



SAPIENZA
UNIVERSITÀ DI ROMA

Reinforcement learning for wall modeling in large-eddy simulation

Scuola di Dottorato in Ingegneria Aeronautica e Spaziale
Dottorato di Ricerca in Ingegneria Aeronautica e Spaziale (XXXVIII cycle)

Maochao Xiao

ID number 2068246

Advisor
Prof. Sergio Pirozzoli

Co-Advisor
Prof. Johan Larsson

Academic Year 2024/2025

Thesis defended on 29-01-2026
in front of a Board of Examiners composed by:

Prof. Emanuele De Angelis

Prof. Maurizio Quadrio

Prof. Pierluigi Di Lizia

Reinforcement learning for wall modeling in large-eddy simulation

PhD thesis. Sapienza University of Rome

© 2026 Maochao Xiao. The present document is distributed under the Creative Commons
CC BY license, attribution.

This thesis has been typeset by L^AT_EX and the Sapthesis class.

Author's email: maochao.xiao@uniroma1.it

"To my wife."

Acknowledgments

I am deeply grateful to Prof. Sergio Pirozzoli, my advisor, for introducing me to academic research and for his steadfast support throughout this journey. Over the past three years, his mentorship has been invaluable. Our discussions have been a constant source of insight. I especially thank him for supporting collaborations with researchers from diverse backgrounds, including high-performance computing and machine learning. I hold the deepest respect for his contributions to turbulence research, and he remains a role model for researchers in the turbulence community. His enthusiasm toward wall-bounded turbulence and its computation will drive me to do my best in my future career. I am equally indebted to Prof. Johan Larsson, my co-advisor, whose guidance and inspiring discussions on wall modeling for large-eddy simulation have profoundly shaped this work. He consistently impressed upon me that turbulence modeling earns its true value through engineering impact rather than publications alone, and his reminders kept me from drifting in the wrong direction.

I am also grateful to Prof. Ricardo Vinuesa for hosting my research stay at KTH and for the productive collaboration with his group. I thank Prof. Paolo Orlandi for his engaging discussions on turbulence. I extend my gratitude to Prof. Jane H. Bae and Dr. Di Zhou for their insightful conversations, to Prof. Pedro Costa for his assistance during the development of the GPU-accelerated CaLES solver, to Prof. Bernat Font, Prof. Andrea Beck, and Dr. Marius Kurz for their support in advancing the SmartFlow framework, and to Prof. Yufei Zhang for always being willing to support me in both academic and personal matters. I also wish to express special thanks to Jean Rabault, Oriol Lehmkuhl, Yuning Wang, Felix Rodach, Pol Suárez, Francisco Alcántara-Ávila, Junle Liu, Ricard Montalà, Jiawei Chen, and Ziyu Zhou for their generous collaboration and insightful discussions that have enriched this work. My sincere thanks go to Alessandro, Mariangela, Alessio, Daniele F., Daniele R., Giulio, Federico, and Gianluca for the camaraderie and laughter that brightened our office. You turned our professional relationship into friendship and made each day more rewarding.

I owe a special debt of gratitude to my family for their unwavering encouragement, which has been instrumental in reaching this milestone. I am especially grateful to my wife, who chose to work in Europe to be with me and frequently came to visit me in Rome during this journey, making countless sacrifices to support my academic pursuits. I also thank my sister and brother for accompanying my parents while I am abroad, allowing me to focus on my research with peace of mind.

I gratefully acknowledge the financial support provided by the China Scholarship Council. The simulations were performed using the EuroHPC Research Infrastructure resource LEONARDO based at CINECA, Casalecchio di Reno, Italy, under a LEAP grant. I further acknowledge the CINECA award under the ISCRA initiative and the EuroHPC Joint Undertaking for providing access to the LEONARDO high-performance computing resources.

Abstract

This work advances the field of large-eddy simulation (LES) for wall-bounded turbulence through four key contributions. We begin by conducting direct numerical simulations (DNS) of rotating turbulent pipe flow at friction Reynolds numbers up to $Re_\tau \approx 3000$, investigating the physical mechanisms underlying drag reduction induced by steady axial rotation. Next, we present CaLES, a graphics processing unit (GPU) accelerated LES solver featuring advanced subgrid-scale (SGS) models and numerical schemes, with validation demonstrated on decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow. We then develop SmartFlow, a nearly solver-agnostic deep reinforcement learning (DRL) framework for computational fluid dynamics (CFD) deployable on high-performance computing (HPC) systems, showcasing its versatility through single-agent synthetic-jet control for cylinder flows, multi-agent wake control, and multi-task multi-agent wall-model training. Finally, we employ multi-task multi-agent DRL to construct wall models for LES of wall-bounded turbulence, focusing on model robustness and generalizability across flow conditions.

DNS of rotating pipe flows up to $Re_\tau \approx 3000$ are carried out to investigate drag reduction effects associated with axial rotation, extending previous studies carried out at a modest Reynolds number [99, 98]. The results show that the drag reduction, which we theoretically show to be equivalent to net power saving assuming no mechanical losses, monotonically increases as either the Reynolds number or the rotation number increases, proportionally to the inner-scaled rotational speed. Net drag reduction up to about 70% is observed, while being far from flow relaminarization. Scaling laws for the mean axial and azimuthal velocity are proposed, from which a predictive formula for the friction factor is derived. The formula can correctly represent the dependency of the friction factor on the Reynolds and rotation numbers, maintaining good accuracy for low-to-moderate rotation numbers. Examination of the turbulent structures highlights the role of rotation in widening and elongating the small-scale streaks, with subsequent suppression of sweeps and ejections. In the core part of the flow, clear weakening of large-scale turbulent motions is observed at high Reynolds numbers, with subsequent suppression of the outer-layer peak in the pre-multiplied spectra. The Fukagata-Iwamoto-Kasagi decomposition indicates that, consistent with a theoretically derived formula, the outer layer yields the largest contribution to drag reduction at increasingly high Reynolds numbers. In contrast, both the inner and the outer layers contribute to drag reduction as the rotation number increases.

We introduce CaLES, a GPU-accelerated finite-difference solver designed for LES of incompressible wall-bounded flows in massively parallel environments. Built upon the existing DNS solver CaNS, CaLES relies on low-storage, third-order Runge-Kutta schemes for temporal discretization, with the option to treat viscous terms via an implicit Crank-Nicolson scheme in one or three directions. A fast direct solver, based on eigenfunction expansions, is used to solve the discretized Poisson/Helmholtz equations. For turbulence modeling, the classical Smagorinsky model with van Driest near-wall damping and the dynamic Smagorinsky model are implemented, along with a logarithmic law wall model. GPU acceleration is achieved through OpenACC

directives, following CaNS-2.3.0. Performance assessments were conducted on the Leonardo cluster at CINECA, Italy. Each node is equipped with one Intel Xeon Platinum 8358 central processing unit (CPU) (2.60 GHz, 32 cores) and four NVIDIA A100 GPUs (64 GB HBM2e), interconnected via NVLink 3.0 (200 GB/s). The inter-node communication bandwidth is 25 GB/s, supported by a DragonFly+ network architecture with NVIDIA Mellanox InfiniBand HDR. Results indicate that the computational speed on a single GPU is equivalent to approximately 15 CPU nodes, depending on the treatment of viscous terms and the SGS model, and that the solver efficiently scales across multiple GPUs. The predictive capability of CaLES has been tested using multiple flow cases, including decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow. The high computational efficiency of the solver enables grid convergence studies on extremely fine grids, pinpointing non-monotonic grid convergence for wall-modeled LES.

