
IMPLEMENTATION AND VERIFICATION OF THE RESOLVED REYNOLDS STRESS TRANSPORT EQUATIONS IN OPENFOAM

Mario J. Rincón^{1,2,*} , Christoffer Hansen¹ , Martino Reclari², Mahdi Abkar^{1,*} 

¹*Department of Mechanical and Production Engineering, Aarhus University, 8200 Aarhus N, Denmark*

²*Quality & Sustainability Department, Kamstrup A/S, 8660 Skanderborg, Denmark*

ABSTRACT

The analysis of the Reynolds Stress Transport Equation (RSTE) provides fundamental physical insights that are essential for the development and validation of advanced turbulence models. However, a comprehensive and validated tool for computing the complete RSTE budget is absent in the widely-used open-source Computational Fluid Dynamics (CFD) framework, OpenFOAM. This work addresses this gap by presenting the implementation and *a posteriori* validation of a function object library for calculating all terms of the resolved RSTE budget in Large-Eddy Simulations (LES). The library is applied to simulate two canonical wall-bounded turbulent flows: a channel flow and a pipe flow, both at a friction Reynolds number of $Re_\tau = 180$. The implementation is validated through a mesh refinement study where the results from the LES simulations are systematically compared against high-fidelity Direct Numerical Simulation (DNS) data. The computed budget terms are observed to converge systematically towards the DNS reference data. This validation demonstrates that the implemented library accurately captures the intricate balance of all budget terms. This contribution provides the open-source CFD community with a powerful utility for detailed turbulence analysis, thereby facilitating deeper physical understanding and accelerating the development of next-generation turbulence models.

1 Introduction

The accurate prediction of turbulent flows remains a cornerstone of modern fluid dynamics, with profound implications for a multitude of scientific and engineering applications [1, 2]. While high-fidelity approaches such as Direct Numerical Simulation (DNS) and Large-Eddy Simulation (LES) offer detailed insights into turbulence physics, their prohibitive computational cost restricts their use to academic studies at low Reynolds numbers [3, 4]. Consequently, Reynolds-Averaged Navier-Stokes (RANS) models continue to be the predominant tool for industrial Computational Fluid Dynamics (CFD) due to their computational efficiency and robustness [5, 6].

The accuracy of RANS simulations is fundamentally dependent on the fidelity of the turbulence model used to approximate the Reynolds Stress Tensor (RST). The majority of widely-used RANS models are linear eddy-viscosity models, which rely on the Boussinesq hypothesis to relate the RST to the mean rate of strain [7]. However, the assumption of isotropy inherent in this hypothesis is known to be invalid in a wide range of complex flows, including those with strong streamline curvature, secondary flows, or significant body forces, leading to predictive inaccuracies [8, 9]. To improve turbulence models and gain a deeper physical understanding of complex flow phenomena, an analysis of the transport equation for the Reynolds stresses is highly valuable [10]. The exact transport equation for the RST, which is called the Reynolds Stress Transport Equation (RSTE) here, but which is also often referred to as the Reynolds stress budget, describes the evolution of the individual stress components. Each term within this equation—such as production, dissipation, pressure-strain correlation, and turbulent and viscous diffusion—represents a distinct physical mechanism responsible for the creation, destruction, and redistribution of the Reynolds stresses. A term-by-term analysis of this budget therefore provides crucial information that is essential for the development and validation of more sophisticated turbulence closures, including second-moment closure models and advanced data-driven approaches [11–17].

Despite its diagnostic power, the detailed evaluation of the RST budget is not a standard feature in many general-purpose CFD software packages. In particular, for the widely-used open-source CFD framework OpenFOAM [18], no readily available and validated utility for a comprehensive RST budget analysis exists. This limitation presents a

significant barrier to researchers and engineers who require detailed turbulence analysis for model development or for understanding the intricate physics of their specific applications. The implementation of such a tool is non-trivial, demanding careful numerical treatment to ensure accuracy and consistency with the flow solver. To address this critical gap, a library for the calculation of all terms in the resolved RSTE in LES has been developed and implemented within the OpenFOAM framework. This new functionality enables a complete decomposition of the resolved RST budget, providing a powerful diagnostic tool for the turbulence modelling community. However, the development of the code alone is insufficient; its credibility relies upon rigorous verification and validation. Therefore, a primary contribution of this work is the comprehensive assessment of the implemented library against benchmark data of the highest available fidelity.

The verification and validation of the developed tool have been performed against high-quality DNS data from studies of canonical turbulent flows [19, 20]. By comparing the resolved RST budget terms computed from OpenFOAM-based highly resolved LES with the reference DNS data, a term-by-term quantification of the implementation's accuracy is achieved. This process ensures that the library provides a reliable and accurate representation of the resolved RST budget, thereby establishing it as a trustworthy tool for scientific research and advanced engineering analysis.

This study is organised as follows. The governing equations and the theoretical formulation of the resolved RSTE are presented in Section 2. The numerical implementation of the budget terms as a function object library in OpenFOAM is detailed in Section 3. In Section 4, the results of the verification and validation studies are presented, where the computed budget terms are compared against DNS data for two benchmark cases. Finally, concluding remarks and an outlook on future work are provided in Section 5.

2 Resolved Reynolds stress transport equations

This section gives an introduction to the resolved RSTE in LES. First, the general form is covered in 2.1, including the Turbulent Kinetic Energy (TKE) budget. After this, the resolved RSTE formulation in Cartesian coordinates is presented in 2.2. Finally, a short discussion of alternative coordinate systems is given in 2.3.

2.1 General form

The total RSTE for the full velocity and pressure fields can be derived from the Navier-Stokes equations [2]. Here, the focus is on the resolved RSTE which can be derived similarly starting from the filtered Navier-Stokes equations [21]. In general index notation, i.e., without specifying a coordinate system, the resolved RSTE can be written as

$$\frac{\partial R_{ij}}{\partial t} = C_{ij} + P_{ij} + T_{ij} + D_{p,ij} + \Phi_{ij} + D_{\nu,ij} + \varepsilon_{ij} + \Pi_{ij}, \quad (1)$$

where R_{ij} is the resolved RST. The remaining terms are: C_{ij} convection, P_{ij} production, T_{ij} turbulent transport, $D_{p,ij}$ pressure-diffusion, Φ_{ij} pressure-strain, $D_{\nu,ij}$ viscous diffusion, ε_{ij} viscous dissipation, Π_{ij} subgrid-scale (SGS) term. Beyond the individual components of the resolved RSTE budget, the TKE budget is often a primary quantity of interest. It is obtained as half the trace of Eq. (1) and reads

$$\frac{\partial k}{\partial t} = C_k + P_k + T_k + D_{p,k} + D_{\nu,k} + \varepsilon_k + \Pi_k, \quad (2)$$

where $k = \frac{1}{2}\text{tr}(R_{ij})$ is the resolved TKE, $C_k = \frac{1}{2}\text{tr}(C_{ij})$ is the TKE convection, etc. It should be noted that the pressure-strain term is absent from the TKE budget as it vanishes due to incompressibility; $\Phi_k = \frac{1}{2}\text{tr}(\Phi_{ij}) = \langle \tilde{p}' \partial_j \tilde{u}_j' \rangle = 0$. Here, $\tilde{\cdot}$ indicates a low pass filtering operation.

In this study, LES using eddy viscosity based SGS models are considered. Therefore, the deviatoric part of the SGS tensor τ_{ij} is modelled as

$$\tau_{ij}^d = \tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = -2\nu_t\tilde{S}_{ij}, \quad (3)$$

where $\tilde{S}_{ij}$ is the filtered rate-of-strain tensor and ν_t is the eddy viscosity. In LES, this deviatoric part is included in the filtered momentum equations, while the isotropic part is absorbed into the a modified pressure $p^* = p - \frac{1}{3}\tau_{kk}$. In remainder of this study, it is understood to be that the pressure and SGS tensor used in the resolved RSTE are the modified pressure and deviatoric part of the SGS tensor. Similarly, for simplicity, the asterisk superscript on the modified pressure is not written explicitly.

2.2 Cartesian coordinates

In Cartesian coordinates and using the Einstein summation convention, the terms on the right-hand side of the resolved RSTE in Eq. (1) take the following form (see, e.g., [21])

$$C_{ij} = -\langle \tilde{u}_k \rangle \frac{\partial \langle \tilde{u}'_i \tilde{u}'_j \rangle}{\partial x_k}, \quad (4)$$

$$P_{ij} = -\left(\langle \tilde{u}'_j \tilde{u}'_k \rangle \frac{\partial \langle \tilde{u}_i \rangle}{\partial x_k} + \langle \tilde{u}'_i \tilde{u}'_k \rangle \frac{\partial \langle \tilde{u}_j \rangle}{\partial x_k} \right), \quad (5)$$

$$T_{ij} = -\frac{\partial \langle \tilde{u}'_i \tilde{u}'_j \tilde{u}'_k \rangle}{\partial x_k}, \quad (6)$$

$$D_{p,ij} = -\left(\frac{\partial \langle \tilde{u}'_i \tilde{p}' \rangle}{\partial x_j} + \frac{\partial \langle \tilde{u}'_j \tilde{p}' \rangle}{\partial x_i} \right), \quad (7)$$

$$\Phi_{ij} = \left\langle \tilde{p}' \left(\frac{\partial \tilde{u}'_i}{\partial x_j} + \frac{\partial \tilde{u}'_j}{\partial x_i} \right) \right\rangle, \quad (8)$$

$$D_{\nu,ij} = \nu \frac{\partial^2 \langle \tilde{u}'_i \tilde{u}'_j \rangle}{\partial x_k \partial x_k}, \quad (9)$$

$$\varepsilon_{ij} = -2\nu \left\langle \frac{\partial \tilde{u}'_i}{\partial x_k} \frac{\partial \tilde{u}'_j}{\partial x_k} \right\rangle, \quad (10)$$

$$\Pi_{ij} = -\left(\left\langle \tilde{u}'_j \frac{\partial \tau_{ik}^d}{\partial k} \right\rangle + \left\langle \tilde{u}'_i \frac{\partial \tau_{jk}^d}{\partial k} \right\rangle \right). \quad (11)$$

Here, $\tilde{u}_i$ is the filtered velocity, $\tilde{p}$ is the filtered modified kinematic pressure, τ_{ij}^d is the deviatoric part of the SGS tensor, ν is the kinematic viscosity, $\langle \cdot \rangle$ is an expectation operation taken as time-averaging in this study, and the fluctuating variables are defined as $\tilde{f}' = \tilde{f} - \langle \tilde{f} \rangle$. Note that this representation of the SGS term used here differs from [21], where it is decomposed into two parts, however, in this study, the full SGS contribution is kept as a single term. The explicit SGS term used here in the resolved RSTE can be written as

$$\Pi_{ij} = 2 \left(\left\langle \tilde{u}'_j \frac{\partial (\nu_t \tilde{S}_{ik})}{\partial k} \right\rangle + \left\langle \tilde{u}'_i \frac{\partial (\nu_t \tilde{S}_{jk})}{\partial k} \right\rangle \right). \quad (12)$$

Before moving on, it is important to highlight that because OpenFOAM uses a Cartesian coordinate system, all derivative operations needed for calculating the terms in the Cartesian RSTE budget can be evaluated directly using OpenFOAM functionality. However, several additional fields, which are not native to OpenFOAM, still need to be added and averaged. Further discussion of these and other implementation details are given in Section 3.

