Using Physics Informed Neural Network (PINN) and Neural Network (NN) to Improve a $k-\omega$ Turbulence Model

Lars Davidson
Div. of Fluid Dynamics
Mechanics and Maritime Sciences (M2)
Chalmers University of Technology, Gothenburg, Sweden
Email: lada@chalmers.se

Abstract

The Wilcox $k-\omega$ turbulence model predicts turbulent boundary layers well, both fully-developed channel flows and flat-plate boundary layers. However, it predicts too low a turbulent kinetic energy. This is a feature it shares with most other two-equation turbulence models. When comparing the terms in the k equations with DNS data it is found that the production and dissipation terms are well predicted but the turbulent diffusion is not. In the present work the poor modeling of the turbulent diffusion is improved by making the turbulent diffusion constant, σ_k, a function of y/δ (y and δ denote wall distance and boundary-layer thickness, respectively). The k equation is turned into an ordinary differential equation for the turbulent viscosity which is solved using Physics Informed Neural Network (PINN). A new coefficient, $C_{K,NN}(y/\delta)$, is added to the dissipation term and another, $C_{\omega 2,NN}(y/\delta)$, to the destruction term in the ω equation. Finally, all three coefficients – $\sigma_{k,NN}$,,$C_{K,NN}$ and $C_{\omega 2,NN}$ – are made functions of two new input parameters, $\overline{|u'v'|}_{tot}/u_\tau^2$ and $\nu_t/(yu_\tau)$, in a Neural Network (NN) model. The new turbulence model, called the $k-\omega$-PINN-NN model, is shown to produce excellent velocity and turbulent kinetic profiles in channel flow at $Re_\tau = 550$, 2 000, 5 200 and $Re_\tau = 10\,000$ as well as in flat-plate boundary layer flow. The $k-\omega$-PINN-NN model is also used for predicting the flow over a periodic hill and the agreement with DNS is very good. The Python PINN and NN scripts and the Python CFD codes can be downloaded (Davidson, 2025a).

Keywords: Machine Learning; RANS; turbulence model; Physics Informed Neural Network; Neural Network; PINN; NN

1 Introduction

Eddy-viscosity RANS (Reynolds-Averaged Navier-Stokes) turbulence models in the literature have been developed with the object of predicting a correct velocity

field. $\bar{v}_i$. The most common turbulence models are the $k-\varepsilon$ and the $k-\omega$ models where k, ε and ω denote the turbulent kinetic energy, its dissipation rate and the specific dissipation $\omega = \varepsilon/(\beta^* k)$ (β^* is a constant), respectively. Both turbulence models include five tuning constants. These constants can be improved using PINN.

Yazdani and Tahani (2024) use PINN to optimize the five constants α, β^*, β, σ_k and σ_ω in the $k-\omega$ model. DNS data of the flow over a periodic hill at $Re = 5600$ are used. Using these DNS data, they compute all terms in the 2D RANS equations comprising of the two momentum equations ($\bar{v}_1$ and $\bar{v}_2$), the continuity equation, the k and ω equations. The loss function is defined as the sum of $(\hat{Q}_i^n - \tilde{Q}_i^n)^2$ where $\hat{Q}$ and $\tilde{Q}$ denote DNS value and PINN value, respectively. Subscript i represents 5000 arbitrary chosen DNS data points and superscript n corresponds to $\bar{v}_1$, $\bar{v}_1$, continuity equation, k or ε. The residuals (square of L_2 norm) of the five governing equations, Q^n, are added to the loss function. They use PINN to find optimal values of the five coefficient in the $k-\omega$ model. PINN gives modified values for $\alpha = 2.9719$ and $\sigma_\omega = 1.2685$ (baseline values are 2 and 0.52) whereas the other three constants are not changed. Then they carry out RANS simulations of the flow over the same periodic hill that was used for training and compare with RANS simulations using the standard values for the coefficients. They find that the new PINN coefficients give somewhat better results.

Luo et al. (2020) improve the five constants. C_μ, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, σ_k and σ_ε in the $k-\varepsilon$ model using PINN. They define the loss function as $(\hat{Q}_i^n - \tilde{Q}_i^n)^2$ (see above) at the DNS data points where Q^n is k or ε. Subscript i represents all DNS data points. The residuals of the transport equations of k and ε – multiplied by penalty functions – are added to the loss function. All terms in the k and ε equations are taken from DNS of a converging-diverging channel. New values of the constants are found by PINN (C_μ keeps its standard value of 0.09). Finally, RANS simulations are carried out for the converging-diverging channel flow (the same flow that was used when training with PINN) using the new PINN-optimized $k-\varepsilon$ constants. Somewhat better results are obtained compared with the standard $k-\varepsilon$ constants.

Thakur et al. (2024) use PINN to predict a diffusion coefficient. They study an unsteady, two dimensional convection-diffusion concentration equation, $c = c(x, y, t)$, with a spatially dependent diffusion coefficient, $D = D(x, y)$. The object is to predict D. The loss function, L, is

$$L = \frac{1}{N\sigma_c}\sum_{i=1}^{N}|\tilde{c} - c|^2 + \frac{1}{N^e\sigma_c}\sum_{i=1}^{N_e}|\tilde{c} - c^p|^2$$

where c denotes true data predicted with CFD using a prescribed (true), varying diffusion coefficient (D_{true} varies linearly, $\sin$, $\tanh$ etc) and c^p is obtained from the explicit unsteady diffusion equation using $\tilde{c}$ at the old time step. σ_c is the standard deviation of the concentration. D and $\tilde{c}$ are obtained from two neural networks.

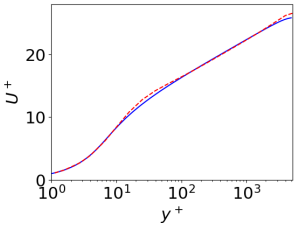
In simple flows including a boundary layer (channel flow, flat-plate boundary

layer flow, jet flow etc) the turbulent shear stress, $\overline{u'v'}$, must be correctly predicted. The turbulent shear stress is in these models linearly coupled to the turbulent viscosity, ν_t, via the Boussinesq assumption. Hence, well-tuned eddy viscosity models correctly predict the mean flow, the turbulent shear stress and the turbulent viscosity. However, the turbulent, kinetic energy, k, is usually poorly predicted. Figure 1 presents a prediction using the Wilcox $k-\omega$ turbulence model (Wilcox, 1988) of fully developed channel flow at $Re_\tau \equiv u_\tau \delta/\nu = 5\,200$ where u_τ and δ denote friction velocity and half-channel width, respectively. It can be seen that the $k-\omega$ model behaves as outlined above: the mean flow (and hence the turbulent shear stress and the turbulent viscosity) is well predicted but the predicted turbulent kinetic energy is much too small.

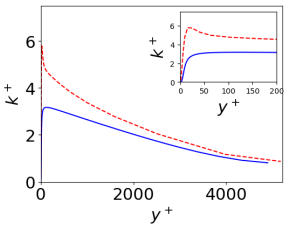
The object of the present paper is to improve the predicted, turbulent kinetic energy while not deteriorating the predicted mean flow. The predicted terms in the k equation using the $k-\omega$ model are compared with DNS in Fig. 1c. It can be seen that the production term, P^k, and the sum of the viscous diffusion and the dissipation term, $D^\nu - \varepsilon$, agree fairly well with DNS but that the agreement for the turbulent diffusion term, D^k, is not so good.

In order to improve the predicted k, we will use Physics Informed Neural Network, i.e. Physics Informed NN – usually called PINN – to improve the predicted diffusion term. In the works of Yazdani and Tahani (2024) and Luo et al. (2020) summarized above, new constant values of turbulent coefficients were optimized. In the present study, one turbulent constant, σ_k, will be turned into a function of y/δ, i.e. $\sigma_{k,PINN}\,(y/\delta)$. The $\sigma_{k,PINN}$ function is obtained as follows. By taking the production, the dissipation terms as well as the turbulent kinetic energy from DNS channel flow data, the k equation is turned into an ordinary differential equation for the turbulent viscosity, $\nu_k = \nu_{t,PINN}/\sigma_{k,PINN}$, in the k equation which is solved using Physics Informed Neural Network (PINN). In order to not change the predicted turbulent viscosity – which is well predicted by the standard Wilcox $k-\omega$ turbulence model – a new function, $C_{K,NN}$, is added to the dissipation term and another, $C_{\omega 2,NN}$, to the destruction term in the ω equation. Finally, $\sigma_{k,NN}$, $C_{K,NN}$ and $C_{\omega 2,NN}$ are made functions of two input parameters, $\overline{|u'v'|}_{tot}/u_\tau^2$ and $\nu_t/(yu_\tau)$, in a Neural Network (NN) model. The difference between $\sigma_{k,PINN}$ and $\sigma_{k,NN}$ is that the former is obtained from PINN and the latter is given by the NN model using $\sigma_{k,PINN}$ as the target. The Python PINN and NN scripts and the Python CFD codes can be downloaded (Davidson, 2025a).