DRL is emerging as a powerful tool for fluid-dynamics research, encompassing active flow control, autonomous navigation, turbulence modeling and discovery of novel numerical schemes. We introduce SmartFlow, a CFD-solver-agnostic framework for both single- and multi-agent DRL algorithms that can easily integrate with message passing interface (MPI) parallel CPU and GPU-accelerated solvers. Built on Relexi and SmartSOD2D, SmartFlow uses the SmartSim infrastructure library and our newly developed SmartRedis-MPI library to enable asynchronous, low-latency, in-memory communication between CFD solvers and Python-based DRL algorithms. SmartFlow leverages PyTorch’s Stable-Baselines3 for training, which provides a modular, Gym-like environment API. We demonstrate its versatility via three case studies: single-agent synthetic-jet control for drag reduction in a cylinder flow simulated by the high-order FLEXI solver, multi-agent cylinder wake control using the GPU-accelerated spectral-element code SOD2D, and multi-agent wall-model learning for LES with the finite-difference solver CaLES. SmartFlow’s CFD-solver-agnostic design and seamless HPC integration are promising to accelerate reinforcement learning (RL) driven fluid-mechanics studies.

Finally, we develop wall models for LES of wall-bounded turbulence using multi-task multi-agent DRL. The wall models are trained within the SmartFlow framework coupled with the GPU-accelerated CaLES solver, utilizing turbulent channel flows across multiple Reynolds numbers to enhance generalizability. Model robustness is evaluated through comprehensive testing at various Reynolds numbers and wall-model matching heights, demonstrating the potential of RL approaches for developing robust wall models.

Contents

1	Introduction	1
1.1	Direct numerical simulation of wall-bounded turbulent flow	3
1.2	GPU-accelerated large-eddy simulation solver	4
1.3	Deep reinforcement learning framework for computational fluid dynamics	6
1.4	Wall modeling in large-eddy simulation	10
1.5	Summary of work	12
2	Direct numerical simulation of wall-bounded turbulence	13
2.1	Navier–Stokes equations in cylindrical coordinates	13
2.2	The numerical dataset	14
2.3	Results	16
2.3.1	Drag reduction	16
2.3.2	Organisation of turbulence	22
2.3.3	Contributions to frictional drag	24
2.4	Numerical issues	27
2.5	Conclusions	28
3	GPU-accelerated large-eddy simulation solver	31
3.1	Methodology	31
3.1.1	Governing equations and subgrid-scale models	31
3.1.2	Numerical methods	33
3.2	Implementation and performance	36
3.2.1	Overall implementation strategy	36
3.2.2	Performance analysis	37
3.3	Validation	40
3.3.1	Decaying isotropic turbulence	41
3.3.2	Wall-resolved turbulent plane channel flow	43
3.3.3	Wall-modeled turbulent plane channel flow	46
3.3.4	Wall-modeled turbulent square duct flow	52
3.4	Conclusion	57
4	CFD-solver-agnostic deep reinforcement learning framework for computational fluid dynamics	59
4.1	Reinforcement learning	59
4.2	Software architecture	61
4.2.1	SmartSim	63

4.2.2	SmartRedis-MPI	66
4.2.3	CFD solver	67
4.2.4	Stable-Baselines3	67
4.3	Illustrative applications	71
4.3.1	Single-agent RL for flow control	71
4.3.2	Multi-agent RL for flow control of bluff body	75
4.4	Conclusions	78
5	Reinforcement learning for wall modeling in large-eddy simulation	79
5.1	Training setup	79
5.2	Results	82
5.3	Conclusions	84
6	Conclusions	85
	Bibliography	89

List of Figures

- 1.1 Typical Reynolds numbers in boundary layer applications. Re_L and Re_θ denote the Reynolds numbers based on the streamwise characteristic length and momentum thickness, respectively. Re_τ denotes the friction Reynolds number. ASL, atmospheric surface layer; PBL, planetary boundary layer. Adapted from Deck *et al.* [27] 2
- 1.2 Classification of unsteady approaches according to levels of modeling and readiness. Reproduced from Deck *et al.* [27]. 2
- 1.3 Implementation of wall-stress-modeled LES where the wall-model is decoupled from the LES grid in complex geometries. The wall-modeled thickness h_{wm} is specified independently from the LES grid, which necessitates interpolation from the LES grid to the top of the wall-modeled layer. Reproduced from Larsson *et al.* [75]. 11
- 2.1 Global flow properties: (a) friction factor as a function of N ; (b) friction factor as a function of Re_b ; (c) drag reduction coefficient as a function of N ; (d) drag reduction coefficient as a function of N^+ . In (a), (c) and (d), we show data for $Re_{\tau,0} = 180$ (black), 495 (red), 1137 (green), 1979 (blue) and 3020 (pink); in (b) we show $N = 0$ (black), 0.25 (red), 0.5 (green), 1.0 (cyan), 2.0 (blue) and 4.0 (pink). In (b), the brown dashed line denotes Prandtl's friction law [108], and the orange dashed line denotes the laminar friction law $\lambda = 64/Re_b$. In (c) and (d), the horizontal dashed lines denote the laminar limits of drag reduction for various Reynolds numbers. The triangle symbols denote DNS data at $Re_{\tau,0} \approx 180$ and 540 [24], and the circles denote experiments at $Re_{\tau,0} \approx 240$ and 570 [65]. 17
- 2.2 Inner-scaled mean axial velocity profiles (a), and corresponding logarithmic diagnostic functions (b). The color codes correspond to the same values of the rotation number: $N = 0$ (black), $N = 0.25$ (red), $N = 0.5$ (green), $N = 1$ (cyan), $N = 2$ (blue), $N = 4$ (pink). The line patterns correspond to the same values of the nominal Reynolds number: $Re_{\tau,0} = 495$ (dash-dotted), $Re_{\tau,0} = 1137$ (dotted), $Re_{\tau,0} = 1979$ (dashed), $Re_{\tau,0} = 3020$ (solid). The dashed black lines in panel (a) denote the expected logarithmic distribution given in Eq. (2.12), and in panel (b) the inverse of the Karman constant ($1/\kappa$). 18

