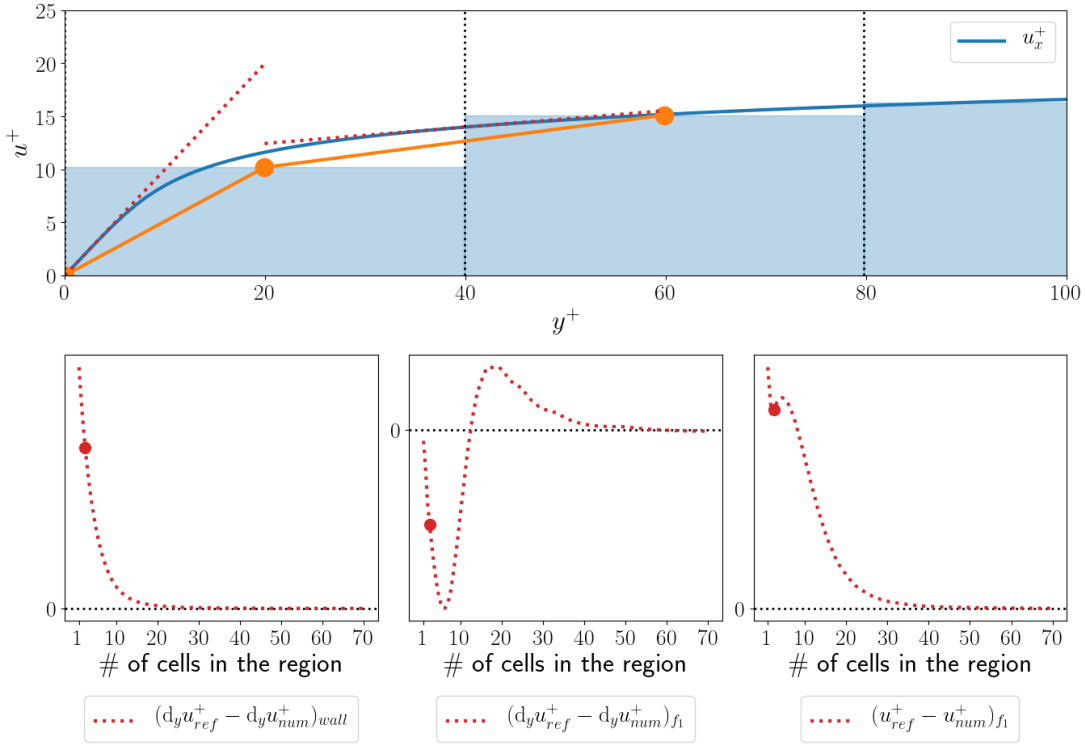

Figure 3.1: Error in the flux approximation with respect to the mesh resolution $y^+ = 40$

For the flux correction methods, Equation (2.6) is used for a starting point, which is recalled as follows

$$\sum_{f \in F_c} (\mathbf{U}_f \otimes \mathbf{U}_f) \cdot \mathbf{S}_f - \sum_{f \in F_c} (\nu \nabla \mathbf{U}_f) \cdot \mathbf{S}_f = - \int_{\Omega_c} \frac{\nabla p}{\rho} dV.$$

A convection-dominated flow is applied in this work as shown in Figure 3.2.

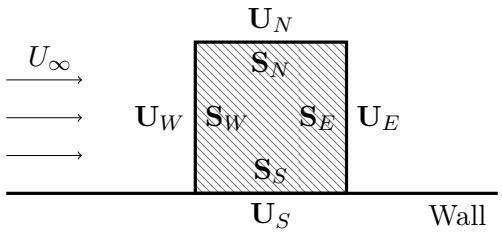

Figure 3.2: Velocities and surfaces in a convection-dominated flow at a wall

Using the notation in Figure 3.2, the convective flux in Equation (2.6) is changed to

$$\sum_{f \in F_c} (\mathbf{U}_f \otimes \mathbf{U}_f) \cdot \mathbf{S}_f = (\mathbf{U}_W \otimes \mathbf{U}_W) \cdot \mathbf{S}_W + (\mathbf{U}_E \otimes \mathbf{U}_E) \cdot \mathbf{S}_E + (\mathbf{U}_N \otimes \mathbf{U}_N) \cdot \mathbf{S}_N + (\mathbf{U}_S \otimes \mathbf{U}_S) \cdot \mathbf{S}_S. \quad (3.1)$$

The difference between the magnitude of $\mathbf{U}_W$ (U_W) and $\mathbf{U}_E$ (U_E) is negligible compared to the difference between $\mathbf{U}_N$ and $\mathbf{U}_S$ because of the boundary layer structure at the wall, and therefore $U_W = U_E$ can be applied. Subsequently, Equation (3.1) is changed to

$$\sum_{f \in F_c} (\mathbf{U}_f \otimes \mathbf{U}_f) \cdot \mathbf{S}_f = (\mathbf{U}_N \otimes \mathbf{U}_N) \cdot \mathbf{S}_N + (\mathbf{U}_S \otimes \mathbf{U}_S) \cdot \mathbf{S}_S. \quad (3.2)$$

On the other hand, the diffusive flux in Equation (2.6) is modified as

$$\sum_{f\in F_c}(\nu\nabla\mathbf{U}_f)\cdot\mathbf{S}_f = (\nu\nabla\mathbf{U}_W)\cdot\mathbf{S}_W + (\nu\nabla\mathbf{U}_E)\cdot\mathbf{S}_E + (\nu\nabla\mathbf{U}_N)\cdot\mathbf{S}_N + (\nu\nabla\mathbf{U}_S)\cdot\mathbf{S}_S. \quad (3.3)$$

The difference between the gradient of the velocities at west and east is also negligible, which leads to the equation as follows

$$\sum_{f\in F_c}(\nu\nabla\mathbf{U}_f)\cdot\mathbf{S}_f = (\nu\nabla\mathbf{U}_N)\cdot\mathbf{S}_N + (\nu\nabla\mathbf{U}_S)\cdot\mathbf{S}_S. \quad (3.4)$$

When it comes to the convective flux for the case of a flat plate, the velocity component in x-direction u is not zero, but the velocity component in y-direction v is almost zero that can be negligible. Thus, Equation (3.2) is changed to

$$\sum_{f\in F_c}(\mathbf{U}_f\otimes\mathbf{U}_f)\cdot\mathbf{S}_f = 0, \quad (3.5)$$

where $U_S = 0$ since the velocity at a wall is zero. Hence, the convective flux is not considered in flat plate cases. However, for the case of an airfoil, v is not zero, and therefore U_N remains in the equation as follows

$$\sum_{f\in F_c}(\mathbf{U}_f\otimes\mathbf{U}_f)\cdot\mathbf{S}_f = (\mathbf{U}_N\otimes\mathbf{U}_N)\cdot\mathbf{S}_N = [(u^2 cos\theta + uv(cos\theta + sin\theta) + v^2 sin\theta)|_N]\mathbf{n}_y, \quad (3.6)$$

where the angle between the surface of the airfoil and the x-axis is θ.

Meanwhile, for the diffusive flux in the flat plate case, both of the velocity gradients at the north and the south faces are not zero and in a large scale. Thus, Equation (3.4) can be expressed again as

$$\sum_{f\in F_c}(\nu\nabla\mathbf{U}_f)\cdot\mathbf{S}_f = (\nu\nabla\mathbf{U}_N)\cdot\mathbf{S}_N + (\nu\nabla\mathbf{U}_S)\cdot\mathbf{S}_S = (\nu\frac{\partial u}{\partial y}|_N - \nu\frac{\partial u}{\partial y}|_S)\mathbf{n}_y. \quad (3.7)$$

Equation (3.7) can be applied to both of the flat plate and the airfoil cases, and hence the velocity gradients at the north and the south faces will be used for the diffusive flux correction.

Consequently, Equations (2.6), (3.5), and (3.7) can be combined to one equation for the flat plate case, which is given by

$$(\nu\frac{\partial u}{\partial y}|_{face} - \nu\frac{\partial u}{\partial y}|_{wall})\mathbf{n}_y = \int_{\Omega_c}\frac{\nabla p}{\rho}dV. \quad (3.8)$$

Analogously, Equations (2.6), (3.6), and (3.7) can also be combined to one equation for the airfoil case as follows

$$[((u^2 cos\theta + uv(cos\theta + sin\theta) + v^2 sin\theta))|_{face}]\mathbf{n}_y - (\nu\frac{\partial u}{\partial y}|_{face} - \nu\frac{\partial u}{\partial y}|_{wall})\mathbf{n}_y = -\int_{\Omega_c}\frac{\nabla p}{\rho}dV. \quad (3.9)$$

3.2.1 Diffusive Flux Correction

A diffusive flux correction method is based on Equation (3.8) or Equation (3.9). According to these equations, the velocity gradients or the slopes at a wall and the first cell face normal to the wall are needed to correct the flux. The slopes influence wall shear stresses because the wall

shear stresses are affected by the wall slopes in accordance with Equation (2.28) that is recalled as follows

$$\tau_w = \nu_{eff}\frac{\partial u}{\partial y}|_{wall} = (\nu + \nu_t)\frac{\partial u}{\partial y}|_{wall}.$$

The diffusive flux correction occurs when the ratio of the ML model slope to the numerical slope for the wall and the first cell face is multiplied to Equation (2.28) in order to improve the prediction of wall shear stresses. Afterward, the numerical velocity gradient is changed to the ML model slope. However, in actual simulations, this direct substitution does not work because the velocity gradient in the `simpleFoam` solver is possibly implicit, which means that the information of the next time step is involved for the gradient by employing the Euler backward method for transient simulations or a relaxation factor. Hence, the ratio of the velocity gradient will be applied to the effective viscosity instead of the gradient term itself. The basic correction method by changing the effective viscosity is given by

$$\nu_{eff,corr} = \nu_{eff,num}\frac{\partial u/\partial y|_{model}}{\partial u/\partial y|_{num}}, \tag{3.10}$$

where *num* stands for numerical, *corr* stands for corrected. The ratio of the slopes modifies the effective viscosity, and the modified effective viscosity changes wall shear stresses.

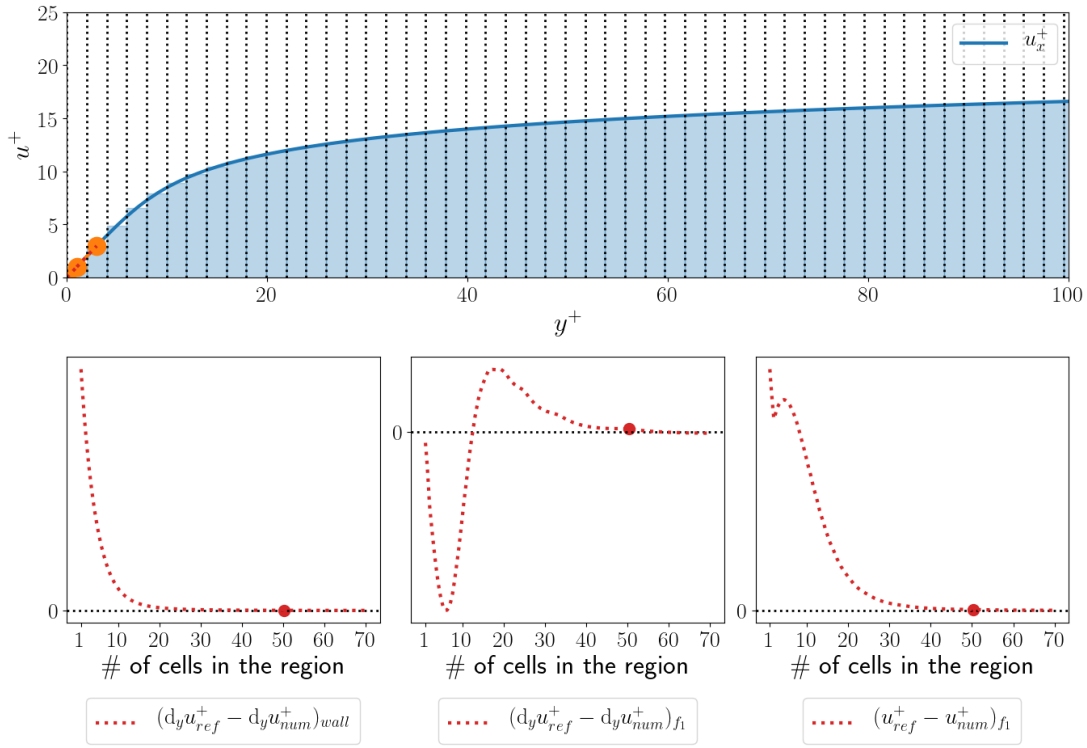

Figure 3.3: Error in the flux approximation with respect to the mesh resolution $y^+ = 2$

Meanwhile, it is obvious that a diffusive correction is not needed if mesh is well resolved according to Figure 3.3 because the difference between the numerical and the actual values are almost zero, but the effect of correction needs to gradually increase as the mesh resolution goes coarser. Therefore, a blending method with a criterion of velocity is introduced here. The criterion of velocity that is also considered as a normalized velocity is as follows

$$\tilde{U} = \frac{U - U_{wall}}{U_\infty - U_{wall}}, \tag{3.11}$$

where U_{wall} is the magnitude of the velocity at the wall, U_∞ is the magnitude of the inlet velocity, and U corresponds to the velocity magnitude at the first cell center. If the mesh goes coarser, the cell center velocity gets closer to the ambient velocity, which leads to $\tilde{U} = 1$. This velocity blending method can be divided into two methods that correspond to a typical blending and a reverse blending. The typical blending method is employed for the wall correction, while the reverse blending method is used for the first face correction. The typical blending equation is as follows

$$\nu_{eff,velBlend}|_{wall} = \nu_{eff,num}|_{wall} \cdot \frac{\tilde{U}^{\gamma_w} \cdot \partial u/\partial y|_{model} + (1-\tilde{U})^{\gamma_w} \cdot \partial u/\partial y|_{num}}{\partial u/\partial y|_{num}}, \tag{3.12}$$

where γ_w is an exponent that can control the extent of blending at the wall, and $\gamma_w = 0.8$ is used for this work. If the center of the first cell normal to the wall is located far from the wall, Equation (3.12) changes to Equation (3.11) since $\tilde{U} = 1$ is applied. In contrast, if the first cell center is near the wall, no correction occurs because $\tilde{U} = 0$ holds here. On the other hand, the typical velocity blending method does not work for the first cell face correction in a simulation because the numerical slopes get infinitely smaller, which leads to very small denominator of the blending equation if the simulation iterates further. Thus, the reverse blending equation is introduced which is given by

$$\nu_{eff,velBlend}|_{face} = \nu_{eff,num}|_{face} \cdot \frac{\partial u/\partial y|_{model}}{\tilde{U}^{\gamma_f} \cdot \partial u/\partial y|_{num} + (1-\tilde{U})^{\gamma_f} \cdot \partial u/\partial y|_{model}}, \tag{3.13}$$

where γ_f is an exponent that can control the extent of blending at the first cell face normal to the wall, and $\gamma_f = 0.7$ is used here. The blending occurs in the denominator part, and therefore the equation is more stable than the typical blending equation. The working scheme is basically the same as the previous one, but the actual value is different since the blended value is inserted into the denominator instead of the numerator. The reverse blending is not employed to the wall correction since the wall correction is already stable, which means that the reverse blending is not needed. In addition, the scale of the wall slopes are much larger than the scale of the face slopes, and thus the reverse blending method yields significantly different values from the typical blending method.

Subsequently, another blending method is introduced based on y^+. If the y^+ value is small which means that mesh is resolved, the simulation itself can capture the actual behavior of wall shear stresses, and thus the flux correction method is not needed. However, if the y^+ value is too large, uncertainties for the first face correction at the front of the plate increases. Therefore, only the wall correction is used for this case. This is why the y^+ blending method is employed, and the method is divided into two equations for the wall and the first face correction, respectively. The y^+ blending equation for the wall correction is given by

$$\nu_{eff,ypBlend}|_{wall} = w_{wall} \cdot \nu_{eff,num}|_{wall} + (1 - w_{wall}) \cdot \nu_{eff,velBlend}|_{wall}, \tag{3.14}$$

where

$$w_{wall} = \frac{1}{1 + exp(-0.95(15 - y^+))}. \tag{3.15}$$

w_{wall} is a sigmoid function that has the y^+ range between 10 and 20. If y^+ is smaller than 10 where $w_{wall} = 1.0$, no blending and correction occur, whereas the full velocity blending works for $y^+ > 20$ where $w_{wall} = 0.0$. On the other hand, the equation for the first face correction is given by

$$\nu_{eff,ypBlend}|_{face} = w_{face} \cdot \nu_{eff,num}|_{face} + (1 - w_{face}) \cdot \nu_{eff,velBlend}|_{face}, \tag{3.16}$$

where

$$w_{face} = \frac{1}{1 + exp(-0.95(15 - y^+))} + \frac{1}{1 + exp(-0.95(y^+ - 30))}. \tag{3.17}$$

In this case, the first cell face correction works between the y^+ range of 10 and 35 because there is a kink at the front of the plate and a large discrepancy of skin friction at the front of the airfoil if the same sigmoid function as Equation (3.15) is used for the first cell face correction. For $y^+ > 35$ where $w_{face} = 1.0$, only wall correction is used as shown in Figure 3.4 because Equation (3.17) turns the face correction off for this range.

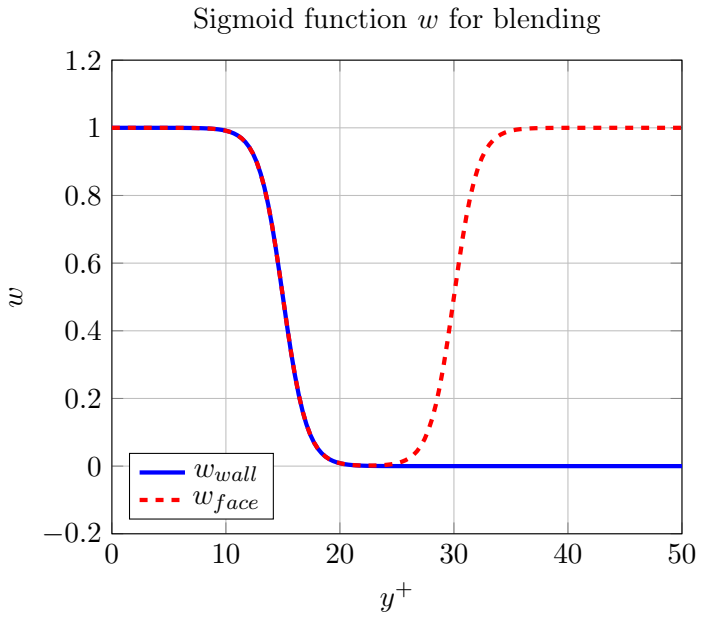

Figure 3.4: Sigmoid function w for y^+ blending

3.2.2 Convective Flux Correction

A correction method for the convective flux is needed for an airfoil case because the velocity component that is perpendicular to a wall is involved unlike flat plate cases. Equation (3.9) shows that the convective flux only at the first cell face is involved for the momentum equation to hold. Therefore, the ratio of the ML model slope to the numerical slope at the first cell face normal to the wall is applied to the convective flux `phi` or ϕ in OpenFOAM as follows

$$\phi_{corr} = \phi_{num} \frac{U_{model}|_{face}}{U_{num}|_{face}}, \tag{3.18}$$

where $U_{model}|_{face}$ and $U_{num}|_{face}$ are the velocity magnitudes at the first cell face normal to the wall. Since the ML model yields the velocity from the 1D setting and the correction needs the velocity ratio, the convective flux correction occurs by the magnitude of the velocity.

The same blending methods for the diffusive flux correction are applied to the convective flux correction. First of all, the velocity blending method is given by

$$\phi_{velBlend}|_{face} = \phi_{num}|_{face} \cdot \frac{\tilde{U}^{\gamma_\phi} \cdot U_{model}|_{face} + (1 - \tilde{U})^{\gamma_\phi} \cdot U_{num}|_{face}}{U_{num}|_{face}}, \tag{3.19}$$

where $\gamma_\phi = 1.0$ is a blending exponent for the convective flux at the first cell face normal to the wall. Here, the typical blending method is used because this part is numerically stable. Secondly, the y^+ blending method is involved as follows

$$\phi_{ypBlend}|_{face} = w_{face} \cdot \phi_{num}|_{face} + (1 - w_{face}) \cdot \phi_{velBlend}|_{face}, \tag{3.20}$$

where the same w_{face} from Equation (3.17) is employed, and therefore the convective flux correction at the first face also works between the y^+ range of 10 and 35.

Chapter 4

Approximating Velocity Profile in 1D Channel Flow

4.1 Simulation Setup

4.1.1 Flow and Boundary Conditions

In order to extract proper data of the features and the labels that will be mentioned in Subsection 4.2.1, a geometry near a wall region is needed. In this work, a 1D channel flow setting will be employed due to two reasons. First of all, if a geometry is as simple as possible, data generation time will markedly be reduced. Secondly, any high fidelity data is not needed regardless of the availability of DNS data if this 1D approach is introduced.

The flow condition in the 1D channel mimics the condition in the 2D flat plate case[1] in OpenFOAM with the Reynolds number (Re_δ) 10 million given in Table 4.1.

Variable	Value
U_∞	$69.4m/s$
ν	$1.388 \cdot 10^{-5}m^2/s$
d	$2.0m$
Re_δ	$1 \cdot 10^7$
Model	Spalart-Allmaras

Table 4.1: Flow condition in a 1D channel case

The geometry is described in Figure 4.1. The channel height is $2.0m$, but only a half of the height will be used by setting the top patch as `symmetryPlane`. The detail of mesh generation will be discussed in Subsection 4.1.2.

In SA model, several boundary conditions are needed for such as `nut`, `nuTilda`, `U`, and `p`. It is a one-equation model, and hence only `nuTilda` is needed instead of `k`, `omega`, and `epsilon`. These boundary conditions are specified in `0` or `0.orig` folder in each case in OpenFOAM. For pressure, the boundary condition at the wall is `zeroGradient`. For `nut`, the value at the wall should be a fixed value of 0.0 since only the molecular viscosity exists at the wall. For `nuTilda`, the value should also be 0.0 at the wall because `nuTilda` is a modified version of `nut` for SA model. For velocity, an inlet velocity condition is needed, but the condition cannot be specified

[1] `https://www.openfoam.com/documentation/guides/latest/doc/verification-validation-turbulent-flat-plate-zpg.html`

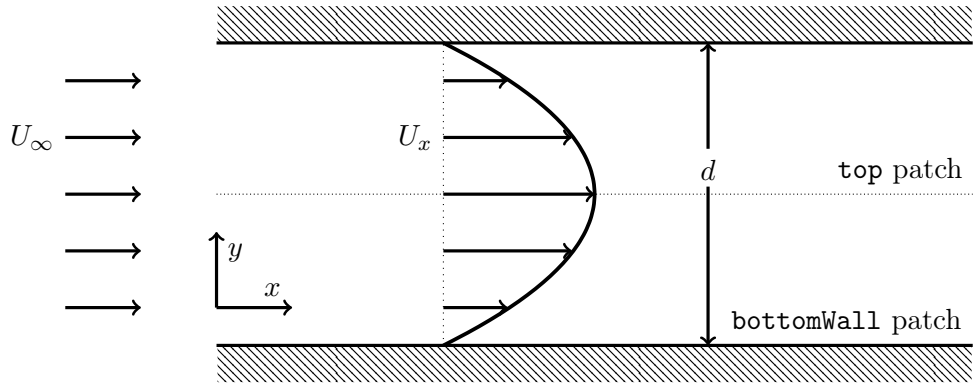

Figure 4.1: Geometry of a channel flow

in the boundary condition section since this simulation is 1D. Therefore, only zero velocity is specified in the `U` dictionary at the wall in this case. Instead, a source term should be added in the `fvOptions` file in the `system` folder. If a momentum source with an average velocity of $U_\infty = 69.4$ is added, the condition with the U_∞ from the inlet in 2D can be mimicked in 1D by keeping the identical Reynolds number.

4.1.2 Mesh Generation

A mesh generation method can determine the range of y^+ that will be used in 2D simulations. If meshes are very coarse, the velocity profile from a simulation will not be defined in a small y^+ range, and thus the slopes in that area cannot be found. In opposite, the velocity profile will not correctly be estimated in a large y^+ range if meshes are immensely resolved. Furthermore, the calculation speed for the exceedingly resolved meshes is dramatically slow. Therefore, the range of y^+ is set from approximately 1.5 to 500 in the 1D channel case.

Figure 4.2 is a conceptual picture that shows how the mesh in the actual 1D simulation looks like. For the range of y^+ to be set as the above, 1600 cells for y-direction is needed with the grading of 400. The mesh generation has a decent procedure because there is no specific geometry in this 1D channel. Thus, by using `blockMesh` in OpenFOAM, the mesh can be generated without any other tools.

Since this simulation corresponds to 1D, the left patch and the right patch are `cyclic`, and thus velocity profiles can be developed as time increases. The solver in the simulation will be `pimpleFoam` that can be introduced for transient simulations. The mapping function from this 1D simulation will be applied to incompressible steady-state 2D cases, and therefore the velocity development concerning the location in 2D is to be considered as the velocity development with regards to the time in 1D channel flow. For instance, a velocity profile in 1D is considered as the profile at the front of the plate in a 2D flat plate case if this 1D profile is at an early time step, whereas a velocity profile at a late time step in 1D is the profile at the end of the plate in 2D.

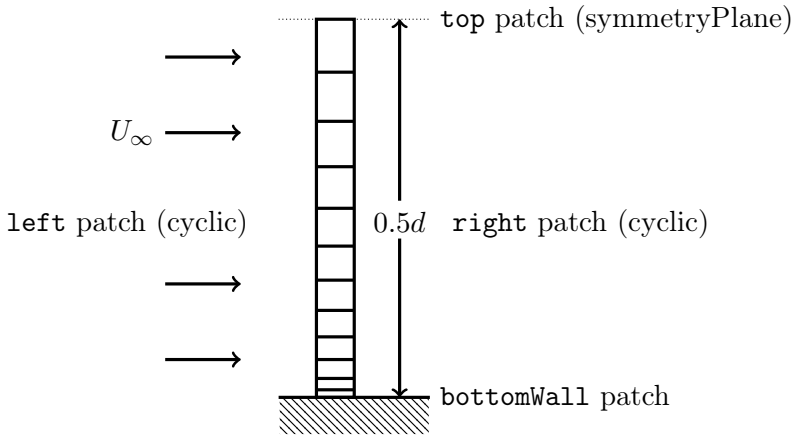

Figure 4.2: Mesh setting for a 1D channel flow geometry

4.2 Learning Parameters

4.2.1 Data Generation

Data generation for the 1D channel flow is defined as saving features and labels in every time step. The features in this case are the height of each cell face with the variable name `y_face` and the integral average velocity in x-direction `avgU` until the certain number of cells as well as the labels are the slope at the wall `dUdy_wall`, the slope at each face `dUdy_face`, and the velocity in x-direction at each face `Ux_face`. Table 4.2 shows the brief description of the features and the labels.