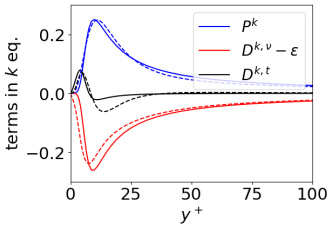
The paper is organized as follows. In the two following sections, the two-dimensional solver (channel flows and flat-plate boundary layer) and three-dimensional solver (periodic hill flow) are briefly described. Then the PINN and NN models are presented. Next, the results are discussed and presented and in the final section we give some concluding remarks.



(a) Mean velocity.



(b) Turbulent kinetic energy.



(c) Terms in k equation.

Figure 1: Fully-developed channel flow. $Re_\tau = 5\,200$. Solid lines: $k-\omega$; dashed lines:DNS (Lee and Moser, 2015).

2 Equations and the numerical method for the 2D RANS simulations

The RANS equations are given by for the fully-developed channel flow and the flat-plate boundary layer read

$$\frac{\partial \bar{v}_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial \bar{v}_i \bar{v}_j}{\partial x_j} = \delta_{1i} - \frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[(\nu + \nu_t)\frac{\partial \bar{v}_i}{\partial x_j}\right] \tag{2}$$

The first term on the right-hand side of Eq. 2 is the driving pressure gradient used in fully-developed channel flow; it is not present in the flat-plate boundary layer. The $k-\omega$ turbulence model reads

$$\begin{aligned}
\frac{\partial \bar{v}_j k}{\partial x_j} &= P^k + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_{k,NN}}\right)\frac{\partial k}{\partial x_j}\right] - C_\mu C_{k,NN} k\omega \\
\frac{\partial \bar{v}_j \omega}{\partial x_j} &= C_{\omega_1}\frac{P^k}{\nu_t} + \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial \omega}{\partial x_j}\right] - C_{\omega 2,NN}\omega^2 \\
P^k &= \nu_t\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right)\frac{\partial \bar{v}_i}{\partial x_j}, \quad \nu_t = \frac{k}{\omega}
\end{aligned} \tag{3}$$

In the standard $k-\omega$ model $C_{k,NN} = 1$, $\sigma_{k,NN} = \sigma_k$ and $C_{\omega 2,NN} = C_{\omega 2}$. The coefficients have the values $C_{\omega 1} = 5/9$, $C_{\omega 2} = 3/40$, $\sigma_k = \sigma_\omega = 2$ and $C_\mu = 0.09$.

The **pyCALC-RANS** code is used (Davidson, 2021a) for solving the discretized equations. It is an incompressible, finite volume code written in Python. It is fully vectorized (i.e. no for loops). The convective terms in the momentum equations are discretized using the MUSCL scheme (van Leer, 1979) (a second-order bounded upwind scheme) and the hybrid 1^{st} order upwind/2^{nd} order central scheme is used for k and ω equations. The numerical procedure is based on the pressure-correction method, SIMPLEC, and a collocated grid arrangement using Rhie-Chow interpolation (Rhie and Chow, 1983).

3 Equations and the numerical method for the hill-flow simulations

The unsteady momentum equations for the periodic hill flow read

$$\begin{aligned} \frac{\partial \bar{v}_i}{\partial t} + \frac{\partial \bar{v}_j \bar{v}_i}{\partial x_j} &= \beta \delta_{1i} - \frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{v}_i}{\partial x_j \partial x_j} - \frac{\partial \overline{v_i' v_j'}}{\partial x_j} \\ \overline{v_i' v_j'} &= -\nu_t \left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i} \right) + \frac{2}{3}\delta_{ij} k \end{aligned} \tag{4}$$

where the term $\beta \delta_{1i}$ is the driving pressure gradient in the streamwise direction; the last term in the expression for $\overline{v_i' v_j'}$ is incorporated in the pressure.

The finite volume code **pyCALC-LES** (Davidson, 2021b) is used. It is written in Python and is fully vectorized (i.e. no `for` loops). The solution procedure is based on fractional step. The second-order MUSCL scheme (van Leer, 1979) is used. The k and ω equations are given in Eq. 3 (adding the unsteady term $\partial/\partial t$) using the same discretization.

The only reason why **pyCALC-RANS** is not used for this flow is that the author did not succeed in adjusting the β coefficient, in a reasonable manner, see Eq. 17. The discretization in space in **pyCALC-LES** is identical to that in **pyCALC-LES**. The main difference is how the pressure-velocity coupling is treated; furthermore the latter code was developed for unsteady and three-dimensional flow whereas the former is used for steady and two-dimensional flow.

Above, we have used tensor notation for the velocity vector $(\bar{v}_1, \bar{v}_2, \bar{v}_3)$ and for the space vector (x_1, x_2, x_3). Below, we will replace them with $\bar{u}, \bar{v}, \bar{w}$ and x, y, z.

4 Physics informed NN (PINN)

Let's create a simple NN that finds a damping function, $Y \equiv f$, as a function of $X \equiv y^+$, see Fig. 2. It has one input ($X = a_1^{(0)}$), one hidden layer with two neurons ($a_1^{(1)}, a_2^{(1)}$) and one output ($Y = a_1^{(2)}$). The target is the correct $f = f_{DNS}$ taken from DNS. In Listing 1 in the Appendix, you find the line la-

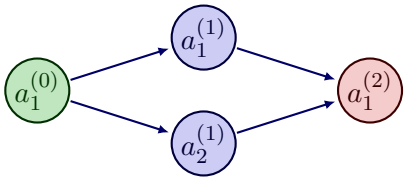


Figure 2: Schematic of Simple NN

beled `Connection 0-1` which connects $a_1^{(0)}$ and $\left(a_1^{(1)}, a_2^{(1)}\right)$ and the line labeled `Connection 1-2` which connects $\left(a_1^{(1)}, a_1^{(1)}\right)$ and $a_1^{(2)}$.

Now we formulate the NN with weights, w, and biases, b and add Sigmoid activators, s, to both neurons, i.e.

$$\begin{aligned}
\text{Activation 1:} \quad & a_1^{(1)} = s_1^{(1)}\left(w_1^{(0)} a_1^{(0)} + b_1^{(0)}\right) \\
\text{Activation 2:} \quad & a_2^{(1)} = s_2^{(1)}\left(w_2^{(0)} a_1^{(0)} + b_2^{(0)}\right) \\
\text{Output:} \quad & a_1^{(2)} = s_1^{(2)}\left(w_1^{(1)} a_1^{(1)} + b_1^{(1)}\right. \\
& \left. + w_2^{(1)} a_2^{(1)} + b_2^{(1)}\right) \equiv Y
\end{aligned}$$

The Python code is given in Listing 2 in the Appendix.

The `loss.backward()` command computes the gradients of the loss, L, with respect to the weights, biases and activators, (i.e. $\partial L/\partial w_1, \partial L/\partial b_1, \partial L/\partial s_1 \ldots$) in order to get new improved $w_1, b_1, s_1 \ldots$

Now, let's involve our differential equation, the k equation (see Eq. 3). In fully-developed channel flow it is a diffusion equation with source terms which reads

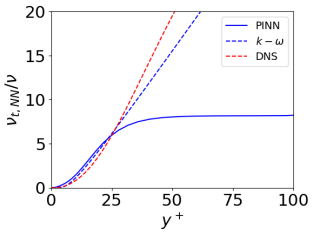
$$\frac{d}{dy}\left(\nu + \nu_{t,PINN}\frac{dk}{dy}\right) + P^k - \varepsilon = Q.$$

It is re-written as

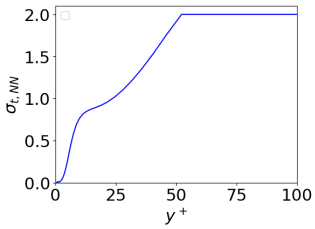
$$\left(\nu + \nu_{t,PINN}\right)\frac{d^2k}{dy^2} + \frac{dk}{dy}\frac{d\nu_{t,PINN}}{dy} + P^k - \varepsilon = Q \tag{5}$$

We want to find a new turbulent viscosity, $\nu_{t,PINN}$, that gives a turbulent diffusion that agrees with the DNS turbulent diffusion term in Fig. 1c. Hence, $\nu_{t,PINN}$ is the unknown variable in Eq. 5 and k, P^k and ε are taken from DNS. Equation 5 is solved in half a channel at $Re_\tau = 5\,200$. The boundary condition at the wall ($y = 0$) is $\nu_{t,PINN} = 0$ and at the center ($y = \delta$) it is $\nu_{t,PINN} = \nu_{t,DNS}$.