-
- 2.3 Mean axial velocity defect profiles (U_{CL} is the pipe centreline velocity), normalized by the friction velocity, u_τ (a), and by the pipe rotational velocity, ΩR (b). The colour codes correspond to the same values of rotation number: $N = 0$ (black), $N = 0.25$ (red), $N = 0.5$ (green), $N = 1$ (cyan), $N = 2$ (green), $N = 4$ (pink). The line patterns correspond to the same values of the nominal Reynolds number: $Re_{\tau,0} = 495$ (dash-dotted), $Re_{\tau,0} = 1137$ (dotted), $Re_{\tau,0} = 1979$ (dashed), $Re_{\tau,0} = 3020$ (solid). The solid black lines in panel (b) indicate the power-law fitting function (2.13), with coefficients given in Eq. (2.14). 19
- 2.4 Inner-scaled bulk velocity (a,b) and friction factor (c,d), as a function of the friction Reynolds number (a,c) and bulk Reynolds number (b,d). The colour codes correspond to the same values of rotation number: $N = 0$ (black), $N = 0.25$ (red), $N = 0.5$ (green), $N = 1$ (cyan), $N = 2$ (blue), $N = 4$ (pink). Symbols denote the DNS data, and lines the corresponding predictions of Eq. (2.18). The solid black line denotes the predictions of Prandtl's friction law for a non-rotating pipe [108]. 20
- 2.5 Radial profiles of mean azimuthal velocity in linear (a) and logarithmic (b) scale. The colour codes correspond to the same values of rotation number: $N = 0.25$ (red), 1.0 (cyan), 2.0 (blue) and 4.0 (pink). The line patterns correspond to the same values of the nominal Reynolds number: $Re_{\tau,0} = 495$ (dash-dotted), $Re_{\tau,0} = 1137$ (dotted), $Re_{\tau,0} = 1979$ (dashed), $Re_{\tau,0} = 3020$ (solid). The solid black lines denote Eq. (2.21), with the coefficient given in Eq. (2.22). The black dashed lines correspond to the case of solid body rotation. 21
- 2.6 Instantaneous fields of axial velocity on a near-wall cylindrical shell and cross-stream plane. The cylindrical shell is located at a wall distance $y^+ \approx 15$. The colour scale is from blue (low speed) to red (high speed). Rotation numbers $N = 0, 0.5$ and 2.0 are reported in panels (a,d), (b,e), (c,f), respectively, for $Re_{\tau,0} = 495$ (a-c) and $Re_{\tau,0} = 3020$ (d-f). The arrows indicate the direction of rotation. . . 23
- 2.7 Distributions of: (a) turbulence kinetic energy for $Re_{\tau,0} = 495$; (b) turbulence kinetic energy for $Re_{\tau,0} = 3020$; (c) individual velocity variances for $Re_{\tau,0} = 3020$. In (c), lines correspond to: $\langle u_\theta^2 \rangle$ (solid); $\langle u_r^2 \rangle$ (dashed); $\langle u_z^2 \rangle$ (dash-dot). Color codes: $N = 0$ (black), 0.25 (red), 0.5 (green), 1.0 (cyan) and 2.0 (blue). 23
- 2.8 Pre-multiplied azimuthal energy spectra of axial velocity $(\kappa_\theta E_{uu})^+$ for $N = 0$ (a, d), $N = 0.5$ (b, e), $N = 2.0$ (c, f), at $Re_{\tau,0} = 495$ (a-c), $Re_{\tau,0} = 3020$ (d-f). The solid lines denote the trend $y = 0.14\lambda_\theta$, and the dashed lines denote the trend $\lambda_\theta/R = 2.35(y/R)^{0.52}$ in panel (c), and $\lambda_\theta/R = 2.20(y/R)^{0.72}$ in panel (f). The crosses denote spectral peaks. 25

2.9	(a) Turbulent shear stress for $N = 0$ (dashed lines) and $N = 2$ (solid lines); (b) local contributions to turbulent friction (as from Eq. (2.24)), for $N = 0$ (dashed lines) and $N = 2$ (solid lines); (c) turbulent contribution to friction as a function of Re_τ ; (d) fraction of drag reduction due to turbulence as a function of Re_b . In (a) and (b), $Re_{\tau,0} = 180$ (black), 495 (red), 1137 (green), 1979 (blue) and 3020 (pink). In (c) and (d), $N = 0$ (black), 0.25 (red), 0.5 (green), 1.0 (cyan), 2.0 (blue) and 4.0 (pink). In (c), the dash-dotted line indicates the prediction of Eq. (2.26).	26
2.10	Radial profiles for $Re_{\tau,0} = 1137$ and $N = 4.0$: (a) axial velocity; (b) azimuthal velocity. Pink: $L = 15R$ and $N_r = 164$; blue: $L = 30R$ and $N_r = 164$; green: $L = 15R$ and $N_r = 328$	28
2.11	Time history of axial pressure gradient for $Re_{\tau,0} = 495$. Black: $N = 0$; pink: $N = 4.0$	28
3.1	Schematic of the wall-model implementation.	35
3.2	Elapsed wall-clock time per time step per grid point for different methods when the viscous terms are handled explicitly (a), or implicitly in the y direction (b).	38
3.3	Comparison of wall-clock time breakdown for different computational components, when the viscous terms are handled explicitly (a), or implicitly in the y direction (b). “RHS” denotes the time spent calculating the right-hand side of the momentum equation, given the eddy viscosity; “Implicit” indicates the time required to solve the Helmholtz equations when the viscous terms are handled implicitly in the y direction; “Solver” corresponds to the time spent solving the Poisson equation; “Correction” denotes the time allocated for the correction procedure; and “SGS” represents the time required to evaluate the eddy viscosity using the classical Smagorinsky model with the van Driest damping function.	39
3.4	Strong (a) and weak (b) code scaling performance. Here, N_{nodes} denotes the number of nodes, T_2 represents the wall-clock time for $N_{nodes} = 2$, and T is the wall-clock time for a given N_{nodes}	40
3.5	Decaying isotropic turbulence: computed velocity spectra with 32^3 (a,b) and 64^3 (c,d) grid points. The experimental spectra [21] are shown unfiltered (a, c) and box-filtered (b, d). In each plot, the results for three time instants $tU_0/M = 42, 98$, and 171 are displayed from top to bottom. The vertical dashed lines denote the Nyquist limits.	42
3.6	Turbulent channel flow: visualization of streamwise velocity obtained with WRLES using the SM model on a mesh with $\Delta_z^+ = 6.8$ (a) and profiles of mean total (solid), resolved and modeled turbulent (dashed), resolved turbulent (dash-dotted), modeled turbulent (dotted), and viscous (solid) shear stress obtained with WRLES using the SM and DSM models on a grid with $\Delta_z^+ = 20.4$ (b). In (a), the wall-parallel plane is located at $y^+ \approx 15$. In (b), normalization is based on the wall units of each simulation.	44