Features	Description
`y_face`	Height of each cell face in y-direction
`avgU`	Average velocity by integral method in x-direction
Labels	Description
`dUdy_wall`	Slope at the wall
`dUdy_face`	Slope at the faces
`Ux_face`	Velocity at each face in x-direction

Table 4.2: Description of the features and the labels

`y_face` is set as the height value at each face including the wall face because the flux correction methods need the information at the wall and the first cell face normal to the wall. For the first cell face, the height of this face is changed over a mesh setting, and therefore the feature `y_face` is employed. `avgU` is calculated not by the arithmetic average but by the area-based average of the numerical velocity since the area-based average velocity is more sensitive for mesh resolution than the arithmetic average velocity. The blue shaded area in Figure 3.1 means the integral value for each cell. The calculation is given by

$$U_{avg} = \frac{u_1 V_1 + u_2 V_2 + ... + u_n V_n}{V_1 + V_2 + ... + V_n}, \tag{4.1}$$

where the index n denotes n-th cell, and V_n is the volume of the n-th cell. Since the mapping in the 1D channel is performed on the high resolved mesh, the blue shaded area for the numerical velocity becomes identical to the area for the true velocity from the simulation. `dUdy_wall` is depicted as the red dotted line at the wall in Figure 3.1, and `dUdy_wall` is the red dotted line at the first cell face in the same figure. They are located at the wall face and the faces at each cell

in the 1D channel, and hence they are predicted by the mapping method with the information of the average velocity. However, in Figure 3.1, the slopes are based on Spalding's function, while the slope labels are based on the velocity profile from the simulation. `Ux_face` is the true velocity value at the first cell face that corresponds to the blue line at the face located at $y^+ = 40$ in Figure 3.1. Analogously, this face velocity is based on Spalding's function in the figure, whereas the velocity label is based on the velocity profile from the simulation.

The appropriate time range should be set to save appropriate values of the features and the labels. In 2D cases, the velocity profile near an inlet is not fully developed, while the profile near an outlet is highly likely to be fully developed. Therefore, the information of velocity profiles at early time steps, which are not fully developed, is strongly needed. First of all, 50 steps with $0.0002s$ write interval are set for the early stage of the flow. Afterward, 90 steps with $0.001s$ write interval are set for the late stage of the flow, which creates total 140 time steps.

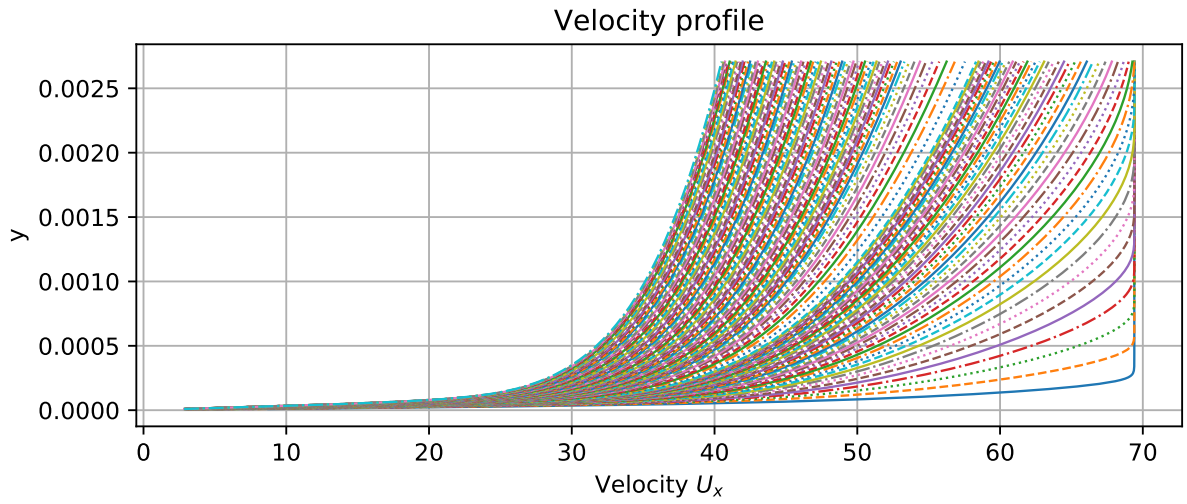

Figure 4.3: Velocity profiles for all time steps

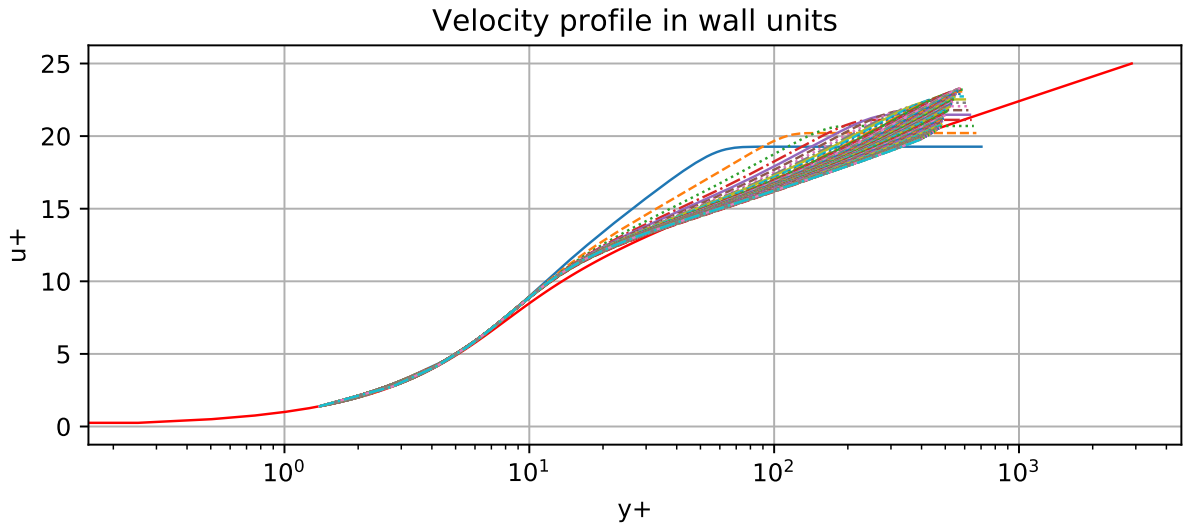

Figure 4.4: Velocity profiles for all time steps in wall units

On the one hand, Figure 4.3 shows how the velocity profiles shape for all the time steps used in the simulation, and it turns out that the data is well distributed. On the other hand, Figure 4.4

demonstrates that the velocity profiles from the beginning to the end of the time steps in wall units keep getting closer to the law of the wall as the time increases.

4.2.2 Generation of Mapping Function

In this subsection, the mapping method will be discussed. Firstly, since arbitrary inlet conditions could be used for other simulations, all the features and the labels should be non-dimensionalized as follows

$$x_1 = \frac{y_{face}}{l_{ref}} \tag{4.2}$$

$$x_2 = \frac{avgU - U_{wall}}{U_\infty - U_{wall}} \tag{4.3}$$

$$y_1 = \left(\left. \frac{dU}{dy} \right|_{wall} \right) \cdot \frac{l_{ref}}{U_\infty} \tag{4.4}$$

$$y_2 = \left(\left. \frac{dU}{dy} \right|_{face} \right) \cdot \frac{l_{ref}}{U_\infty} \tag{4.5}$$

$$y_3 = \frac{U_{x,face} - U_{wall}}{U_\infty - U_{wall}}, \tag{4.6}$$

where

$$l_{ref} = \frac{\nu}{U_\infty}. \tag{4.7}$$

Afterward, all the features and the labels are to be scaled as 0-1 scale by `min-max scaling`.

Regarding MLP, a simple NN model structure will be used, which has the identical number of neurons per each hidden layer. The related datasets are divided into three parts such as training sets, validation sets, and test sets with the percentage of 60%, 20%, and 20%, respectively. The indices for each set are to be determined by using a multinomial method in `PyTorch` so that the datasets will randomly be divided. Currently, only one NN model is employed, which consists of two inputs and three outputs, but from Section 4.3, this model is divided into three models for better accuracy.

The optimization of the model is performed by the Adam optimizer, and the best model for the training sets and the validation sets will be saved during the training if a new loss value is smaller than the loss of the previous best model. No stopping criterion will be used, but the training will be executed with the enough number of epochs.

4.2.3 Investigation of Uncertainties

Here, an investigation is needed to find the hyper-parameters that yield the least uncertainties. In this study, the inlet velocity and the molecular viscosity are $50.0 m/s$ and $1.0 \cdot 10^{-5} m^2/s$, while the Reynolds number is kept identical as shown in Table 4.3.

Variable	Value
U_∞	$50.0m/s$
ν	$1.0 \cdot 10^{-5} m^2/s$
d	$2.0m$
Re_δ	$1 \cdot 10^7$
Model	Spalart-Allmaras

Table 4.3: Flow condition for uncertainty check

The investigation consists of four parts that are pertinent to activation functions, the number of layers, learning rates, and the number of neurons per layer, which can be hyper-parameters of the model. For each section, 10 random seeds are to be introduced with 10000 epochs, and the range of MSE loss values are investigated.

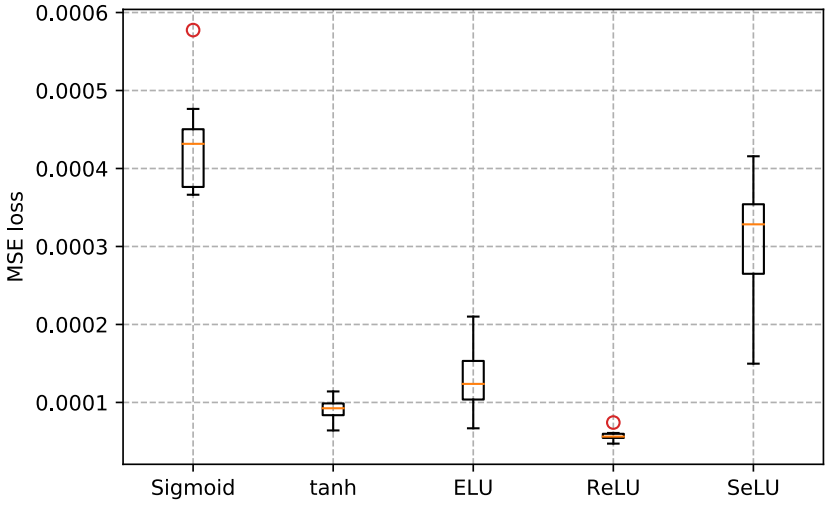

Figure 4.5: MSE loss values for activation functions by box plots

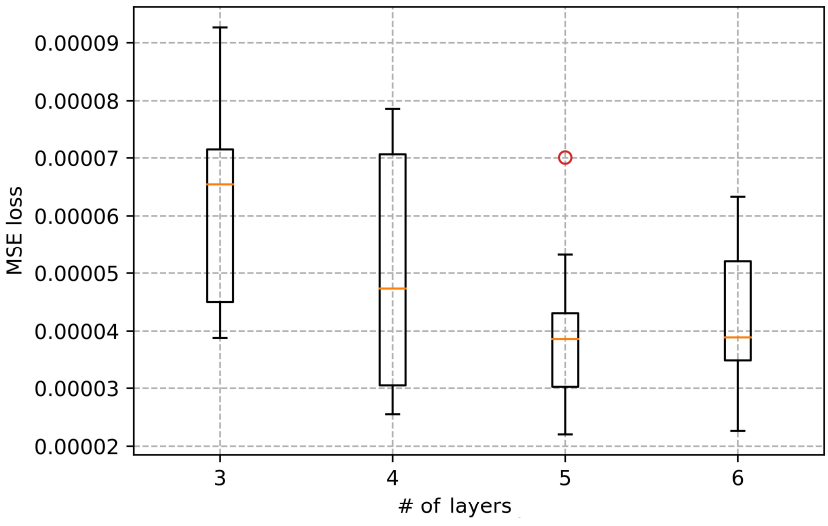

Figure 4.6: MSE loss values for the number of layers by box plots

Figure 4.5 demonstrates the MSE loss values for each activation function with box diagrams. It shows that the `ReLU` function has the shortest range and the smallest mean value of MSE loss,

and therefore it can be concluded that the `ReLU` function yields the most certain and similar value to the original one compared to the other functions. The next part is in terms of the number of layers. The model with 5 layers (4 hidden layers) has the best performance in accordance with Figure 4.6 in spite of one outlier. Figure 4.7 shows the loss values for various learning rates, and then the models with the learning rate 0.001 and with the rate 0.003 yield the similar best results. In this work, 0.001 will be used for more stable training. Lastly, when it comes to the number of neurons per layer shown in Figure 4.8, the model with 50 neurons shows the least loss value and the smallest error range. Consequently, it turns out that the best model combination should be with the `ReLU` activation function, 5 layers, the learning rate 0.001, and 50 neurons per layer.

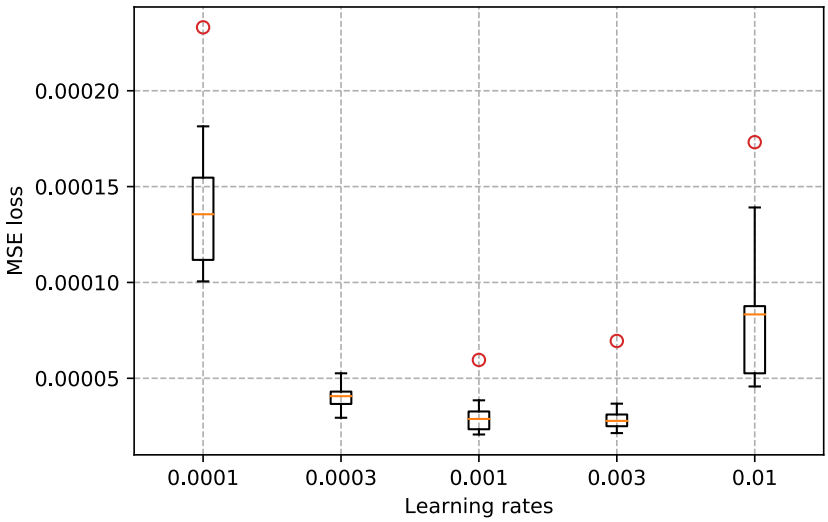

Figure 4.7: MSE loss values for learning rates by box plots

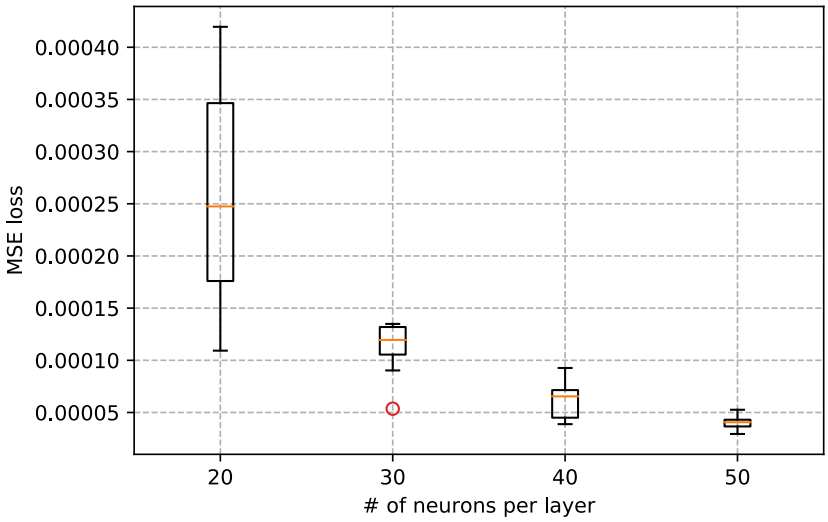

Figure 4.8: MSE loss values for the number of neurons by box plots

Currently, uncertainties of the best combination is to be investigated. According to Figure 4.9, the box plot of the best model shows the least MSE loss compared to all the values in the other box plot figures.

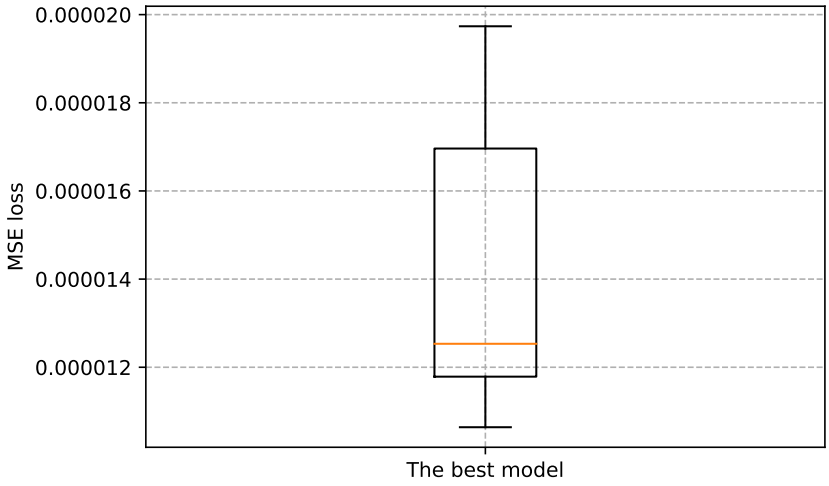

Figure 4.9: Box plot of MSE for the best model

The histogram in Figure 4.10 demonstrates that most of the datasets have the relative prediction error for all the labels with less than 3% where a relative error for a label y_i is defined as follows

$$RE(y_i) = \frac{|(y_i - y_{i,min}) - (\hat{y}_i - y_{i,min})|}{y_{i,max} - y_{i,min}} = \frac{|y_i - \hat{y}_i|}{y_{i,max} - y_{i,min}}. \tag{4.8}$$

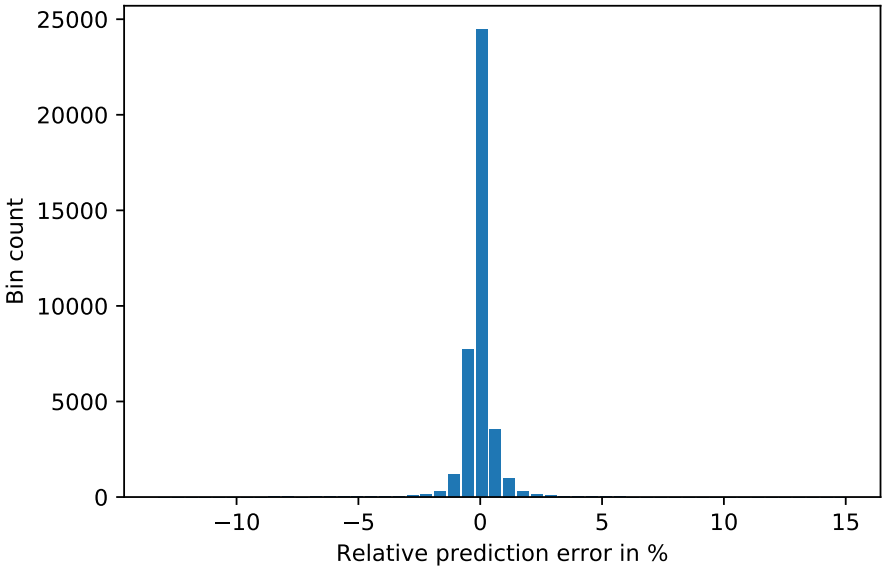

Figure 4.10: Relative error for the best model by histogram

On the other hand, heat maps for the prediction are also investigated as per Figures 4.11, 4.12, and 4.13 for three labels. The heat map for wall slopes in Figure 4.11 shows that the maximum relative error occurs near the wall. In Figure 4.12, it is obvious that the maximum relative error is located at the fast velocity region that corresponds to the early time steps. Lastly, Figure 4.13 demonstrates that the relative error for the whole region is fairly small. Comprehensively, regardless of the labels, the error occurs at the early time step region. Hence, the front part of a 2D flat plate should carefully be investigated at a later stage.

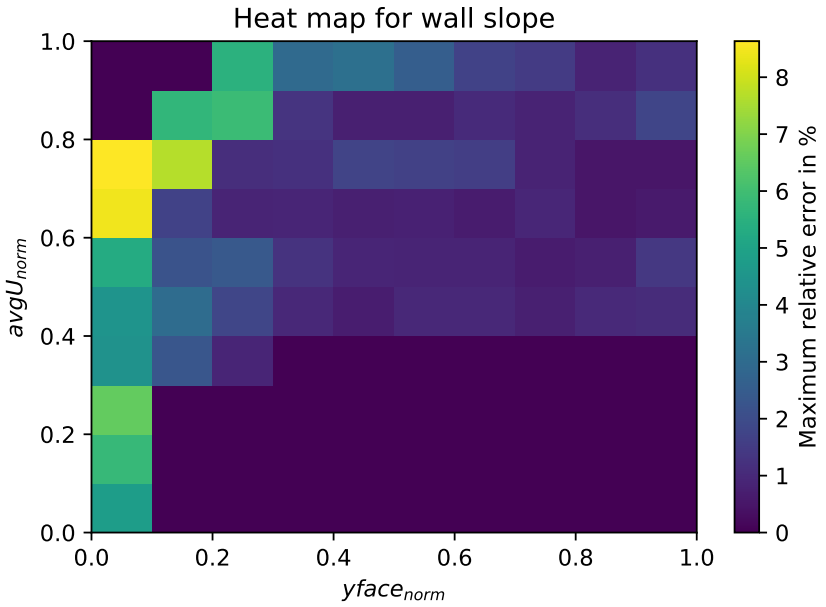

Figure 4.11: Relative error heat map for wall slopes

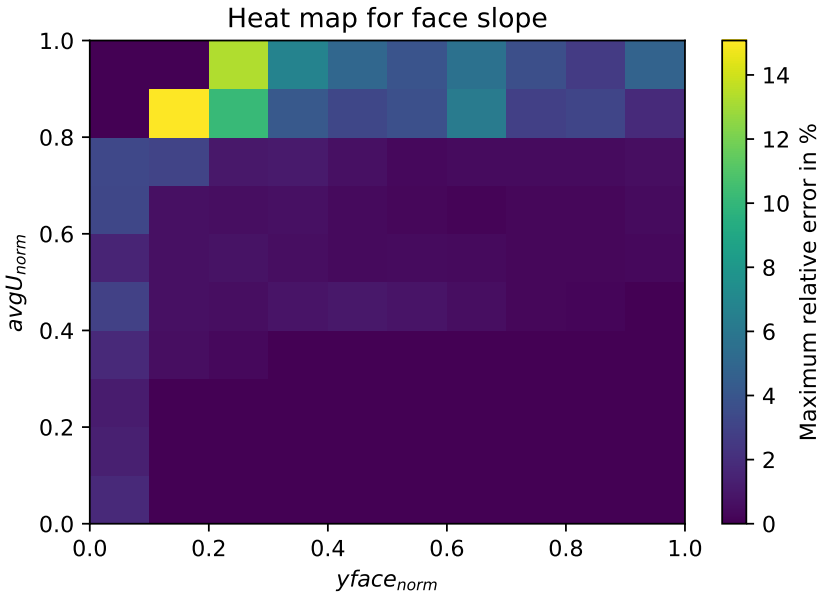

Figure 4.12: Relative error heat map for face slopes

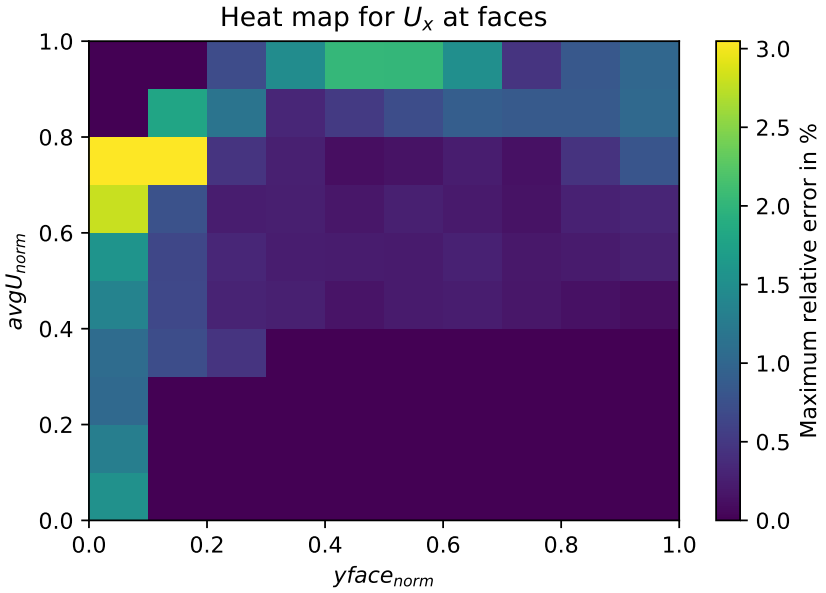

Figure 4.13: Relative error heat map for velocity at faces

4.3 Results

In this section, training results of the mapping function will be shown. As mentioned in Subsection 4.2.2, one modification is executed for the model to yield better results. The NN model is divided into three NN models so that the training accuracy can be improved. The features `y_face` and `avgU` predict `dUdy_wall` for one model, `dUdy_face` for the other model, and `Ux_face` for the last model. The values of the hyper-parameters of the previous best model are kept in the training phase, and the flow condition is identical to the condition mentioned in Table 4.1. After the training of three models, they will be combined together by using a tracing method from `PyTorch` and used in 2D simulations in OpenFOAM.

4.3.1 Model for Wall Slopes

The previous model is already divided into three models, and thus all the three models have two features and one label, respectively. This model receives `y_face` and `avgU`, and yields the prediction of `dUdy_wall`.

Figure 4.14 shows the convergence behavior of the loss function of this model with 10000 epochs, and both of the loss values for the training sets and the validation sets are within the scale of $1 \cdot 10^{-4}$. The mapping performance heat map for wall slopes is plotted as shown on the left side in Figure 4.17. The maximum relative error for the entire region is approximately 0.16, and therefore the mapping performance is fairly good compared to the original data. The discrepancies mostly occur near the wall and at the fastest range of the velocity, which implies that the error occurs at the early stage of the 1D simulation.