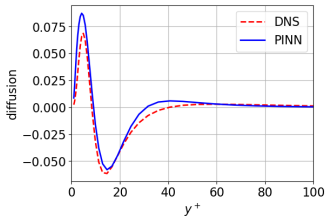
The turbulent viscosity, $\nu_{t,PINN}$ in Eq. 5, will be predicted by PINN while minimizing the error Q^2. The loss function, `loss_fn`, in Listing 2 in the Appendix is replaced with Eq. 5 and the Python code is given in Listing 3.



(a) Turbulent viscosity.



(b) Prandtl number.



(c) Turbulent diffusion.

Figure 3: Prediction by PINN compared to DNS and the Wilcox $k-\omega$ model. $Re_\tau = 5\,200$.

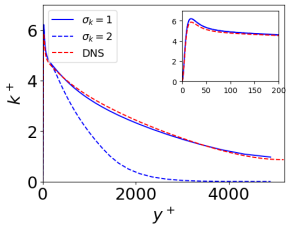


Figure 4: CFD predictions of Eq. 8 with different $\sigma_{k,max}$.

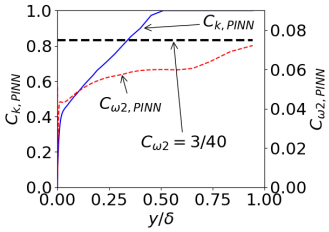


Figure 5: $C_{k,PINN}$ and $C_{\omega2,PINN}$ vs. y/δ.

`nut` and `k` in Listing 3 are $\nu_{t,PINN}$ and k in Eq. 5, respectively. `k_y`, for example, is dk/dy. Note that `k_y` and `k_yy` are known (taken from DNS). There are two losses in Listing 3, one, `ODE_loss`, which relates to the ODE and one, `BC_loss`, which relates to the boundary conditions. In this way the PINN is forced to satisfy both the ODE and the boundary conditions.

Figures 3a and 3b present the predicted turbulent viscosity in the k equation, $\nu_{t,PINN}$, and the turbulent Prandtl number, $\sigma_{k,PINN}$, where $\sigma_{k,PINN} = \nu_t/\nu_{t,PINN}$ (ν_t is the turbulent viscosity predicted by the Wilcox $k-\omega$). It is compared to the DNS turbulent diffusion in Fig. 3c, and the agreement is good. The turbulent diffusion goes to zero at $y^+ \simeq 40$ and the turbulent diffusion is negligible for $y^+ \gtrsim 40$ (see Fig. 1c). Hence, the value of $\nu_{t,PINN}$ (and $\sigma_{k,PINN}$) is not relevant in the outer region and $\sigma_{k,PINN}$ is set to a constant in this region (to be determined).

Now we have modified the turbulent Prandtl number in the k equation so that we – with exact source terms taken from DNS, P^k and ε – predict a correct near-wall behaviour of k. In the next step, we have to take the source terms from the $k-\omega$ model. Recall that P^k in the Wilcox $k-\omega$ model is correct since the model does predict the velocity profile correctly, see Fig. 1 and hence also the turbulent viscosity (recall that the total shear stress is predicted as $y-1$ by any turbulence models in a finite volume method). The k predicted by DNS near the wall is much larger than that predicted by the Wilcox $k-\omega$ model, see Fig. 1b. This means that ω must be modified in order to give the same turbulent viscosity as the Wilcox $k-\omega$ model, i.e.

$$\nu_{t,k-\omega} = \frac{k_{k-\omega}}{\omega_{k-\omega}} = \nu_{t,DNS} = \frac{k_{DNS}}{\omega_{DNS}} \tag{6}$$

which gives

$$\omega_{DNS} = k_{DNS}/\nu_{t,k-\omega}. \tag{7}$$

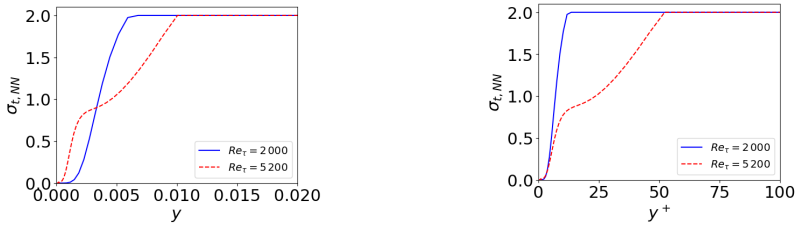
The turbulent Prandtl number in the k equation has now been modified (Fig. 3b) and we have found an expression for a correct ω (Eq. 7). Let's verify that a CFD solver does predict a correct turbulent kinetic energy. Equation 5 is formulated as an equation for k, i.e.

$$\frac{d}{dy}\left(\left(\nu + \frac{\nu_{t,k-\omega}}{\sigma_{k,PINN}}\right)\frac{dk}{dy}\right) + P^k - \varepsilon = 0 \tag{8}$$

where P^k and ε are again taken from DNS. The turbulent viscosity, $\nu_{t,k-\omega}$, is taken from the standard $k-\omega$ model and the turbulent Prandtl number is computed using the DNS value of ω, i.e.

$$\sigma_{k,PINN} = \frac{k/\omega_{DNS}}{\nu_{t,PINN}} \tag{9}$$

We solve Eq. 8 using the CFD solver **pyCALC-RANS**. Figure 4 presents the predicted k profiles using two different turbulent Prandtl numbers. We find that when using $\sigma_{k,max} = 1$ the agreement is excellent. The larger value, $\sigma_{k,max} = 2$, also gives excellent agreement near the wall for $y^+ < 200$ but away from the wall



(a) $\sigma_{k,PINN}$ vs. y. (b) $\sigma_{k,PINN}$ vs. y^+.

Figure 6: Turbulent Prandtl number vs. y and y^+.

it gives much too small a k. The reason is that the diffusion in the outer region is too small because of a small dk/dy. However, when solving the full equation system (i.e. $\bar{u}$, k and ω) the k profile in the outer region must adapt so that the total shear stress satisfies $y - 1$. In Section 5 it is found that $\sigma_{k,max} = 1$ and using no limit give identical predictions. The reason is that the turbulent diffusion in the k equation is negligible in the outer region, see Fig. 1c. In the Wilcox $k - \omega$ model $\sigma_k = 2$ and hence $\sigma_{k,max}$ is set to two.

Now we know ω_{DNS}. Next, the dissipation term $C_\mu k\omega$ in the k equation in Eq. 3 must be modified so that it agrees with $\varepsilon_{DNS} = C_\mu k_{DNS}\omega_{DNS}$. This is achieved by multiplying the dissipation term by a damping function, $C_{k,PINN} = C_{k,PINN}\,(y/\delta)$, so that the k equation in Eq. 3 is satisfied with $k = k_{DNS}$ and $\omega = \omega_{DNS}$, i.e. (see Eq. 8)

$$C_{k,PINN} = \frac{\frac{d}{dy}\left(\frac{\nu_t}{\sigma_{k,PINN}}\frac{dk_{DNS}}{dy}\right) + P^k_{DNS}}{C_\mu k_{DNS}\omega_{DNS}} \tag{10}$$

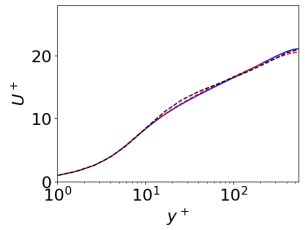
Note that the viscous diffusion has been omitted in order to prevent a large gradient of $C_{k,PINN}$ close to the wall.

Finally, we must make sure that the ω equation in Eq. 3 predicts $\omega = \omega_{DNS}$. This is achieved by making $C_{\omega 2,PINN}$ = $C_{\omega 2,PINN}\,(y/\delta)$. From the ω equation in Eq. 3 we get

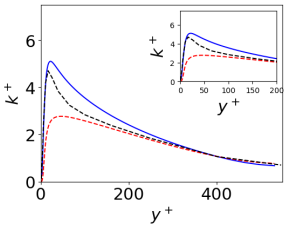
$$C_{\omega 2,PINN} = \frac{\frac{d}{dy}\left(\frac{\nu_t}{\sigma_\omega}\frac{d\omega_{DNS}}{dy}\right) + C_{\omega 1}\frac{P^k_{DNS}}{\nu_{t,DNS}}}{\omega^2_{DNS}} \tag{11}$$

Again, the viscous diffusion has been omitted.