3.7	Turbulent channel flow: profiles of mean streamwise velocity obtained with WRLES using the SM (a) and DSM (b) models. Normalization is based on wall units of each simulation. The DNS data is from [76].	45
3.8	Turbulent channel flow: profiles of resolved turbulent normal stress (a,b) and shear stress (c,d) obtained with WRLES using the SM (a,c) and DSM (b,d) models. Normalization is based on the DNS wall units. The DNS data is from [76]. Line codes: $\langle uu \rangle$ (solid), $\langle vv \rangle$ (dashed), $\langle ww \rangle$ (dash-dotted).	45
3.9	Turbulent channel flow: visualization of streamwise velocity obtained with WMLES using the SM model on a mesh with $\Delta_z/h = 0.012$ (a) and profiles of mean total (solid), resolved and modeled turbulent (dashed), resolved turbulent (dash-dotted), modeled turbulent (dotted) and viscous (solid) shear stress computed with WMLES using the SM and DSM models, for the grid with $\Delta_z/h = 0.050$ and $AR = 1.0$ (b). In (a), the wall-parallel plane is located at $y/h = 0.1$. In (b), the dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of each simulation.	46
3.10	Turbulent channel flow: profiles of mean streamwise velocity obtained with WMLES using the SM (a,c) and DSM (b,d) models on the grids with $AR = 1.0$ (a,b) and $AR = 2.0$ (c,d). The dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of each simulation. The DNS data is from [76].	48
3.11	Turbulent channel flow: profiles of resolved turbulent normal stress (a,b) and shear stress (c,d) obtained with WMLES using the SM (a,c) and DSM (b,d) models on the grids with $AR = 1.0$. The dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of the DNS (a,b) and the wall units of each simulation (c,d). Normalization is based on the DNS wall units. The DNS data is from [76]. Line codes: $\langle uu \rangle$ (solid), $\langle vv \rangle$ (dashed), $\langle ww \rangle$ (dash-dotted).	49
3.12	Turbulent channel flow: prediction error as a function of $\Delta z/h$ or Δz^+ for various statistical properties, as obtained from WMLES with SM (a,c) and DSM (b,d) models for grids with $AR = 1.0$ (a,b) and $AR = 2.0$ (c,d).	51
3.13	Computational setup for the square duct flow (a) and visualization of turbulent flow obtained with WMLES using the SM model on a mesh with $\Delta_z/h = 0.006$ (b). The planes display contours of the streamwise velocity, with the wall-parallel plane located at $y/h = 0.1$.	53
3.14	Turbulent square duct flow: cross-stream contours of mean velocity U^+ (row 1) and V^+ (row 2), turbulent normal stresses $\langle uu \rangle^+$ (row 3) and $\langle vv \rangle^+$ (row 4), and shear stress $\langle uv \rangle^+$ (row 5), computed on different grids using WMLES with the SM model. Normalization is based on the DNS wall units. Only a quarter of the full domain is presented. The horizontal axis is z and the vertical axis is y . $\Delta_z/h = 0.025$ (column 1), $\Delta_z/h = 0.017$ (column 2), $\Delta_z/h = 0.013$ (column 3), $\Delta_z/h = 0.006$ (column 4) and DNS [107] (column 5).	54

-
- 3.15 Turbulent square duct flow: profiles of mean streamwise velocity along the wall bisector (a,b) and corner bisector (c,d) obtained from WMLES with the SM (a,c) and DSM (b,d) models. The dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of each simulation. The DNS data is from [107]. 55
- 3.16 Turbulent square duct flow: profiles of resolved turbulent normal stress (a,b) and shear stress (c,d) along the wall bisector as obtained from WMLES with the SM (a,c) and DSM (b,d) models. The dashed line denotes $y/h = 0.1$. Normalization is based on the DNS wall units. The DNS data is from [107]. Line codes: $\langle uu \rangle$ (solid), $\langle vv \rangle$ (dashed), $\langle ww \rangle$ (dash-dotted). 56
- 4.1 Schematic of the interaction between the SmartFlow framework (middle box) and two distinct CFD environments (top and bottom boxes) within a multi-agent deep reinforcement learning (MARL) setup. Top: In the wall-modeling case, each agent receives a local state $s_t^{(i)} = \{h_{wm}^*, U^*\}$. The shared policy $\pi(a_t | s_t)$ selects actions $a_t^{(i)} = \tau_w(t_{n+1})/\tau_w(t_n)$, and rewards are computed as $r_t^{(i)} = f(\tau_w(t_{n+1}), \tau_w(t_n), \tau_{w,ref})$. Bottom: In the flow-control case, agents positioned around a cylinder observe local pressure states $s_t^{(i)} = \{P\}$ and output control actions $a_t^{(i)} = f(Q_{jet})$, modulating the jet mass-flow rate to perturb the wake and reduce drag. Rewards are defined from aerodynamic coefficients, $r_t^{(i)} = f(C_l, C_d)$, where C_l and C_d are lift and drag coefficients. This illustration is to highlight that the middle-box SmartFlow framework can be easily connected to distinct environments easily. The illustration is inspired by Figure 1 in Suárez *et al.* [128]. 62
- 4.2 SmartFlow architecture. Before entering the training loop, the SmartSim IL is used to launch the SmartSim orchestrator, which provides an in-memory database for data exchange between SmartFlow and the HPC workload. At the start of each iteration, the SmartSim IL launches a batch of CFD instances and distributes them to the worker nodes. Each CFD instance could be initialized from a random state and then advanced in time. For every RL update interval Δt_{RL} , each CFD environment sends its current states to the agents and receives new actions. We apply a synchronous RL algorithm, which means that SmartFlow waits until all simulations have terminated, then aggregates the collected episodes $\tau^{(i)}$ and performs an update on the policy weights θ . Here, $\nabla_{\theta} J(\tau^{(i)})$ denotes the gradient of the expected return evaluated on the current batch of experience. The loop of collect–update is then repeated with the new policy until the policy converges. The figure is adapted from Kurz *et al.* [70]. 65
- 4.3 Simulation setup for the flow around a two-dimensional cylinder. Pressure probe locations are indicated by black circles. 73

4.4	Evolution of the average training return for the MLP policies of Kurz <i>et al.</i> [73] (—) and the present study (—). The x-axis is truncated to skip intermediate episodes. The plot resumes at episode 1350 to show the converged values of the reference.	75
4.5	Long-term evolution of the drag (c_D) and lift (c_L) coefficients, comparing the uncontrolled case (—) to the MLP results from Kurz <i>et al.</i> [73] (—) and the present study (—).	75
4.6	(a) Training curve of the policy networks for drag-reduction control using SOD2D. Note that the $\bar{r}_i$ denotes the mean reward of all trajectories for a single episode. (b) Temporal evolution of the drag (C_d) and lift coefficient (C_l) before and after applying DRL control in deterministic mode, where the gray regions denote the uncontrolled evolution of C_d and C_l . Note that tU_∞/D is the dimensionless time unit.	77
5.1	Schematic of the multi-task multi-agent reinforcement learning (MT-MARL) setup. Top: Zero-pressure-gradient turbulent boundary layer. The curve shows the variation of Re_τ with Re_x , i.e., $Re_\tau \sim Re_x^{0.7}$. The turbulent boundary layer is used to demonstrate the variation of Re_τ in the streamwise direction. The true range of Re_τ values at the marked locations do not match the marked values. Middle: Turbulent channel flows at different Re_τ values, as marked in the top turbulent boundary layer at different streamwise locations. Bottom: The interaction between the agents and the environments. The agents share a single policy neural network. The agents receive local states $s_t^{(i)} = \{h_{wm}^*, U_{wm}^*\}$. The shared policy $\pi(a_t s_t)$ selects actions $a_t^{(i)} = \tau_w(t_{n+1})/\tau_w(t_n)$, and rewards are computed as $r_{t+1}^{(i)} = f(\tau_w(t_{n+1}), \tau_w(t_n), \tau_{w,ref})$. In the MT-MARL setup, the training includes multiple tasks, i.e., channel flows at multiple Re_τ values. For each time step, the states and rewards are obtained from a range of turbulent channel flows, and the shared policy samples actions, which are imposed on the corresponding channel environments. The goal is to train a single policy that is valid for different flow environments.	80
5.2	Channel environment setup. Agents are distributed evenly along the wall; each agent obtains state information at a wall-normal height h_{wm} , computes the reward at the wall, and provides the state to the policy π_θ , which then outputs the action for the next time step. Dashed lines denote grid lines. The figure is inspired by the illustration in Bae and Koumoutsakos [7].	81
5.3	Evolution of the agent's episode reward versus agent episode number. The reward is averaged over one batch of CFD simulations; the values are raw and no moving average is applied.	83
5.4	Mean streamwise velocity U^+ as a function of the wall-normal distance y^+ , where the superscript + indicates normalization by mean wall units. The different colors from bottom left to top right denote $Re_\tau = 10^3, 5200, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}$. The log law (black line) corresponds to $u^+ = 1/\kappa \ln(y^+) + B$, with $\kappa = 0.41$ and $B = 5.2$	83