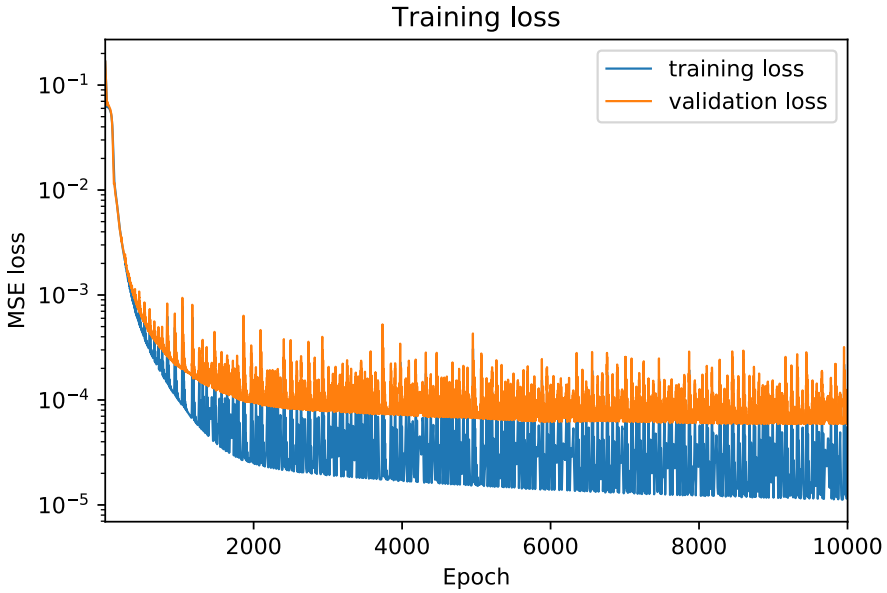

Figure 4.14: Training loss for the wall slope model

4.3.2 Model for Face Slopes

This model receives `y_face` and `avgU`, and yields the prediction of `dUdy_face`. Since the range of the datasets is quite broad compared to the other models, the training was conducted with 20000 epochs.

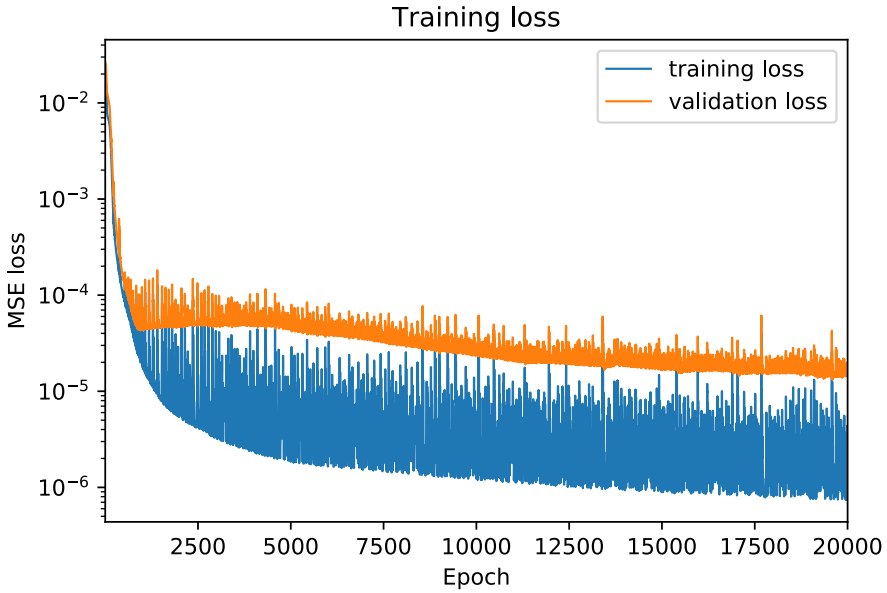

Figure 4.15: Training loss for the face slope model

Figure 4.15 shows the training loss function convergence. The scale of the training sets' loss is $1 \cdot 10^{-6}$, while the scale of the validation sets' loss is $1 \cdot 10^{-4}$. The mapping performance heat map for face slopes is plotted as shown on the middle in Figure 4.17. The maximum error for the entire region is approximately 0.08 near the wall. Otherwise, the error is extremely small. Therefore, it can be concluded that the mapping performance is markedly good.

4.3.3 Model for Velocities at Faces

This model receives `y_face` and `avgU`, and yields the prediction of `Ux_face`. The training was conducted with 10000 epochs again as the first model. The training loss convergence is displayed in Figure 4.16, and both of the losses for the training sets and the validation sets are within the scale of $1 \cdot 10^{-5}$.

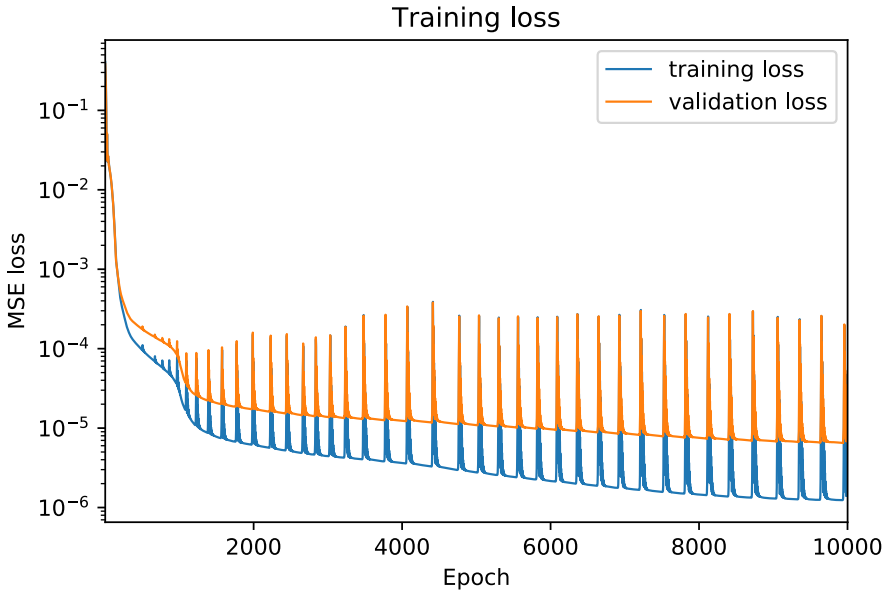

Figure 4.16: Training loss for the wall slope model

The mapping performance heat map for `Ux_face` is plotted as shown on the right side in Figure 4.17. The maximum relative error is approximately 0.045, which implies that the mapping performance is also remarkably good.

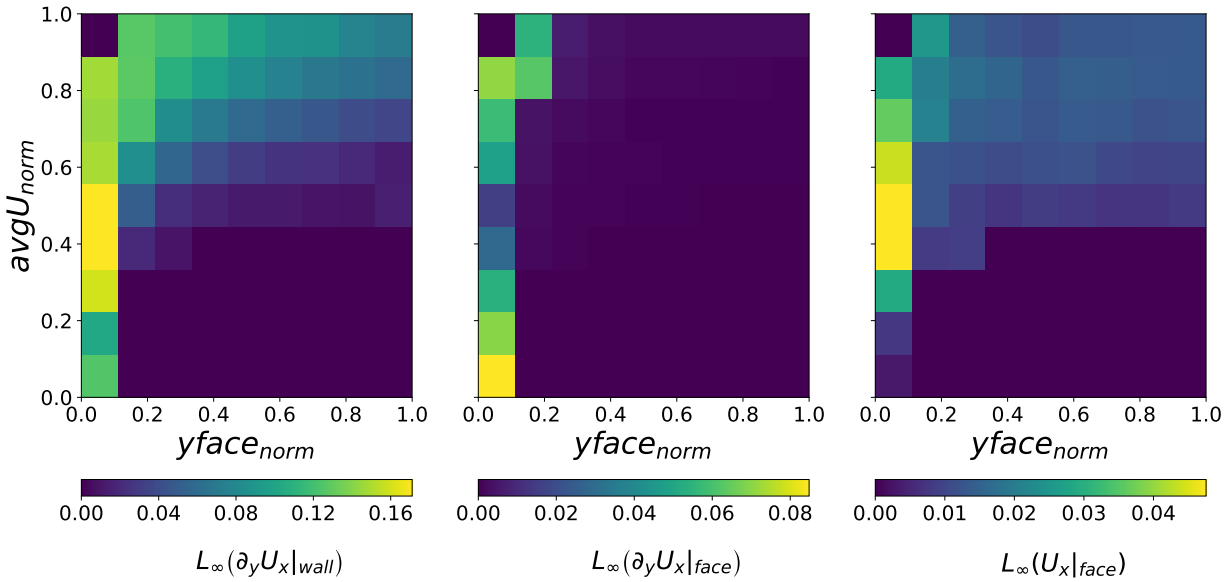

Figure 4.17: Heat map for the maximum relative error in different segments of the feature space (integral average velocity and height of each cell face); Prediction of wall slopes (left), prediction of face slopes (middle), prediction of face velocity in x-direction (right)

Chapter 5

Wall Modeling in 2D Flat Plate

5.1 Simulation Setup

5.1.1 Flow and Boundary Conditions

In this section, the trained NN models will be applied to a 2D flat plate case[1]. The geometry is shown in Figure 5.1, and the detailed boundary conditions are also indicated. The Reynolds number (Re_L is used in the NASA case, but Re_x is used in this project.) 5 million is for the plate length of $1m$. However, a $2m$-plate will be used in this case, and therefore the actual Reynolds number is 10 million that is the same as the Reynolds number in Section 4. The Mach number is mentioned as 0.2 in Figure 5.1, but the inlet velocity is slower than one third of the speed of sound. Thus, the simulation in this work will be incompressible.

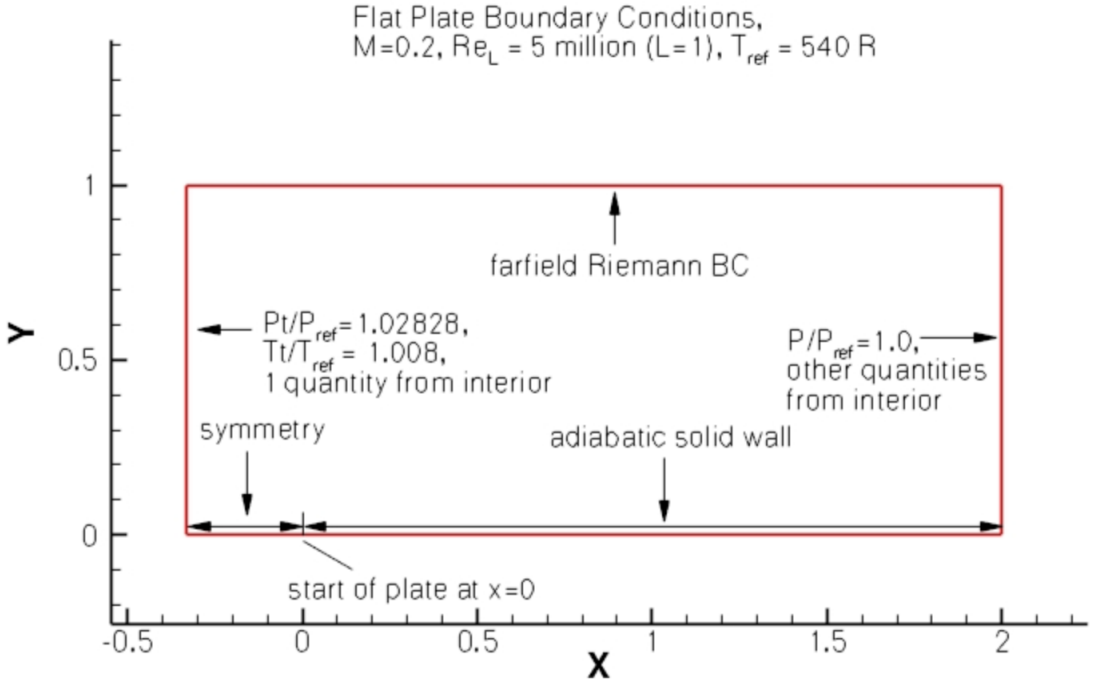

Figure 5.1: A 2D flat plate geometry[1]

Before the starting point of the plate, there is a small space for flow development from the inlet.

[1] `https://turbmodels.larc.nasa.gov/flatplate.html`

This is an open space that is not blocked by any objects, and thus this part will be set as the `symmetryPlane` patch. Other patches will be explained in the next subsection.

Since the flow condition in the 1D channel was imitated from this case, the inlet velocity and the molecular viscosity for this case are identical to the values in Table 4.1. Regarding the height of the control volume, it is sufficiently set as $1m$, and hence this patch will not interrupt the validity of the simulation. Here, the SA model is also introduced, which means that the same variables as those in Section 4.1.1 are involved in this simulation. However, the boundary conditions for each variable are different because two cases have different types of patches. For instance, the initial velocity condition, $69.4m/s$, can be defined to the `inlet` patch without creating any source terms in the `fvOptions` dictionary since this case is 2D. The pressure boundary condition is zero pressure gradient.

5.1.2 Mesh Generation

The mesh of this flat plate case is designed for various y^+ values, and it looks similar to the 1D case mesh shown in Figure 4.2 but has one dimension more. For the small space before the plate starts, 96 cells in x-direction and 385 cells in y-direction, whereas 449 cells in x-direction and 385 cells in y-direction for the $2m$ plate. The variation of y^+ is determined by various values of grading. For example, the grading factor of 50000 is for $y^+ = 0.05$, the factor of 950 is for $y^+ = 2$, and the factor of 5 is for $y^+ = 100$ as indicated in Table 5.1. This indicates that the mesh near the wall is more resolved if the grading factor is larger. For the flat plate case, total eight y^+ values will be involved that correspond to 0.05, 1, 2, 5, 10, 30, 50, and 100. The y^+ values for mesh setting are based on the standard wall function. The actual y^+ values without wall functions will be smaller than the values with wall functions because the linear interpolation is used for the numerical slopes for the case without wall functions, which leads to underestimation of skin friction.

y^+	`Grading`
0.05	50000
1	2200
2	950
5	300
10	130
30	30
50	15
100	5

Table 5.1: Mesh setting for a flat plate case

With regards to boundary patches, the `inlet` patch and the `outlet` patch have `patch` type patches because the actual flow comes and goes through these boundaries. The `topWall` patch has the `wall` type patch with a slip condition so that this location would not interrupt the simulation. The small space before the plate is called `symmetry` in the case, and this space has the `symmetryPlane` type patch as explained above. Finally, the `bottomWall` patch also has the `wall` type but with a non-slip condition so that turbulent behavior at the wall can be investigated.

5.1.3 Related Coefficients

For the flat plate case, the performance of the trained ML models can be determined by calculating one of the coefficients known as skin friction C_f. The formula of the skin friction is given

by

$$C_f = \frac{\tau_w}{\frac{1}{2}\rho U_\infty^2}. \tag{5.1}$$

Here, the density ρ is not considered because the simulation is incompressible. Therefore, $\rho = 1.0$ holds or it can be deleted from Equation (5.1). Then, the equation is given by

$$C_f = \frac{\tau_w}{\frac{1}{2}U_\infty^2}. \tag{5.2}$$

5.2 Results

In this section, a comparison of skin friction for four scenarios will be shown. The four scenarios are as follows.

1. No wall function
2. Standard wall function
3. Data-driven wall function with blended wall correction
4. Data-driven wall function with blended wall and first cell face correction

The third scenario is also called a wall correction case, while the fourth scenario can be called a wall/face correction case. Regarding the standard wall function, `nutUSpaldingWallFunction` is used for ν_t.

5.2.1 Comparison of Skin Friction for Different Scenarios

For $y^+ = 0.05$, 1, and 2, no correction occurs regardless of the scenarios. Hence, the skin friction graphs look the same as shown in Figures 5.2, 5.3, and 5.4.

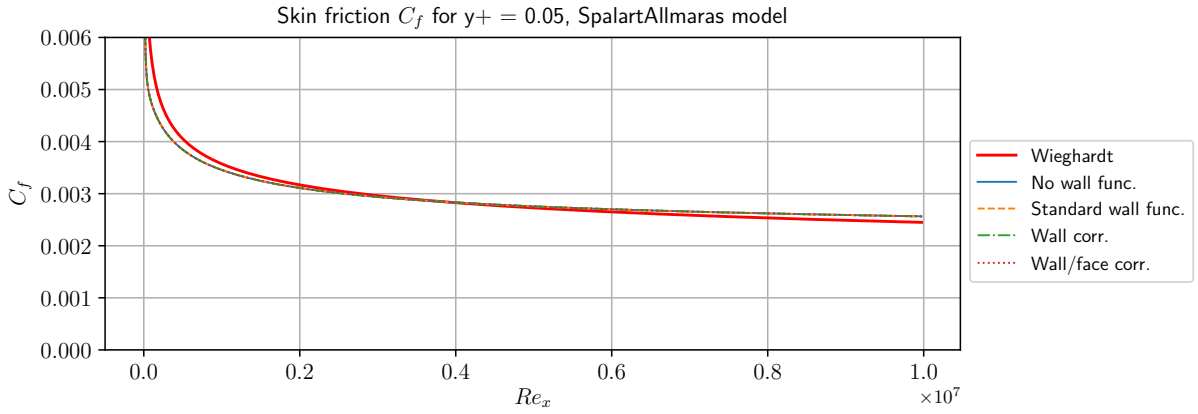

Figure 5.2: Comparison of skin friction values for $y^+ = 0.05$

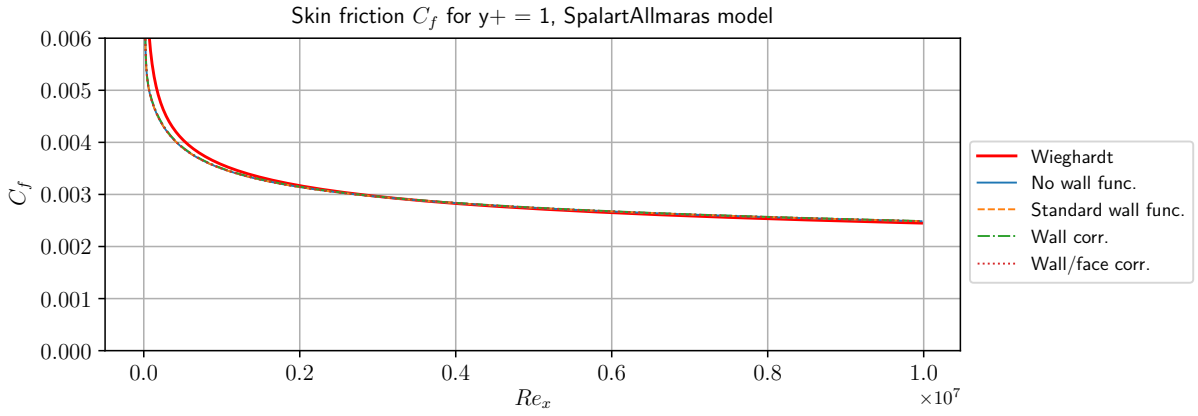

Figure 5.3: Comparison of skin friction values for $y^+ = 1$

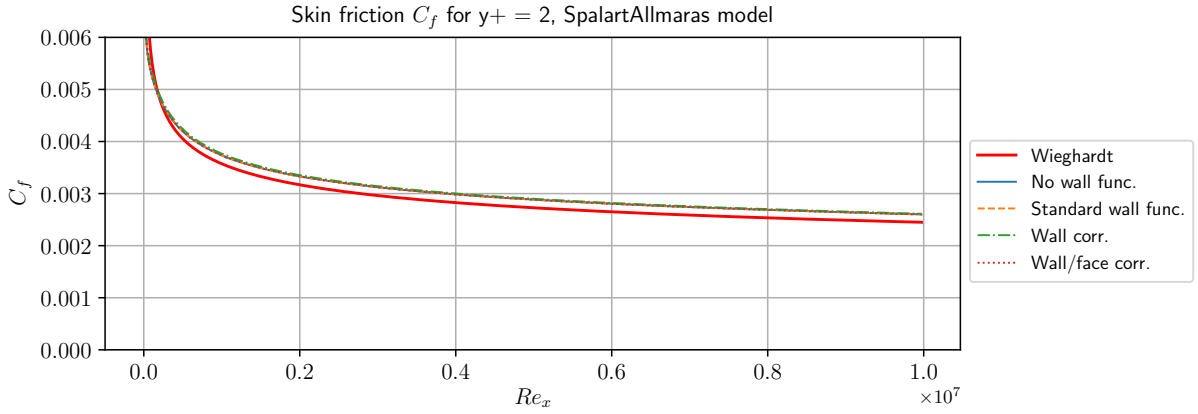

Figure 5.4: Comparison of skin friction values for $y^+ = 2$

For $y^+ = 5$, the local y^+ values are larger than 10 at the front of the plate in the case of the wall correction and the wall/face correction. Thus, these local y^+ values let the correction happen, and the correction influences an increase of skin friction at the back of the plate for the wall correction scenario as shown in Figure 5.5. For the wall/face correction scenario, the skin friction values are back to normal again.

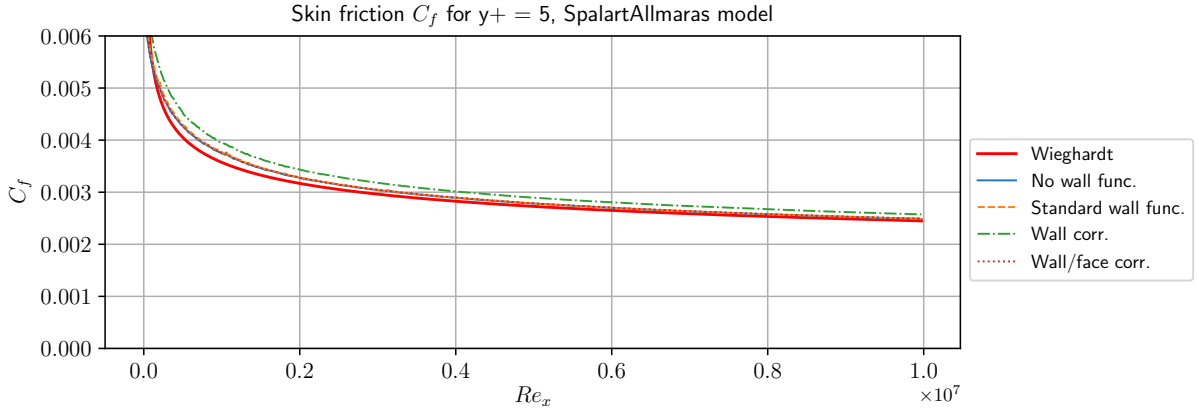

Figure 5.5: Comparison of skin friction values for $y^+ = 5$

For $y^+ = 10$, the skin friction fluctuates for the first three scenarios, but the last scenario that is the wall/face correction shows the stable behavior as shown in Figure 5.6.

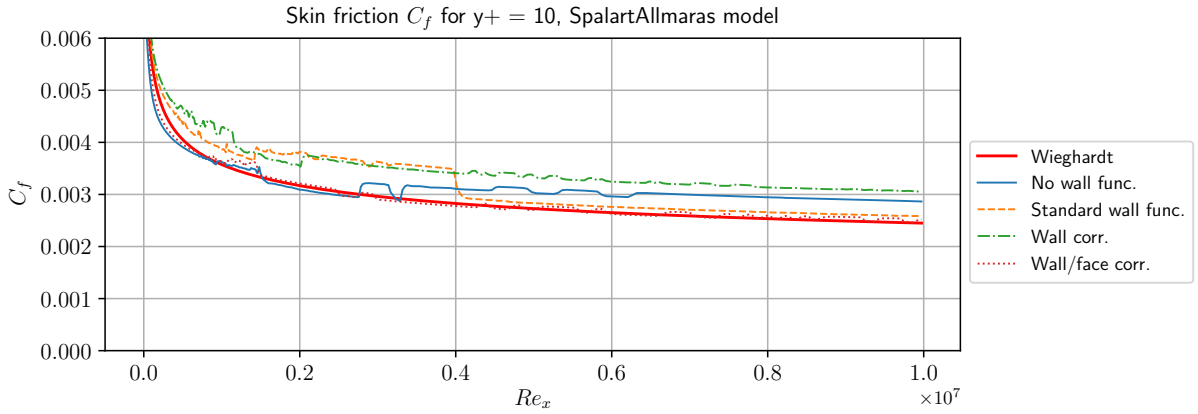

Figure 5.6: Comparison of skin friction values for $y^+ = 10$

From $y^+ = 30$, the scenario without wall functions shows very bad estimations, whereas the other scenarios yield the fine results as shown Figures 5.7, 5.8, and 5.9. Particularly, the wall correction scenario shows the superior result except at the very front of the plate. However, there is a kink at the front of the plate although the first cell face correction is excluded for the higher y^+.

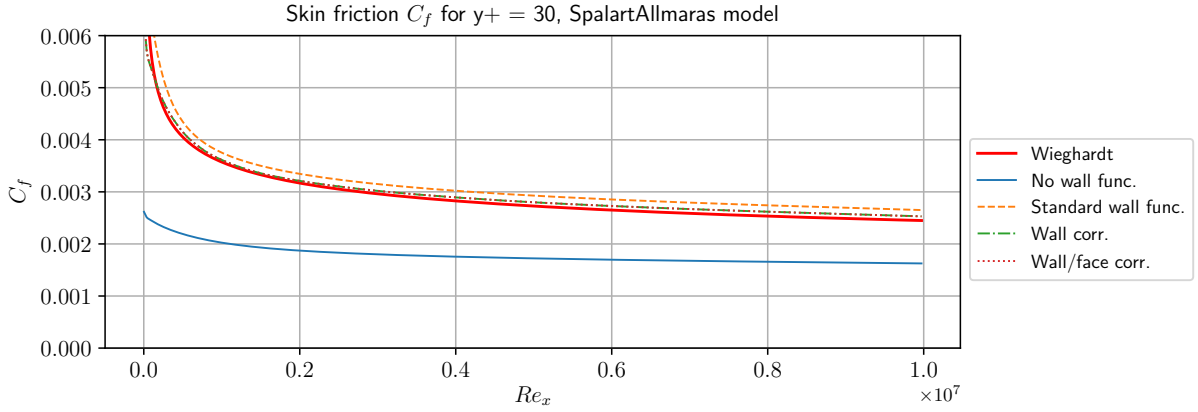

Figure 5.7: Comparison of skin friction values for $y^+ = 30$

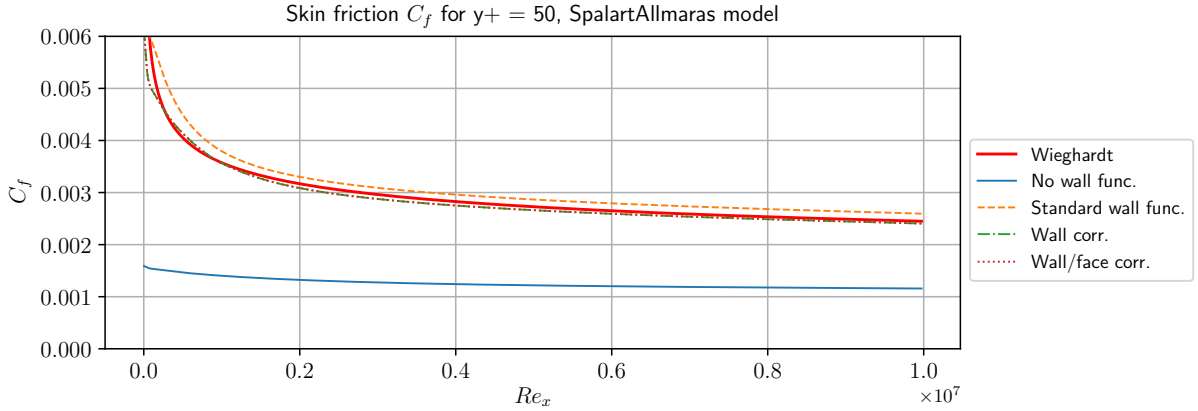

Figure 5.8: Comparison of skin friction values for $y^+ = 50$

The wall/face correction scenario is basically the same as the wall correction scenario for $y^+ \geq 35$ because the y^+ blending turns the face correction off for that region.