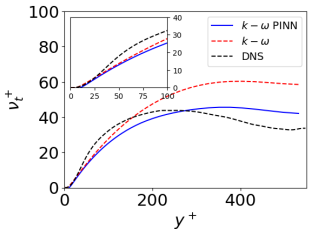
Figure 5 shows the two damping functions. The thick, black, dashed line indicates the standard value of $C_{\omega,2} = 3/40$, see below Eq. 3. The damping function, $C_{k,PINN}$, is in the outer region much affected by the dominator in Eq. 10 which essentially is $(k_{DNS}/k_{k-\omega})^2$: first, k_{DNS} is larger than $k_{k-\omega}$ (see Fig. 4), and, second, ω_{DNS} is larger than $\omega_{k-\omega}$, see Eq. 6. This explains why $C_{k,PINN} < 1$ for $y/\delta \lesssim 0.5$.



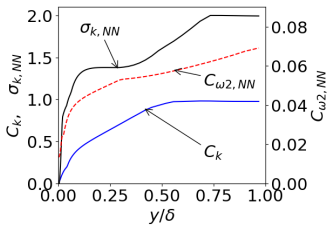
(a) Velocity.



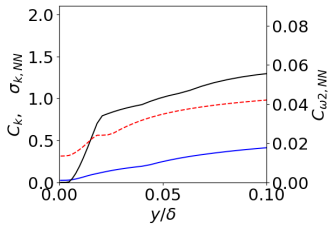
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.



(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

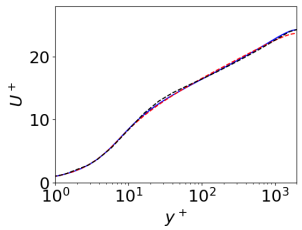


(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

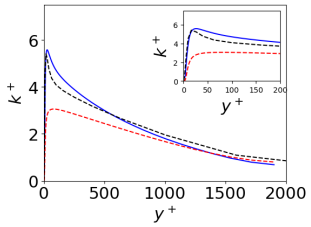
Figure 7: Fully-developed channel flow. $Re_\tau = 550$.

Re_τ	N_y	s_y	y^+
550	60	1.07	0.3
2000	60	1.11	0.2
5 200	70	1.13	0.07
10 000	150	1.05	0.2

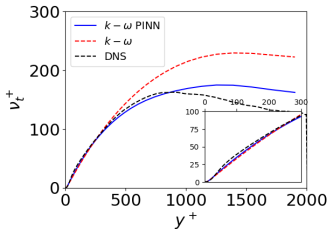
Table 1: Channel flow. Reynolds number (Re_τ), number of cells in the wall-normal direction (N_y), grid stretching (s_y) and location of wall-adjacent cell center (y^+).



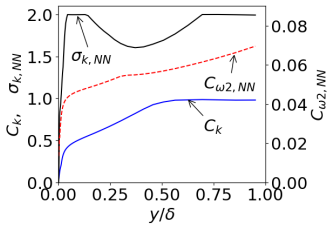
(a) Velocity.



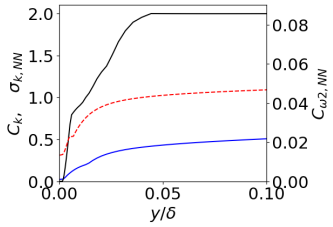
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.

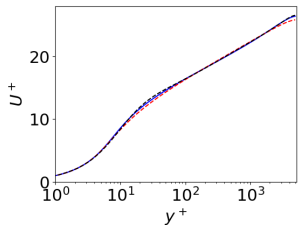


(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

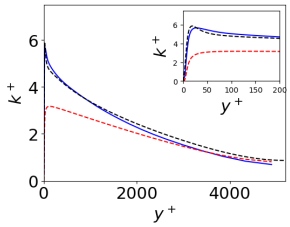


(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

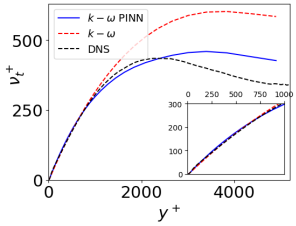
Figure 8: Fully-developed channel flow. $Re_\tau = 2\,000$.



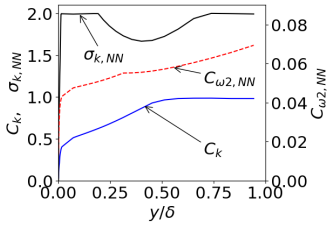
(a) Velocity.



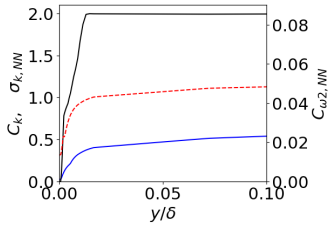
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.

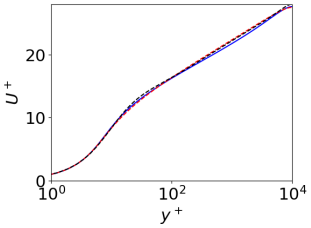


(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

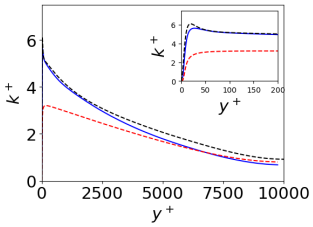


(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

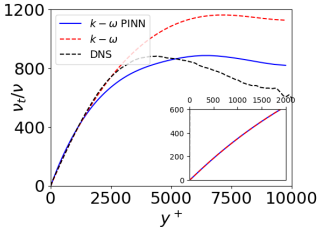
Figure 9: Fully-developed channel flow. $Re_\tau = 5\,200$.



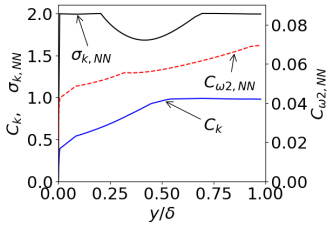
(a) Velocity.



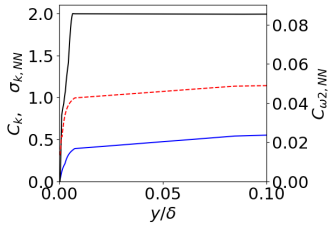
(b) Turbulent kinetic energy.



(c) Turbulent viscosity.



(d) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.



(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model. Zoomed-in view.

Figure 10: Fully-developed channel flow. $Re_\tau = 10\,000$.

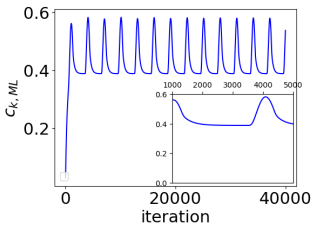
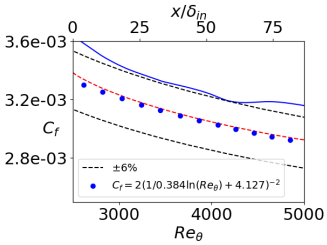
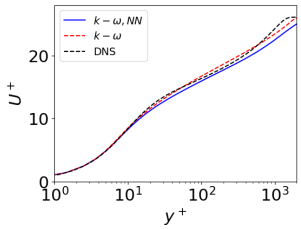


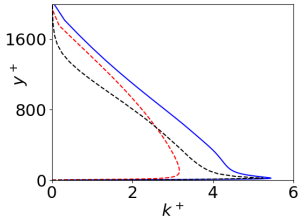
Figure 11: Fully-developed channel flow. $Re_\tau = 10\,000$. $C_{k,NN}$ vs. iteration.



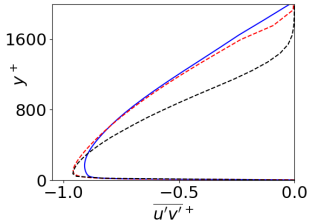
(a) Skin friction.



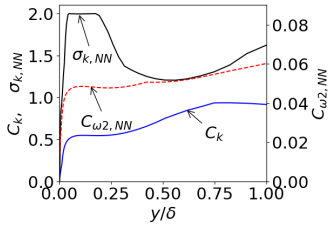
(b) Velocity.



(c) Turbulent kinetic energy.



(d) Turbulent shear stress.



(e) $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$ predicted by the NN model.