-
- 5.5 Mean streamwise velocity U^+ as a function of the wall-normal distance y^+ , where the superscript $+$ indicates normalization by mean wall units. The different colors from bottom left to top right denote $Re_\tau = 10^3, 5200, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}$. The log law (black line) corresponds to $u^+ = 1/\kappa \ln(y^+) + B$, with $\kappa = 0.41$ and $B = 5.2$. The wall-modeled layer thickness is set as $h_{wm} = 0.05h, 0.075h, 0.1h, 0.15h, 0.2h$ 84

List of Tables

1.1	Open-source DNS/LES solvers for academic research.	5
1.2	Selected recent deep reinforcement learning applications in fluid mechanics. This table highlights representative state-of-the-art applications and is not intended to be exhaustive.	7
1.3	Open-source DRL frameworks for fluid mechanics. Data Comm.: Data communication between the DRL agent and the CFD solver. Parallel Env.: Parallel CFD environments. GPU: GPU-accelerated CFD solvers.	9
2.1	Flow parameters for DNS. $Re_b = 2Ru_b/\nu$ is the bulk Reynolds number, $Re_\tau = u_\tau R/\nu$ is the friction Reynolds number, $Re_{\tau,0}$ is the friction Reynolds number for the non-rotating case, $N = \Omega R/u_b$ is the rotation number, N_θ , N_r and N_z are the number of grid points along the azimuthal, radial and axial directions, respectively, $N^+ = \Omega R/u_{\tau,0}$ is the inner-scaled rotational speed; λ is the friction factor, DR is the percentage of drag reduction and #ETT is the time-averaging interval, expressed in terms of the eddy turnover time, R/u_τ . The symbol * denotes DNS with pipe length $L = 30R$, and ** denotes DNS with doubled resolution in the radial direction. The standard uncertainty of the friction factor is estimated using a modified batch means method [118], and the uncertainty of drag reduction is evaluated by propagating the standard uncertainties of the friction factors.	15
3.1	Estimated memory footprint per grid point for different methods. “SM” denotes the Smagorinsky model with the van Driest damping function, and “DSM” is the dynamic Smagorinsky model.	39
3.2	Computational parameters for WRLES of channel flow.	43
3.3	Computational parameters for WMLES of turbulent channel flow.	47
3.4	Parameters for the WMLES of turbulent square duct flow.	52
4.1	Evolution of the SmartFlow framework from Relaxi through SmartSOD2D.	64
4.2	Key API functions of the SmartRedis-MPI library.	66
4.3	Computing systems on which SmartFlow has been tested. NIC refers to the network interface card.	71

4.4	SmartFlow usage examples provided in the SmartFlow GitHub repository. NIC refers to the network interface card, N_{cases} denotes the number of parallel cases (tasks) for training, N_{cfd} is the number of CFD simulations per case, and N_{procs} is the number of MPI processes used per CFD simulation.	72
4.5	Illustrative applications of SmartFlow integrated with different CFD solvers	73

Chapter 1

Introduction

Wall-bounded turbulence is ubiquitous in engineering applications, including aircraft, vehicles, ships, and wind turbines. Direct numerical simulation (DNS) and wall-resolved large-eddy simulation (LES), known as WRLES, remain computationally prohibitive for high-Reynolds-number turbulence. Fig. 1.1 shows typical Reynolds number ranges in mechanical engineering applications, while Fig. 1.2 presents available turbulence modeling approaches. Spalart [127] estimated that DNS and WRLES may be practical for real aircraft simulations by 2045 and 2080, respectively. One promising approach to reduce LES computational cost is wall-modeled LES (WMLES) [104, 103, 75, 15], which models small-scale turbulent structures in the inner layer rather than resolving them directly. NASA Langley Research Center and the Center for Turbulence Research (CTR) have successfully applied WMLES to high-lift aircraft configurations, including the Japan Aerospace Exploration Agency Standard Model (JSM) at the 3rd AIAA computational fluid dynamics (CFD) High Lift Prediction Workshop (HLPW-3) [45] and the NASA High-Lift Common Research Model (CRM-HL) at HLPW-4 [149, 44].

This work explores deep reinforcement learning (DRL) for wall modeling in LES. To achieve this goal, the research addresses four specific objectives:

1. Investigate the fundamental physics of rotating wall-bounded turbulence through direct numerical simulation, providing insights into drag reduction mechanisms and turbulence modification;
2. Develop a graphics processing unit (GPU) accelerated LES solver for incompressible wall-bounded flows, enabling efficient training of data-driven wall models through high-performance computing (HPC);
3. Create a CFD-solver-agnostic DRL framework capable of seamless integration with diverse CFD codes on HPC platforms;
4. Construct robust and generalizable wall models for LES using multi-task multi-agent reinforcement learning, increasing applicability across a wide range of flow conditions.

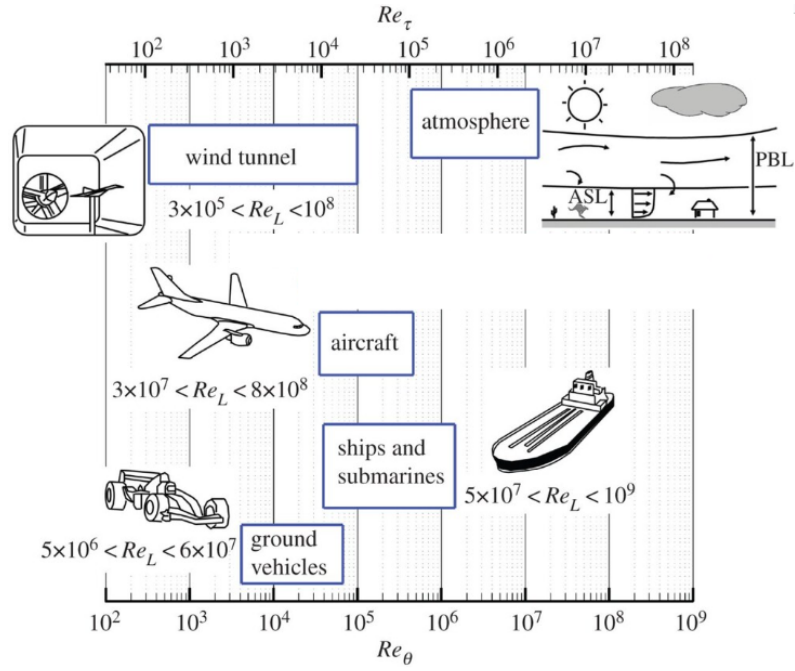


Figure 1.1. Typical Reynolds numbers in boundary layer applications. Re_L and Re_θ denote the Reynolds numbers based on the streamwise characteristic length and momentum thickness, respectively. Re_τ denotes the friction Reynolds number. ASL, atmospheric surface layer; PBL, planetary boundary layer. Adapted from Deck *et al.* [27]

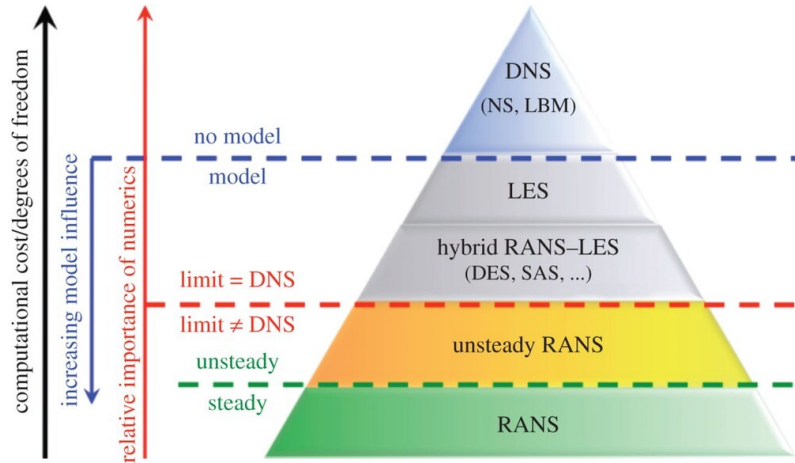


Figure 1.2. Classification of unsteady approaches according to levels of modeling and readiness. Reproduced from Deck *et al.* [27].