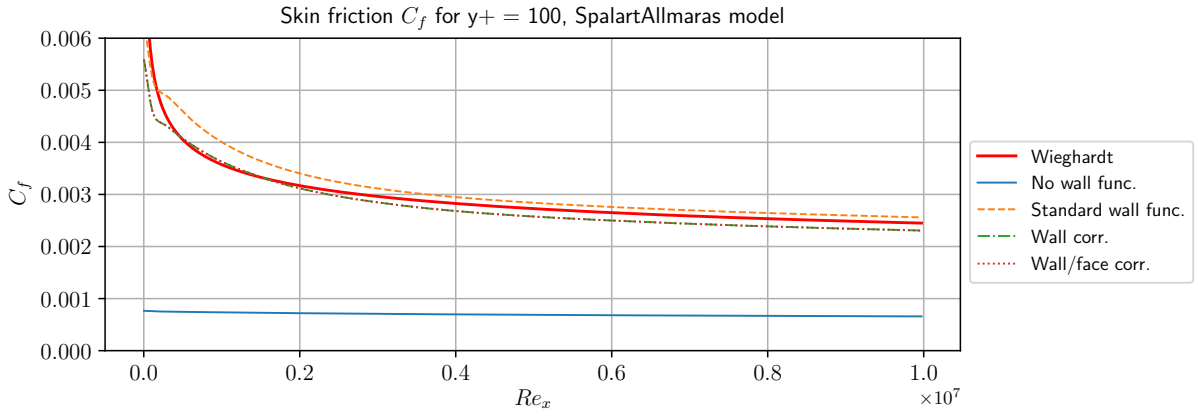

Figure 5.9: Comparison of skin friction values for $y^+ = 100$

5.2.2 Comparison of Skin Friction for Different y^+

The skin friction values for the first scenario that corresponds to no wall function for each y^+ are depicted in Figure 5.10. The case of no wall function cannot capture the behavior of the reference value for the higher y^+. Moreover, there is a small discrepancy at the buffer layer, $y^+ = 10$.

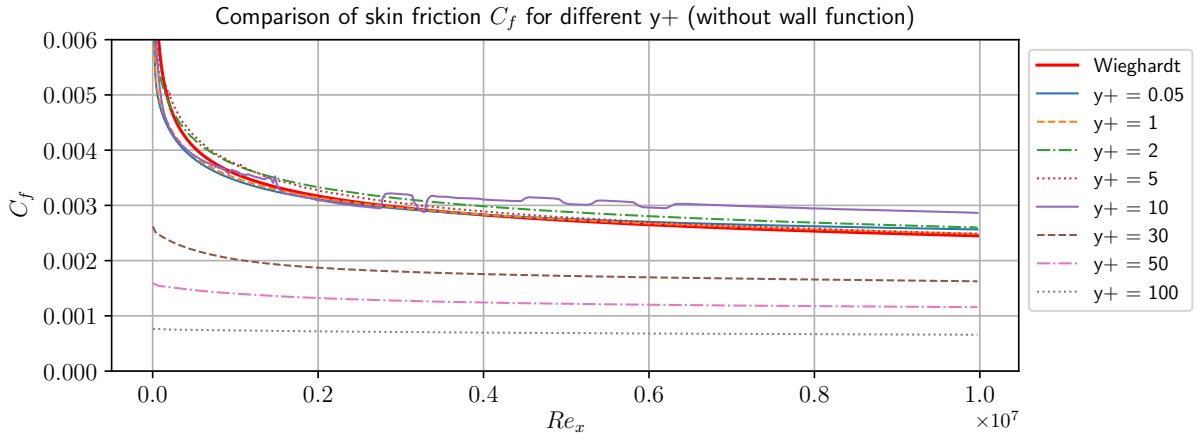

Figure 5.10: Skin friction for the no wall function scenario

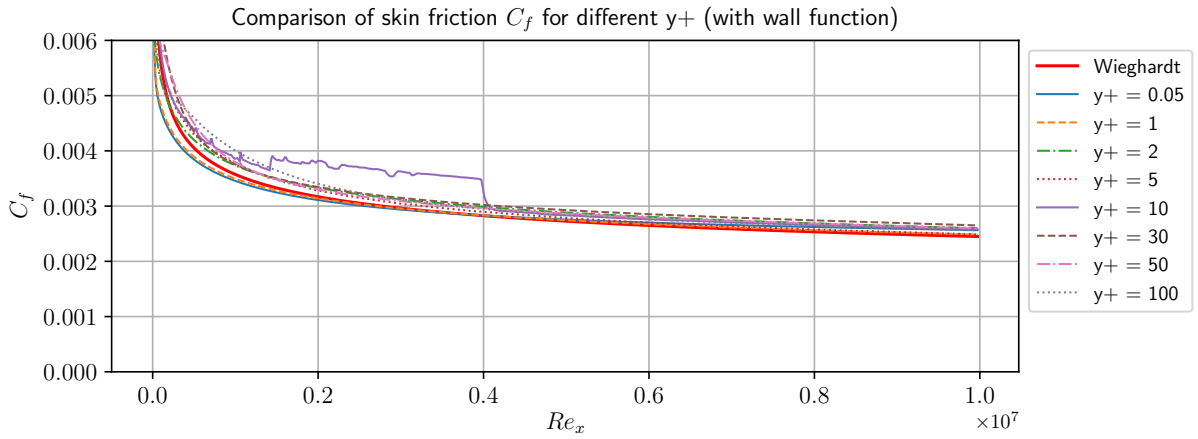

Figure 5.11: Skin friction for the standard wall function scenario

The second scenario that is with the standard wall function yields the results as shown in Figure 5.11. The skin friction value looks well except at $y^+ = 10$, and there are small differences from the reference value at the front of the plate.

The third scenario is the data-driven approach with the blended wall correction, and the result is shown in Figure 5.12. The blended wall correction works well, but there is still an outlier case at $y^+ = 10$.

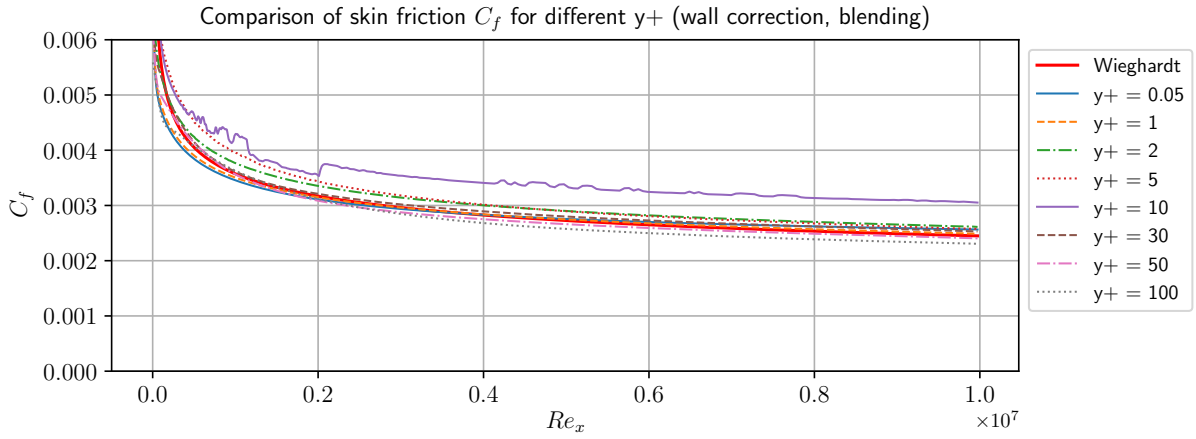

Figure 5.12: Skin friction for the data-driven wall function with wall correction scenario

The result of the last scenario that corresponds to the data-driven wall function with wall and face correction case is shown in Figure 5.13.

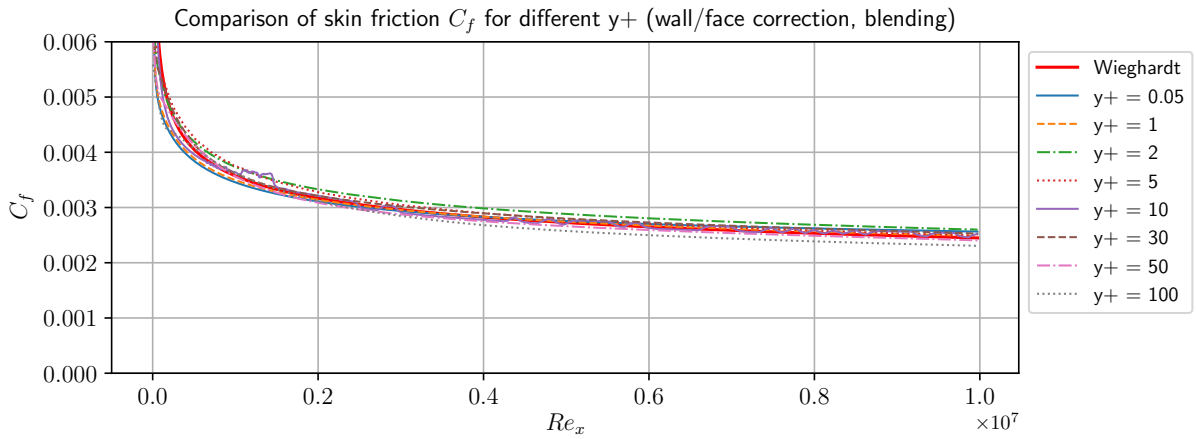

Figure 5.13: Skin friction for the data-driven wall function with wall/face correction scenario

This approach mitigates the discrepancy at $y^+ = 10$ and are almost mesh-independent. However, the distribution of the skin friction values should be investigated to determine the performance of the correction. Therefore, Figure 5.14 depicts the distribution of the skin friction for three scenarios by employing the standard deviation σ. The graph shows the range of 2σ which has 95% of the confidence interval. Here, the first scenario is excluded because the skin friction values are spread for higher y^+. As shown in the figure, the data-driven wall function with wall and face correction yields the best performance amongst the other cases. the data-driven wall function with wall correction gives the moderate performance, but the performance is less better

than that of the standard wall function case.

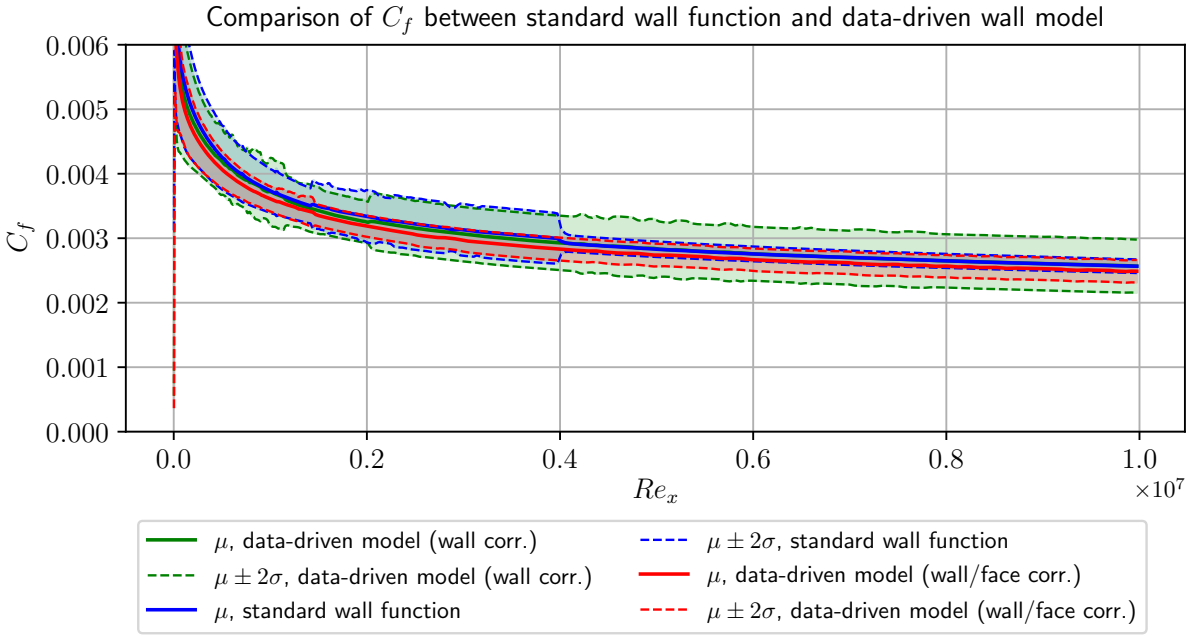

Figure 5.14: Distribution of skin friction for each scenario

There is one more method to evaluate the performance that corresponds to the mean value of the standard deviation range for each scenario. Table 5.2 indicates the representative mean value and standard deviation for each scenario, which can show the performance of each model or wall function in one figure. According to this table, the wall correction scenario has slightly broader range of the confidence interval than the standard wall function scenario. The wall/face correction scenario yields the smallest range of the confidence interval.

Scenario	$\mu_{rep} \pm 2\sigma_{rep}$
No wall func.	0.002718 ± 0.002234
Standard wall func.	0.003674 ± 0.000429
Wall corr.	0.003606 ± 0.000532
Wall/face corr.	0.003505 ± 0.000284

Table 5.2: Representative mean value with the confidence interval for the flat plate case

Chapter 6

Application of Modeling to NACA-0012 Airfoil

6.1 Simulation Setup

6.1.1 Flow and Boundary Conditions

In this section, the trained NN models will be applied to a 2D airfoil case. The geometry is shown in Figure 6.1[1]. However, the boundary conditions used here are different from the original reference case. First of all, the Reynolds number (Re_c) for the chord length of $1m$ is 3 million. The original Mach number is 0.15, and then the inlet velocity is also slower than one third of the speed of sound. Thus, the airfoil simulation will be incompressible. Since the simulation is incompressible, the speed of sound and the Mach number need not to be considered. Therefore, the molecular viscosity and the inlet velocity are set to be $1.0 \cdot 10^{-5} m^2/s$ and $30.0 m/s$, respectively.

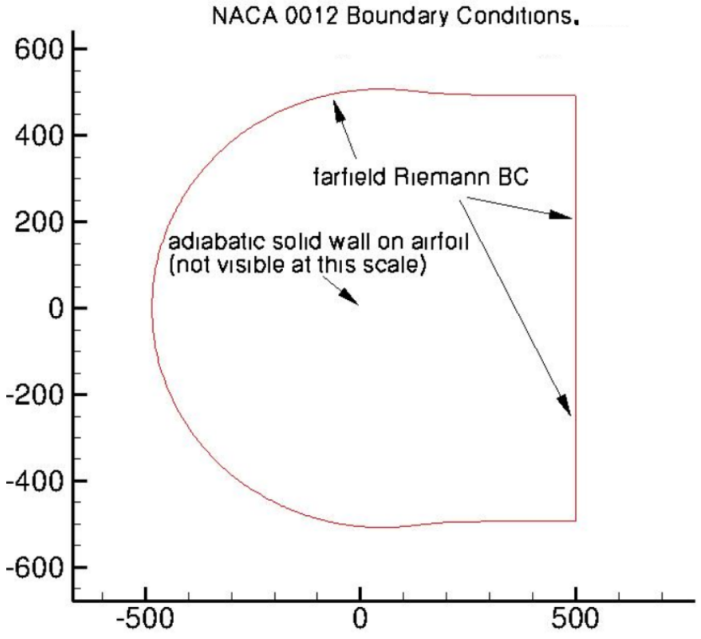

Figure 6.1: Boundary conditions for airfoil case[1]

[1] https://turbmodels.larc.nasa.gov/naca0012_val.html

As shown in Table 6.1, the angle of attack is 0°, the chord length of the airfoil is $1.0m$, and the SA model is used. The pressure boundary condition is also zero pressure gradient.

Variable	Value
U_∞	$30.0m/s$
ν	$1.0 \cdot 10^{-5} m^2/s$
c	$1.0m$
Angle of Attack	0°
Re_c	$3 \cdot 10^6$
Model	Spalart-Allmaras

Table 6.1: Flow condition in a 2D airfoil case

6.1.2 Mesh Generation

The mesh of the airfoil case is also designed for various y^+ values, but it is much more complicated due to the airfoil shape itself that is divided into several sections. The mesh in OpenFOAM is based on a `plot3d` format, but the `blockMesh` format is also available used for the other project[2]. The mesh of the `blockMesh` format is employed for this work.

For y^+ to be modified, only three sections are to be revised among all the mesh regions in the control volume, which correspond to `yMid`, `yGradAirfoil`, and `yTrail` in the `blockMeshDict` file as mentioned in Figure 6.2[2]. For the airfoil case, total eight y^+ values will also be involved, but they correspond to 0.05, 1, 2, 3.5, 5, 10, 50, and 100 because the buffer layer range for this case is a bit different from the flat plate case, which carefully needs to be investigated. Table 6.2 demonstrates how many cells (`yMid` and `yTrail`) or grading values (`yGradAirfoil`) are used to set proper y^+ in this airfoil case.

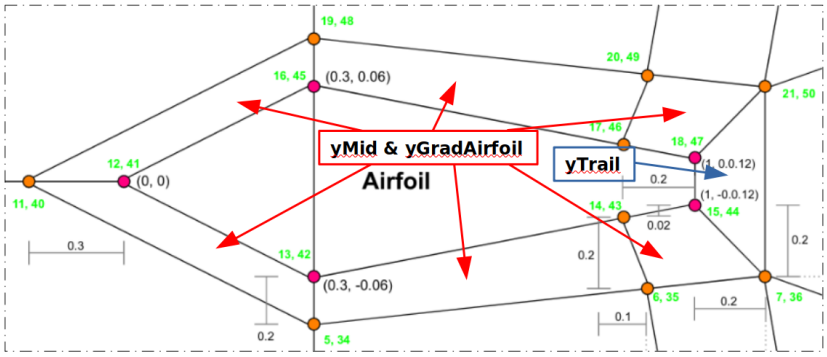

Figure 6.2: Modifiable section of the mesh near the airfoil[2]

With regards to boundary patches, the `inlet` patch and the `outlet` patch have the `patch` type patches with a freestream setting. Since all the direction of the control volume is open, the remaining patch is `airfoil` that has the `wall` patch type.

On the other hand, there is a stability issue for the smaller y^+ in the airfoil case. The velocity near the wall becomes exceedingly fast at the early time steps, approximately until 100 iterations. Afterward, due to the rapid velocity, the ratio of the ML model slope to the numerical slope becomes immensely large, which leads to the simulation crash. Therefore, the numerical slopes at the wall and the first cell face are limited in this simulation before the velocity reduces to a valid range.

[2] `https://github.com/AndreWeiner/naca0012_shock_buffet`

y^+	yGradAirfoil	yMid	yTrail
0.05	8000	200	30
1	350	200	30
2	150	200	40
3.5	65	200	40
5	50	200	40
10	15	200	40
50	1	180	40
100	1	100	40

Table 6.2: Mesh setting for a 2D airfoil case

6.1.3 Related Coefficients

For the airfoil case, the performance of the ML model can also be determined by skin friction C_f. In addition, pressure coefficient C_p is investigated that is given by

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho U_\infty^2}. \tag{6.1}$$

The density ρ is not considered here because the simulation is incompressible. Therefore, $\rho = 1.0$ holds or it can be deleted from Equation (6.1) which is given by

$$C_p = \frac{p - p_\infty}{\frac{1}{2}U_\infty^2}. \tag{6.2}$$

Unlike skin friction, pressure coefficient is virtually mesh-independent since pressure is much less sensitive to the location than wall shear stresses. Hence, this coefficient will be calculated to examine if the simulation is valid.

6.2 Results

In this section, a comparison of skin friction for four scenarios that are the same as the flat plate case will be shown. For the standard wall function, `nutUSpaldingWallFunction` is also used here. When it comes to the reference data for skin friction, the data at $Re_c = 3 \cdot 10^6$ is not available. Hence, the simulation data of the finest mesh setting that corresponds to $y^+ = 0.05$ at $Re_c = 3 \cdot 10^6$ is used for the airfoil case. For pressure coefficient, the experimental data[3] is available so that the data can be indicated in the plots. Since the angle of attack of this simulation is 0°, the results at the upper part and the lower part of the airfoil will mostly be the same if the simulation is sufficiently stable. The line style of the graph for the upper airfoil is mentioned in all the figures, and all the dotted lines with corresponding colors are for the lower airfoil. Therefore, for the unstable case, the dotted lines that correspond to the lower part of the airfoil are shown in the figures although the angle of attack is 0°.

6.2.1 Comparison of Pressure Coefficient for Different Scenarios

The pressure coefficient graphs show the consistent behavior and well fit to the experimental data regardless of the scenarios as depicted in Figures 6.3, 6.4, 6.5, 6.7, 6.8, 6.9, and 6.10. However, a strange behavior is found for the wall correction scenario and the wall/face correction scenario at $y^+ = 3.5$ as shown in Figure 6.6. Thus, it can be expected that the skin friction values for the correction methods cannot capture the behavior of the reference graph for the case of $y^+ = 3.5$.

[3] `https://turbmodels.larc.nasa.gov/NACA0012_validation/CP_Gregory_expdata.dat`