2.3 Alternative coordinate systems

While the Cartesian formulation of the RSTE discussed in 2.2 above is the one most commonly used, the RSTE can in principle be derived in any coordinate system. This can be done by deriving the RSTE in general tensor form, valid for arbitrary curvilinear coordinate systems, using tools from tensor calculus [22]. The particular form for a given coordinate system then follows directly by specifying the relevant geometric and differential quantities, e.g., the metric tensor and the nabla (∇) operator. However, the resulting RSTE formulations for non-Cartesian cases are typically much more involved. Furthermore, in OpenFOAM, everything is required to be calculated using the Cartesian quantities available during the simulation, which adds an additional level of complexity. The implementation of the RSTE in non-Cartesian coordinate systems is therefore not pursued in this work. However, while non-Cartesian coordinate systems are not pursued directly, the current implementation still gives access to the TKE budget, which is coordinate-independent (the trace is the first invariant of a second-rank tensor). This will be illustrated further in 4.3 for the case of pipe flow.

3 Implementation details

This section discusses the resolved RSTE implementation details. Initially, a quick overview of the standard fields available in OpenFOAM and how their means are calculated is provided for reference. The resolved mean velocity, mean pressure, and Reynolds stresses are defined as follows

$$\langle \tilde{u}_i \rangle, \quad \langle \tilde{p} \rangle, \quad \langle \tilde{u}'_i \tilde{u}'_j \rangle. \quad (13)$$

While it is clear how $\langle \tilde{u}_i \rangle$ and $\langle \tilde{p} \rangle$ can be calculated on the fly, however, it is less immediately apparent for the Reynolds stresses and other higher-order statistics. In order to achieve this, and following the methodology in OpenFOAM, the resolved Reynolds stresses are rewritten as

$$\langle \tilde{u}'_i \tilde{u}'_j \rangle = \langle \tilde{u}_i \tilde{u}_j \rangle - \langle \tilde{u}_i \rangle \langle \tilde{u}_j \rangle, \quad (14)$$

and then $\langle \tilde{u}_i \tilde{u}_j \rangle$ are calculated on the fly, similarly to $\langle \tilde{u}_i \rangle$ and $\langle \tilde{u} \rangle$. The Reynolds stresses $\langle \tilde{u}'_i \tilde{u}'_j \rangle$ can then be calculated from $\langle \tilde{u}_i \rangle$ and $\langle \tilde{u}_i \tilde{u}_j \rangle$ when needed for write-out or additional calculations. Beyond the mean velocity and Reynolds stresses, several additional fields are required for the RSTE budget. The general strategy taken in the implementation is to use built-in OpenFOAM functionality as much as possible. However, this comes at the cost of the current implementation being somewhat memory inefficient. Specifically, defining additional fields is done using the *components*, *grad*, and *multiply* functionalities. The averaging of these new fields is then performed using *fieldAverage*.

Below, each term in the resolved RSTE budget is reviewed, any potential additional fields required are identified, and the calculation of their averaging is discussed. Likewise, examples of the OpenFOAM code for calculating the different terms are also given to illustrate the implementation.

3.1 Convective term

The resolved convective term is given by

$$C_{ij} = -\langle \tilde{u}_k \rangle \frac{\partial \langle \tilde{u}'_i \tilde{u}'_j \rangle}{\partial x_k}. \quad (15)$$

For the implementation, the convective term is written in conservative form as follows

$$C_{ij} = -\frac{\partial}{\partial x_k} (\langle \tilde{u}_k \rangle \langle \tilde{u}'_i \tilde{u}'_j \rangle). \quad (16)$$

The implementation then boils down to calculating the vectors (k being the vector index)

$$\langle \tilde{u}_k \rangle \langle \tilde{u}'_i \tilde{u}'_j \rangle, \quad (17)$$

for all i and j in the upper diagonal, taking the divergence of these vectors using the *div* functionality, and then collecting them appropriately in a *volSymmTensorField*. The OpenFOAM code for calculating C_{11} (denoted C_{xx} in the code) is given below as an example:

```

1 // ----- Calculate Cxx component ----- //
2
3 // Define field to store vector
4 volVectorField u_uu
5 (
6     IOobject
7     (
8         "u_uu_tmp",
9         mesh().time().timeName(),
10        mesh(),
11        IOobject::NO_READ,
12        IOobject::NO_WRITE
13    ),
14    mesh(),
15    dimensionedVector
16    (
17        "zero",
18        dimVelocity*dimVelocity*dimVelocity,
19        vector::zero
20    )
21 );
22
23 // Calculate vector
24 // Here u, v, and w are vel. comps. and uu is the xx Reynolds stress comp.
25 vectorField& resultxx = u_uu.ref();
26 forAll(resultxx, i)
27 {
28     resultxx[i].x() = u[i] * uu[i];
29     resultxx[i].y() = v[i] * uu[i];
30     resultxx[i].z() = w[i] * uu[i];

```

```

31 }
32 u_uu.correctBoundaryConditions();
33
34 // Calculate divergence
35 volScalarField Cxx
36 (
37     IOobject
38     (
39         "Cxx",
40         mesh().time().timeName(),
41         mesh(),
42         IOobject::NO_READ,
43         IOobject::AUTO_WRITE
44     ),
45     fvc::div(u_uu)
46 );
47 Cxx.correctBoundaryConditions();

```

3.2 Production term

The resolved production term is given by

$$P_{ij} = - \left(\langle \tilde{u}'_j \tilde{u}'_k \rangle \frac{\partial \langle \tilde{u}_i \rangle}{\partial x_k} + \langle \tilde{u}'_i \tilde{u}'_k \rangle \frac{\partial \langle \tilde{u}_j \rangle}{\partial x_k} \right). \quad (18)$$

For the implementation, the velocity gradient tensor is first calculated from the mean velocity using *grad* and is then contracted with the Reynolds stress tensor. The resulting tensor is then made symmetric using the *symm* functionality, and the negative sign is added. Note that *symm* includes a factor of a half which is compensated by multiplying the result by two. The OpenFOAM code for calculating the full production term is given below:

```

1 // ----- Calculate production term ----- //
2
3 // Calculate velocity gradient tensor
4 volTensorField gradUMean = fvc::grad(UMean);
5
6 // Calculate production
7 volSymmTensorField P_ij_RSTE
8 (
9     IOobject
10    (
11        prodOutNm, mesh().time().timeName(),
12        mesh(), IOobject::NO_READ,
13        IOobject::AUTO_WRITE
14    ),
15    -2.0 * symm(RMean & gradUMean)
16 );
17 P_ij_RSTE.correctBoundaryConditions();

```

3.3 Turbulent transport term

The resolved turbulent transport term is given by

$$T_{ij} = - \frac{\partial \langle \tilde{u}'_i \tilde{u}'_j \tilde{u}'_k \rangle}{\partial x_k}. \quad (19)$$

Here, the triple products $\langle \tilde{u}'_i \tilde{u}'_j \tilde{u}'_k \rangle$ are additional fields that needs to be calculated and averaged. The idea is to expand them, similar to the Reynolds stresses, to allow on-the-fly averaging. Specifically, using $\tilde{u}'_i = \tilde{u}_i - \langle \tilde{u}_i \rangle$, the triple correlation can be expanded as

$$\begin{aligned}
 \langle \tilde{u}'_i \tilde{u}'_j \tilde{u}'_k \rangle &= \langle \tilde{u}_i \tilde{u}_j \tilde{u}_k \rangle - [\langle \tilde{u}_i \rangle \langle \tilde{u}_j \tilde{u}_k \rangle + \langle \tilde{u}_j \rangle \langle \tilde{u}_i \tilde{u}_k \rangle + \langle \tilde{u}_k \rangle \langle \tilde{u}_i \tilde{u}_j \rangle] \\
 &\quad + 3 \langle \tilde{u}_i \rangle \langle \tilde{u}_j \rangle \langle \tilde{u}_k \rangle - \langle \tilde{u}_i \rangle \langle \tilde{u}_j \rangle \langle \tilde{u}_k \rangle \\
 &= \langle \tilde{u}_i \tilde{u}_j \tilde{u}_k \rangle - [\langle \tilde{u}_i \rangle \langle \tilde{u}_j \tilde{u}_k \rangle + \langle \tilde{u}_j \rangle \langle \tilde{u}_i \tilde{u}_k \rangle + \langle \tilde{u}_k \rangle \langle \tilde{u}_i \tilde{u}_j \rangle] + 2 \langle \tilde{u}_i \rangle \langle \tilde{u}_j \rangle \langle \tilde{u}_k \rangle.
 \end{aligned} \quad (20)$$

The terms in the square parentheses can be further rewritten using $\tilde{u}_i = \langle \tilde{u}_i \rangle + \tilde{u}'_i$ to get

$$[\langle \tilde{u}_i \rangle \langle \tilde{u}_j \tilde{u}_k \rangle + \langle \tilde{u}_j \rangle \langle \tilde{u}_i \tilde{u}_k \rangle + \langle \tilde{u}_k \rangle \langle \tilde{u}_i \tilde{u}_j \rangle] = 3\langle \tilde{u}_i \rangle \langle \tilde{u}_j \rangle \langle \tilde{u}_k \rangle + [\langle \tilde{u}_i \rangle \langle \tilde{u}'_j \tilde{u}'_k \rangle + \langle \tilde{u}_j \rangle \langle \tilde{u}'_i \tilde{u}'_k \rangle + \langle \tilde{u}_k \rangle \langle \tilde{u}'_i \tilde{u}'_j \rangle]. \quad (21)$$

Putting it all together, the following relation is obtained

$$\langle \tilde{u}'_i \tilde{u}'_j \tilde{u}'_k \rangle = \langle \tilde{u}_i \tilde{u}_j \tilde{u}_k \rangle - [\langle \tilde{u}_i \rangle \langle \tilde{u}'_j \tilde{u}'_k \rangle + \langle \tilde{u}_j \rangle \langle \tilde{u}'_i \tilde{u}'_k \rangle + \langle \tilde{u}_k \rangle \langle \tilde{u}'_i \tilde{u}'_j \rangle] - \langle \tilde{u}_i \rangle \langle \tilde{u}_j \rangle \langle \tilde{u}_k \rangle. \quad (22)$$