1.1 Direct numerical simulation of wall-bounded turbulent flow

Turbulent flow in circular pipes has always attracted the interest of scientists, owing to its prominent importance in engineering practice and because of the beautiful simplicity of the setup. The study of flows in circular pipes holds particular significance not only for its fundamental insights into fluid dynamics but also for its practical applications, especially in industries such as oil and gas transportation. Understanding the behavior of turbulent flow in pipelines is crucial for optimizing transport efficiency, ensuring structural integrity, and minimizing energy consumption. As such, research into flows in circular pipes continues to be an active area of study, driving innovations in both theoretical understanding and engineering practice.

Specifically, the research on drag reduction techniques in pipe flow holds significant importance due to its potential for reducing energy consumption in the transportation of gases or liquids, thereby leading to decreased pollutant emissions into the atmosphere. The potential of axial rotation for drag reduction in turbulent pipe flow was first shown by White [154]. In those experiments, dye was injected from a hypodermic tube into the central core of the flow in a pipe, and then diffused radially outwards to fill the whole pipe. When rotation was imposed on the pipe wall, the dye moved along the central core of the pipe without as much radial diffusion. The rate of rotation is typically measured in terms of the rotation number, $N = \Omega R / u_b$, where Ω is the rotational speed, R is the pipe radius, and u_b is the bulk velocity. The experiments of Kikuyama *et al.* [65] showed that as N increases, the wall friction decreases, with the mean axial velocity profiles approaching the parabolic Poiseuille solution. These findings were subsequently confirmed from the direct numerical simulations of Orlandi and Fatica [99]. Those authors mainly attributed the mechanism of drag reduction to the stabilizing effect of the radially growing centrifugal force (or angular momentum). Indeed, according to the Rayleigh criterion [115, 28], the radially growing pressure gradients impede the radial motion of fluid particles. Orlandi and Fatica [99] further pointed out that axial rotation induces a long columnar structure in the core region which transports the streamwise vortical structures away from the wall, while tilting and widening the near-wall streaks. Also, wall rotation disrupts the symmetry between right- and left-handed helical structures, and the resulting high helicity density yields less dissipation, with incurred drag reduction [97]. Zhang and Wang [167] showed that the sweep events are systematically suppressed by rotation, which further impedes the formation of hairpin structures. However, they found that rotation amplifies the azimuthal and radial velocity variances, and that the pressure-strain and Coriolis production terms become the leading terms in the budget of these two Reynolds normal stresses.

Davis *et al.* [24] reported that the drag reduction effect increases with the bulk Reynolds number, $Re_b = u_b D / \nu$, where $D = 2R$ is the pipe diameter, and ν is the kinematic viscosity. However, their highest friction Reynolds number, $Re_\tau = u_\tau R / \nu$ (with $u_\tau = (\tau_w / \rho)^{1/2}$ the friction velocity), was about 540, at which extrapolation to real-world scenarios of fully developed turbulent flow is far from clear, and well less than that achieved in DNS of non-rotating pipe flow [108].

Other techniques for drag reduction in turbulent pipe flow have been proposed

based on more complex wall actuation. Quadrio and Sibilla [110] studied turbulent flow in a circular pipe oscillating around its longitudinal axis through DNS and found that the maximum amount of drag reduction achievable with appropriate oscillations of the pipe wall is on the order of 40%, comparable to what was found in planar geometries. They observed that the transverse shear layer induced by wall oscillation induces substantial modifications to the turbulence statistics in the near-wall region, indicating a strong effect on the vortical structures. Auteri *et al.* [6] assessed the practical use of travelling waves of spanwise velocity, as suggested by Quadrio *et al.* [111] and further analyzed by Gatti and Quadrio [42], to achieve drag reduction in pipe flow. In their experiments, the pipe wall was subdivided into thin slabs that could rotate independently in the azimuthal direction, confirming the possibility of achieving drag reduction of up to 33%.

1.2 GPU-accelerated large-eddy simulation solver

LES has become an important tool in the design processes of spacecraft, aircraft, automobiles, ships, and other engineering systems, for cases where traditional Reynolds-averaged Navier-Stokes (RANS) models fall short. RANS models often struggle to accurately predict complex flow phenomena, such as flow separation, rotational turbulence, and three-dimensional turbulent boundary layers [155]. DNS can provide accurate results but becomes computationally prohibitive at high Reynolds numbers due to the high mesh resolutions required. As a result, LES has gained popularity for offering a reasonable balance between computational cost and predictive accuracy, particularly for complex aerodynamic and hydrodynamic applications. In recent years, LES has been increasingly applied to external aerodynamic flows, especially at the edges of flight envelopes, such as high-lift aircraft aerodynamics [45] and iced-wing separated flows [157]. It has also been used in internal flows, including those within aircraft engines [5]. These applications establish LES as a critical component in the industry push toward certification by analysis [84]. Despite significant advancements, both wall models and subgrid-scale (SGS) closures for LES remain areas of active development. For wall modeling, researchers have been exploring robust models that account for more complex flows, such as laminar-turbulent transitions [75] and flow separation [30] among others. Concurrently, advancements in SGS models have been made, with some of the recent studies emphasizing robust SGS models suited to anisotropic grids [3].

It is well established that LES is significantly more computationally demanding than RANS, making the development of faster LES solvers a critical need. An efficient LES solver is not only essential for large-scale production simulations, but also invaluable for the development and testing of subgrid-scale and wall models. Indeed, as data-driven approaches and machine learning techniques are increasingly applied to turbulence modeling, the demand for efficient LES solvers becomes even more pressing. Previous studies have trained SGS models using filtered DNS data [39, 62]. However, relying on DNS data can be problematic, as the SGS tensor in implicitly filtered LES does not perfectly align with the SGS stress terms derived from the filtered Navier-Stokes equations [8]. This inconsistency is especially significant when grid sizes are significantly larger than the Kolmogorov scale, as the numerical and

modeling errors can then be comparable. As a result, SGS models trained on filtered DNS data may perform well in *a priori* tests but fail in *a posteriori* assessments, whereby performance is evaluated in actual LES simulations. Consequently, research has increasingly shifted toward generating training data directly from LES, and turbulence models are optimized for accurate statistical metrics such as mean velocity and wall shear stress [81]. In the same vein, reinforcement learning has been explored for both SGS and wall modeling [95, 7]. Such strategies have been referred to as “model-consistent” approaches [31]. While these strategies show promise, they often require hundreds or even thousands of LES runs to generate training data or complete one-time training, underscoring the need for fast and efficient solvers. Moreover, the grid convergence properties of WMLES have become increasingly studied [168, 165], which also requires many LES runs on a series of refined meshes for thorough analysis. Fast LES solvers are therefore desirable to enable grid convergence investigations at high Reynolds numbers.