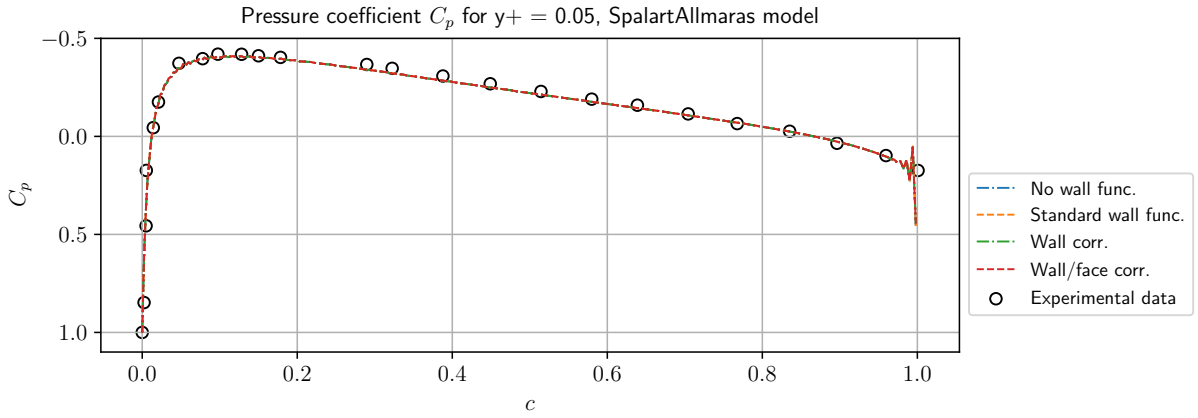

Figure 6.3: Comparison of pressure coefficient for $y^+ = 0.05$

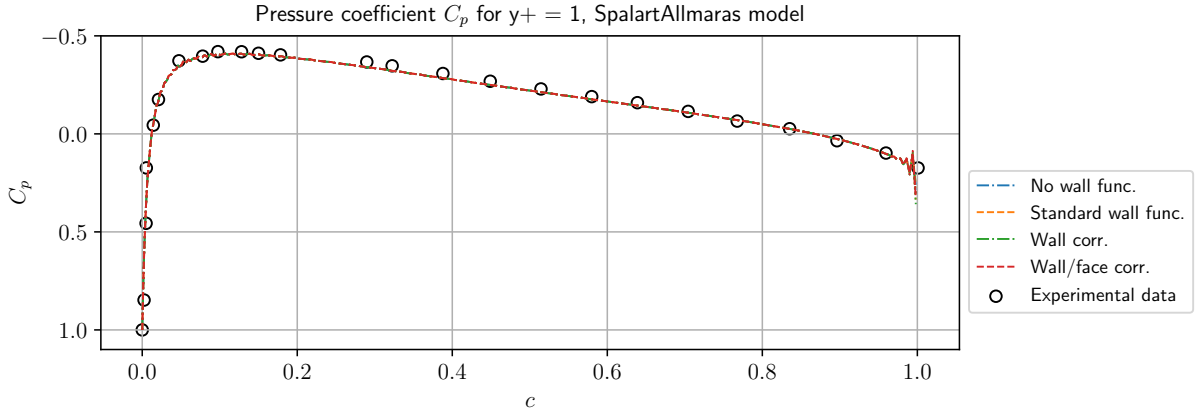

Figure 6.4: Comparison of pressure coefficient for $y^+ = 1$

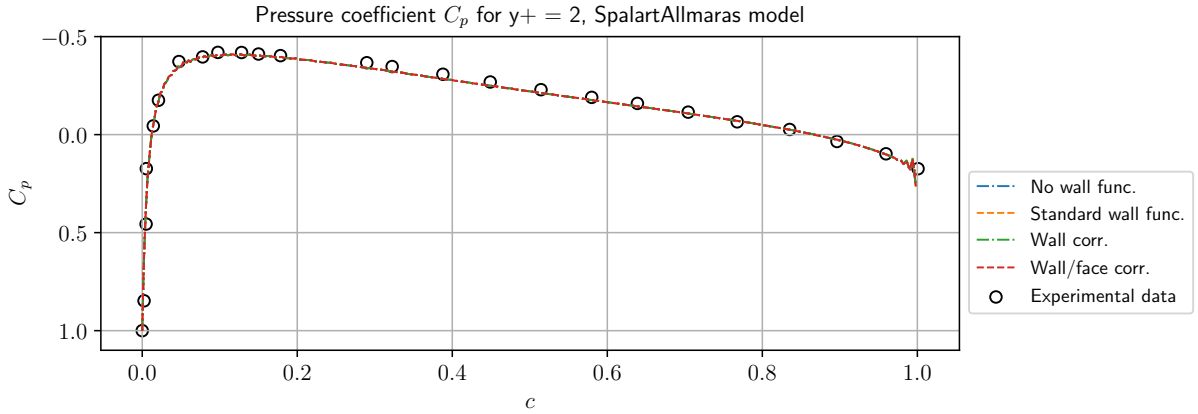

Figure 6.5: Comparison of pressure coefficient for $y^+ = 2$

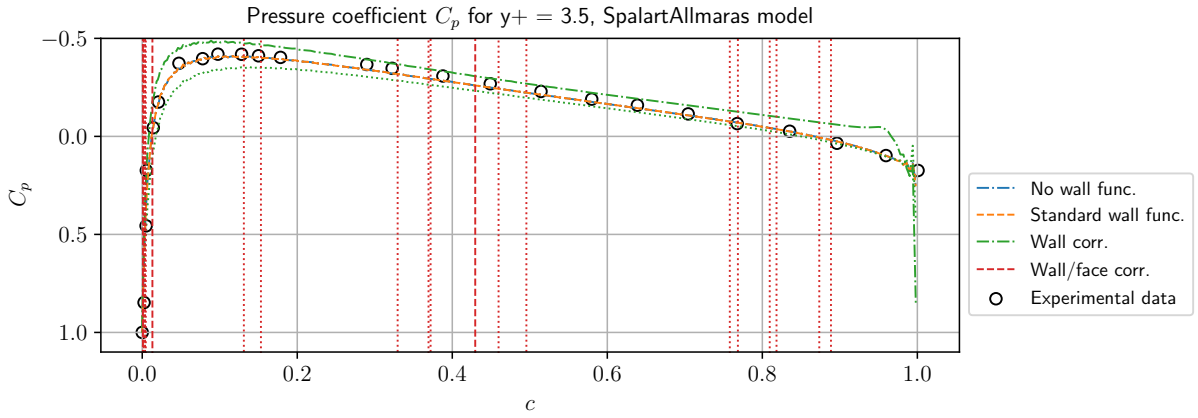

Figure 6.6: Comparison of pressure coefficient for $y^+ = 3.5$

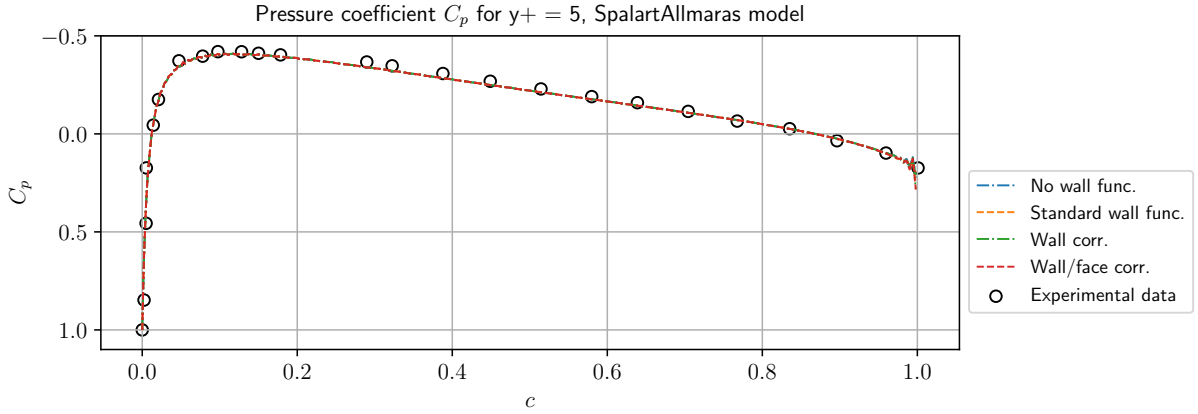

Figure 6.7: Comparison of pressure coefficient for $y^+ = 5$

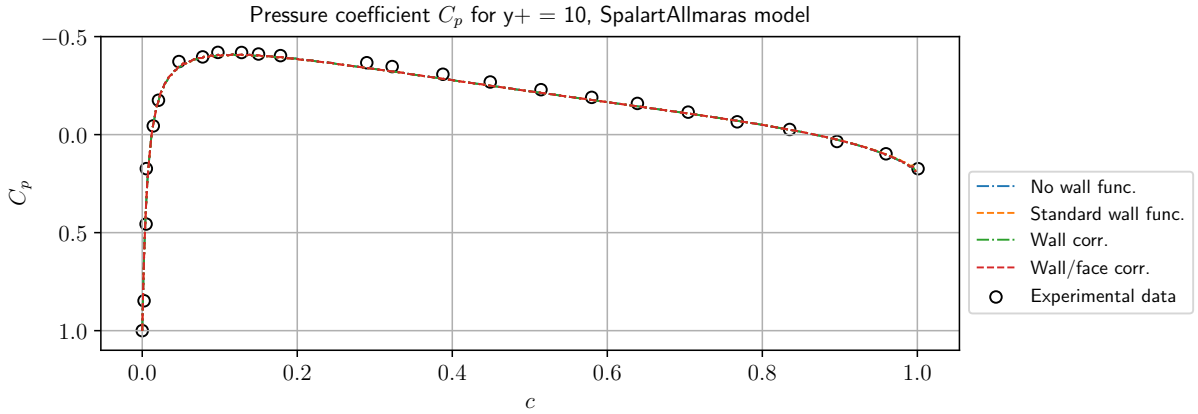

Figure 6.8: Comparison of pressure coefficient for $y^+ = 10$

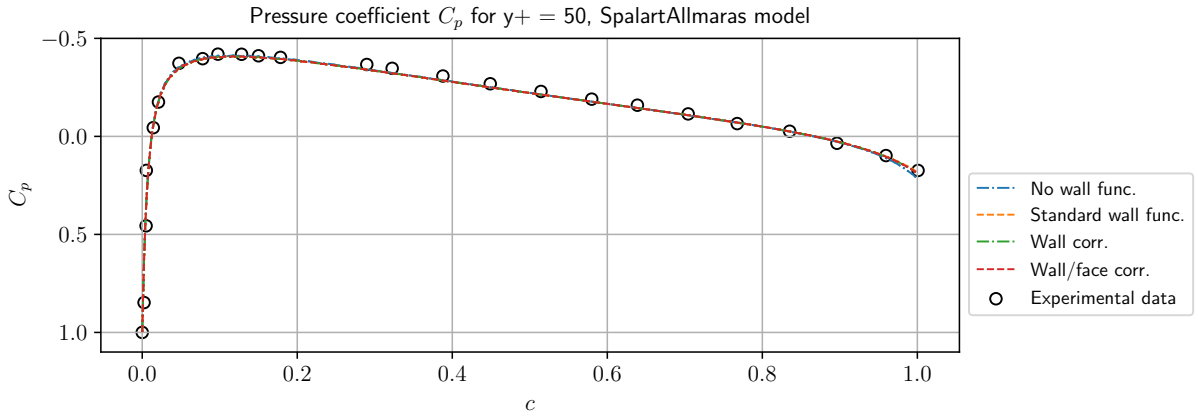

Figure 6.9: Comparison of pressure coefficient for $y^+ = 50$

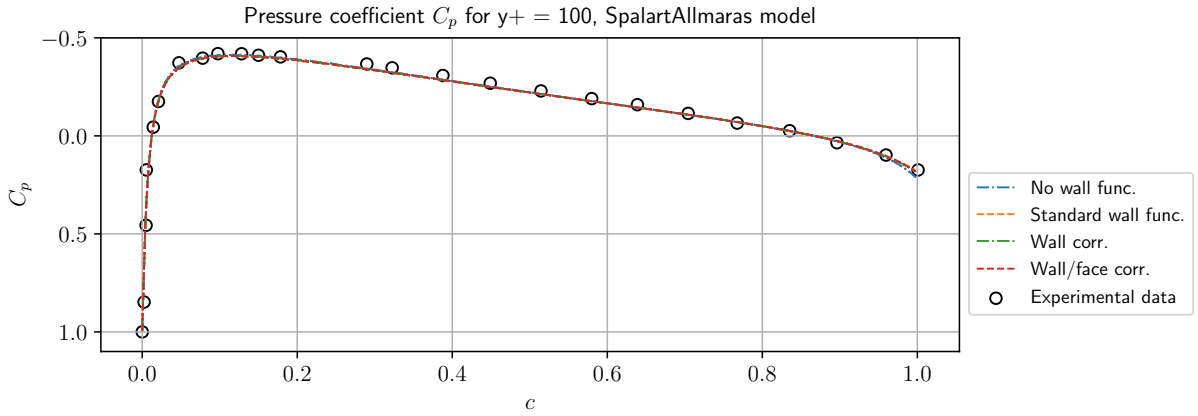

Figure 6.10: Comparison of pressure coefficient for $y^+ = 100$

6.2.2 Comparison of Skin Friction for Different Scenarios

For $y^+ = 0.05$ and 1, no correction occurs regardless of the scenarios. Hence, the skin friction graphs look the same as shown in Figures 6.11 and 6.12. For $y^+ = 2$ shown in Figure 6.13, there is also no correction, but the slight discrepancy between the simulation and the reference data exists due to the turbulence model itself.

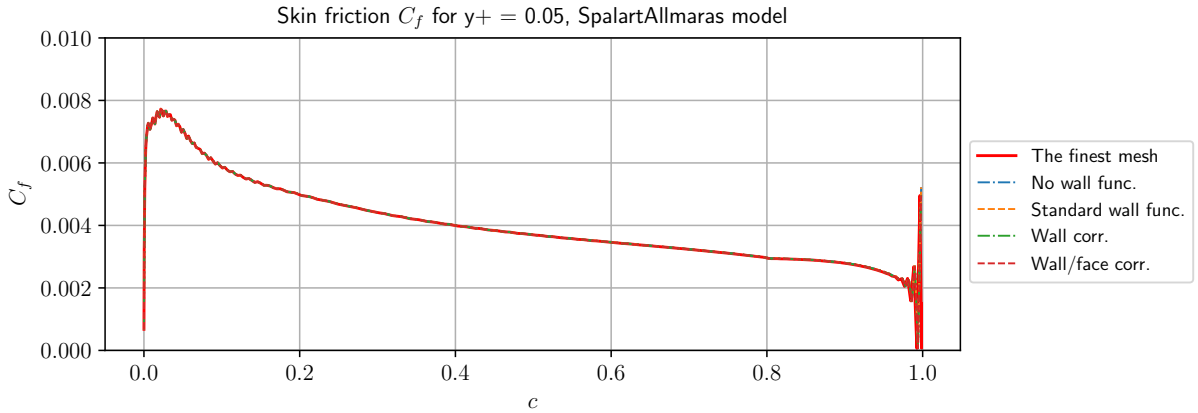

Figure 6.11: Comparison of skin friction values for $y^+ = 0.05$

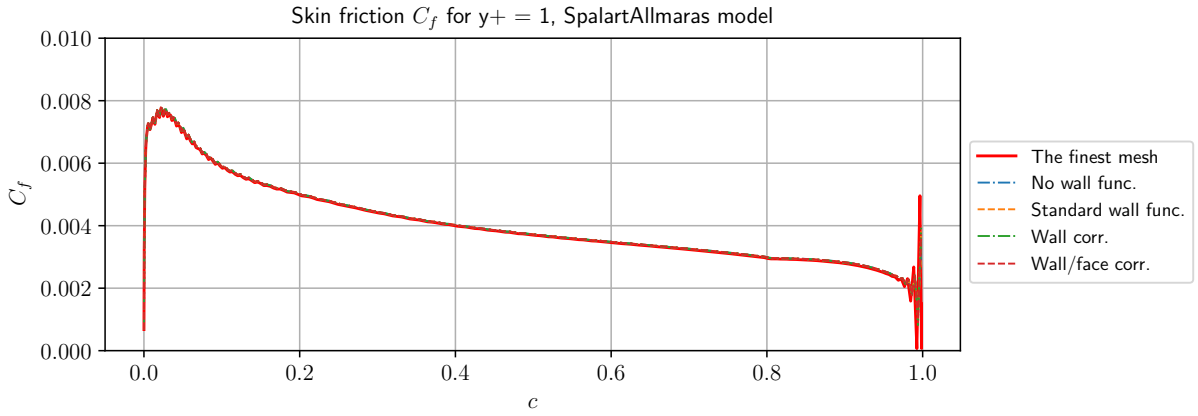

Figure 6.12: Comparison of skin friction values for $y^+ = 1$

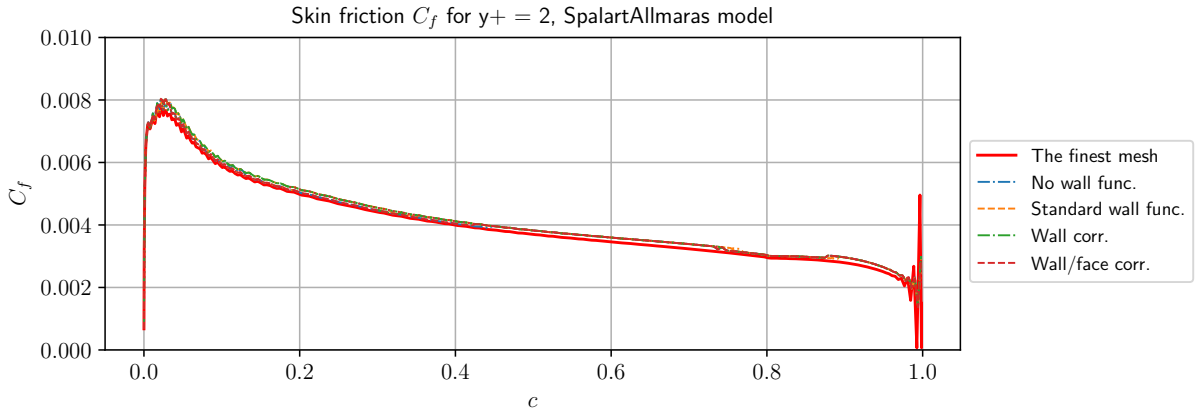

Figure 6.13: Comparison of skin friction values for $y^+ = 2$

For $y^+ = 3.5$, fluctuations are generally found in Figure 6.14. The first two scenarios (without/with wall functions) show the small fluctuations and estimate the skin friction a bit larger. However, the wall correction case illustrates a really unstable behavior particularly at the end of the airfoil. The wall/face correction case is even depicted out of the range of the plot.

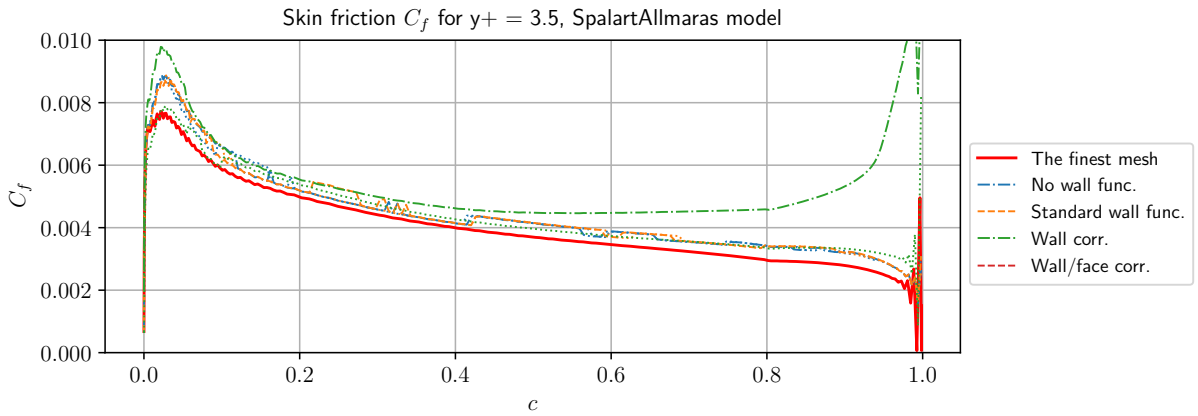

Figure 6.14: Comparison of skin friction values for $y^+ = 3.5$

Unlike the flat plate case, the local y^+ values do not exceed 10 before $y^+ = 10$, and therefore the correction does not occur at $y^+ = 5$. As shown in Figure 6.15, there are small fluctuations

for all the scenarios, but the graphs are much more stable than the results at $y^+ = 3.5$.

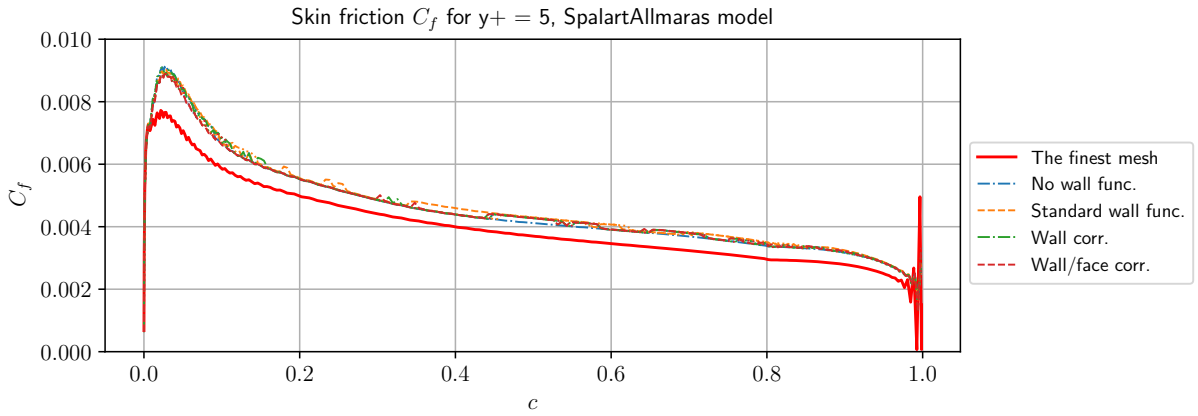

Figure 6.15: Comparison of skin friction values for $y^+ = 5$

As indicated in Figure 6.16, three scenarios that correspond to no wall function, wall correction, and wall/face correction can capture the behavior of the reference graph compared to the standard wall function scenario.

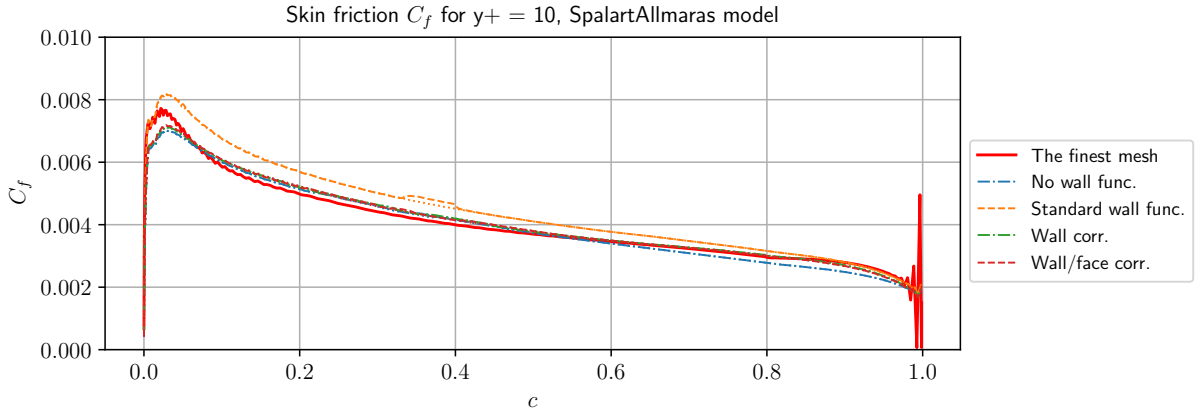

Figure 6.16: Comparison of skin friction values for $y^+ = 10$

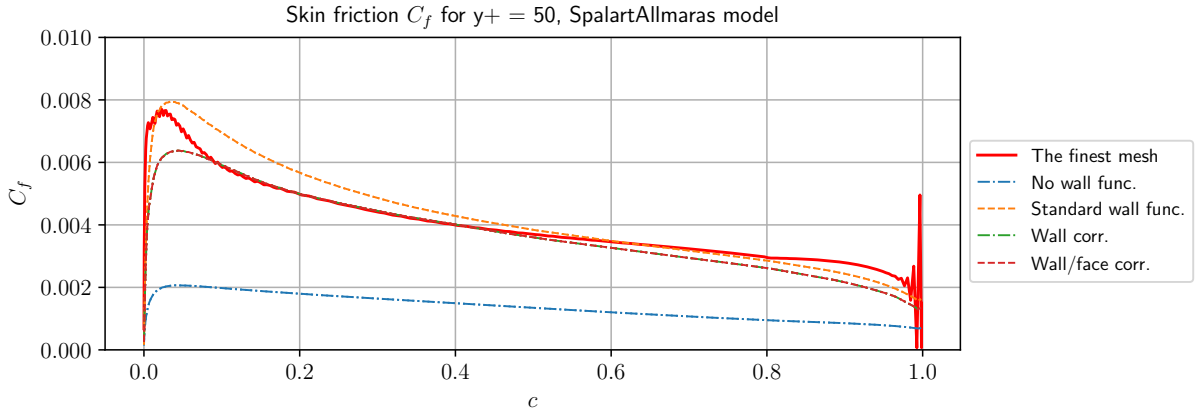

Figure 6.17: Comparison of skin friction values for $y^+ = 50$

From $y^+ = 50$ shown in Figures 6.17 and 6.18, the scenario without wall functions for the airfoil case also shows very bad estimations, whereas the other scenarios yield the fine results. However,

the wall correction scenario and the wall/face correction scenario show high discrepancies from the reference graph. The y^+ blending turns the face correction off for the wall/face correction scenario for $y^+ \geq 35$ also in the airfoil case.

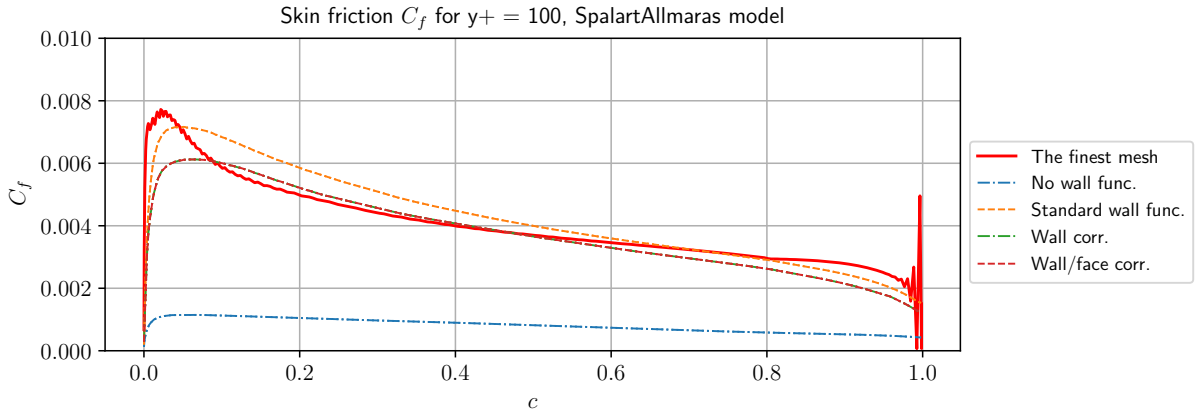

Figure 6.18: Comparison of skin friction values for $y^+ = 100$

6.2.3 Comparison of Skin Friction for Different y^+

The skin friction values for the no wall function scenario for each y^+ are depicted in Figure 6.19. This scenario cannot capture the behavior of the reference graph for the higher y^+. There are discrepancies from the reference data at $y^+ = 3.5$ and 5.

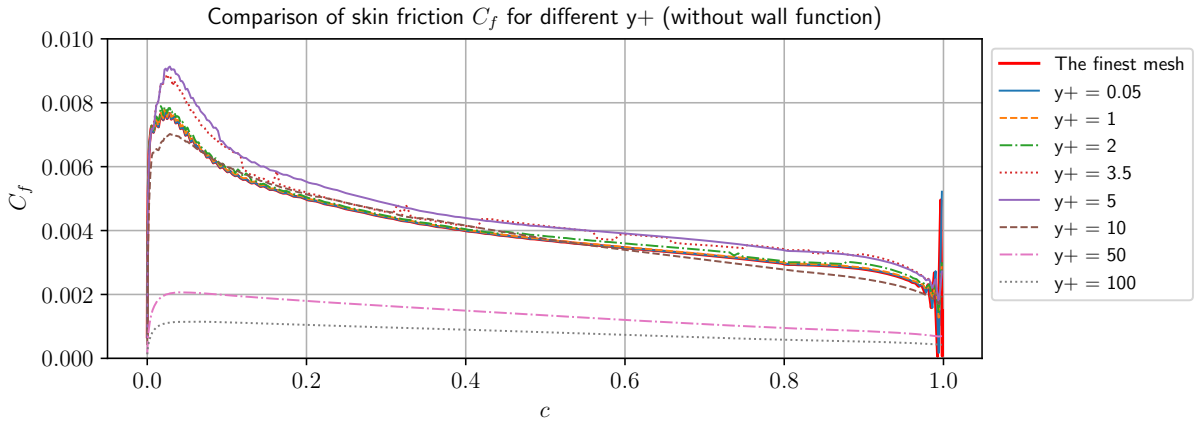

Figure 6.19: Skin friction for the no wall function scenario (airfoil)

The second scenario that is with the standard wall function yields the results as shown in Figure 6.20. The skin friction values for higher y^+ are estimated greater than the reference.

The result of the wall correction scenario is shown in Figure 6.21. The skin friction values are estimated worse than the standard wall function case. Especially, large discrepancies generally appear at the front of the airfoil. For $y^+ = 3.5$, the skin friction estimation is unstable.

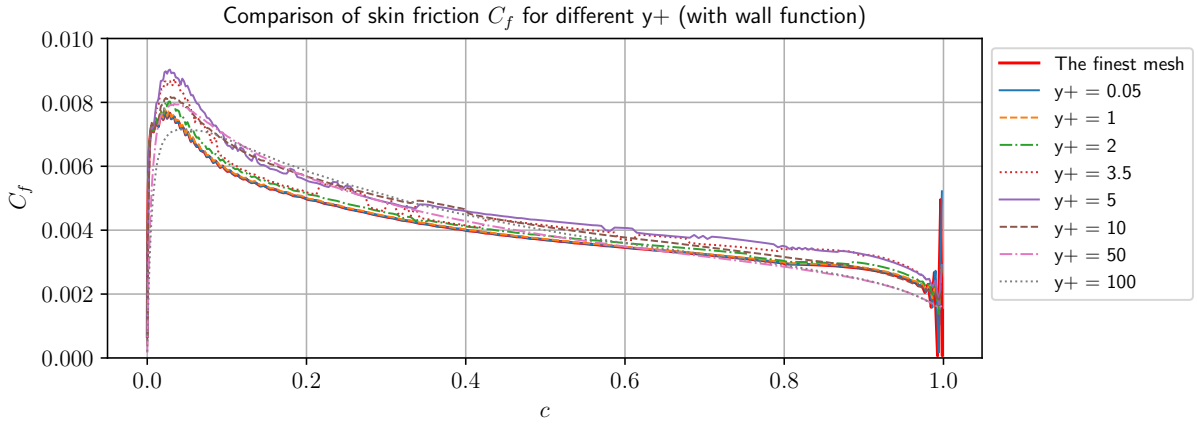

Figure 6.20: Skin friction for the standard wall function scenario (airfoil)

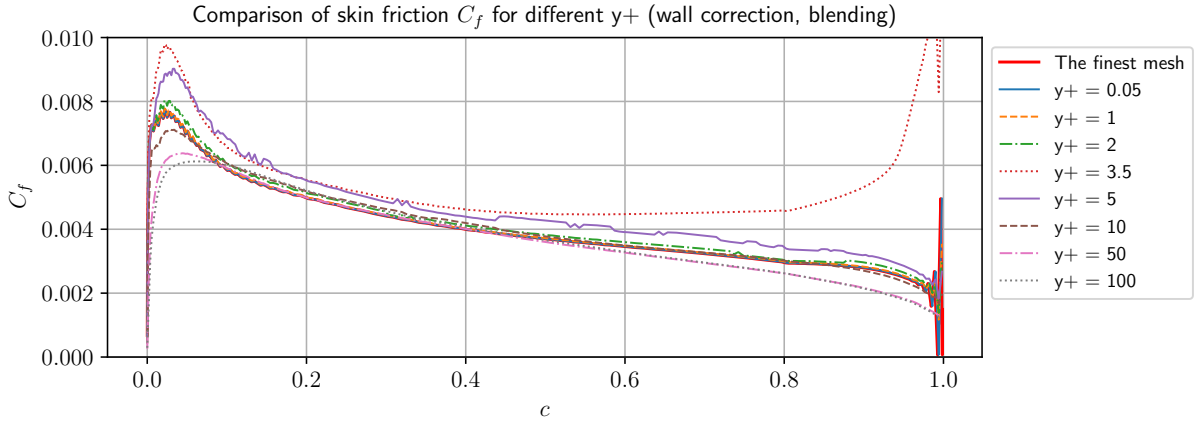

Figure 6.21: Skin friction for data-driven wall function with wall correction case (airfoil)

The result of the last scenario that corresponds to the wall/face correction case is shown in Figure 6.22. Basically, the same problem as the third scenario occurs here, and the skin friction estimation is even much more unstable for $y^+ = 3.5$.