Thus, to summarise, the resolved mean velocities $\langle \tilde{u}_i \rangle$, the Reynolds stresses $\langle \tilde{u}'_i \tilde{u}'_j \rangle$, and the triple-velocity-products $\langle \tilde{u}_i \tilde{u}_j \tilde{u}_k \rangle$ are needed. The triple-products are constructed by using *components* on the velocity and then using the *multiply* functionality to get the required products. The averaging is done using *fieldAverage*. After calculating the fluctuating triple-products $\langle \tilde{u}'_i \tilde{u}'_j \tilde{u}'_k \rangle$ using Eq. (22), they are collected together in vectors (k is the vector index) for every i and j , similar to the convective term. The divergences of these vectors are then calculated using *div*, the negative sign is added, and, finally, everything is collected in a *volSymmTensorField*. The OpenFOAM code for calculating T_{11} (denoted T_{xx} in the code) is given below as an example:

```

1  // ----- Calculate Txx component ----- //
2
3  // Define field to store vector
4  volVectorField UxUxUk
5  (
6      IOobject
7      (
8          "UxUxUk_tmp",
9          mesh().time().timeName(),
10         mesh(), IOobject::NO_READ,
11         IOobject::NO_WRITE
12     ),
13     mesh(),
14     dimensionedVector
15     (
16         "zero",
17         dimVelocity*dimVelocity*dimVelocity,
18         vector::zero
19     )
20 );
21
22 // Calculate vector (<u'u'u'>, <u'u'v'>, <u'u'w'>)
23 vectorField& resultxx = UxUxUk.ref();
24 forAll(resultxx, i)
25 {
26     resultxx[i].x() = ( uu[u[i]]
27                         - (u[i] * uu[i] + u[i] * uu[i] + u[i] * uu[i])
28                         - u[i] * u[i] * u[i] );
29     resultxx[i].y() = ( uv[u[i]]
30                         - (u[i] * uv[i] + u[i] * uv[i] + v[i] * uu[i])
31                         - u[i] * u[i] * v[i] );
32     resultxx[i].z() = ( uw[u[i]]
33                         - (u[i] * uw[i] + u[i] * uw[i] + w[i] * uu[i])
34                         - u[i] * u[i] * w[i] );
35 }
36 UxUxUk.correctBoundaryConditions();
37
38 // Calculate divergence
39 volScalarField Txx
40 (
41     IOobject
42     (
43         outputTxx,
44         mesh().time().timeName(),
45         mesh(),
46         IOobject::NO_READ,
47         IOobject::AUTO_WRITE
48     ),
49     fvc::div(UxUxUk)

```

```

50     );
51     Txx.correctBoundaryConditions();

```

3.4 Viscous diffusion term

The resolved viscous diffusion term is given by

$$D_{\nu,ij} = \nu \frac{\partial^2 \langle \tilde{u}'_i \tilde{u}'_j \rangle}{\partial x_k \partial x_k}, \quad (23)$$

and does not require any new fields. It only requires the calculation of the Laplacian of the Reynolds stresses using *laplacian*. However, an explicit calculation of the Laplacian was observed to give spurious results at the first off-wall cell centre. Therefore, in the implementation, these values are overwritten with the value from the nearest wall-normal neighbour. Hence, the OpenFOAM code for calculating the full viscous diffusion term is given below:

```

1 // ----- Calculate viscous diffusion term ----- //
2
3 volSymmTensorField D_ij_RSTE
4 (
5     IOobject
6     (
7         viscDiffOutNm, mesh().time().timeName(),
8         mesh(),
9         IOobject::NO_READ,
10        IOobject::AUTO_WRITE
11    ),
12    nu * fvc::laplacian(RMean)
13 );
14
15 // ----- Additional code for first cell interpolation ----- //
16 // ... code not included for brevity...

```

3.5 Pressure diffusion term:

The resolved pressure diffusion term is given by

$$D_{p,ij} = - \left(\frac{\partial \langle \tilde{u}'_j \tilde{p}' \rangle}{\partial x_i} + \frac{\partial \langle \tilde{u}'_i \tilde{p}' \rangle}{\partial x_j} \right), \quad (24)$$

where it should be remembered that $\tilde{p}$ is the modified kinematic pressure. The double-correlations are then expanded as follows

$$\langle \tilde{u}'_i \tilde{p}' \rangle = \langle \tilde{u}_i \tilde{p} \rangle - \langle \tilde{u}_i \rangle \langle \tilde{p} \rangle. \quad (25)$$

Next, *multiply* is used to create a new field for the velocity-pressure product and the averaging is done using *fieldAverage*. The gradient tensor corresponding to $\langle u'_i p' \rangle$ is calculated, symmetrized using *symm*, and the negative sign is added. The factor of a half from *symm* is compensated for by multiplying with two. The OpenFOAM code for calculating the full pressure diffusion term is given below:

```

1 // ----- Calculate pressure diffusion term ----- //
2
3 // Calculate fluctuating velocity-pressure product
4 volVectorField UPrimePPrimeMean
5 (
6     IOobject
7     (
8         "UPrimePPrimeMean",
9         mesh().time().timeName(),
10        mesh(),
11        IOobject::NO_READ,
12        IOobject::NO_WRITE
13    ),
14    UpMean - (UMean * pMean)
15 );
16

```

```

17 // Calculate corresponding gradient tensor
18 volTensorField gradUPrimePPrimeMean = fvc::grad(UPrimePPrimeMean);
19
20 // Calculate pressure diffusion
21 volSymmTensorField Dp_ij_RSTE
22 (
23     IOobject
24     (
25         outputDpij,
26         mesh().time().timeName(),
27         mesh(),
28         IOobject::NO_READ,
29         IOobject::AUTO_WRITE
30     ),
31     -2.0 * symm(gradUPrimePPrimeMean)
32 );
33 Dp_ij_RSTE.correctBoundaryConditions();

```

3.6 Pressure-strain term

The resolved pressure-strain term is given by

$$\Phi_{ij} = \left\langle \tilde{p}' \left(\frac{\partial \tilde{u}'_i}{\partial x_j} + \frac{\partial \tilde{u}'_j}{\partial x_i} \right) \right\rangle = 2 \langle \tilde{p}' \tilde{S}'_{ij} \rangle, \quad (26)$$

where it should be remembered that $\tilde{p}$ is the modified kinematic pressure and $\tilde{S}'_{ij}$ is the filtered fluctuating rate-of-strain tensor. The double correlation is expanded as follows

$$\langle \tilde{p}' \tilde{S}'_{ij} \rangle = \langle \tilde{p} \tilde{S}_{ij} \rangle - \langle \tilde{p} \rangle \langle \tilde{S}_{ij} \rangle. \quad (27)$$

In terms of implementation, a new field consisting of the product of the pressure and the velocity gradient tensor is introduced using *multiply*. This field is subsequently averaged using *fieldAverage*. The remaining calculations are given in the OpenFOAM code below:

```

1 // ----- Calculate pressure-strain term ----- //
2
3 // Calculate SijMean from UMean
4 volSymmTensorField SijMean = symm(fvc::grad(UMean));
5
6 // Calculate pSijMean from pGradUMean
7 volSymmTensorField pSijMean = symm(pGradUMean);
8
9 // Calculate pressure-strain
10 volSymmTensorField Phi_ij_RSTE
11 (
12     IOobject
13     (
14         Phi_ij_RSTE,
15         mesh().time().timeName(),
16         mesh(),
17         IOobject::NO_READ,
18         IOobject::AUTO_WRITE
19     ),
20     2.0 * (pSijMean - pMean * SijMean)
21 );
22 Phi_ij_RSTE.correctBoundaryConditions();

```

3.7 Dissipation term

The resolved dissipation term is given by

$$\varepsilon_{ij} = 2\nu \left\langle \frac{\partial \tilde{u}'_i}{\partial x_k} \frac{\partial \tilde{u}'_j}{\partial x_k} \right\rangle. \quad (28)$$

The double correlations are then expanded as

$$\left\langle \frac{\partial \tilde{u}'_i}{\partial x_k} \frac{\partial \tilde{u}'_j}{\partial x_k} \right\rangle = \left\langle \frac{\partial \tilde{u}_i}{\partial x_k} \frac{\partial \tilde{u}_j}{\partial x_k} \right\rangle - \frac{\partial \langle \tilde{u}_i \rangle}{\partial x_k} \frac{\partial \langle \tilde{u}_j \rangle}{\partial x_k}. \quad (29)$$

Thus, there is a need to introduce new fields for the velocity gradient products. This is done using *grad* to get the velocity gradients and *multiply* to construct the products. The averages are then calculated using *fieldAverage*. The implementation then primarily involves summing up all the scalar contributions. The OpenFOAM code is given below:

```

1  // ----- Calculate dissipation term ----- //
2
3  // Define volSymmTensorField
4  volSymmTensorField eps_ij
5  (
6      IOobject
7      (
8          outputeps_ij,
9          mesh().time().timeName(),
10         mesh(),
11         IOobject::NO_READ,
12         IOobject::AUTO_WRITE
13     ),
14     mesh(),
15     dimensionedSymmTensor
16     (
17         "zero",
18         dimensionSet(0, 2, -3, 0, 0, 0, 0),
19         symmTensor::zero
20     )
21 );
22
23 // Calculate dissipation
24 symmTensorField& eps_ij_Field = eps_ij.ref();
25 forAll(eps_ij_Field, i)
26 {
27
28     // Only eps_xx calculation for illustration
29     eps_ij_Field[i].xx() = ( 2.0 * nu.value() *
30                             ( (dUxxdUxx[i] + dUyxdUyx[i] + dUzxdUzx[i])
31                               - ( gradUxx[i] * gradUxx[i]
32                                   + gradUyx[i] * gradUyx[i]
33                                   + gradUzx[i] * gradUzx[i] )
34                             )
35     );
36
37     // Remaining components
38     // ... not included for brevity...
39 }
40 eps_ij.correctBoundaryConditions();

```

3.8 SGS term

The SGS term for an eddy viscosity model is given by

$$\Pi_{ij} = 2 \left(\left\langle \tilde{u}'_j \frac{\partial(\nu_t \tilde{S}_{ik})}{\partial k} \right\rangle + \left\langle \tilde{u}'_i \frac{\partial(\nu_t \tilde{S}_{jk})}{\partial k} \right\rangle \right). \quad (30)$$

Then, using $\tilde{u}'_i = \tilde{u}_i - \langle \tilde{u}_i \rangle$, the double-correlations are decomposed as follows

$$\Pi_{ij}^* = 2 \left(\left\langle \tilde{u}_j \frac{\partial(\nu_t \tilde{S}_{ik})}{\partial k} \right\rangle + \left\langle \tilde{u}_i \frac{\partial(\nu_t \tilde{S}_{jk})}{\partial k} \right\rangle \right) - 2 \left(\langle \tilde{u}_j \rangle \left\langle \frac{\partial(\nu_t \tilde{S}_{ik})}{\partial k} \right\rangle + \langle \tilde{u}_i \rangle \left\langle \frac{\partial(\nu_t \tilde{S}_{jk})}{\partial k} \right\rangle \right). \quad (31)$$