Table 1.1 gives some popular open-source solvers used in academic research. Whereas all those solvers are capable of performing DNS for canonical flow cases, only URANOS and LESGO currently support LES capabilities. It is well-known that incompressible solvers, such as LESGO, are typically more efficient than compressible solvers like URANOS for simulations of low-Mach-number flows. This efficiency mainly derives from allowing much larger time steps when explicit time-integration schemes are used. However, a key limitation of LESGO is the absence of GPU acceleration, which restricts its scalability and efficiency on modern high-performance computing platforms. Given the high demand for LES in simulating incompressible flows, particularly for machine-learning-based turbulence modeling and grid convergence studies, the development of a GPU-accelerated incompressible LES solver is highly desirable in academia.

Table 1.1. Open-source DNS/LES solvers for academic research.

Solver	Governing equations	GPU-supported	Purposes
STREAmS-2 [13]	Compressible NS	Yes	DNS
URANOS [25]	Compressible NS	Yes	DNS/LES
AFiD [171]	Incompressible NS	Yes	DNS
LESGO [2]	Incompressible NS	No	DNS/LES
CaNS [23]	Incompressible NS	Yes	DNS
CaLES	Incompressible NS	Yes	DNS/LES

The present work introduces CaLES, a GPU-accelerated incompressible LES solver specifically designed for wall-bounded flows. CaLES builds on the capabilities of CaNS [23, 22], an open-source DNS solver known for its efficiency in solving the incompressible Navier-Stokes equations. The solver uses a fractional-step method [66]. Temporal discretization is carried out using a low-storage, third-order Runge-Kutta scheme [156]. Spatial discretization is performed using a second-order finite-difference method on staggered grids [52], which avoids odd-even decoupling phenomena and preserves energy at the discrete level in the inviscid limit [140]. The solver employs eigenfunction expansions [124] to efficiently solve the Poisson equation. GPU acceleration is achieved using a combination of CUDA Fortran and OpenACC directives, and performance benchmarks demonstrate that the code per-

formance on 4 NVIDIA Tesla V100 GPUs in a DGX-2 system is roughly equivalent (0.9 times slower to 1.6 times faster) to 2048 cores on state-of-the-art CPU-based supercomputers, and 3.1 to 5.6 times faster when all 16 GPUs in the DGX-2 cluster are used [23]. CaLES extends CaNS by supporting LES through the inclusion of the classical Smagorinsky model [126] with the van Driest damping function [136], and the dynamic Smagorinsky model [43, 78], along with a logarithmic-law wall model. The solver can simulate various canonical flows in Cartesian single-block domains, including isotropic turbulence, temporally-evolving turbulent boundary layers, channel flows, duct flows, cavity flows, etc. Flexibility and efficiency make it an ideal platform to develop subgrid-scale and wall models, particularly those based on machine learning techniques, and to perform grid convergence studies at high Reynolds numbers. In this work, we present CaLES as an extension of CaNS, which was designed to maintain simplicity and adaptability, facilitating its use across diverse applications.

1.3 Deep reinforcement learning framework for computational fluid dynamics

DRL has emerged as a versatile and powerful approach in fluid mechanics research, enabling, for instance, the discovery of active control strategies, autonomous navigation, shape optimization, turbulence models and numerical algorithms as summarized in Table 1.2. Active flow control applications include separation control on flat plates [35], bluff bodies [129, 73] and airfoils [40, 89], as well as frictional drag reduction [50, 11], and wake stabilization [160]. Beyond control, DRL agents have also been applied to navigate complex fluid environments, such as autonomous urban navigation [131] and biological swimming [67]. Shape optimization via DRL has been demonstrated for airfoils [77]. In turbulence modeling, DRL has been employed to develop LES subgrid-scale closures and wall models for the modeling of unresolved turbulent stresses [95, 72, 53] and wall shear stress [7, 169, 135]. DRL-based methods have also been used to construct zone-adaptive combustion models [162], RANS closures [37], and nonlinear numerical schemes for conservation laws [148, 64].

Despite the wide range of DRL applications in fluid mechanics, existing DRL frameworks typically exhibit tight coupling to specific CFD solvers and lack full support for deployment on HPC infrastructures, especially GPU-accelerated supercomputers. A comprehensive overview of frameworks is given in Table 1.3. DRLFEniCS [113] uses the FEniCS central processing unit (CPU) based finite element framework to build its CFD solver. FEniCS has a Python application programming interface (API), so that the bridging can be performed either directly through native Python code importing both the FEniCS and DRL frameworks when running on a single machine or node, or UNIX sockets mechanisms if distributing over several nodes or machines. Guastoni *et al.* [50] used an interface based on the message-passing interface (MPI) to couple the CPU-based solver SIMSON and DRL environment, which are coded in Fortran 77/90 and Python, respectively. The CFD solver is dynamically spawned as a child process, and an intercommunicator is created between the solver and the Python main program. DRLinFluids [152] integrates the CPU-based OpenFOAM with DRL libraries such as Tensorforce

Table 1.2. Selected recent deep reinforcement learning applications in fluid mechanics. This table highlights representative state-of-the-art applications and is not intended to be exhaustive.

Field of research	Applications
Active flow control	Separation control [35, 129, 73, 40, 89, 128, 161, 88, 137] Frictional drag reduction [50, 11, 18] Wake stabilization [160] Suppression of chaotic dynamics [100] Rayleigh–Bénard convection control [138]
Autonomous navigation	Navigation in urban flows [131] Swimming [67, 139, 51, 159, 109]
Shape optimization	Airfoil shape optimization [77, 145]
Turbulence modeling	LES subgrid-scale and wall models [95, 72, 53, 7, 169, 135, 34, 146] RANS models [37] Turbulent flame modeling [162]
Numerical algorithms	Nonlinear numerical schemes for conservation laws [148, 64, 9]

and Tianshou, facilitating active flow control and optimization studies within fluid mechanics. Its adaptable architecture makes DRLinFluids a valuable tool for researchers and engineers. However, the requirement for I/O operations at each control step may introduce a potential bottleneck in training efficiency, though this can be mitigated with the use of, for example, a RAM disk. DRLFluent [82] is a distributed co-simulation framework that links Python-based deep reinforcement learning agents with the commercial C++ CFD solver Ansys-Fluent via a CORBA object-request broker (omniORB), enabling low-latency, cross-node communication on HPC clusters. Its architecture exposes two high-level interfaces that handle most CFD–RL interactions, reducing the need for labor-intensive secondary coding. DRLFluent integrates seamlessly with Fluent’s closed-source core; no modifications to the solver itself are required. However, it might require a non-trivial amount of work to replace the CFD solver with a different one. Novati *et al.* [95] proposed the scientific multi-agent reinforcement learning (SciMARL) framework based on the open-source library Smarties [94], which is designed to enable efficient coupling between DRL algorithms and flow solvers. The framework employs the remember and forget experience replay (ReF-ER) method to stabilize learning and reduce the variance in policy updates caused by the simultaneous actions of multiple agents. However, its scalability may be constrained by centralized coordination and the volume of data exchanged between the simulation and learning components, particularly when large flow fields or high-frequency interactions are involved in simulations distributed across many compute nodes. Kurz *et al.* [69] developed the TF-Agents-based Relexi platform. Its advantage is that it offers SmartRedis-backed scalability, though it is tied to the CPU-based FLEXI CFD solver. Wind-RL [87] employed a similar strategy, using the TorchRL library for reinforcement learning algorithms and the high-order finite-difference Fortran-based Xcompact3d as the backend CFD solver

to train active flow controllers for wind turbines. SmartSOD2D [35] is developed on top of Relexi and supports GPU-accelerated CFD simulations. However, it remains tightly coupled to the SOD2D CFD solver.