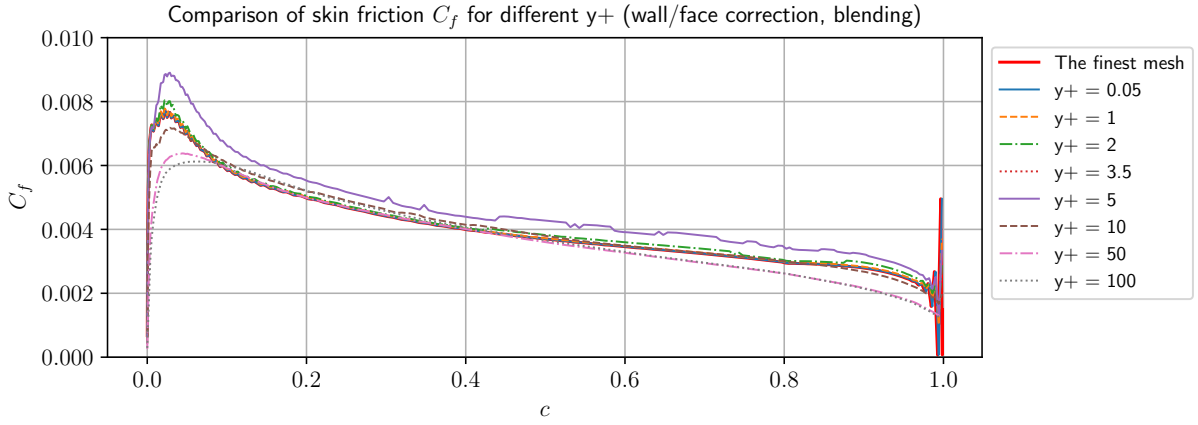

Figure 6.22: Skin friction for the data-driven wall function with wall/face correction scenario (airfoil)

The distribution of the skin friction values are also to be investigated for the airfoil case as indi-

cated in Figure 6.23. The $y^+ = 3.5$ case for all the scenarios is excluded because the simulations of the wall correction and the wall/face correction scenarios diverge, which leads to an enormous increase of the mean value for two scenarios. As shown in the figure, the wall correction and the wall/face correction scenarios show similar performance. The performance for two scenarios is better than that for the standard wall function at the front-middle of the airfoil, but worse at the very front of the airfoil.

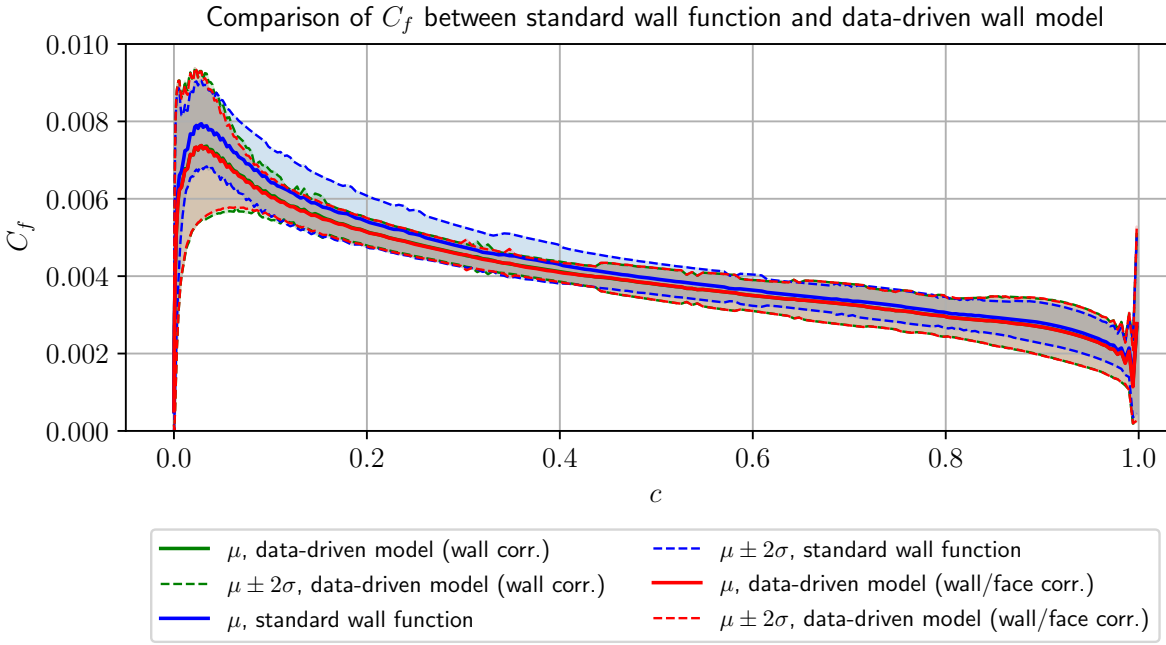

Figure 6.23: Distribution of skin friction for each scenario

Here, the representative mean and standard deviation are to be calculated since the performance comparison is unclear only with the plot. As indicated in Table 6.3, the wall correction and the wall/face correction scenarios yield slightly worse performance than the standard wall function numerically.

Scenario	$\mu_{rep} \pm 2\sigma_{rep}$
No wall func.	0.003531 ± 0.003122
Standard wall func.	0.004578 ± 0.000701
Wall corr.	0.004355 ± 0.000782
Wall/face corr.	0.004345 ± 0.000760

Table 6.3: Representative mean value with the confidence interval for the airfoil case

Chapter 7

Generalization of Wall Models for Various Reynolds Numbers

A generalization of the data-driven wall function will be demonstrated in this section. Two more flat plate cases with different Reynolds numbers will be investigated that correspond to $Re_x = 3 \cdot 10^6$ and $6 \cdot 10^6$. Since the comparison plots of skin friction for the different scenarios for these cases yield virtually the same implication, these plots are located in Appendix A.

7.1 Simulation Setup

7.1.1 Flow and Boundary Conditions

The flow settings and the boundary conditions for these two cases are basically the same as those in Section 5, but the molecular viscosity and the inlet velocity are different. Table 7.1 shows the flow conditions for the case at $Re_x = 3 \cdot 10^6$.

Variable	Value
U_∞	$15.0 m/s$
ν	$1.0 \cdot 10^{-5} m^2/s$
d	$2.0 m$
Re_x	$3 \cdot 10^6$
Model	Spalart-Allmaras

Table 7.1: Flow condition in the flat plate case at $Re_x = 3 \cdot 10^6$

Table 7.2 shows the flow conditions for the case at $Re_x = 6 \cdot 10^6$.

Variable	Value
U_∞	$30.0 m/s$
ν	$1.0 \cdot 10^{-5} m^2/s$
d	$2.0 m$
Re_x	$6 \cdot 10^6$
Model	Spalart-Allmaras

Table 7.2: Flow condition in the flat plate case at $Re_x = 6 \cdot 10^6$

7.1.2 Mesh Generation

The mesh generation method is virtually the same as Section 5.1.2. However, grading settings must be modified to keep the identical y^+ values for every flat plate case. Table 7.3 shows the corresponding mesh settings for the flat plate case at $Re_x = 3 \cdot 10^6$.

y^+	Grading
0.05	12500
1	600
2	250
5	75
10	30
30	10
50	3
100	1

Table 7.3: Mesh setting for a flat plate case at $Re_x = 3 \cdot 10^6$

Table 7.4 shows the settings for the flat plate case at $Re_x = 6 \cdot 10^6$.

y^+	Grading
0.05	20000
1	1300
2	550
5	160
10	70
30	20
50	8
100	2

Table 7.4: Mesh setting for a flat plate case at $Re_x = 6 \cdot 10^6$

7.2 Results

7.2.1 Comparison of Skin Friction for Different y^+ at $Re_x = 3 \cdot 10^6$

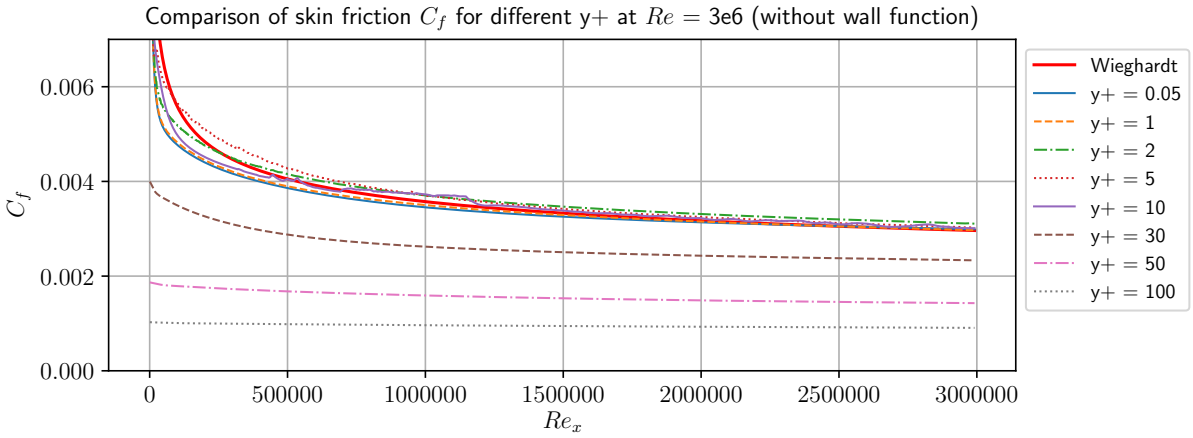

Figure 7.1: Skin friction for the no wall function scenario at $Re_x = 3 \cdot 10^6$

The skin friction values for the no wall function scenario for each y^+ are depicted in Figure 7.1.

The case without wall functions cannot capture the behavior of the reference value for $y^+ \geq 30$. However, for the other y^+, the skin friction values are sufficiently stable. There are still discrepancies at the front of the plate.

The standard wall function scenario yields the results as shown in Figure 7.2. The skin friction value fluctuates at $y^+ = 10$, and the skin friction at the other y^+ comprehensively shows larger deviations from the reference graph than the skin friction for the no wall function scenario except from $y^+ = 30$.

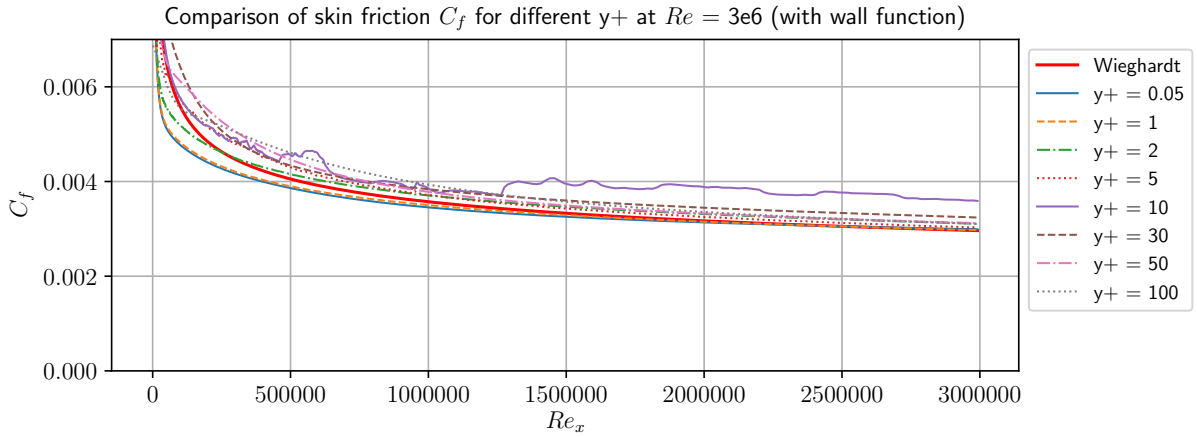

Figure 7.2: Skin friction for standard the wall function scenario at $Re_x = 3 \cdot 10^6$

The result of the wall correction scenario is shown in Figure 7.3. The wall correction yields the best results compared to the other scenarios.

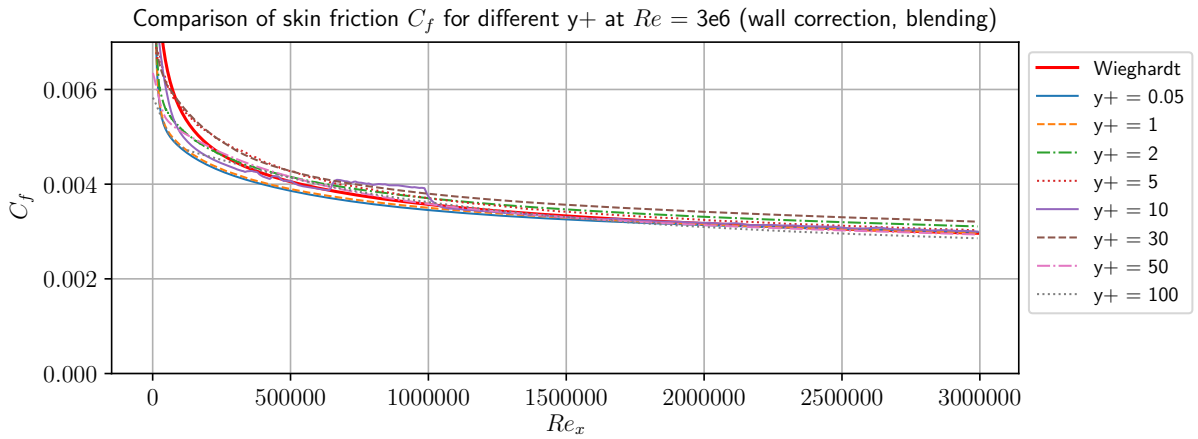

Figure 7.3: Skin friction for the data-driven wall function with wall correction scenario at $Re_x = 3 \cdot 10^6$

The result of the wall/face correction scenario is shown in Figure 7.4. This is almost the same as the previous scenario, but the skin friction for $y^+ = 30$ is overestimated at the very front of the plate.

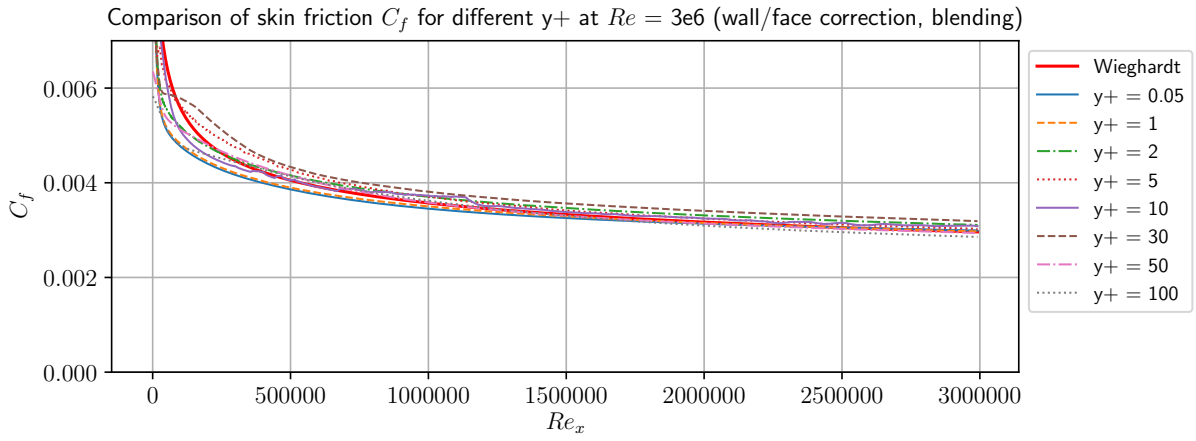

Figure 7.4: Skin friction for the data-driven wall function with wall/face correction scenario at $Re_x = 3 \cdot 10^6$

Figure 7.5 depicts the distribution of the skin friction for three scenarios except the no wall function scenario. The data-driven wall function with wall correction yields the best performance amongst the other scenarios. The performance of the data-driven wall function with wall/face correction is less better than that of the wall correction scenario.

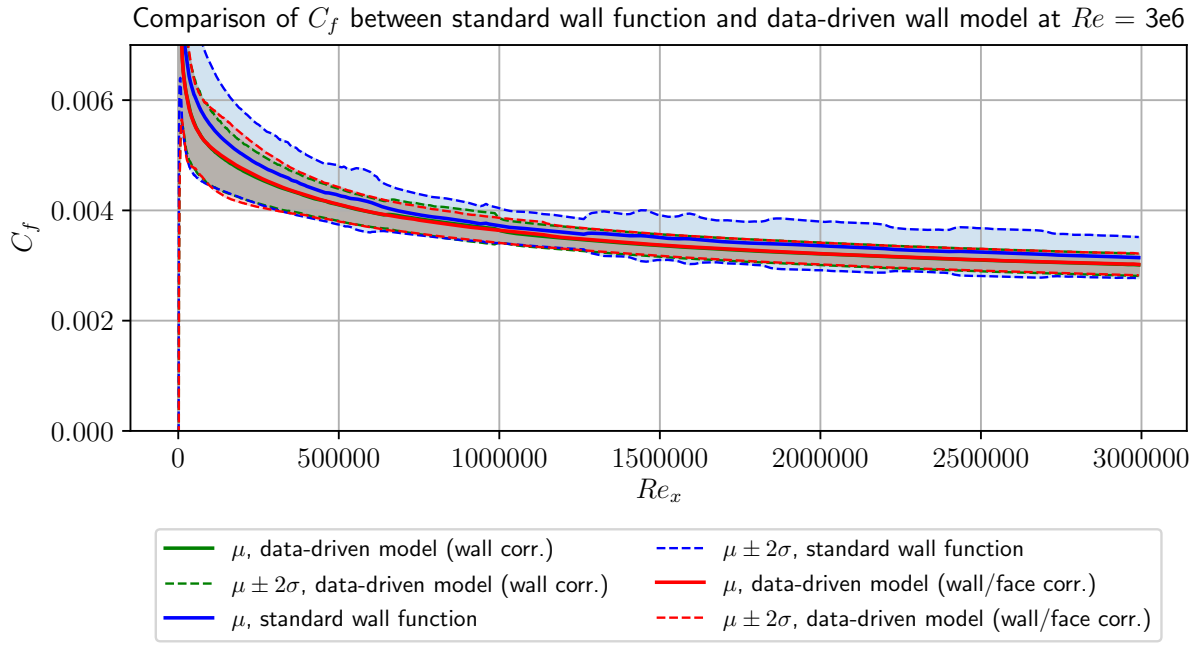

Figure 7.5: Distribution of skin friction for each scenario at $Re_x = 3 \cdot 10^6$

Table 7.5 indicates the representative mean value and standard deviation for each scenario as a numeric. According to the table, the wall correction scenario has the smallest range of the confidence interval than the other scenarios.

Scenario	$\mu_{rep} \pm 2\sigma_{rep}$
No wall func.	0.003381 ± 0.002597
Standard wall func.	0.004542 ± 0.000762
Wall corr.	0.004295 ± 0.000504
Wall/face corr.	0.004303 ± 0.000529

Table 7.5: Representative mean value with the confidence interval for the flat plate case at $Re_x = 3 \cdot 10^6$

7.2.2 Comparison of Skin Friction for Different y^+ at $Re_x = 6 \cdot 10^6$

The skin friction values for the no wall function scenario for each y^+ are depicted in Figure 7.6. The case of no wall function yields bad estimations for $y^+ \geq 30$. However, for the other y^+, the skin friction values are also stable as the case in Subsection 7.2.1.

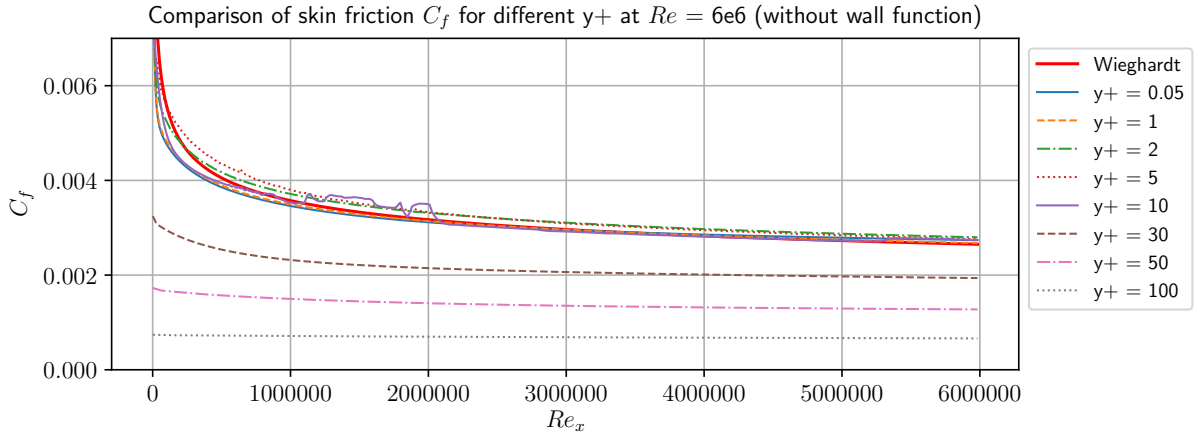

Figure 7.6: Skin friction for the no wall function scenario at $Re_x = 6 \cdot 10^6$

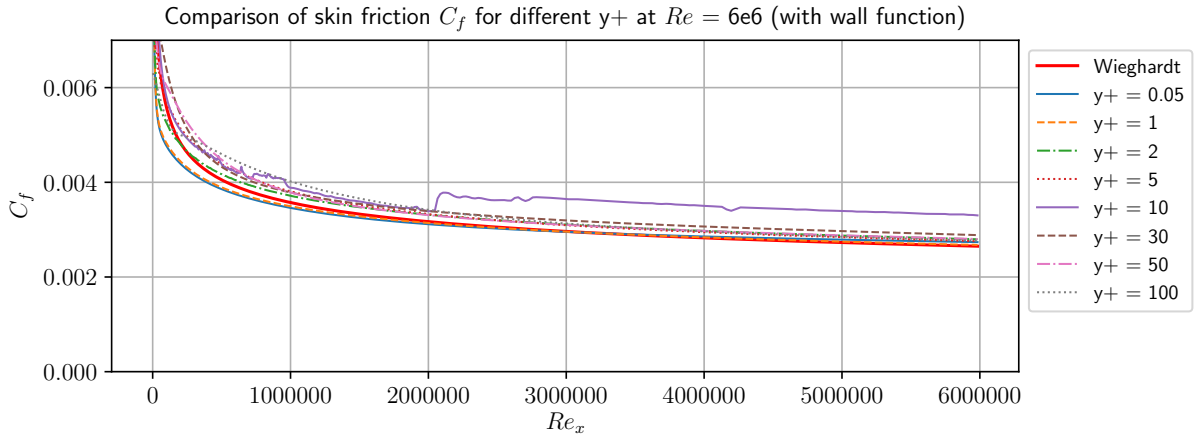

Figure 7.7: Skin friction for the standard wall function scenario at $Re_x = 6 \cdot 10^6$

The standard wall function scenario yields the results as shown in Figure 7.7. The skin friction value is overestimated at $y^+ = 10$, but the skin friction looks acceptable at the other y^+.

The result of the wall correction scenario is shown in Figure 7.8. The wall correction also yields the best results compared to the other scenarios as Subsection 7.2.1.

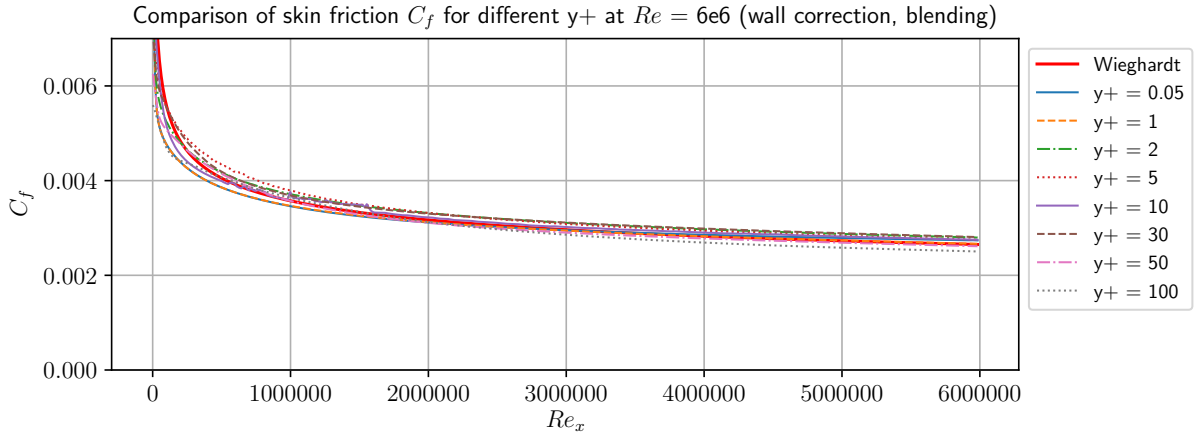

Figure 7.8: Skin friction for the data-driven wall function with wall correction scenario at $Re_x = 6 \cdot 10^6$

The result of the wall/face correction scenario is shown in Figure 7.9. This shows almost identical results to the previous scenario, but the skin friction for $y^+ = 10$ shows a bit overestimation.