Thus, several new fields need to be introduced. Further, the divergences $\partial(\nu_t \tilde{S}_{ik})/\partial k$ cannot be calculated directly using built-in OpenFOAM functionality. Therefore, a custom coded function object is made for this purpose. In full,

$\partial(\nu_t \tilde{S}_{ik})/\partial_k$ is constructed by first calculating the velocity gradient tensor using *grad* and then multiplied with ν_t using *multiply*. The coded function object then symmetrizes this product using *symm* and calculates the divergence using *div*. Next, the products $\tilde{u}_j \partial(\nu_t \tilde{S}_{ik})/\partial_k$ are constructed using *multiply* on the velocity components and $\partial(\nu_t \tilde{S}_{ij})/\partial_k$. These fields, i.e., $\partial(\nu_t \tilde{S}_{ik})/\partial_k$ and $\tilde{u}_j \partial(\nu_t \tilde{S}_{ik})/\partial_k$, are then averaged using *fieldAverage*. The products $\langle \tilde{u}_j \rangle \langle \partial(\nu_t \tilde{S}_{ik})/\partial_k \rangle$ are also constructed using *multiply*. This gives all the components needed for calculating the SGS term. The OpenFOAM code putting all the components together is given below:

```

1 // ----- Calculate SGS term ----- //
2
3 // Constructing SGS as a volSymmTensorField
4 volSymmTensorField SGS_ij_RSTE
5 (
6     IOobject
7     (
8         sgsOutNm,
9         mesh().time().timeName(),
10        mesh(),
11        IOobject::NO_READ,
12        IOobject::AUTO_WRITE
13    ),
14    2.0 * symm(2.0 * divNutSijUMean) - 2.0 * symm(2.0 * divNutSijMean_UMean)
15 );
16 SGS_ij_RSTE.correctBoundaryConditions();

```

4 Results

The resolved RSTE budget implementation is tested on two different wall-bounded turbulent flows: channel and pipe. An overview of the simulation setups is given in 4.1. This is followed by the channel flow results in 4.2 and the pipe flow results in 4.3. Since second-order-accurate numerical simulations following the finite volume method are performed in this study, it is not expected that the results fully match the DNS reference data. The reader is reminded that the aim of this study is to show, provide, and validate a seamless and correct calculation of the resolved RSTE budget with OpenFOAM, not to match spectral-high-order data from DNS results. For brevity and simplicity of the presentation of the results, the "resolved" before, e.g., RSTE, is dropped below together with the $\tilde{\cdot}$ notation for filtered variables.

4.1 Simulation details

An overview of the solvers is provided, as well as numerical schemes, SGS modeling, and meshing used in the simulations. Note that the numerical schemes, SGS modeling, and meshes take inspiration from previous efforts in OpenFOAM-based LES [23–26].

The *pimpleFoam* solver based on the PIMPLE algorithm is used with a single outer correction (PISO-like mode), two inner correctors for the pressure-velocity coupling, one non-orthogonal corrector per iteration, and the momentum predictor is enabled. Time derivatives are discretised using the second-order *backward* scheme, and adaptive time stepping is used to ensure a Courant-Friedrichs-Lewy (CFL) number below 0.5. Spatial gradients and divergence terms are computed with the *Gauss linear* method. Viscous terms also use the *Gauss linear* scheme with non-orthogonal correction. Linear interpolation is used for cell-face values, and surface-normal gradients are corrected for mesh non-orthogonality. To drive the flow, a momentum source is added using *meanVelocityForce* to enforce a bulk velocity of $U_b = 1$. The forcing is time-dependent but uniform in space. To ensure $Re_\tau = 180$ in the simulations, the corresponding bulk Reynolds numbers must be matched. These are $U_b = 2857$ for channel (from [20]) and $U_b = 5300$ for pipe (from [19]), which gives the resulting kinematic viscosities as $\nu = 3.5 \times 10^{-4}$ and $\nu = 1.9 \times 10^{-4}$, respectively. The momentum equations are solved using a preconditioned biconjugate gradient stabilised (PBiCGStab) solver with a diagonal incomplete LU (DILU) precondition. Furthermore, the linear system for the pressure is solved using a preconditioned conjugate gradient (PCG) solver with a geometric-algebraic multigrid (GAMG) preconditioner and a diagonal incomplete-Cholesky Gauss-Seidel (DICGaussSeidel) smoother with two post-sweeps. Finally, tight absolute numerical tolerances are applied to the final iterations for both velocity and pressure.

For SGS modeling, the Wall-Adapting Local Eddy-viscosity (WALE) model [27] is used. This SGS model expresses the turbulent eddy-viscosity using the square of the velocity gradient tensor. Denoting the filtered velocity gradient tensor as $\tilde{g}_{ij} = \partial \tilde{u}_j / \partial x_i$, the traceless symmetric part of the square of the velocity gradient tensor is

$$\tilde{\epsilon}_{ij} = \frac{1}{2} (\tilde{g}_{ij}^2 + \tilde{g}_{ji}^2) - \frac{1}{3} \tilde{g}_{kk}^2, \quad (32)$$

where $\tilde{g}_{ij}^2 = \tilde{g}_{ik}\tilde{g}_{kj}$. The turbulent eddy-viscosity for the WALE model can then be written as

$$\nu_t^{\text{WALE}} = (C_w \Delta)^2 \frac{(\tilde{\xi}_{ij}\tilde{\xi}_{ij})^{3/2}}{(\tilde{S}_{ij}\tilde{S}_{ij})^{5/2} + (\tilde{\xi}_{ij}\tilde{\xi}_{ij})^{5/4}}, \quad (33)$$

where C_w is a constant, i.e., it is not tuned dynamically.

For both channel and pipe, structured hexahedral meshes are used. In both cases, the meshes are close to isotropic in the core of the flow and are then stretched towards the walls to provide sufficient near-wall resolution. An illustration of the meshes is shown in Fig. 1.

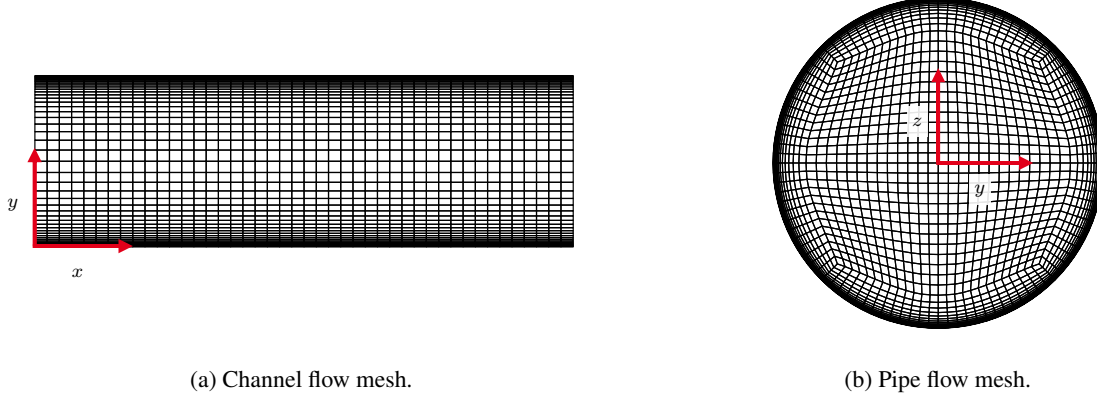


Figure 1: Mesh topology for channel (Fig. 1a) and pipe flow (Fig. 1b). The number of cells has been reduced in this figure for visualisation purposes.

To assess the resolution requirements where the RSTE from LES start to faithfully reproduce the full RSTE from DNS, a mesh refinement campaign for both channel and pipe has been performed. A summary of the meshes used for the different cases is given in Table 1.

Channel cases	N_{tot}	Centre			Wall		
		Δx_c^+	Δy_c^+	Δz_c^+	Δx_w^+	Δy_w^+	Δz_w^+
L_1	131 072	17.7	17.4	17.7	17.7	0.87	17.7
L_2	389 344	12.3	13.7	12.3	12.3	0.43	12.3
L_3	1 048 576	8.8	10.5	8.8	8.8	0.26	8.8
L_4	2 916 000	6.3	7.9	6.3	6.3	0.16	6.3

Pipe cases	N_{tot}	Centre			Wall		
		Δz_c^+	Δr_c^+	$R\Delta\theta_c^+$	Δz_w^+	Δr_w^+	$R\Delta\theta_w^+$
L_1	536 640	15	3.9	3.1	15	0.07	2.7
L_2	1 173 120	7.5	3.9	3.1	7.5	0.04	2.7
L_3	3 400 800	2.7	3.9	3.1	2.7	0.02	2.7

Table 1: Mesh resolution summary for channel and pipe flow simulations. The table shows the total number of grid points (N_{tot}) and the dimensionless cell sizes in wall units (Δ^+) for both centre and wall regions. Centre cell sizes represent the mesh resolution in the bulk flow region, while wall cell sizes correspond to the near-wall mesh refinement. The wall-normal direction (y for channel, r for pipe) shows the finest resolution near the walls.

4.2 Channel Flow

To ensure a correct implementation of the RSTE budget in LES, results for different meshes are compared against DNS data from [20] of channel flow at $\text{Re}_\tau = 180$. This section displays 1D-mean values obtained by performing a

stream-span-wise spatial average of the first and second order statistics of the velocity— $\langle u \rangle$, k , R_{ij} —as well as the RSTE budget terms for the 4 main components of the tensor—11, 22, 33, 12. Here, 1, 2, and 3 refer to the streamwise, wall-normal, and spanwise directions, respectively. All values are shown in wall units.