Figure 12: Flat-plate boundary layer. Profiles at $Re_\theta = 4\,500$. DNS (Sillero et al., 2014)

4.1 Compute $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ using Neural Network (NN)

Note that $\sigma_{k,PINN}$, $C_{k,PINN}$ and $C_{\omega 2,PINN}$ in the previous section are expressed as function of y/δ rather than of y^+. The reason is that when repeating the predictions in the previous section using data at $Re_\tau = 2\,000$ it is found that $\sigma_{k,PINN}$ scales much better with y/δ than with y^+, see Fig. 6.

It was shown in Davidson (2025b) that this model accurately predicts fully-developed channel flow at $Re_\tau = 2\,000$. $Re_\tau = 5\,200$, $Re_\tau = 10\,000$ as well as a flat-plate boundary layer. But the drawback is that this model is not applicable to re-circulating flow since the three coefficients are expressed in y/δ. In this section, we will use NN models to predict $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$. Many different input parameters to the NN models were investigated such as P_k/ε, P_k^+, ε^+, $\nu_t/(yu_\tau)$, etc, It is important that they are properly non-dimensionalized so that they can be used in other flows as well at other Reynolds numbers. In the end, a good combination was found: the input parameters to the NN models for $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ were taken as

$$\frac{\overline{|u'v'|}_{tot}}{u_\tau^2} \quad \text{and} \quad \frac{\nu_t}{(yu_\tau)} \tag{12}$$

where y is the distance to the nearest wall and

$$\overline{|u'v'|}_{tot} = 2\left(\nu + \nu_t\right)\left(\bar{s}_{ij}\bar{s}_{ij}\right)^{1/2}, \quad \bar{s}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right) \tag{13}$$

The targets/outputs are $\sigma_{k,PINN}$ (Eq. 9), $C_{k,PINN}$ (Eq. 10) and $C_{\omega 2,PINN}$ (Eq. 11)

Minimum and maximum of the input parameters, $\overline{|u'v'|}_{tot}/u_\tau^2$ and $\nu_t/(yu_\tau)$, as well as the output parameters, $\sigma_{k,NN}$, $C_{k,NN}$, $C_{\omega 2,NN}$, are computed and stored on disk. They are then used in the NN model in the CFD code to set limits on the CFD input parameters, see Lines 42-49 in Listing 4 in the Appendix, and the output parameters, see Lines 65-67. The difference between $\sigma_{k,PINN}$ and $\sigma_{k,NN}$ is that the former is obtained from Eq. 9 where $\nu_{t,PINN}$ is given by solving an ODE (Eq. 8) using PINN; the latter is predicted by the NN model using $\sigma_{k,PINN}$ as the target. The differences between $C_{k,PINN}$/$C_{k,NN}$ and $C_{\omega 2,PINN}$/$C_{\omega 2,NN}$ are essentially the same: $C_{k,PINN}$ and $C_{\omega 2,PINN}$ are given by Eqs. 10 and 11 whereas $C_{k,NN}$ and $C_{\omega 2,NN}$ are predicted by the NN models using $C_{k,PINN}$ and $C_{\omega 2,PINN}$ as targets.

5 Results

The turbulent Prandtl number, $\sigma_{k,NN}$, and the functions $C_{k,NN}$ and $C_{\omega 2,NN}$ are in the previous section predicted by NN models. They are in this section implemented in a $k-\omega$ model in Python CFD codes. The new model is called the $k-\omega$-PINN model.

The NN models are incorporated in the CFD code **pyCALC-RANS** as follows (the procedure is identical in **pyCALC-LES**).

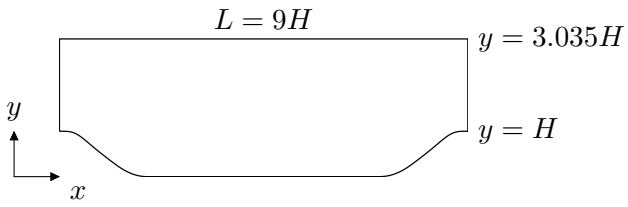
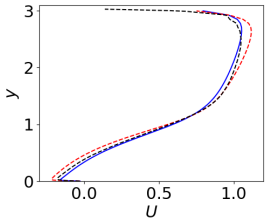


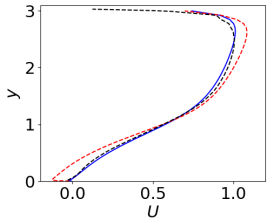
Figure 13: The geometry of the hill.

(a) $x = 0.05$.

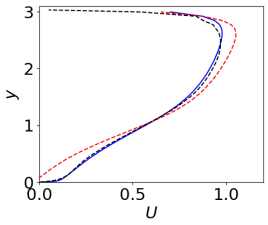
(b) $x = 1$.



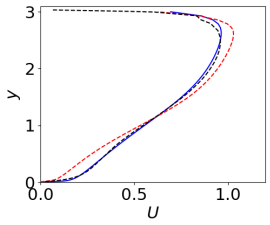
(c) $x = 3$.



(d) $x = 4$.

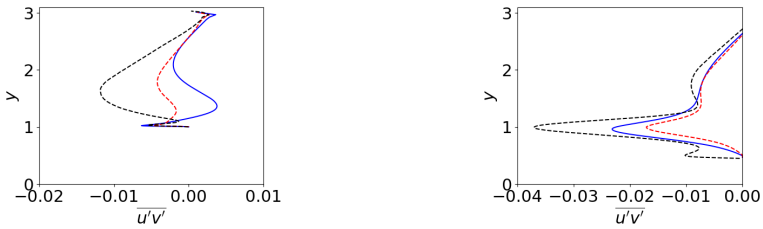


(e) $x = 5$.

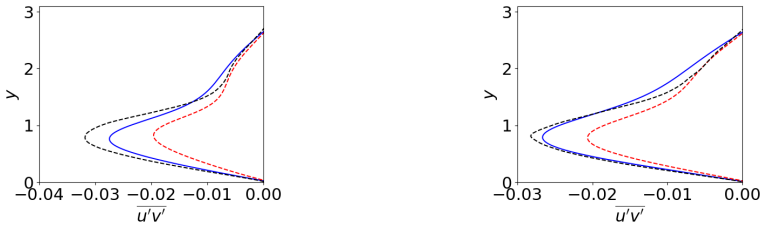


(f) $x = 7$.

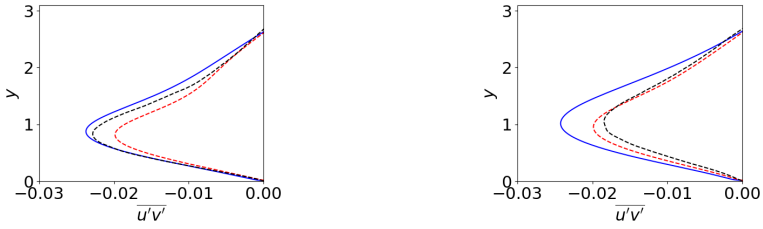
Figure 14: Hill flow. Velocity. —— : $k - \omega$-PINN-NN; - - : standard $k - \omega$; ▬ ▬ : DNS by Froehlich et al. (2005)



(a) $x = 0.05$. (b) $x = 1$.



(c) $x = 3$. (d) $x = 4$.

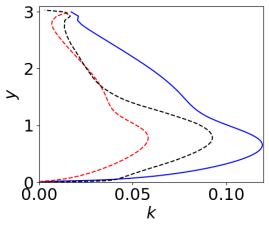


(e) $x = 5$. (f) $x = 7$.

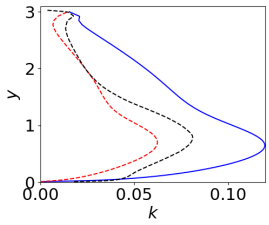
Figure 15: Hill flow. Turbulent shear stresses. —— : $k - \omega$-PINN-NN; – – : standard $k - \omega$; ▬ ▬ : DNS by Froehlich et al. (2005)

(a) $x = 0.05$.

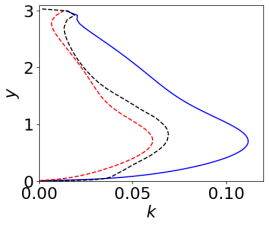
(b) $x = 1$.



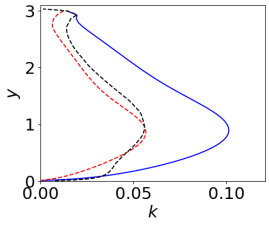
(c) $x = 3$.



(d) $x = 4$.