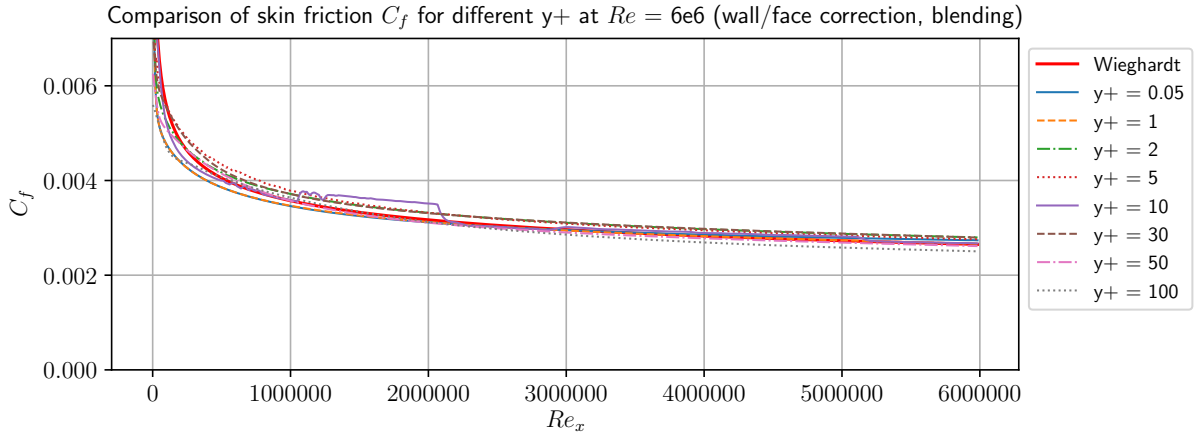

Figure 7.9: Skin friction for the data-driven wall function with wall/face correction scenario at $Re_x = 6 \cdot 10^6$

Figure 7.10 depicts the distribution of the skin friction for three scenarios except the no wall function scenario. The wall correction scenario yields the best performance amongst the other scenarios. The performance of the wall/face correction scenario is less better than that of the wall correction scenario.

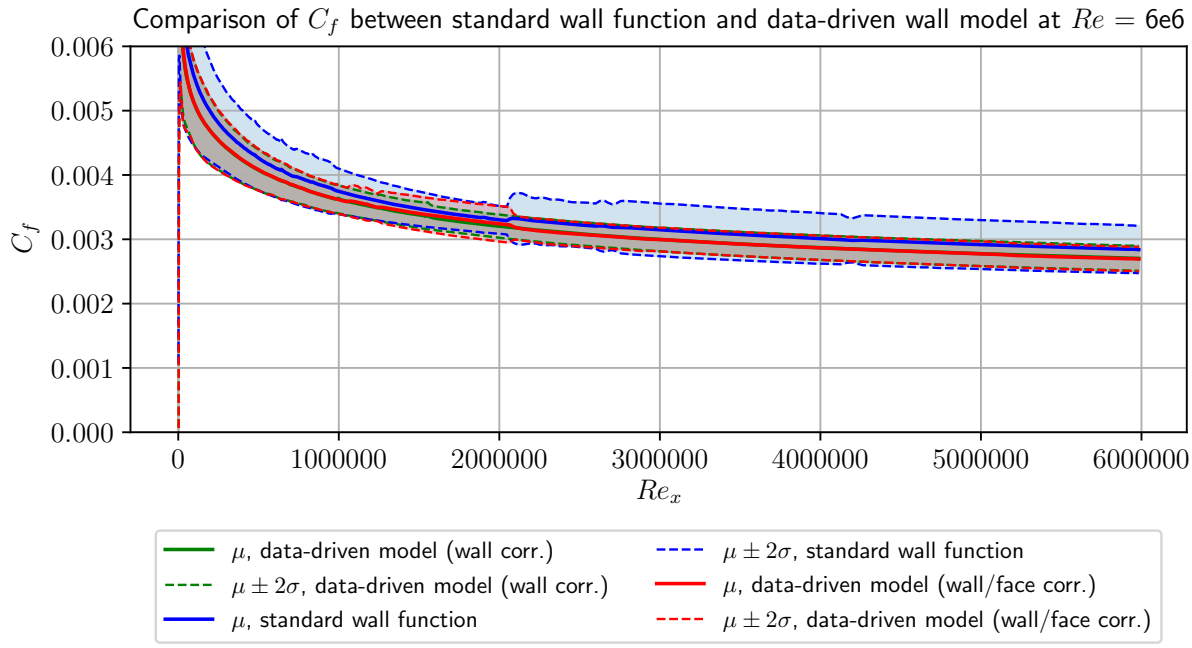

Figure 7.10: Distribution of skin friction for each scenario at $Re_x = 6 \cdot 10^6$

Table 7.6 indicates the representative mean value and standard deviation for each scenario as a numeric. According to the table, the wall correction scenario has the smallest range of confidence interval than the other scenarios.

Scenario	$\mu_{rep} \pm 2\sigma_{rep}$
No wall func.	0.002975 ± 0.002417
Standard wall func.	0.004030 ± 0.000572
Wall corr.	0.003826 ± 0.000368
Wall/face corr.	0.003832 ± 0.000387

Table 7.6: Representative mean value with the confidence interval for the flat plate case at $Re_x = 6 \cdot 10^6$

Chapter 8

Discussion

This project aims to create a data-driven wall function from a very simple geometry. Section 4 demonstrates the first stage of the procedure of the work that is about how to make ML models in a 1D channel geometry. For the data generation part, the label of face slopes in datasets contains a lot of zero values for the early time steps as shown in Figure 4.3, which means that the totally same label values regardless of the feature `y_face`. This made the ML model for face slopes difficult to train. However, the integral average velocity `avgU` can distinguish all the zero points, which can mitigate the training difficulty. The ML model for face slopes yields less maximum relative error than the ML model for wall slopes. The largest error region is mainly near the wall for all the models because the scale of the slopes is the largest there, and the distance of the value between two points near the wall are the largest as well.

Regarding the actual simulation in a flat plate for application of the trained models in Section 5, there is no issue for smaller y^+ values since no correction is applied, which leads to the same results for all the scenarios. Differences occur from the case $y^+ = 5$, and only the wall correction scenario shows the slightly larger skin friction compared to the others. This implies that the skin friction is reduced when the face correction turns on, which is the same as the original expectation of the slope correction according to Figure 3.1. The wall correction makes the skin friction larger because the real slope is larger than the numerical slope. Afterward, the face correction reduces the skin friction since the real slope is smaller than the numerical one. For $y^+ = 10$, the situation is almost identical. The skin friction for three scenarios even fluctuate, but the wall/face correction method mitigates this fluctuation, and capture the behavior of the reference one. From $y^+ = 30$, the wall/face correction scenario is almost the same as the wall correction scenario because no face correction occurs from the local $y^+ = 35$. This setting is applied due to two reasons. Firstly, the underestimation of skin friction and the kink exist at the very front of the plate. This could happen because the ML model slope at this region is almost zero, and thus the skin friction could be underestimated. However, the true reason is yet uncertain. For the wall correction case, there is still a kink, but it is much more reduced compared to the case with involvement of face correction. Secondly, the correction is differently executed for higher y^+. The wall correction increases the skin friction from the case without wall functions, but the face correction slightly increases the skin friction as well. Figure 8.1 shows the ν_{eff} correction ratio that corresponds to Equation (3.10). After a number of iterations, the ratio becomes larger than 1.0 and stably maintain 2.0 at the middle and the end of the plate for $y^+ = 100$, which means that the face correction increases the skin friction. For $y^+ = 10$, the ratio is also larger than 1.0 but fluctuates between 1.0 and 2.0 at the middle and the end of the plate. Due to the velocity blending method, the correction method works well as per the original concept at the buffer layer, whereas the method yields the unexpected results compared to the concept at the logarithmic layer. However, this phenomenon would not be because the

correction ratio at $y^+ = 10$ is smaller than 1.0, but the fluctuation of the ratio could somehow mitigate the unstable behavior of the skin friction. At the front of the plate for both y^+, the skin friction at $y^+ = 10$ can capture the reference graph due to the large correction ratio, whereas the skin friction at $y^+ = 100$ shows the kink at the same location. Hence, it turns out that the ν_{eff} correction ratio should be much larger than 1.0 (8.0 in this case) to avoid a creation of a kink at the front of the plate. With regards to the performance of each scenario, it can be concluded that the wall/face correction scenario yields the best performance because the 95% confidence interval for that scenario is the smallest. However, at the middle and the end of the plate, the confidence interval of the wall and the wall/face correction cases is slightly larger than the interval of the standard wall function case. This implies that the data-driven wall function for the flat plate case can generally mitigate the mesh-dependency, but shows worse performance than the standard wall function at the middle and the end of the plate.

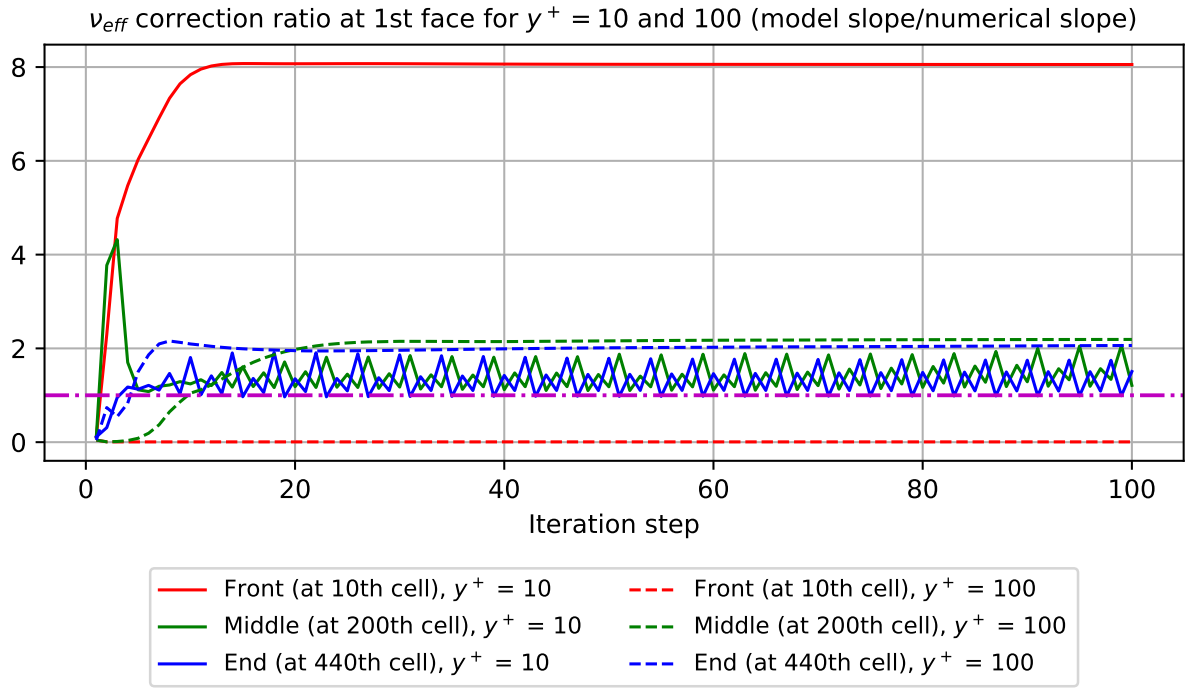

Figure 8.1: ν_{eff} correction ratio for $y^+ = 10$ and 100

In Section 6, the trained ML models are applied to a 2D NACA-0012 airfoil case. The pressure coefficient graphs are plotted well compared to the experiment data except the case $y^+ = 3.5$ because the local pressure values are virtually mesh-independent along with y-axis in a zero pressure gradient condition. For the skin friction, the buffer layer region is different from the flat plate case. The unstable fluctuations occur at $y^+ = 3.5$ for the airfoil case, whereas they occur at $y^+ = 10$ for the flat plate case. Since the same blending methods are used in the airfoil case, no correction happens at $y^+ = 3.5$ and 5 which means that the skin friction values cannot be mitigated. They are all overestimated, and even the skin friction for the wall/face correction scenario is out of range at $y^+ = 3.5$, but this problem is due to the turbulence model itself, which is not related to the data-driven approach. Meanwhile, the skin friction at $y^+ = 10$ that is still in the buffer layer is well mitigated compared to the standard wall function, and the wall/face correction case shows slightly better performance than the wall correction case. For higher y^+, the analogous problem occurs at the front of the airfoil that corresponds to the underestimation of skin friction. The gap between the reference and the data-driven approach in the airfoil case is larger than the gap in the flat plate case. The reason might be that the wall slopes at the front are underestimated, and thus the wall shear stresses are also underestimated compared to

the reference. When it comes to the convective flux correction, the result can be found in the $y^+ = 10$ case because the range of the local y^+ is between 10 and 35. However, the skin friction is not much different between the wall and the wall/face correction scenarios, which means that the diffusive flux correction and the convective flux correction at the first cell face are not markedly influenceable. The reason would be that the application of the ratio of the velocity magnitude (speed) for the flux `phi` is based on 1D. The magnitude of the velocity for the convective flux at the first cell face is actually $(u^2 cos\theta + uv(cos\theta + sin\theta) + v^2 sin\theta)$, which means that the velocities in x-direction and y-direction are involved together. However, the ML models are trained in the 1D channel, and hence only the velocity magnitude in x-direction is used for training. The ratio of the velocity magnitude is only between 0.9 and 1.1, which is quite a small amount of correction. Currently, the exact method is not yet found whether the speed in x-direction and y-direction can be used to correct ϕ as a ratio basis or the velocity vector should be calculated to create another FVM equation in the first place. It can be concluded that at least a 2D geometry setting must be employed when convective fluxes are involved to be correctly calculated. The performance of each scenario for the airfoil case is given by Figure 6.23 and Table 6.3. Although no correction is applied to the case $y^+ = 3.5$, the turbulence model itself yields the bad results for the wall correction and the wall/face correction scenarios. Hence, for this performance evaluation, the case $y^+ = 3.5$ is excluded. The wall correction and wall/face correction scenarios show worse results than the standard wall function scenario even though they have the smaller confidence interval range approximately between the range $c = 0.1$ and 0.4. The deviation at the front airfoil is large, which is the reason of the worse performance.

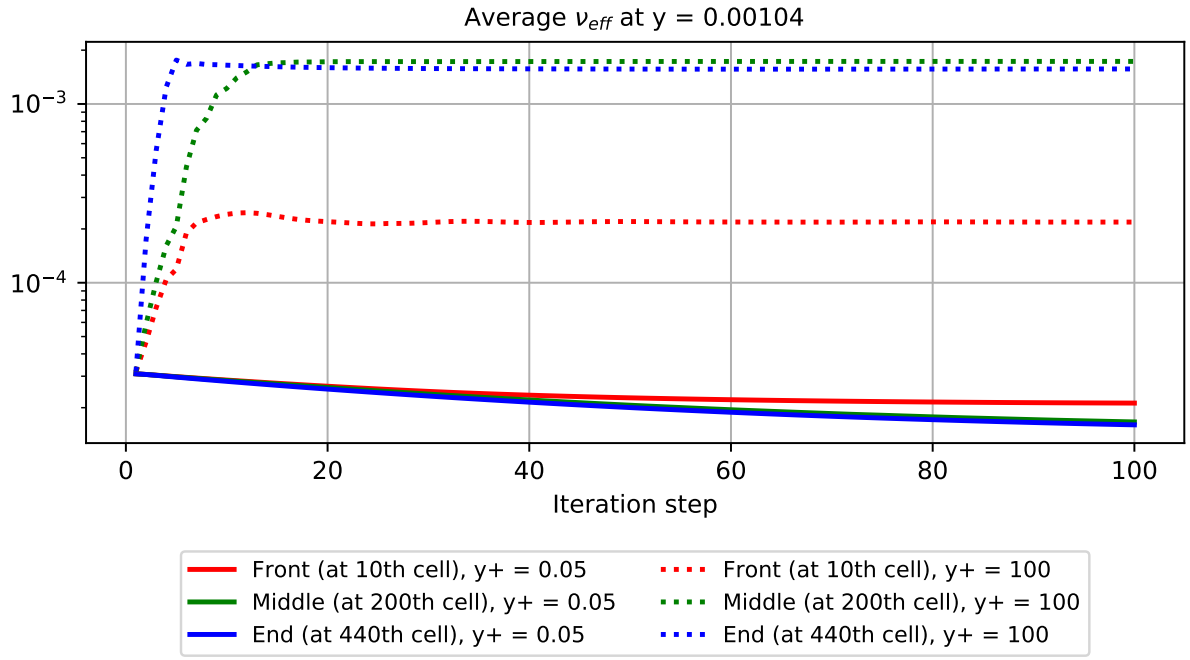

Figure 8.2: Average ν_{eff} for $y^+ = 0.05$ and 100

In Section 7, two additional flat plate cases are investigated that correspond to be at $Re_x = 3 \cdot 10^6$ and $6 \cdot 10^6$. The results are virtually identical to the results in Section 5. Regarding the case at $Re_x = 3 \cdot 10^6$, the skin friction for the wall/face scenario at $y^+ = 30$ is slightly overestimated at the front of the plate compared to the wall correction scenario. This is because the local y^+ at the front is still less than 30, and thus the additional correction at the first cell face is executed. On the other hand, the skin friction for the wall/face scenario at $y^+ = 10$ is slightly increased compared to the wall correction scenario for the case at $Re_x = 6 \cdot 10^6$. The face correction in this case also increases the skin friction. According to the results in Section 7, it can be con-

cluded that the data-driven approach is effective for various Reynolds numbers in flat plate cases since the confidence intervals for the data-driven wall function are smaller than the standard wall function scenario. Therefore, this data-driven approach can be used for mesh-independent results at least in flat plate cases.

The possible error sources are already explained in this section. Firstly, the ML model slopes for the first cell face normal to the wall at the front of the plate are almost zero for higher y^+, which could be an error source for the diffusive flux correction. Secondly, only one dimension is involved for the convective flux correction. Furthermore, there is one general source of showing inconsistent results that could be supposed. When it comes to the equation of the SA turbulence model, ν_{eff} is calculated from the equation. The ν_{eff} used in the equation is a cell-centered value, and therefore it is mesh-dependent. Figure 8.2 shows the average ν_{eff} values at $y^+ = 0.05$ and 100 with almost the same distance ($y_{face,0.05} = 0.001043330676$ and $y_{face,100} = 0.001044526213$). The ν_{eff} difference between two cases is enormously huge, and thus this could be the reason of the inconsistent results of simulations. In this work, the correction is executed only by multiplying the ratio of the slopes or the velocities. If ν_{eff} from the turbulence equation itself yields such a discrepancy, the simulation will yield bad results for coarser meshes regardless of the label calculation of the data-driven wall function.

Chapter 9

Summary

A data-driven approach is one of the popular methods in engineering and academic fields. This work uses the approach to invent a new data-driven wall function for mesh-independent behavior in turbulent flows. The related data is extracted from a 1D channel geometry by simulating 140 time steps shown in Section 4. Afterward, the training is performed with a simple NN structure that has the same number of neurons per hidden layer. Two features that are `y_face` and `avgU` and three labels that correspond to `dUdy_wall`, `dUdy_face`, and `Ux_face` are used for training, and the ML model is divided into three models for accuracy. After training, the maximum relative error near the wall region for three models is approximately 16%, particularly for the wall slope model, but the relative error for the other region is less than 5%.

The trained models are applied to a flat plate case at $Re_x = 1 \cdot 10^7$ in Section 5. By changing grading settings, skin friction values for total eight y^+, which correspond to 0.05, 1, 2, 5, 10, 30, 50, 100, are investigated. Since there is no vertical velocity component, only diffusive fluxes are corrected. The skin friction for the wall/face correction scenario is the best result which means that the data is not much spread from the mean value. The wall correction scenario shows worse result than the standard wall function scenario due to the case at $y^+ = 10$. The length of the confidence interval of the wall/face correction scenario is approximately 66.2% of the length of the standard wall function scenario, which implies that the data-driven approach is more mesh-independent than the standard wall function.

The next case is an airfoil case with a chord length of $1m$ at $Re_c = 3 \cdot 10^6$ mentioned in Section 6. This simulation investigates skin friction values and pressure coefficients for eight y^+ that are 0.05, 1, 2, 3.5, 5, 10, 50, 100. Since the buffer layer region is different from the flat plate case, $y^+ = 3.5$ is included instead of $y^+ = 30$. The vertical velocity component is involved in this airfoil case, and therefore convective fluxes are also corrected. For the pressure coefficient, the results of all the scenarios can capture the behavior of the experimental data except the case $y^+ = 3.5$ for the wall correction and the wall/face correction scenarios. Regarding the skin friction, the data-driven wall correction scenarios yield the bad estimations because of the huge underestimation at the front of the airfoil. Moreover, the simulations at $y^+ = 3.5$ for these two scenarios yield the really bad results, which means that the correction at the buffer layer is enormously unstable. Except the case at $y^+ = 3.5$, the standard wall function yields the best results compared to the other scenarios. The length of the confidence interval of the wall/face correction scenario is approximately 108.4% of the length of the standard wall function scenario, which means that the data-driven approach is not that effective for airfoil cases.

Section 7 demonstrates how the trained ML models work for various Reynolds numbers in flat plate cases, and two cases at $Re_x = 3 \cdot 10^6$ and $Re_x = 6 \cdot 10^6$ are investigated. Skin friction

values for $y^+ = 0.05$, 1, 2, 5, 10, 30, 50, 100 are found again. The wall correction scenario shows the best result, and the length of the confidence interval of the wall correction scenario is approximately 66.1% of the length of the standard wall function scenario for $Re_x = 3 \cdot 10^6$. For $Re_x = 6 \cdot 10^6$, the wall correction scenario shows the best result, and shows slightly better performance than the wall/face correction scenario. The length of the confidence interval of the wall correction scenario for this case is approximately 64.3% of the length of the standard wall function scenario, which also implies that the data-driven approach is more mesh-independent than the standard wall function. Generally, the best scenario of the data-driven approach yields more mesh-independent behavior, approximately 65% of the confidence interval for three flat plate cases.

In this project, the convective flux correction was not properly implemented. It is obvious that the convective flux correction rarely affected the results because the correction is based on the 1D geometry that does not have any vertical velocity components. Hence, a new approach that introduces a simple 2D geometry would be implemented for an appropriate convective flux correction method. On the other hand, it was found that the effective viscosity ν_{eff} at the first cell center (approximately $y = 0.00104$) for the coarser meshes was exceedingly larger than that at the same location for the finer meshes. This yielded the inconsistent results in terms of refinement of meshes. In the future, this uncertainty that is caused by the turbulence model in OpenFOAM is to be investigated. Finally, this project studied the flows that are with the zero pressure gradient condition. The identical ML model will be able to be used later on for various flow situations such as a flat plate case with a back pressure gradient condition, a backward facing step case, etc. The working pipeline is already set, and thus it is possible that the performance of the trained ML models is investigated for other cases as well.

Bibliography

[1] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[2] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, feb 2000.

[3] Peter Bradshaw and George P. Huang. The law of the wall in turbulent flow. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 451(1941):165–188, oct 1995.

[4] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering*. Cambridge University Press, jan 2019.

[5] Georgi Kalitzin, Gorazd Medic, Gianluca Iaccarino, and Paul Durbin. Near-wall behavior of RANS turbulence models and implications for wall functions. *Journal of Computational Physics*, 204(1):265–291, mar 2005.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. December 2014.

[7] J. Ling and J. Templeton. Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged navier stokes uncertainty. *Physics of Fluids*, 27(8):085103, aug 2015.

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

[9] Shuaiqiang Liu, Anastasia Borovykh, Lech A. Grzelak, and Cornelis W. Oosterlee. A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9(1), sep 2019.

[10] Tomislav Marić, Jens Höpken, and Kyle G. Mooney. *The OpenFOAM Technology Primer*. Zenodo, 2021.

[11] Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. A turbulent eddy-viscosity surrogate modeling framework for reynolds-averaged navier-stokes simulations. *Computers & Fluids*, 227:104777, sep 2021.

[12] Tsan Hsing Shih, Louis Povinelli, Nan-Suey Liu, Mark Potapczuk, and John Lumley. A generalized wall function. *NASA Technical Memorandum*, aug 1999.

[13] D. B. Spalding. A single formula for the "law of the wall". *Journal of Applied Mechanics*, 28(3):455–458, sep 1961.

[14] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, jan 2020.

[15] Yifeng Tian, Daniel Livescu, and Michael Chertkov. Physics-informed machine learning of the lagrangian dynamics of velocity gradient tensor. *Physical Review Fluids*, 6(9):094607, sep 2021.

[16] Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - ACL-IJCNLP '09*. Association for Computational Linguistics, 2009.

[17] Pedro Stefanin Volpiani, Morten Meyer, Lucas Franceschini, Julien Dandois, Florent Renac, Emeric Martin, Olivier Marquet, and Denis Sipp. Machine learning-augmented turbulence modeling for RANS simulations of massively separated flows. *Physical Review Fluids*, 6(6):064607, jun 2021.

[18] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on DNS data. *Physical Review Fluids*, 2(3):034603, mar 2017.

[19] Andre Weiner. Modeling and simulation of convection-dominated species transfer at rising bubbles. *Ph.D. Thesis at Technical University of Darmstadt*, 2020.

[20] Andre Weiner, Dennis Hillenbrand, Holger Marschall, and Dieter Bothe. Data-driven subgrid-scale modeling for convection-dominated concentration boundary layers. *Chemical Engineering & Technology*, 42(7):1349–1356, may 2019.

[21] X. I. A. Yang, S. Zafar, J.-X. Wang, and H. Xiao. Predictive large-eddy-simulation wall modeling via physics-informed neural networks. *Physical Review Fluids*, 4(3):034602, mar 2019.

[22] Ning Zhang, Shui-Long Shen, Annan Zhou, and Ye-Shuang Xu. Investigation on performance of neural networks using quadratic relative error cost function. *IEEE Access*, 7:106642–106652, 2019.

[23] Zhideng Zhou, Guowei He, and Xiaolei Yang. Wall model based on neural networks for LES of turbulent flows over periodic hills. *Physical Review Fluids*, 6(5):054610, may 2021.

List of Figures

List of Tables

Appendix A

Additional Plots for Comparison of Skin Friction

A.1 Comparison of Skin Friction for Different Scenarios at $Re_x = 3 \cdot 10^6$

For $y^+ = 0.05$, 1, and 2, no correction occurs as shown in Figures A.1, A.2, and A.3.