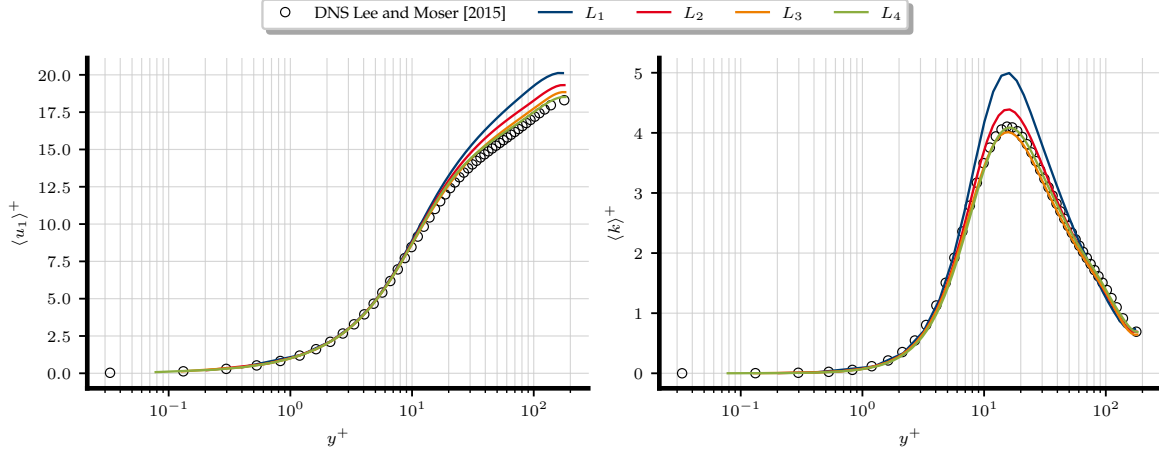


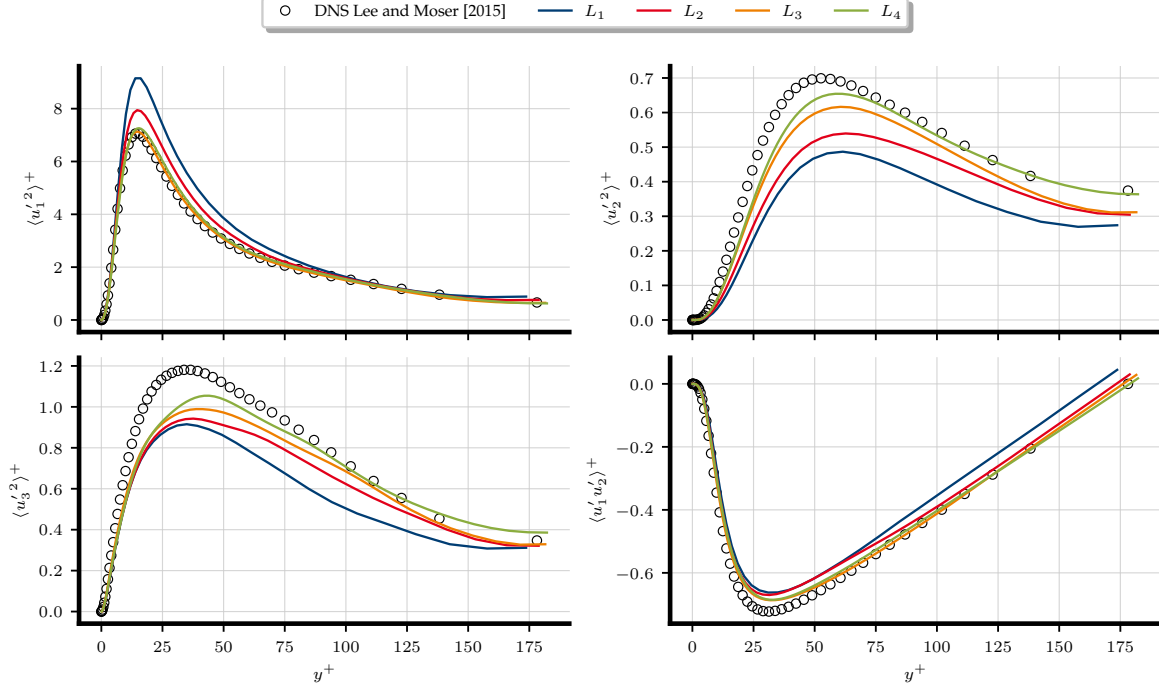
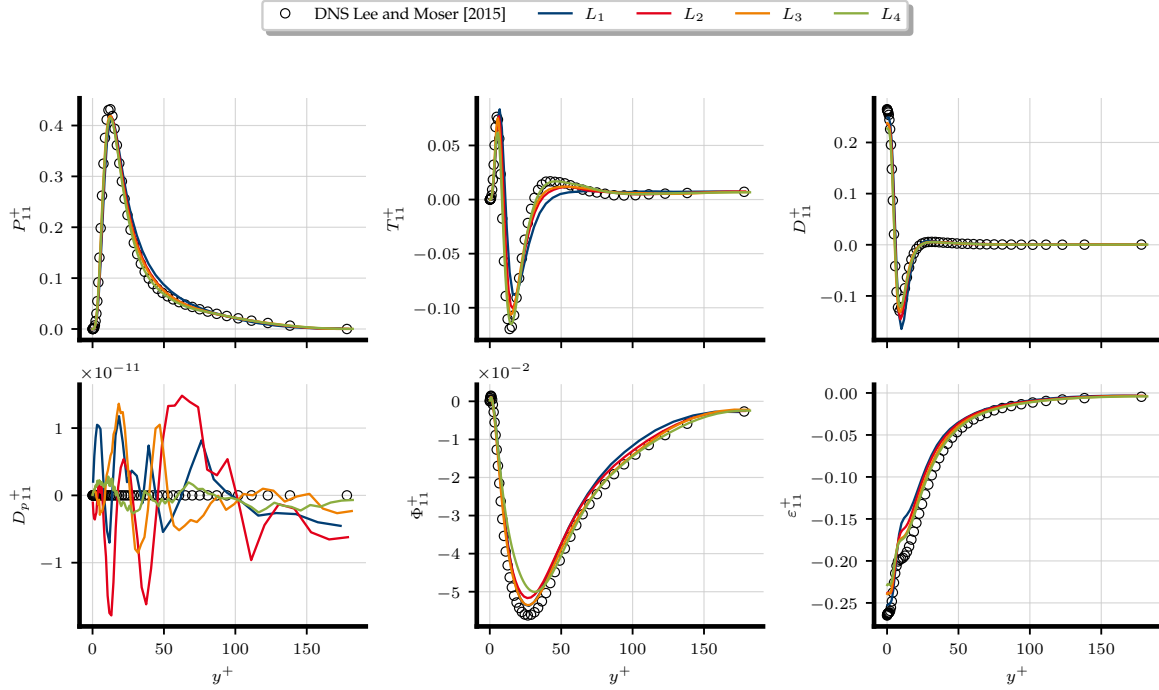
Figure 2: Velocity $\langle u \rangle$ and TKE k profiles.

As seen in Fig. 2, the turbulent flow profile of the velocity and TKE are predicted with increased accuracy by the finer meshes. Both in magnitude and gradients, meshes with fineness above L_3 predict the velocity and TKE with marginal deviations from DNS data and display physical consistency. The mean velocity profile exhibits the characteristic logarithmic region in the range $y^+ \in [30, 100]$, where the velocity follows the law of the wall. The TKE profile shows a known maximum in the buffer region around $y^+ \approx 15$; this maximum arises from the intense shear production of turbulence, which is highest in this region, coupled with the onset of viscous dissipation, which becomes dominant closer to the wall. In terms of TKE prediction, the finer meshes capture the near-wall behaviour more accurately, particularly in the buffer layer ($5 < y^+ < 30$), where gradients are steepest.

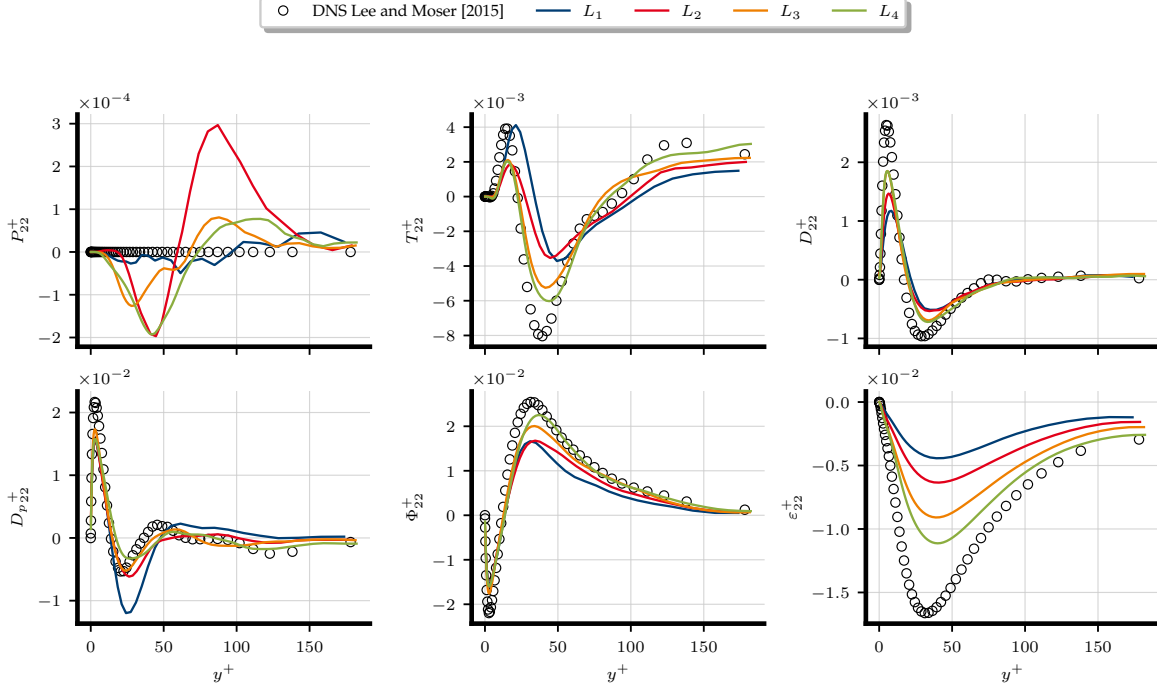
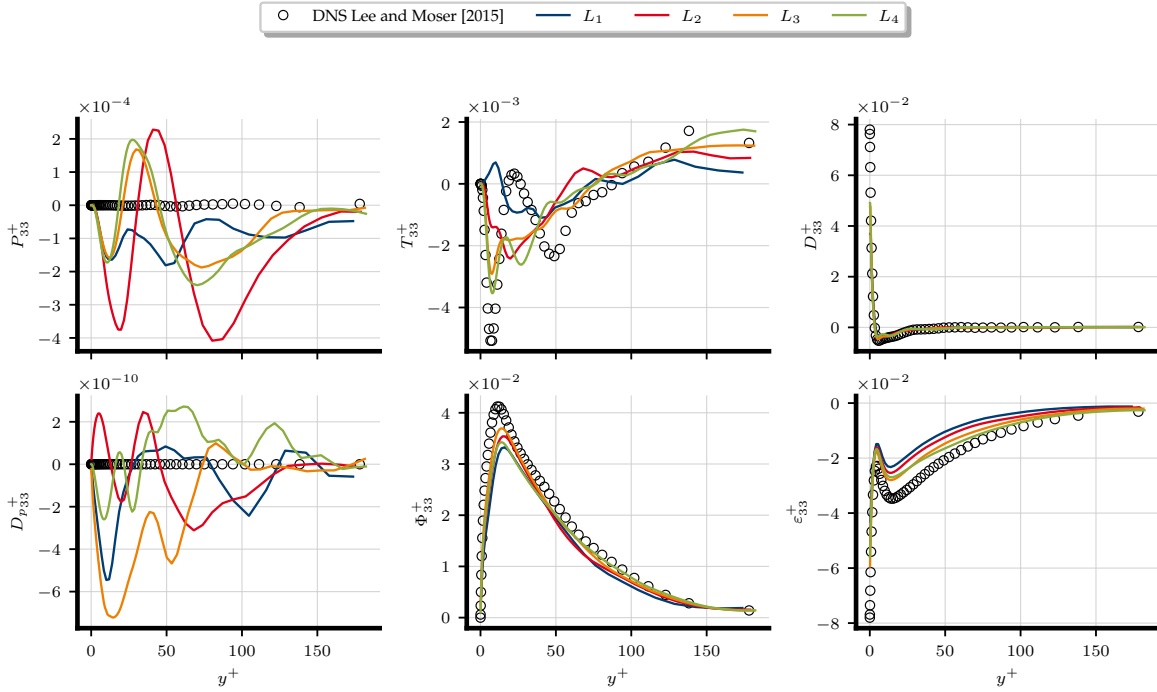
Similarly, in Fig. 3, the 11, 22, 33, and 12 components of the R_{ij} are predicted with increased accuracy by the finer meshes. Gradients are predicted following DNS data, although the magnitudes of the 22 and 33 components do not fully match DNS results at regions $y^+ \in [25 - 75]$. It should be noted that some discrepancies are expected in turbulence simulations when comparing results from solvers with different numerics [28]. Nonetheless, these uncertainties associated with numerics do not fully account for observed differences. Instead, the major contributor to the observed difference is likely the dissipative numerics in OpenFOAM, which is discussed in further detail in [29]. The R_{11} component (streamwise normal stress) shows the largest magnitude, peaking in the buffer region due to the strong mean shear production. The R_{22} and R_{33} components (wall-normal and spanwise normal stresses) are smaller in magnitude, reflecting the anisotropic nature of wall-bounded turbulence. The R_{12} component (Reynolds shear stress) is crucial for momentum transport and shows a maximum around $y^+ \approx 30$, where the production term balances the pressure-strain correlation.