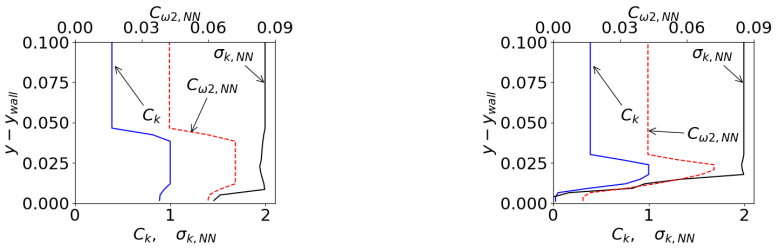


(e) $x = 5$.



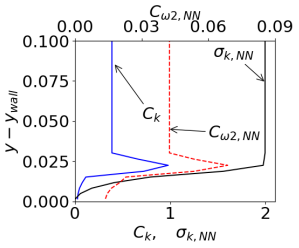
(f) $x = 7$.

Figure 16: Hill flow. Turbulent kinetic energy. —: $k - \omega$-PINN-NN; – –: standard $k - \omega$; ▬ ▬: DNS by Froehlich et al. (2005)

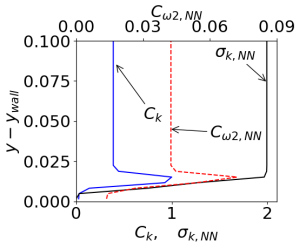


(a) $x = 0.05$.

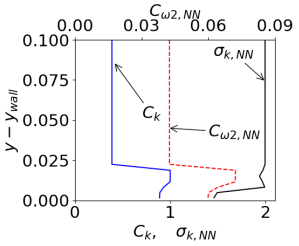
(b) $x = 1$.



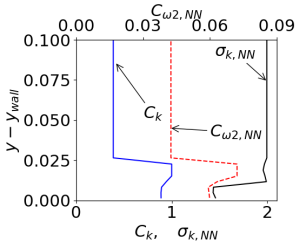
(c) $x = 3$.



(d) $x = 4$.

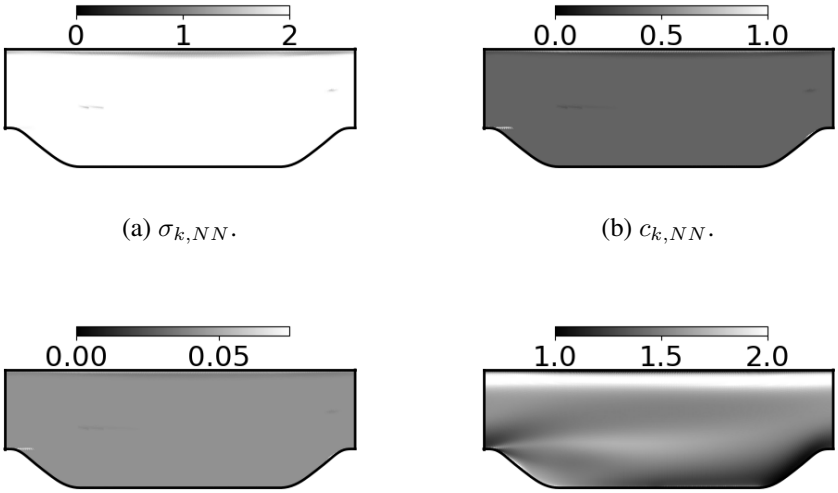


(e) $x = 5$.



(f) $x = 7$.

Figure 17: Hill flow. Turbulent Prandtl number, $\sigma_{k,NN}$, $\sigma_{k,NN}$ and $\sigma_{omega2,NN}$ predicted by the NN model. Zoomed-in view.



(a) $\sigma_{k,NN}$.

(b) $c_{k,NN}$.

(c) $c_{\omega 2,NN}$.

(d) Ratio of $\nu_{tot,NN}$ to $\nu_{tot,k-\omega}$.

Figure 18: Hill flow. Gray-scale plots of $\sigma_{k,NN}$, $c_{k,NN}$, $c_{\omega 2,NN}$ predicted by the NN model and ratio of total viscosities.

1. At the first iteration: Load the NN model into **pyCALC-RANS**
2. Solve U, V and PP equations.
3. Call the NN models to compute $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$
4. Solve k and ω equations.
5. End of global iteration. Repeat from Step 2 until convergence (1000s of iterations)

The new model is validated in three types of flows: fully-developed channel flow ($Re_\tau = 550$, 2 000, 5 200 and $Re_\tau = 10\,000$), flat-plate boundary layer flow and flow over a hill. The boundary conditions at the walls are $\bar{u} = \bar{v} = k = 0$; at the center of the wall-adjacent cells $\omega = 6\nu/(C_{\omega 2} y^2)$.

5.1 Fully-developed channel flow.

Fully-developed, i.e. one-dimensional, channel flows are simulated using **pyCALC-RANS**. The $\bar{u}$, k and ω equations are solved. The domain covers only the lower half of the channel. The wall-normal gradient is set to zero for all variables at the upper boundary. The flow is driven by a prescribed pressure gradient (the first term on the right-hand side of Eq. 2). Details of the grids are provided in Table 1.

The predicted velocity, turbulent kinetic energy and turbulent viscosity using the Wilcox $k-\omega$ model and the $k-\omega$-PINN-NN model are compared with DNS in Figs. 7, 8, 9 and 10. The velocity profiles are well predicted with both models but the turbulent kinetic energy is much better predicted with the $k-\omega$-PINN-NN model. It can be seen that the turbulent viscosities predicted by the two turbulence models are very similar in the inner part of the boundary layer ($y^+ \lesssim 100$, $y^+ \lesssim 300$, $y^+ \lesssim 1\,000$ and $y^+ \lesssim 2\,000$ for $Re_\tau = 550$, $2\,000$, $5\,200$ and $Re_\tau = 10\,000$, respectively). It may be recalled that this was indeed a requirement when developing the $k-\omega$-PINN-NN model in the previous section, see Eq. 6.

Figures 7d, 8d, 9d and 10d present the turbulent Prandtl number, $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ predicted by the NN model. They are fairly similar for all Reynolds numbers. They are also similar to $\sigma_{k,PINN}$, $C_{k,PINN}$ and $C_{\omega 2,PINN}$ (see Figs. 5 and 6a) as they should be. The NN model predicts an non-physical dip in $\sigma_{k,NN}$ at $y/\delta \simeq 0.5$ for all Reynolds number except the lowest. However, this is not a large drawback since the turbulent diffusion is anyway negligible in this region, see Fig. 1c. From the zoomed-in views in Figs. 7e, 8e, 9e and 10e it is clearly seen that the extent of the near-wall region in which $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ are reduced diminishes as the Reynolds number increases. This was also seen in Fig. 6 where $\sigma_{k,PINN}$ was predicted from Eq. 5 for two different Reynolds numbers.

It should be mentioned that $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ exhibit slow oscillations with respect to iteration number. Figure 11 shows how $\sigma_{k,NN}$ oscillates with respect to iteration number; the oscillation period is approximately $3\,000$ iterations. This makes it difficult to converge the ω equation properly. The remedy is to introduce an exponentially weighted averaging operator acting on $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ over m iterations. The average of $\sigma_{k,NN}$, for example, is defined as (Xiao and Jenny, 2012; Davidson, 2021c)

$$\langle \sigma_{k,NN} \rangle^n = a \langle \sigma_{k,NN} \rangle^{n-1} + (1-a)\sigma^n_{k,NN}, \quad a = \exp(-1/m) \tag{14}$$

Taking guidance from Fig. 11 we set m to $m = 3\,000$.

The issue of slow variations of $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ does not occur at the lowest Reynolds number nor in the flat-plat boundary layer or the hill flow. This issue is probably related to the strong elliptic character of fully-developed channel flow at high Reynolds number, i.e. the transport takes place only through diffusion.

5.2 Flat-plat boundary layer flow.

This flow is simulated using **pyCALC-RANS**. The Reynolds number at the inlet is $Re_\theta = 2\,550$. The mean profiles are taken from a pre-cursor 2D RANS. The grid has 150×90 cells (x, y). The domain size is $92\delta_{in} \times 20\delta_{in}$ where δ_{in} denotes the boundary-layer thickness at the inlet. The grid is stretched in the wall-normal direction by 10% up to $y = 6.7\delta_{in}$ where $\Delta_y = 0.5\delta_{in}$; it is stretched by 1% in the streamwise direction.

Figure 12 presents comparisons of predictions by the $k-\omega$-PINN-NN model and the Wilcox $k-\omega$ with DNS. It can be seen that the skin friction with the

$k-\omega$-PINN-NN model is in slightly worse agreement with DNS (6% too large) than the Wilcox $k-\omega$ model. However, the predicted peak of the turbulent kinetic energy by the $k-\omega$-PINN-NN model is superior compared to that predicted by the Wilcox $k-\omega$ model. The predicted $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ (see Fig. 12e) are similar to those for the channel flows. The predicted $\sigma_{k,NN}$, however, exhibits a larger and more extensive dip ($y/\delta \gtrsim 0.2$) than for the channel flow. As mentioned above, this is not large issue since the turbulent diffusion is anyway negligible in this region.