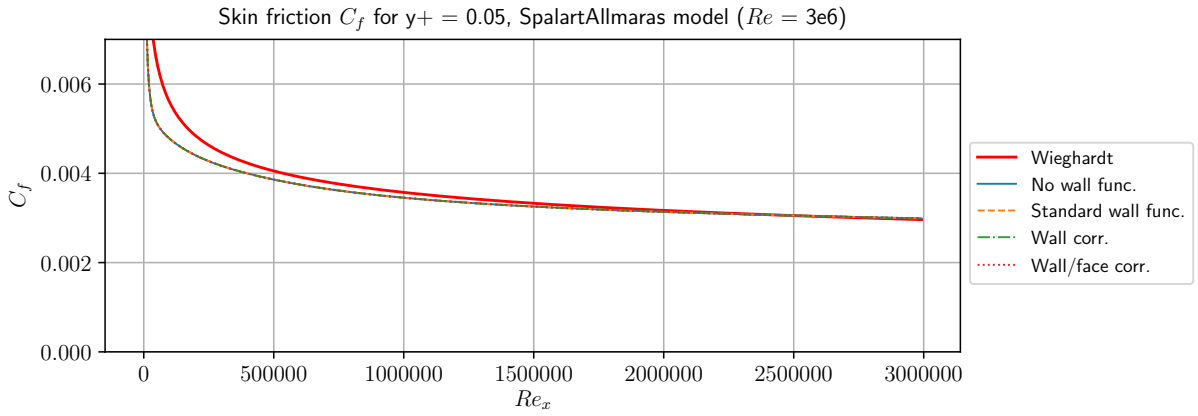

Figure A.1: Comparison of skin friction values for $y^+ = 0.05$ at $Re_x = 3 \cdot 10^6$

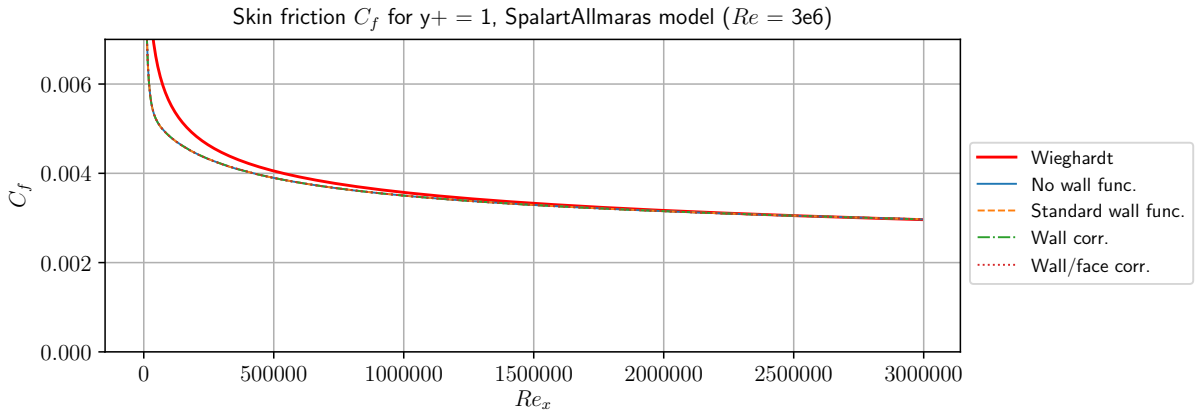

Figure A.2: Comparison of skin friction values for $y^+ = 1$ at $Re_x = 3 \cdot 10^6$

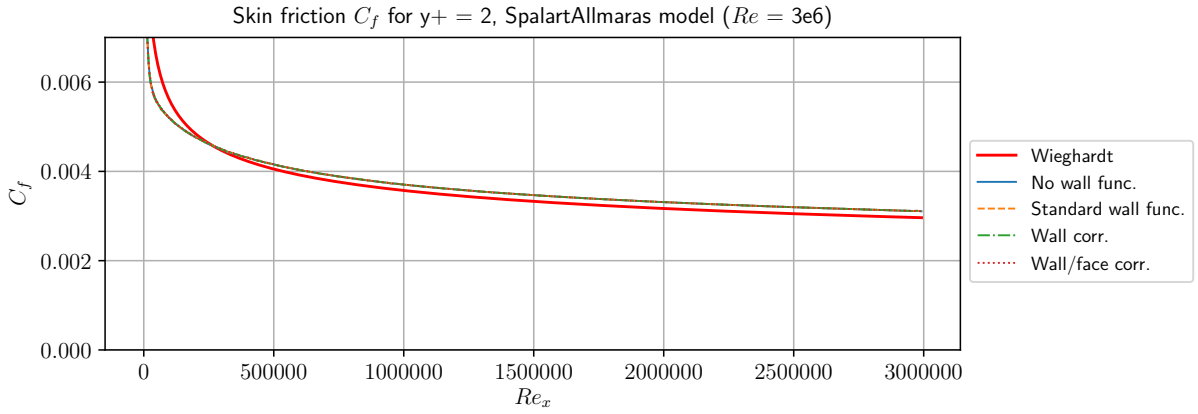

Figure A.3: Comparison of skin friction values for $y^+ = 2$ at $Re_x = 3 \cdot 10^6$

For $y^+ = 5$ shown in Figure A.4, there is no difference amongst all the scenarios since the local y^+ values are less than 10 for all regions unlike Section 5.

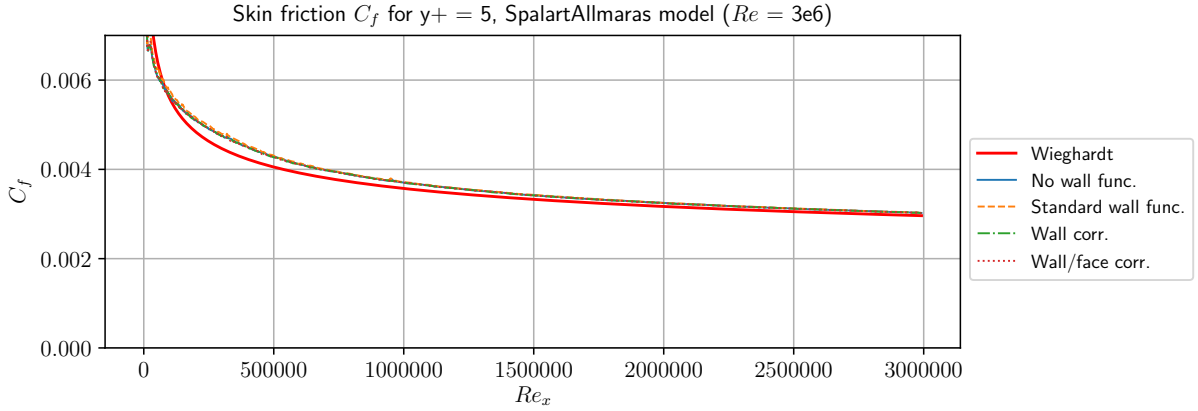

Figure A.4: Comparison of skin friction values for $y^+ = 5$ at $Re_x = 3 \cdot 10^6$

For $y^+ = 10$ shown in Figure A.5, the skin friction fluctuates for the standard wall function scenario, but the other scenarios show the stable behavior except small fluctuations at $Re_x = 1 \cdot 10^6$.

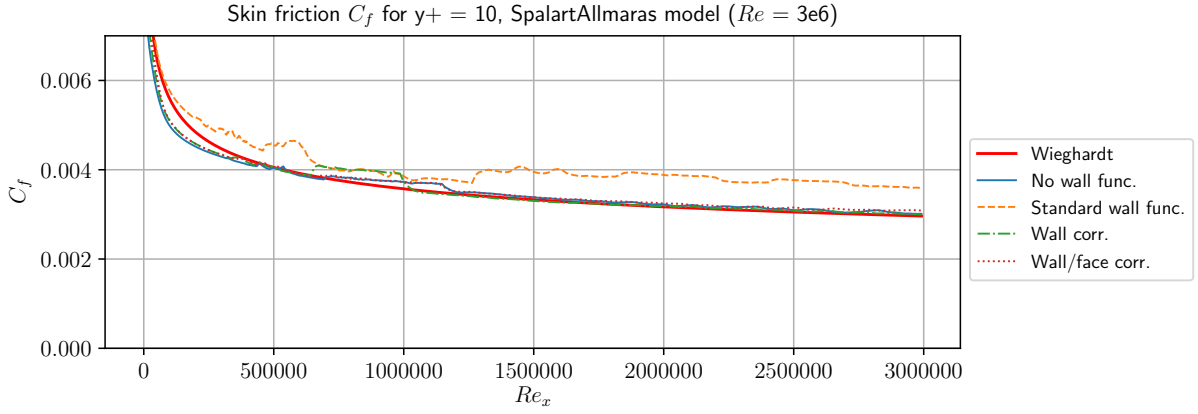

Figure A.5: Comparison of skin friction values for $y^+ = 10$ at $Re_x = 3 \cdot 10^6$

From $y^+ = 30$ depicted in Figures A.6, A.7, and A.8, the scenario without wall functions also shows very bad estimations, and the standard wall function scenario yields the overestimated results. The other two scenarios show the stable results apart from the area of the front plate. The wall/face correction scenario is the same as the wall correction scenario for $y^+ \geq 35$.

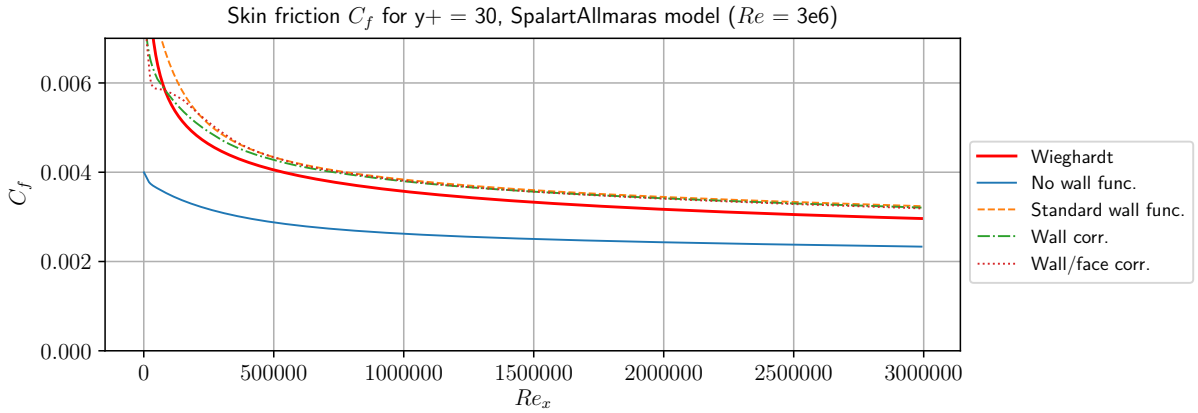

Figure A.6: Comparison of skin friction values for $y^+ = 30$ at $Re_x = 3 \cdot 10^6$

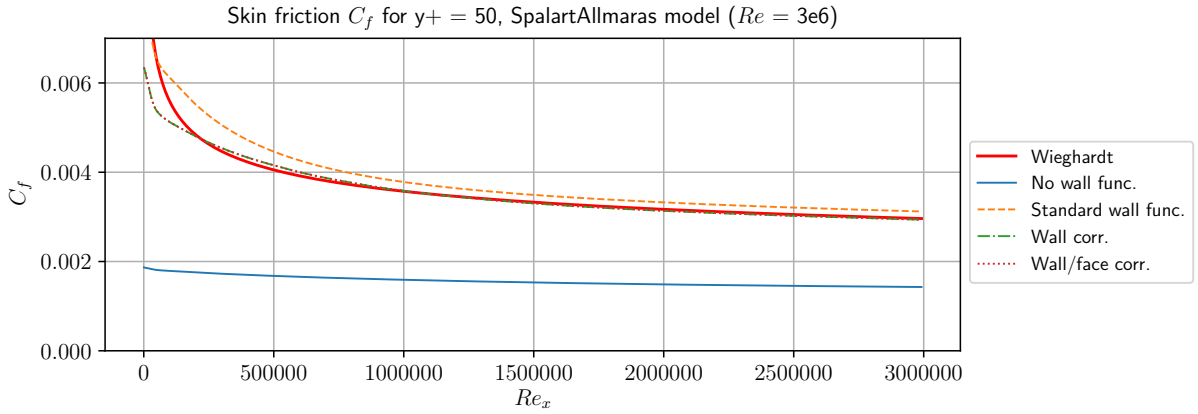

Figure A.7: Comparison of skin friction values for $y^+ = 50$ at $Re_x = 3 \cdot 10^6$

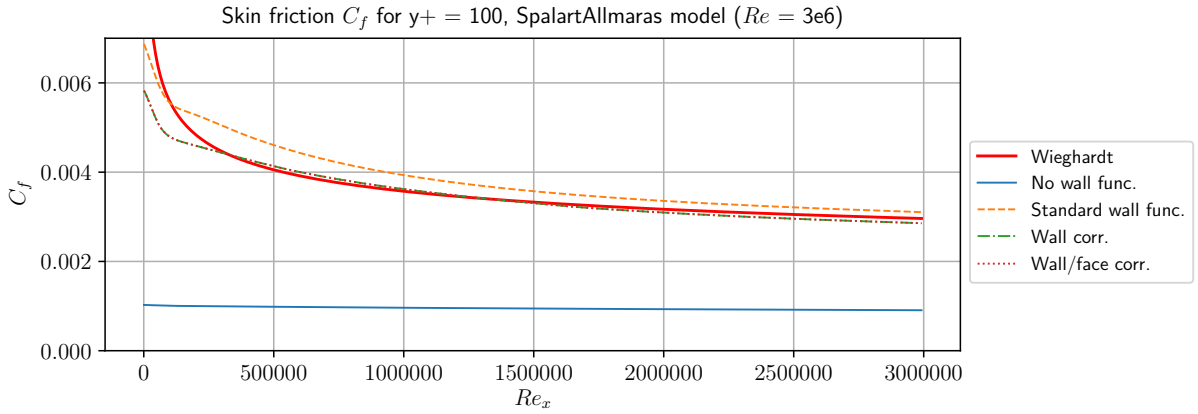

Figure A.8: Comparison of skin friction values for $y^+ = 100$ at $Re_x = 3 \cdot 10^6$

A.2 Comparison of Skin Friction for Different Scenarios at $Re_x = 6 \cdot 10^6$

For $y^+ = 0.05$, 1, and 2, no correction occurs as shown in Figures A.9, A.10, and A.11.

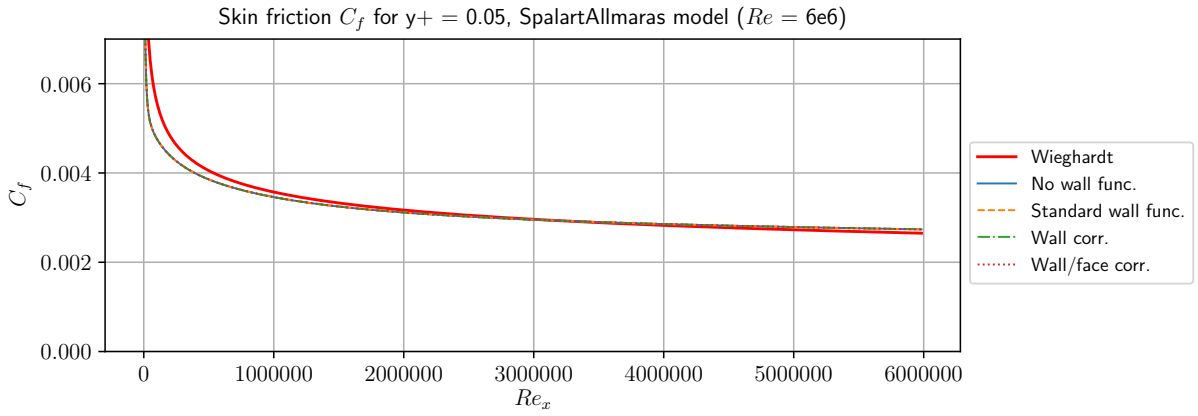

Figure A.9: Comparison of skin friction values for $y^+ = 0.05$ at $Re_x = 6 \cdot 10^6$

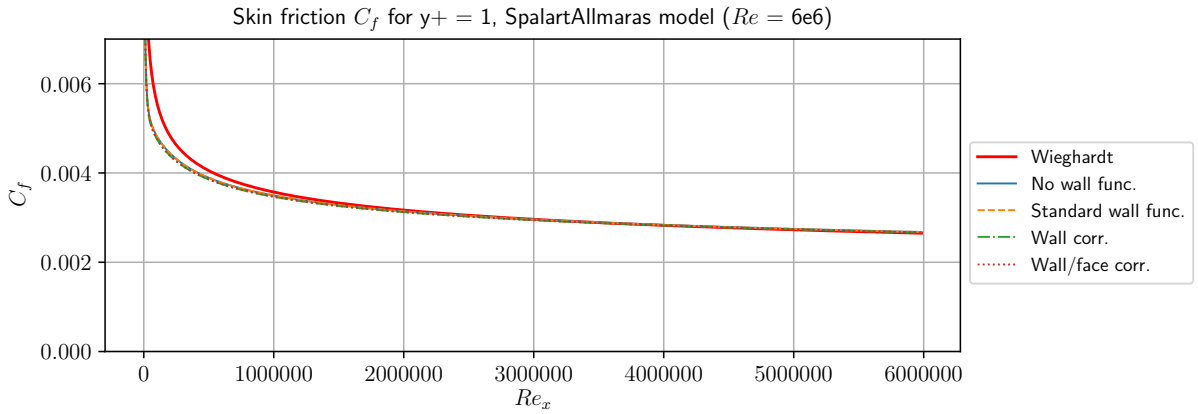

Figure A.10: Comparison of skin friction values for $y^+ = 1$ at $Re_x = 6 \cdot 10^6$

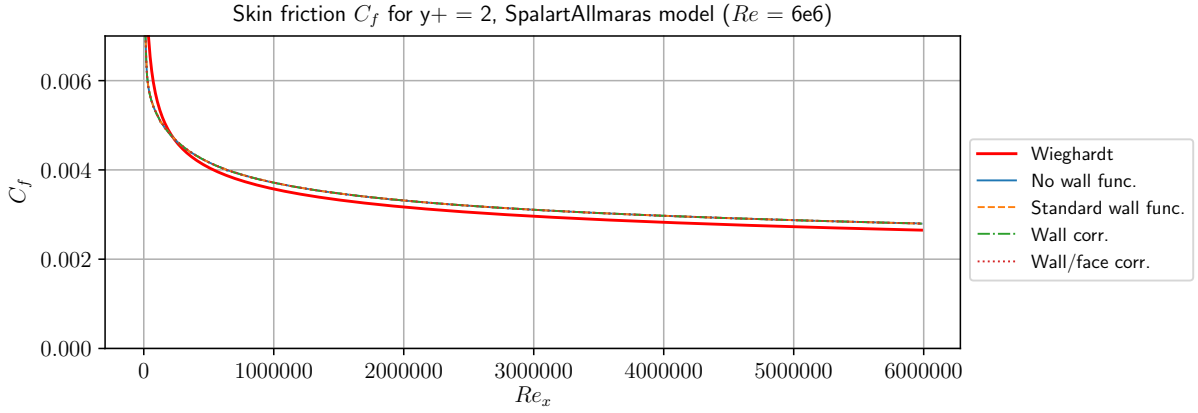

Figure A.11: Comparison of skin friction values for $y^+ = 2$ at $Re_x = 6 \cdot 10^6$

For $y^+ = 5$ shown in Figure A.12, no correction occurs for all the scenarios since the local y^+ values are still less than 10 for this Reynolds number.

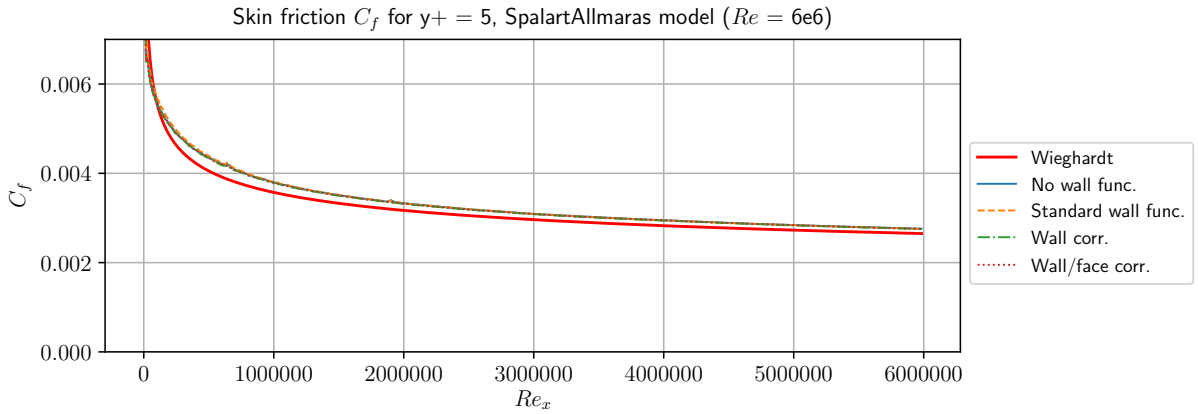

Figure A.12: Comparison of skin friction values for $y^+ = 5$ at $Re_x = 6 \cdot 10^6$

For $y^+ = 10$ shown in Figure A.13, the skin friction fluctuates for the standard wall function scenario, but the other scenarios show the stable behavior except small fluctuations at around $Re_x = 1.5 \cdot 10^6$.

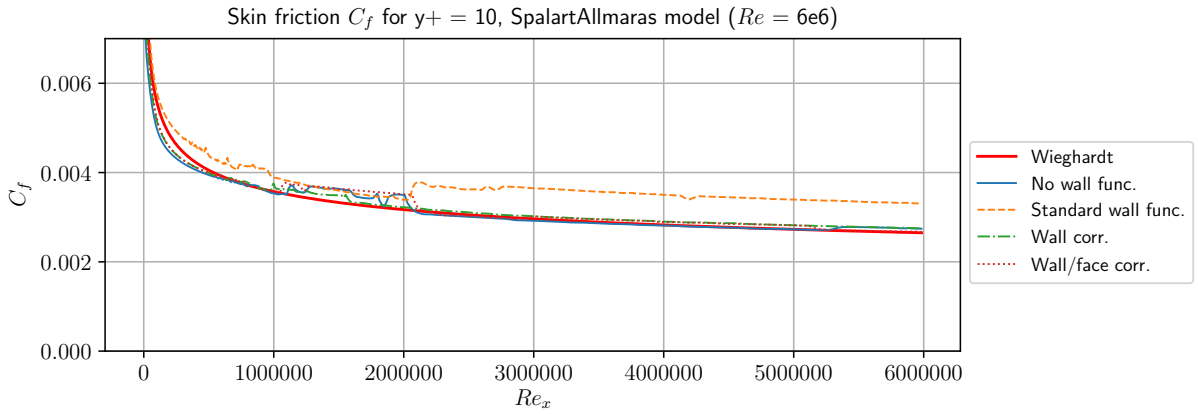

Figure A.13: Comparison of skin friction values for $y^+ = 10$ at $Re_x = 6 \cdot 10^6$

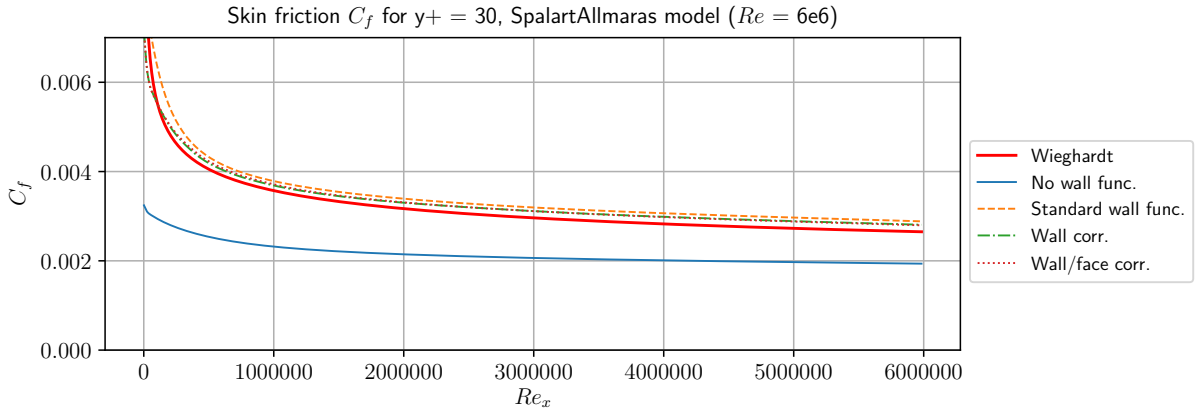

Figure A.14: Comparison of skin friction values for $y^+ = 30$ at $Re_x = 6 \cdot 10^6$

From $y^+ = 30$ depicted in Figures A.14, A.15, and A.16, the behavior of all the scenarios is virtually the same as the case at $Re_x = 3 \cdot 10^6$.

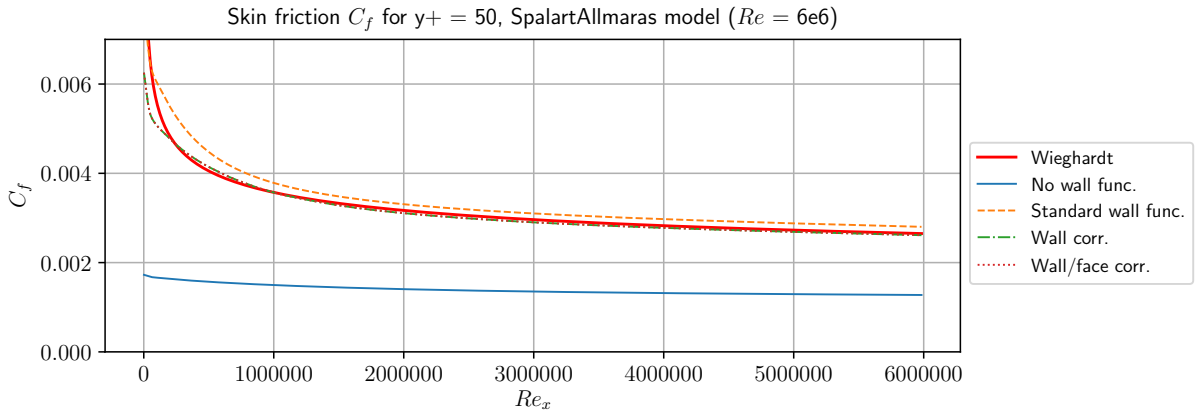

Figure A.15: Comparison of skin friction values for $y^+ = 50$ at $Re_x = 6 \cdot 10^6$

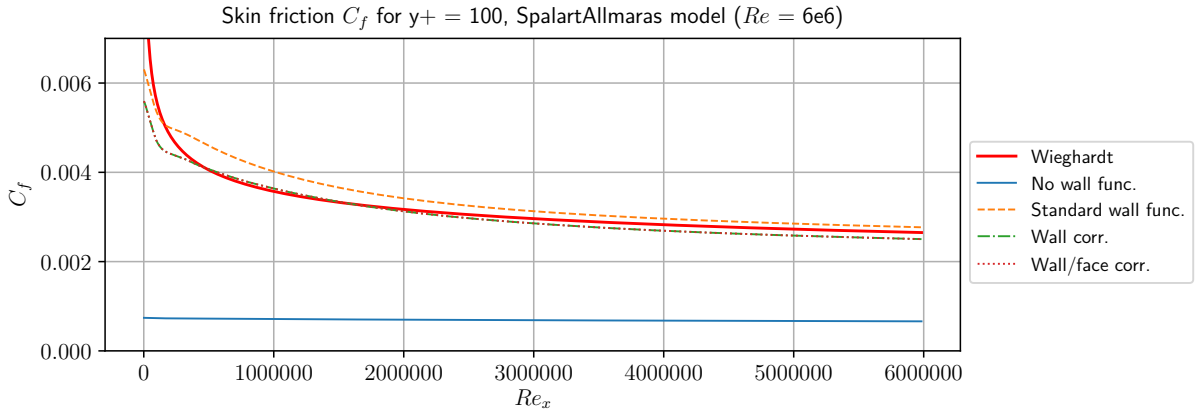

Figure A.16: Comparison of skin friction values for $y^+ = 100$ at $Re_x = 6 \cdot 10^6$