Figure 4 presents the RSTE budget for the streamwise component R_{11} . The budget terms include production P_{11}^+ , turbulent transport T_{11}^+ , viscous diffusion D_{11}^+ , pressure-strain correlation Φ_{11} , pressure-diffusion $D_{p,11}^+$, and viscous dissipation ε_{11}^+ . The production term is the dominant source, converting mean flow energy into turbulent fluctuations. This is balanced primarily by the pressure-strain correlation, which acts as a sink term here by redistributing energy from the streamwise direction to the wall-normal and spanwise components, and by the dissipation term. The turbulent transport and pressure diffusion terms are responsible for spatially redistributing the energy away from the maximum production region near the wall. The simulation predictions for all terms follow the previously seen trends — results on finer meshes more accurately follow DNS data, and all results show accurate gradients and magnitudes. In the DNS reference data, the pressure-diffusion term is given as zero; however, the simulation data shown is displayed with the remaining numerical fluctuations on the order of 10^{-11} , which is close to the numerical precision used.

The budget for the wall-normal component R_{22} is shown in Fig. 5. It is important to highlight that R_{22} has no direct production term since the mean wall-normal velocity is zero. Its energy is supplied entirely by the pressure-strain correlation Φ_{22} , which acts as a source term by redirecting the excess energy from the streamwise component. A significant portion of this term is explained by the *wall-reflection* effect, which damps wall-normal fluctuations and enhances in-plane fluctuations. This energy gain is then balanced by dissipation and spatial transport. Once again, finer meshes more accurately predict DNS data and DNS trends are followed in all predictions. The greatest discrepancy in


 Figure 3: Profiles of R_{ij} , specifically components 11, 22, 33, and 12.

 Figure 4: RSTE budget for the streamwise normal stress component $\langle u_1' u_1' \rangle$.

the data is shown at ε_{22}^+ ; nonetheless the data shows that further mesh refinement tends towards the DNS reference. For P_{22}^+ , the numerical fluctuations are on the order of 10^{-4} , which are considered acceptable.


 Figure 5: RSTE budget for the wall-normal stress component $\langle u'_2 u'_2 \rangle$.

 Figure 6: RSTE budget for the spanwise normal stress component $\langle u'_3 u'_3 \rangle$.

The spanwise component budget in Fig. 6 displays the transport equation for R_{33} . Similar to the wall-normal component, it has no direct production and relies on energy redistribution via the pressure-strain correlation Φ_{33} . The

spanwise component typically shows the smallest magnitude among the normal stresses, reflecting the quasi-two-dimensional nature of the large-scale structures in wall-bounded turbulence. The pressure-strain correlation acts to maintain a degree of isotropy by feeding the spanwise component, although the overall anisotropy is preserved due to the wall constraint. The other budget terms, such as dissipation ε_{33}^+ , act as sinks and balance the budget.

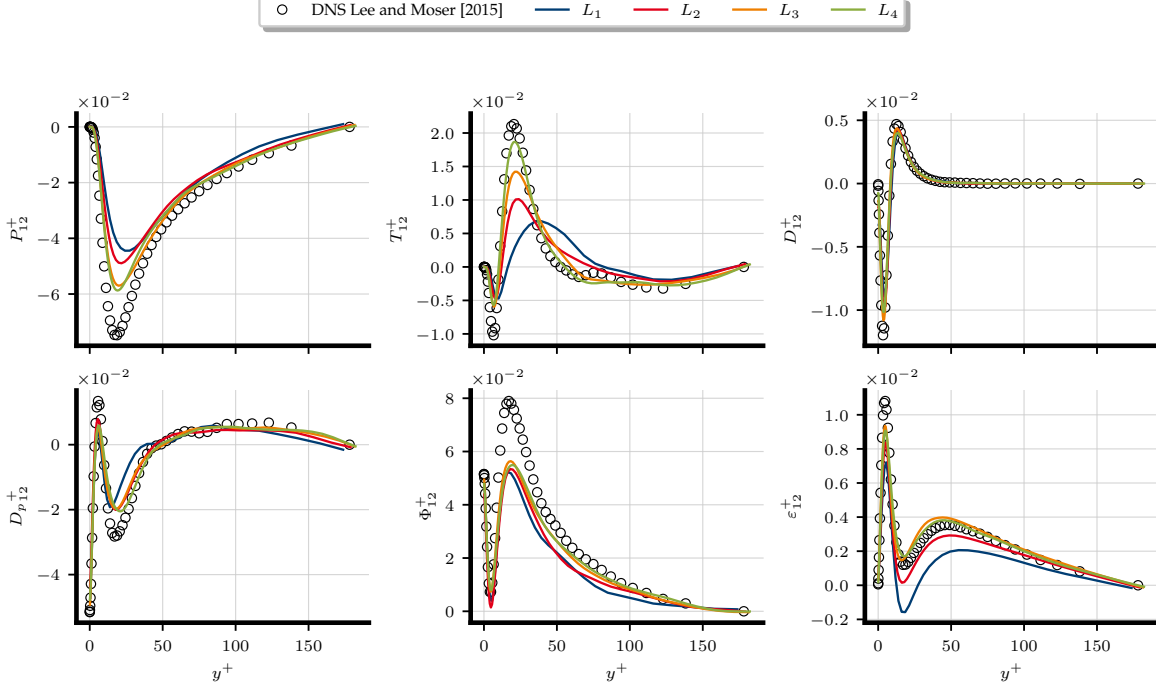


Figure 7: RSTE budget for shear stress component $\langle u_1' u_2' \rangle$.

Figure 7 shows the Reynolds shear stress budget for R_{12} , which is fundamental to understanding momentum transport. The production term P_{12}^+ arises from the interaction between the mean velocity gradient and the wall-normal Reynolds stress. This production is primarily balanced by the pressure-strain correlation Φ_{12} , which acts to destroy the shear stress. The balance between these two terms is what eddy-viscosity models attempt to approximate with an algebraic relation. The finer meshes capture this balance more accurately, particularly in the near-wall region where viscous diffusion becomes important. As seen in previous results, the 12 term of the RSTE burger follows similar trends where finer meshes are more accurate and simulation results predict DNS data with marginal accuracy.

The turbulent kinetic energy budget in Fig. 8 provides the overall energy balance for the turbulent fluctuations. The production term P_k^+ represents the energy extracted from the mean flow and is balanced by dissipation ε^+ , which converts turbulent energy to heat. The remaining terms represent the spatial redistribution of the energy. In the log-law region, production and dissipation are in approximate equilibrium, a key assumption in many turbulence models. Near the wall, transport by viscous diffusion becomes a critical source of energy, while dissipation remains the primary sink. The finer meshes capture this balance more accurately, particularly in the near-wall region where the gradients are steepest. Similarly, the pressure-strain terms are observed to converge towards zero as the mesh is refined.

To end this section, the SGS contributions to each of the RSTE budgets are also considered. These are shown in Fig. 9. As expected, the SGS contributions converge towards zero as the mesh is refined, highlighting the diminishing contribution from unresolved fluctuations. Additionally, the SGS contributions are seen to be small compared with the remaining terms in the RSTE budgets, which is especially evident on the finer meshes.

4.3 Pipe Flow

To further illustrate the correct implementation of the RSTE budget, another canonical flow case is simulated in the same fashion as the previous channel flow results for different meshes. The results are then compared against DNS reference data from [19] of pipe flow at $Re_\tau = 180$. Therefore, this section displays 1D-mean values of the first and second order statistics of the velocity— $\langle u \rangle$, k , R_{ij} —as well as the TKE budget, which is the trace of the RSTE budget. As the trace is a scalar, it is an invariant and issues related to a change in coordinate-change are avoided. All values are shown in wall units.

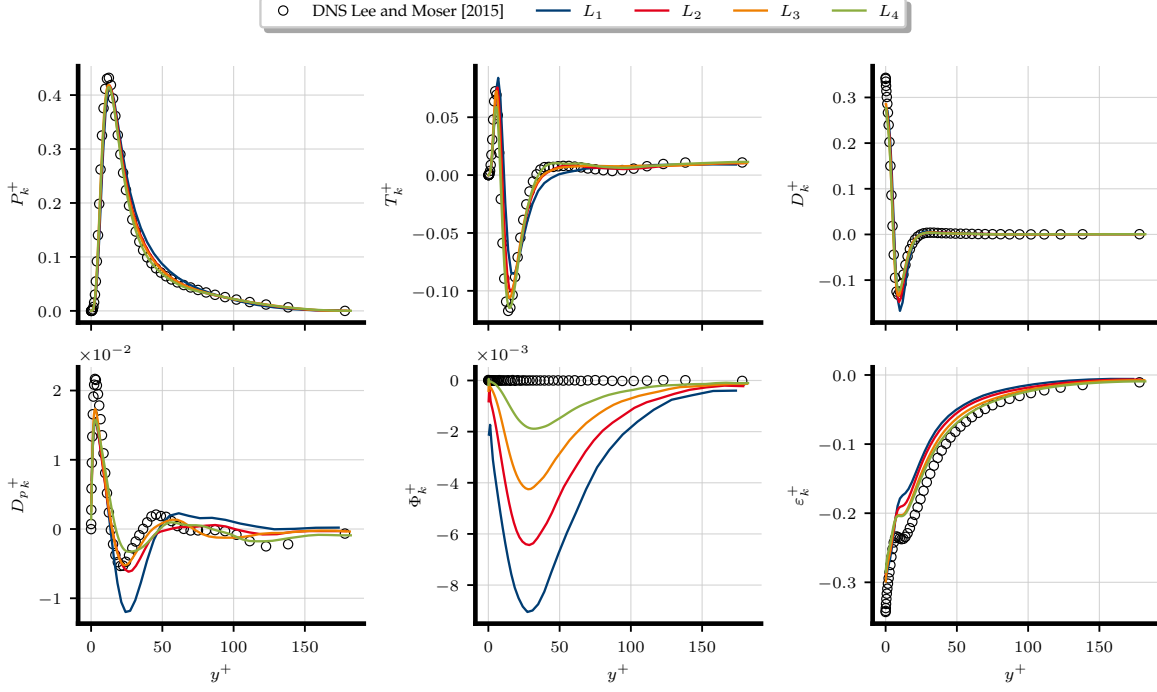


Figure 8: Comparison for the TKE budget.