5.3 Hill flow.

The final test case is the flow over a two-dimensional periodic hill, see Fig. 13. The size of the domain is $9H \times 3.035H \times H$ in the streamwise (x) and the wall-normal (y) direction (z), respectively. The Reynolds number is $Re = 10\,600$ based on the hill height and the bulk velocity U_b at the top of the hill. Periodic boundary conditions are used in the x direction. Slip conditions are prescribed in the z direction.

Unsteady simulations are carried out marching to steady-state conditions. The time step is set to 0.01 which gives a maximum CFL of 0.4. Two global iterations (solving $\bar{u}$, $\bar{v}$, $\bar{w}$, k and ω) are carried out each time step. The flow is solved over $40\,000$ time steps which corresponds to approximately 44 through-flows; the flow variables are time-averaged over the last 100 time steps.

A wall function is used at the upper wall for the hill flow. It is based on Reichardt's law

$$\frac{\bar{u}_P}{u_\tau} \equiv U_P^+ = \frac{1}{\kappa}\ln(1-0.4y_P^+) + 7.8\left[1-\exp\left(-y_P^+/11\right) - (y_P^+/11)\exp\left(-y_P^+/3\right)\right] \tag{15}$$

The friction velocity is then obtained by re-arranging Eq. 15 and solving the implicit equation

$$u_\tau - \bar{u}_P\left\{\ln(1-0.4y_P^+)/\kappa + 7.8\left[1-\exp\left(-y_P^+/11\right) - (y_P^+/11)\exp\left(-y_P^+/3\right)\right]\right\}^{-1} = 0 \tag{16}$$

using the Newton-Raphson method `scipy.optimize.newton` in Python. $\bar{u}_P$ and y_P^+ denote the wall-parallel velocity and non-dimensional wall distance, respectively, at the first wall-adjacent cells. The boundary condition for the wall-parallel velocity is the shear stress boundary condition, ρu_τ^2. The turbulent kinetic energy, k, and its specific dissipation, ω, are set at the wall-adjacent cells centers as

$$C_\mu^{-1/2}u_\tau^2 : k \text{ equation}$$
$$\frac{u_\tau}{\kappa y} : \omega \text{ equation}$$

where $\kappa = 0.4$.

The bulk velocity is then kept constant by adjusting β in Eq. 4 at each time step and iteration. The coefficient in Eq. 4 is computed as

$$\beta^n = \beta^{n-1} + 0.001(U_T - 2U_b^{n-1} + U_b^n) \tag{17}$$

where $U_T = 1$ is the target bulk velocity at the hill crest and n is the time step number.

The predictions are compared with DNS by Froehlich et al. (2005). The grid is the same as in the DNS in the $x - y$ plane, i.e. 196×128 ($x \times y$) cells. Two cells are used in the z direction. All quantities are normalized using the height of the hill (H) and the bulk velocity (U_b) at the crest of the hill.

Figures 14 compare the predicted velocity profiles with DNS and as can be seen the agreement with the $k - \omega$-PINN-NN model is much better with the standard $k - \omega$ model. The velocity profiles predicted with the $k - \omega$-PINN-NN model are better than those obtained with a steady full Reynolds-stress closure (Jakirlic and Maduta, 2015). Also the shear stresses are better predicted by the $k - \omega$-PINN-NN model model than by the standard $k - \omega$ model (which is logical as the velocity profiles are well predicted by the former model). However, it is seen that both models predict very poor Reynolds shear stresses near the crest of the hill (Fig. 14a). The reason that $\overline{u'v'}$ at $x = 0.05$ even has the wrong sign for $y > 1.15$ is that $\partial\bar{v}/\partial x < 0$ and $|\partial\bar{v}/\partial x| > |\partial\bar{u}/\partial y|$ in the expression for the shear stress (see Eq. 4)

$$\overline{u'v'} = -\nu_t \left(\frac{\partial \bar{u}}{\partial y} + \frac{\partial \bar{v}}{\partial x} \right)$$

Figures 16 show the predicted turbulent kinetic energy profiles. As can be seen the agreement is rather poor as the predicted k with the $k - \omega$-PINN-NN model is in general too large; the agreement for the standard $k - \omega$ model is actually better for $x \geq 5$. It should be mentioned that also the steady full Reynolds-stress closures gives inaccurate turbulent kinetic energy (Jakirlic and Maduta, 2015) (much too small values).

Figure 17 presents a zoomed-in view of the predicted $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ by the NN model. It can be seen that for $y - y_{wall} \gtrsim 0.025$ they take a constant value at all six x locations. Gray-scale plots in Figs. 18a, 18b and 18c confirm that they are constant in almost the entire domain. Away from the wall, $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ take the values 2, 0.39 and 0.043, respectively. The reason is that the input parameters to the NN model are limited to the minimum and maximum values during the training process of the NN model (see end of Section 4.1). The maximum values of input parameters $|\overline{u'v'}|_{tot}/u_\tau^2$ and $\nu_t/(yu_\tau)$ in the NN model are 0.995 and 0.37, respectively. The question now arises: what happens if we use these constant values ($\sigma_{k,NN}, C_{k,NN}, C_{\omega 2,NN} = 2, 0.39, 0.043$) in the entire domain? The answer is that the predicted profiles in Figs. 14 – 16 remain virtually exactly the same (not shown). However, the $k - \omega$-PINN-NN model with these constants fails in the channel flows and the flat-plate boundary layer flow; for example, the centerline velocity in channel flow at $Re_\tau = 10\,000$ and the skin friction

in the flat-plate boundary layer are 13% too low and 28% too high, respectively (not shown).

Figure 18d presents the ratio of $\nu_{tot,NN}$ to $\nu_{tot,k-\omega}$. The peak value is 2.3 and it is smaller than one in a few points new the lower wall. Integrated over the entire domain, the ratio is 1.6.

6 Conclusions

The present work proposes a methodology for using PINN and NN for improving turbulence models. A new $k-\omega$ turbulence model, $k-\omega$-PINN-NN, is presented. The k equation is turned into an ODE for the turbulent viscosity, $\nu_{t,PINN}$, by taking k, P^k and ε from DNS. Then the turbulent Prandtl number in the k equation is given by $\sigma_{k,PINN} = \nu_{t,PINN}/\nu_t$. In order to keep the turbulent viscosity, ν_t, the same as in the standard $k-\omega$ model, two new functions are introduced: one, $C_{k,NN}$, in front of the dissipation term in the k equation and another one, $C_{\omega 2,NN}$, in front of the destruction term in the ω equation. They are both obtained from the k and ω equations using DNS data.

Next, the turbulent Prandtl number, $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$ are predicted with NN models using the two input parameters, $\overline{|u'v'|}_{tot}/u_\tau^2$ and $\nu_t/(yu_\tau)$. The difference between $\sigma_{k,PINN}$ and $\sigma_{k,NN}$ is that the former is obtained from $\nu_{t,PINN}$ (which is predicted by PINN) whereas the latter is predicted by the NN model using $\sigma_{k,PINN}$ as the target. The differences between $C_{k,PINN}$ (or $C_{\omega 2,PINN}$) and $C_{k,NN}$ (or $C_{\omega 2,NN}$) is similar: the former, subscript $PINN$, are obtained via PINN whereas the latter, subscript NN, are predicted by the NN models using the former as targets.

The new $k-\omega$-PINN-NN is shown to predict excellent results in all four channel flow and the periodic hill flow. It gives good results also for the flat-plate boundary layer (C_f is over-predicted by 6%). For the hill flow the NN model predicts constant values of $\sigma_{k,NN} = 2$, $C_{k,NN} = 0.39$ and $C_{\omega 2,NN} = 0.043$ at $y - y_{wall} \gtrsim 0.025$. The reason is that the input parameters in the NN model are not allowed to exceed the values that were using when training NN model. An additional simulation of the hill flow was carried out using these constant values for $\sigma_{k,NN}$, $C_{k,NN}$ and $C_{\omega 2,NN}$. The predicted results were virtually the same. But when these constant values were used for the channel flow and flat-plate boundary layer, the agreement with DNS was very poor.

References

L. Davidson. Using Physical Informed Neural Network (PINN) and Neural Network (NN) to improve a $k-\omega$ turbulence model: Python CFD code and PINN script. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2025a. URL https://www.tfd.chalmers.se/~lada/Using-Physical-

Informed-Neural-Network-PINN-and-NN-improve-a-k-omega-turbulence-model.html.