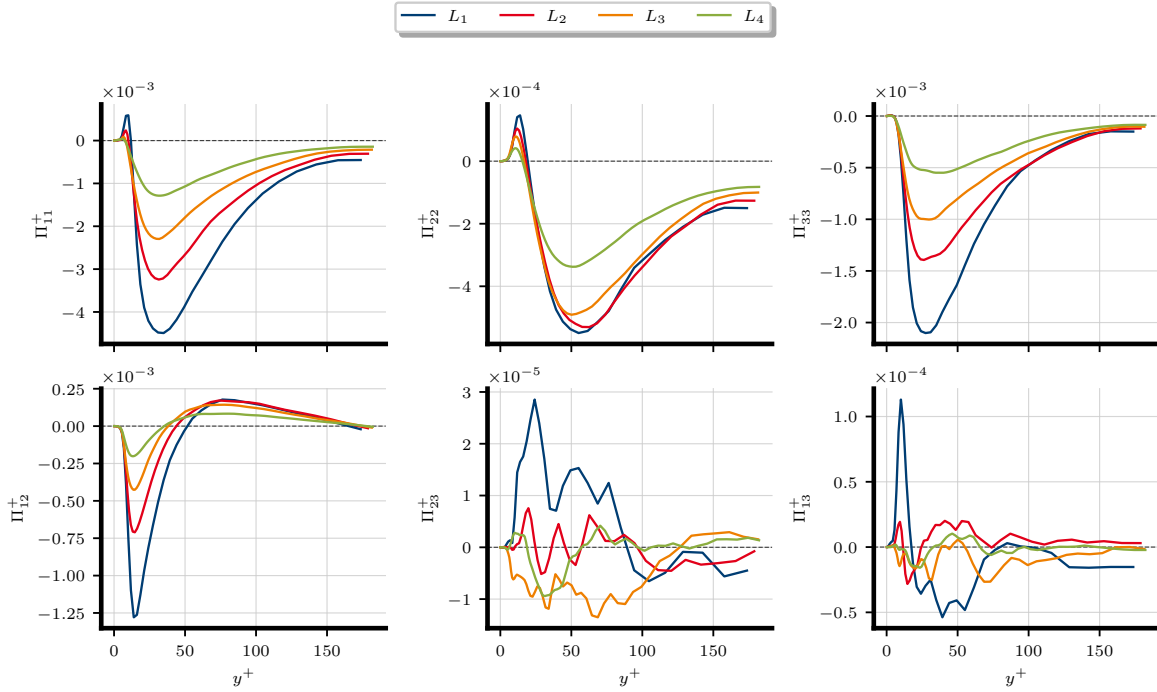
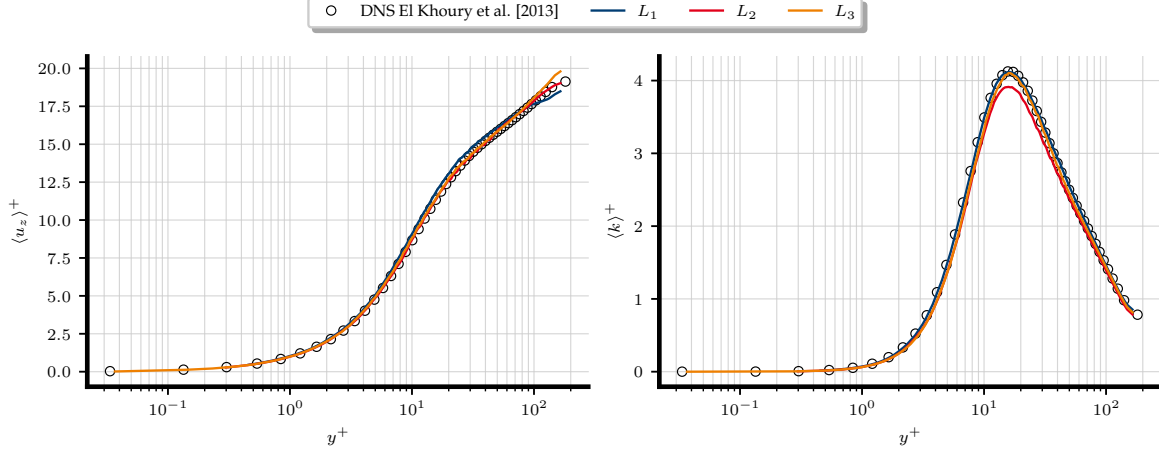
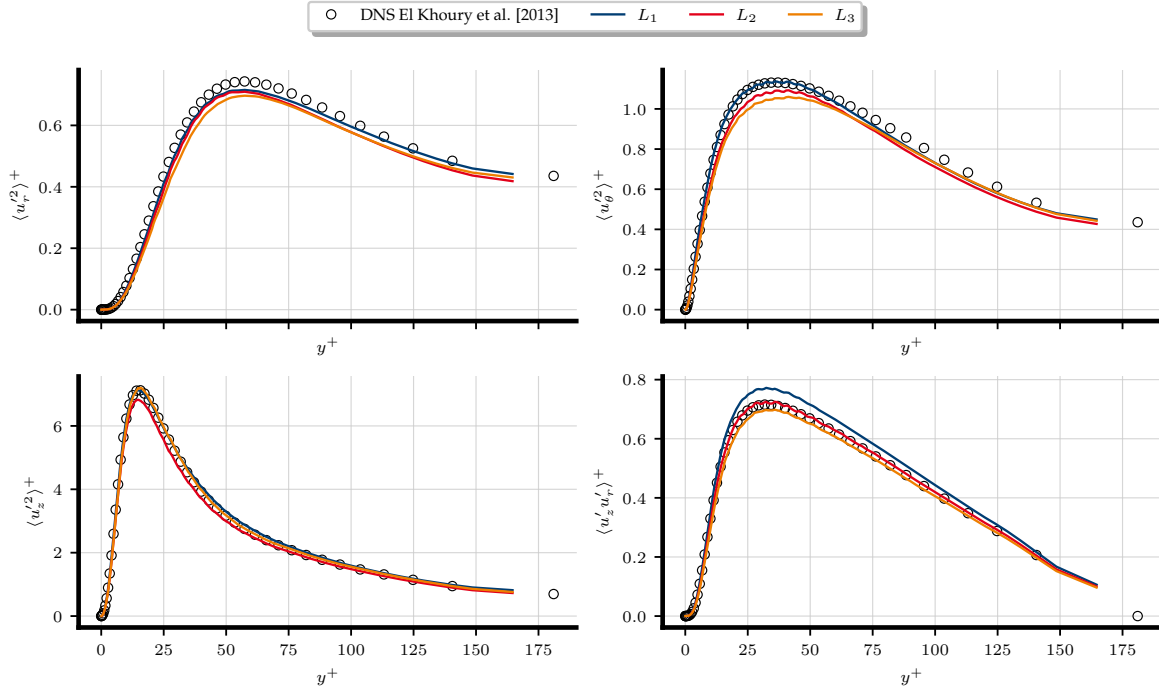

 Figure 9: The SGS term Π_{ij} from each of the RSTE budgets.

Figure 10 presents the velocity and TKE profiles for pipe flow, which can be directly compared with the channel flow results. The pipe flow exhibits similar characteristics to channel flow, with the mean velocity following the law of


 Figure 10: Streamwise velocity $\langle u_z \rangle$ and TKE k profiles.

the wall in the logarithmic region. However, the pipe geometry introduces subtle differences mainly due to the curvature of the pipe walls, which is not present in the channel case. The TKE profile shows a maximum in the buffer region, similar to channel flow, but this maximum location may differ slightly due to the curvature of the conduit. The finer meshes capture both the mean velocity and TKE profiles with improved accuracy, particularly in the near-wall region.


 Figure 11: Profiles of R_{ij} , specifically components zz , rr , $\theta\theta$, and zr .

The Reynolds stress components in pipe flow, shown in Fig. 11, demonstrate the same general trends as channel flow, with the streamwise component R_{zz} being the largest and the circumferential component $R_{\theta\theta}$ being the smallest. The wall-normal radial component R_{rr} and the radial shear stress component R_{zr} show similar behaviour to channel flow, though the coordinate system differences (cylindrical vs. Cartesian) may introduce subtle variations in the interpretation of these components. The finer meshes provide improved prediction of the Reynolds stress profiles, particularly in the buffer region where the stresses peak and the gradients are significant.

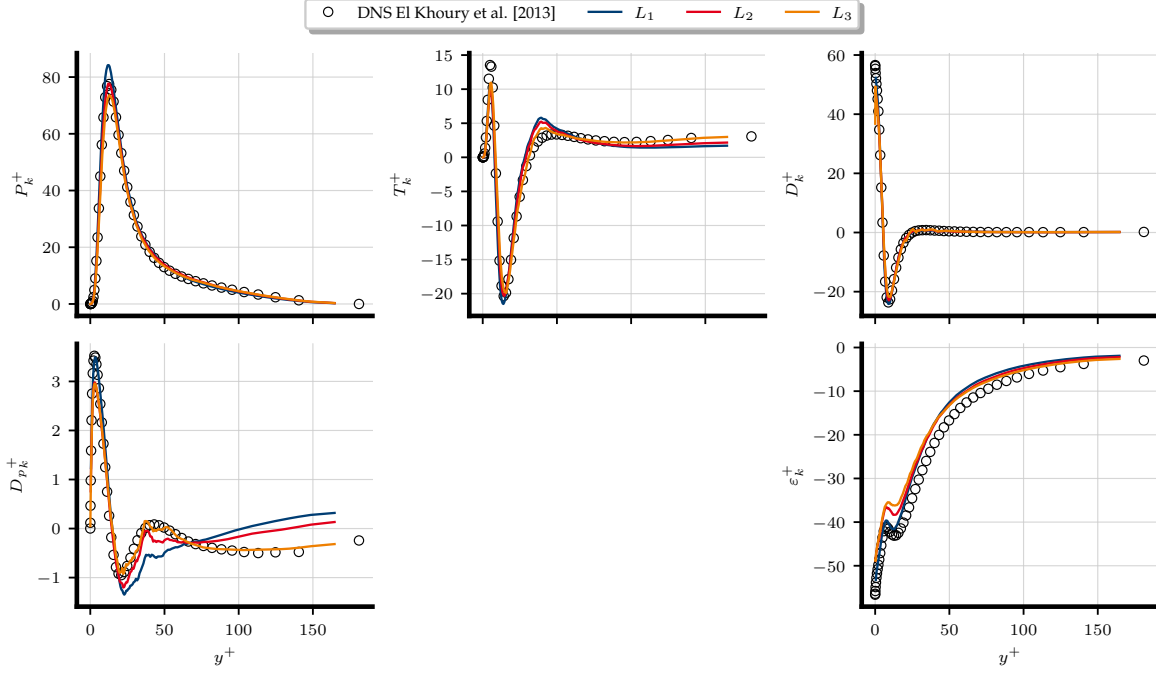


Figure 12: TKE budget of the pipe-flow simulation.

Figure 12 displays the trace of the RSTE budget components, which is equivalent to the TKE budget. As the trace of a tensor is an invariant, this quantity is independent of the coordinate system used. This makes it an excellent tool for validation, as it confirms that the fundamental energy balances of the implementation are correct, irrespective of the geometric complexities introduced by the pipe’s cylindrical coordinate system. The TKE budget shows the overall energy balance for the turbulent fluctuations, with production balancing dissipation in the log-law region and viscous effects dominating near the wall. The remaining terms act to redistribute the energy in space. As seen in previous cases, the finer meshes capture this balance more accurately, demonstrating the correct implementation.

5 Conclusion

In this study, a comprehensive methodology for calculating the complete resolved RSTE budget from LES within the OpenFOAM framework has been successfully implemented and validated. The implementation relies on standard OpenFOAM utilities and post-processing functions to compute all terms of the resolved RSTE budget in Cartesian coordinates, including convection, production, turbulent transport, pressure-diffusion, pressure-strain, viscous diffusion, and, finally, the SGS contribution.

The fidelity of the implementation was rigorously assessed by comparing results against DNS data for two canonical wall-bounded turbulent flows: channel flow and pipe flow, both at a friction Reynolds number of $Re_\tau = 180$. For the channel flow case, the predicted mean velocity, TKE, and Reynolds stress profiles demonstrated reasonable agreement with DNS data. A mesh refinement study confirmed that the LES results systematically converge towards the reference data. Furthermore, the individual budgets for the main components of the Reynolds stress tensor (R_{11} , R_{22} , R_{33} , and R_{12}) were also considered, and it was found that the implementation correctly reproduces the complex interplay of energy production, dissipation, and redistribution mechanisms that characterise near-wall turbulence. While minor discrepancies with DNS data were observed, these are consistent with the inherent limitations of the second-order-accurate finite volume numerics employed. The primary objective was the validation of the resolved RSTE budget library, not the replication of high-order spectral DNS results. The pipe flow simulation further corroborates the correct implementation of the RSTE budget. By analysing the TKE budget—which is coordinate-invariant—it has been confirmed that the underlying energy balances are correctly computed, irrespective of the geometry. This result highlights the robustness of the developed RSTE budget tool.

In conclusion, this work provides the scientific and engineering community with a validated, open-source tool for conducting detailed RSTE budget analysis in OpenFOAM. This utility enables deeper insights into the physics of turbulence and serves as a valuable resource for the development and validation of advanced turbulence models. Future

efforts could be directed towards optimising the memory footprint of the implementation and extending its capabilities to directly handle cylindrical and/or spherical coordinate systems, thereby broadening its applicability to more complex geometries.

Acknowledgements

M.J.R. and M.R. acknowledge financial support from Innovation Fund Denmark (IFD) under Grant No. 3130-00007B and Kamstrup A/S. C.H. and M.A. acknowledge funding from VILLUM FONDEN under Grant No. VIL57365. The authors would also like to acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LUMI, hosted by CSC (Finland) and the LUMI consortium through a EuroHPC Regular Access call. This work was also partially supported by the Danish e-Infrastructure Cooperation (DeiC) National HPC under grant number DeiC-AU-N5-2025130.

Author Contributions: Conceptualisation, M.J.R. and C.H.; methodology, M.J.R. and C.H.; software, M.J.R. and C.H.; validation, M.J.R. and C.H.; formal analysis, M.J.R., C.H., M.R. and M.A.; investigation, M.J.R. and C.H.; resources, M.R. and M.A.; data curation, M.J.R. and C.H.; writing—original draft preparation, M.J.R. and C.H.; writing—review and editing, M.R. and M.A.; visualisation, M.J.R. and C.H.; supervision, M.R. and M.A.; project administration, M.J.R., C.H., M.R. and M.A.; funding acquisition, M.R. and M.A. All authors have read and agreed to the published version of the manuscript.

References

- [1] Marcel Lesieur. *Turbulence in fluids: stochastic and numerical modelling*. M. Nijhoff Boston, 1987.
- [2] Stephen B Pope. *Turbulent flows*. Cambridge University Press, (2000).
- [3] Parviz Moin and Krishnan Mahesh. Direct numerical simulation: a tool in turbulence research. *Annual review of fluid mechanics*, 30(1):539–578, 1998.
- [4] Pierre Sagaut. *Large eddy simulation for incompressible flows: an introduction*. Springer Science & Business Media, 2005.
- [5] J.P. Slotnick, A. Khodadoust, A. Juan, D. Darmofal, W. Gropp, E. Lurie, and D.J. Mavriplis. CFD vision 2030 study: A path to revolutionary computational aerosciences. Technical Report NASA/CR-2014-218178, NASA, 2014.
- [6] Florian Menter, Andreas Hüppe, Alexey Matyushenko, and Dmitry Kolmogorov. An overview of hybrid rans–les models developed for industrial cfd. *Applied Sciences*, 11(6):2459, 2021.
- [7] Joseph Boussinesq. *Essai sur la théorie des eaux courantes*. Impr. nationale, 1877.
- [8] Charles G Speziale. Analytical methods for the development of reynolds stress closures in turbulence. Technical report, NASA, 1990.
- [9] TB Gatski. Constitutive equations for turbulent flows. *Theoretical and Computational Fluid Dynamics*, 18:345–369, 2004.
- [10] Brian Edward Launder, G Jr Reece, and W Rodi. Progress in the development of a reynolds-stress turbulence closure. *Journal of fluid mechanics*, 68(3):537–566, 1975.
- [11] Karthik Duraisamy and Vishal Srivastava. Chapter 8 - Machine learning augmented modeling of turbulence. In Karthik Duraisamy, editor, *Data Driven Analysis and Modeling of Turbulent Flows*, Computation and Analysis of Turbulent Flows, pages 311–354. Academic Press, 2025.
- [12] Brian Edward Launder. Second-moment closure and its use in modelling turbulent industrial flows. *International journal for numerical methods in fluids*, 9(8):963–985, 1989.
- [13] Mario Javier Rincón, Ali Amarloo, Martino Reclari, Xiang IA Yang, and Mahdi Abkar. Progressive augmentation of Reynolds stress tensor models for secondary flow prediction by computational fluid dynamics driven surrogate optimisation. *International Journal of Heat and Fluid Flow*, 104:109242, 2023.
- [14] Ali Amarloo, Mario Javier Rincón, Martino Reclari, and Mahdi Abkar. Progressive augmentation of turbulence models for flow separation by multi-case computational fluid dynamics driven surrogate optimization. *Physics of Fluids*, 35(12), 2023.
- [15] Christoffer Hansen, Xiang IA Yang, and Mahdi Abkar. A pod-mode-augmented wall model and its applications to flows at non-equilibrium conditions. *Journal of Fluid Mechanics*, 975:A24, 2023.
- [16] Christoffer Hansen, Jens N Sørensen, Xiang IA Yang, and Mahdi Abkar. Extension of the law of the wall exploiting weak similarity of velocity fluctuations in turbulent channels. *Physics of Fluids*, 36(1), 2024.

- [17] Mario Javier Rincón, Martino Reclari, Xiang IA Yang, and Mahdi Abkar. A generalisable data-augmented turbulence model with progressive and interpretable corrections for incompressible wall-bounded flows. *International Journal of Heat and Fluid Flow*, 116:109970, 2025.
- [18] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object orientated techniques. *Computers in Physics*, 12:620–631, 1998.
- [19] George K El Khoury, Philipp Schlatter, Azad Noorani, Paul F Fischer, Geert Brethouwer, and Arne V Johansson. Direct numerical simulation of turbulent pipe flow at moderately high reynolds numbers. *Flow, turbulence and combustion*, 91(3):475–495, 2013.
- [20] Myoungkyu Lee and Robert D Moser. Direct numerical simulation of turbulent channel flow up to. *Journal of Fluid Mechanics*, 774:395–415, 2015.
- [21] Charles Meneveau. Statistics of turbulence subgrid-scale stresses: Necessary conditions and experimental tests. *Physics of Fluids*, 6(2):815–833, 1994.
- [22] R. D. Moser and P. Moin. Direct numerical simulation of curved turbulent channel flow. Technical Report NASA-TM-85974, NASA Ames Research Center, 1984.
- [23] Racheet Matai and Paul Durbin. Large-eddy simulation of turbulent flow over a parametric set of bumps. *Journal of Fluid Mechanics*, 866:503–525, 2019.
- [24] Timofey Mukha, Saleh Rezaeiravesh, and Mattias Liefvendahl. A library for wall-modelled large-eddy simulation based on openfoam technology. *Computer Physics Communications*, 239:204–224, 2019.
- [25] Sina Nozarian, Christoffer Hansen, Pourya Forooghi, and Mahdi Abkar. Laminarization through annular gap injection in turbulent pipe flows: A large eddy simulation study. *Physics of Fluids*, 37(5), 2025.
- [26] Christoffer Hansen, Xiang IA Yang, and Mahdi Abkar. Wall-modeled large eddy simulation of turbulent smooth body separation using the openfoam flow solver. *Journal of Fluids Engineering*, 148(1):011502, 2026.
- [27] Frederic Ducros, F Nicoud, and Thierry Poinsot. Wall-adapting local eddy-viscosity models for simulations in complex geometries. *Numerical Methods for Fluid Dynamics VI*, 6:293–299, 1998.
- [28] Peng ES Chen, Xiaowei Zhu, Yipeng Shi, and Xiang IA Yang. Quantifying uncertainties in direct-numerical-simulation statistics due to wall-normal numerics and grids. *Physical Review Fluids*, 8(7):074602, 2023.
- [29] Matteo Montecchia, Geert Brethouwer, Stefan Wallin, Arne V Johansson, and Thilo Knacke. Improving les with openfoam by minimising numerical dissipation and use of explicit algebraic sgs stress model. *Journal of Turbulence*, 20(11-12):697–722, 2019.