S. Yazdani and M. Tahani. Data-driven discovery of turbulent flow equations using physics-informed neural networks. *Physics of Fluids*, 36(3):035107, 03 2024. ISSN 1070-6631. doi: 10.1063/5.0190138. URL https://doi.org/10.1063/5.0190138.

S. Luo, M. Vellakal, S. Koric, V. Kindratenko, and J. Cui. Parameter identification of RANS turbulence model using physics-embedded neural network. In H. Jagode, H. Anzt, G. Juckeland, and H. Ltaief, editors, *High Performance Computing*, pages 137–149, Cham, 2020. Springer International Publishing. URL https://link.springer.com/book/10.1007/978-3-030-59851-8.

S. Thakur, E. Esmaili, S. Libring, L. Solorio, and A. M. Ardekani. Inverse resolution of spatially varying diffusion coefficient using physics-informed neural networks. *Physics of Fluids*, 36(8):081915, 08 2024. ISSN 1070-6631. doi: 10.1063/5.0207453. URL https://doi.org/10.1063/5.0207453.

D. C. Wilcox. Reassessment of the scale-determining equation. *AIAA Journal*, 26 (11):1299–1310, 1988.

M. Lee and R. D. Moser. Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. *Journal of Fluid Mechanics*, 774:395–415, 2015. doi: 10.1017/jfm.2015.268. URL https://doi.org/10.1017/jfm.2015.268.

L. Davidson. pyCALC-RANS: a 2D Python code for RANS. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021a. URL https://www.tfd.chalmers.se/~lada/pyCALC-RANS.html.

B. van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of Computational Physics*, 32(1):101–136, 1979. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(79)90145-1. URL https://www.sciencedirect.com/science/article/pii/0021999179901451.

C. M. Rhie and W. L. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21:1525–1532, 1983.

L. Davidson. pyCALC-LES: a Python code for DNS, LES and Hybrid LES-RANS. Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021b. URL https://www.tfd.chalmers.se/~lada/postscript_files/py-calc-les.pdf.

J.A. Sillero, J. Jimenez, and R.D. Moser. One-point statistics for turbulent wall-bounded flows at Reynolds numbers up to $\delta^+ \simeq 2000$. *Physics of Fluids*, 25(105102), 2014. doi: https://doi.org/10.1063/1.4823831. URL https://doi.org/10.1063/1.4823831.

L. Davidson. Using Physical Informed Neural Network (PINN) to improve a $k - \omega$ turbulence model. In *15th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements (ETMM15), Dubrovnik on 22-24 September*, 2025b. URL https://www.tfd.chalmers.se/~lada/Using-Physical-Informed-Neural-Network-PINN-improve-a-k-omega-turbulence-model.html.

J. Froehlich, C. Mellen, W. Rodi, L. Temmerman, M. A., and Leschziner. Highly-resolved large eddy simulations of separated flow in a channel with streamwise periodic constrictions. *Journal of Fluid Mechanics*, 526:19–66, 2005.

H. Xiao and P. Jenny. A consistent dual-mesh framework for hybrid LES/RANS modeling. *Journal of Computational Physics*, 231:1848–1865, 2012.

L. Davidson. Non-zonal detached eddy simulation coupled with a steady rans solver in the wall region. *International Journal of Heat and Fluid Flow*, 92: 108880, 2021c. ISSN 0142-727X. doi: https://doi.org/10.1016/j.ijheatfluidflow.2021.108880. URL https://www.sciencedirect.com/science/article/pii/S0142727X21001107.

S. Jakirlic and R. Maduta. Extending the bounds of 'steady' rans closures: Toward an instability-sensitive reynolds stress model. *International Journal of Heat and Fluid Flow*, 51:175–194, 2015. ISSN 0142-727X. doi: https://doi.org/10.1016/j.ijheatfluidflow.2014.09.003. URL https://www.sciencedirect.com/science/article/pii/S0142727X14001180. Theme special issue celebrating the 75th birthdays of Brian Launder and Kemo Hanjalic.

A Simple Python scripts

```
class NN(nn.Module):
 def __init__(self):
  self.lay_1=nn.Linear(1, 2) # Connection 0-1
  self.lay_2=nn.Linear(2, 1) # Connection 1-2
 def forward(self, x):
  x = torch.nn.functional.sigmoid(self.lay_1(x))
  out = torch.nn.functional.sigmoid(self.lay_2(x))
```

Listing 1: Simple NN. Initiation and forward step

```
#  initiate the NN model
model = NN()
# define input, X
X=np.zeros(nj,1))
```

```
X[:,0] = scaler_yplus.fit_transform(yplus)[:,0]
# define target f
f = f_DNS
# Training loop
for epoch in range(max_no_epoch):
# Compute prediction and loss, L
 Y = model(X) #prediction
 L = loss_fn(f, Y)  # L = |f-Y|_2
 L.backward()
```

Listing 2: Simple NN. Prediction and backward step

```
def ODE(y, nut):
 nut_y = grad(nut, y, torch.ones\
   (y.size()[0], 1,),create_graph=True)[0]
# Differential equation loss
 ODE_loss = (nu+nut)*k_yy + k_y*nut_y + Pk - eps
 ODE_loss = torch.sum(ODE_loss ** 2)
# b.c. loss
 BC_loss = (nut[0] - nut_0) ** 2
 return ODE_loss, BC_loss
nut = model(x)
loss_ODE, loss_bc = ODE(y,nut)
L = loss_ODE+100*loss_bc
L.backward()
```

Listing 3: Python code for PINN.

```
def modify_PINN(prand_k_ML, c_k_ML, c_omega_2_ML):
# load packages etc
.
.
.
    if itstep == 0 and iter == 0:
# load data c_k
      NN_c_k = torch.load('NN-model.pth',weights_only=False)
      scaler_vist_over_y = load('model-scaler-vist_over_yin')
      scaler_uv = load('model-scaler-uv_tot.bin')

      name='min-max.txt'
      uv_min,uv_max,vist_over_y_min,vist_over_y_max = xp.loadtxt(name)

# load NN models for prand_k, c_omega_2
.
.
.

# compute ustar, south wall
    ustar_s=(abs(u2d[:,0])*viscos/dist2d[:,0])**0.5
#make it 2D
    ustar_s=xp.repeat(ustar_s[:,None], repeats=nj, axis=1)
# compute ustar, north wall (from wall functions)
    ustar_n=cmu**0.25*k2d[:,-1]**0.5
#make it 2D
    ustar_n=xp.repeat(ustar_n[:,None], repeats=nj, axis=1)
    ywall_s=0.5*(y2d[0:-1,0]+y2d[1:,0])
    dist_s=yp2d-ywall_s
# dist2d = distance to nearest wall
    ustar = xp.where(dist_s > dist2d[:,0],ustar_n,ustar_s)

# strain-rate tensor
    s11 = dudx
    s12 = 0.5**(dudy+dvdx)
    s21 = s12
    s22 = dvdy
    ss = (2*(s11**2+s12**2+s21**2+s22**2))**0.5
    uv = vis2d*ss/ustar**2
    vist_over_y = (vis2d - viscos)/dist2d/ustar

# limit min/max
# set limits on uv
```

```
    uv=xp.minimum(uv,uv_max)
    uv=xp.maximum(uv,uv_min)

# set limits on vist_over_y
    vist_over_y=xp.minimum(vist_over_y,vist_over_y_max)
    vist_over_y=xp.maximum(vist_over_y,vist_over_y_min)

# give input parameters to NN model
    vist_over_y = vist_over_y.reshape(-1,1)
    uv= uv.reshape(-1,1)
    X=xp.zeros((len(uv),2))
    X[:,0] = scaler_vist_over_y.transform(vist_over_y)[:,0]
    X[:,1] = scaler_uv.transform(uv)[:,0]
    X_tensor = torch.tensor(X, dtype=torch.float32)

# predict prand_k
    prand_k_pred = NN_prand_k(X_tensor)
# transform from tensor to numpy
    prand_k_ML = prand_k_pred.detach().numpy()[:,0]
    prand_k_ML = xp.reshape(prand_k_ML,(ni,nj,nk))

# set limits om prand_k_ML
    prand_k_ML=np.minimum(prand_k_ML,prand_k_ML_max)
    prand_k_ML=np.maximum(prand_k_ML,prand_k_ML_min)

# predict c_K_ML and c_omega_2_ML
.
.
.
    return prand_k_ML, c_k_ML, c_omega_2_ML
```

Listing 4: NN model in the CFD code (hill flow).