Beyond these frameworks, several libraries have been proposed to aid DRL-CFD coupling. It is important to distinguish between frameworks and libraries in this context: a framework defines a structured pipeline for DRL-CFD workflow, while a library offers reusable classes or functions that can be integrated into frameworks (pipelines). For example, Shams and Elsheikh [125] introduced the Gym-preCICE library, which provides a generic adapter class to couple reinforcement learning agents with numerical solvers governed by partial differential equations. The adapter offers a Gymnasium (formerly OpenAI Gym)-like API and uses the preCICE coupling library for backend communication. It supports multiple communication backends, including MPI, TCP/IP, and shared memory. Although flexible, Gym-preCICE requires users to design and implement their own DRL-CFD framework around the adapter, which might involve a non-trivial amount of coding effort. HydroGym [74] is a library designed for the application of reinforcement learning to computational flow control research. It offers Gymnasium-compatible environments encompassing different flow-control benchmarks that include both non-differentiable and differentiable formulations. These environments have been evaluated using RL algorithms implemented through different reinforcement learning libraries. HydroGym supports deployment from desktop workstations to high-performance computing clusters. However, HydroGym requires that the target CFD solver expose a Python-callable interface, which might involve non-trivial development effort to adapt legacy CFD solvers written in Fortran, C or C++, due to the so-called “two-language gap” [35, 69].

In addition to these code architecture and hardware and software support considerations, several features have emerged as critical to enabling effective use of DRL for flow control. In particular, i) the parallel environment approach (see e.g. Rabault and Kuhnle [112]) and ii) the multi-agent reinforcement learning (MARL, see e.g. Belus *et al.* [10]), discussed in more detail in the next section, are key factors to be able to apply DRL to complex flow problems. There too, disparities exist in the level of support offered by different frameworks, and a unified approach would be beneficial.

To address these limitations, we have developed SmartFlow, a nearly CFD-solver-agnostic multi-agent DRL framework designed for modern CPU and GPU-accelerated HPC clusters. Here, the term “multi-agent” means multiple agents (corresponding to multiple pseudo-environments) with a single shared policy. Built on Relexi [70] and SmartSOD2D [35], SmartFlow uses the SmartSim [102] infrastructure and our newly developed SmartRedis-MPI library to provide asynchronous low-latency in-memory data exchange between Fortran-based CFD solvers and Python-based DRL agents. The newly developed SmartRedis-MPI library makes it trivial to link SmartFlow to a wide variety of CPU or GPU-accelerated CFD solvers. We replace the TensorFlow-based TF-Agents [49] backend with PyTorch’s Stable-Baselines3 [114] for its simplicity and usability, exposing all algorithms through a modular, Gym-inspired [133] environment API. As Table 1.3 shows, SmartFlow can be applied with minimal boilerplate code to single- and multi-agent tasks in the fields of active flow control, autonomous navigation, shape optimization, turbulence modeling, and even numerical scheme discovery. A comparison of SmartFlow with existing frameworks is

Table 1.3. Open-source DRL frameworks for fluid mechanics. Data Comm.: Data communication between the DRL agent and the CFD solver. Parallel Env.: Parallel CFD environments. GPU: GPU-accelerated CFD solvers.

Framework	CFD solver	DRL library	Data Comm.	Parallel Env.	MARL	GPU
DRLFEniCS [113]	FEniCS	TensorForce	Native Python / UNIX sockets	Yes	No	No
Guastoni et al. [50]	SIMSON	Stable-Baselines3 (PyTorch)	MPI	No	Yes	No
Suárez et al. [129]	Alya	TensorForce	MPI and file I/O stream	Yes	Yes	No
DRLinFluids [152]	OpenFoam	Tensorforce/Tianshou	File I/O stream and shared memory	Yes	No	No
DRLFluent [82]	Ansys-Fluent	TensorForce	omniORB	Yes	No	Yes
sciMARL [95]	In-house code	Smarties	MPI	Yes	Yes	No
Relaxi [69]	FLEXI	TF-Agents	SmartRedis	Yes	Yes	No
Wind-RL [87]	Xcompact3d	TorchRL	SmartRedis	Yes	No	No
SmartSOD2D [35]	SOD2D	TF-Agents	SmartRedis	Yes	Yes	Yes
SmartFlow (present)	CFD-solver-agnostic	Stable-Baselines3	SmartRedis-MPI	Yes	Yes	Yes

provided in Table 1.3. SmartFlow employs the above-mentioned two DRL techniques to accelerate training from a computational point of view. The first is to simulate multiple environments in parallel, also known as multi-environment DRL, so that the agent collects multiple experiences in parallel [112]. The second, orthogonal to the multi-environment approach, is a multi-agent reinforcement-learning method [10, 142].

1.4 Wall modeling in large-eddy simulation

Wall-modeled LES methods can be broadly classified into hybrid LES/RANS approaches and LES with wall-stress models, though other variants exist [75, 15]. This work focuses on the latter category, in which wall-stress models provide approximate wall-stress boundary conditions to the LES. Fig. 1.3 illustrates the implementation of wall-stress-modeled LES in complex geometries, where the wall model operates on a grid decoupled from the outer LES grid.

Wall-stress models can be categorized as equilibrium and non-equilibrium models. Equilibrium approaches assume that the sum of the pressure gradient and acceleration terms is negligible, resulting in a constant-stress layer that corresponds to the logarithmic law in the inner region. Early implementations by Deardorff [26], Schumann [123] and Piomelli *et al.* [105] applied the logarithmic law to the mean flow, while Mason and Callen [83] enforced it on the instantaneous flow. This instantaneous approach was extensively investigated by Werner and Wengle [153], who combined a power law with a linear law, and by Kawai and Larsson [63], who solved the instantaneous equilibrium ordinary differential equation (ODE) in the wall-normal direction. The validity of the instantaneous logarithmic law assumption strongly depends on the near-wall grid resolution, which must be sufficiently coarse to contain numerous inner-layer eddies [104, 16]. A fundamental limitation of equilibrium approaches is their restriction to monotonically varying velocity profiles, which can lead to incorrect wall shear stress directions in separated flows. Non-equilibrium models extend equilibrium approaches by accounting for pressure gradient and acceleration effects. Early developments by Hoffmann and Benocci [55], Wang [150], Wang and Moin [151], Duprat *et al.* [29], and Griffin and Fu [48] incorporated pressure gradients. However, Hickel *et al.* [54] argued that including only the pressure term without the acceleration term is inconsistent with the momentum balance in the outer layer, where these terms are approximately in equilibrium. Consequently, Wang and Moin [151], Kawai and Larsson [63], and Park and Moin [101] incorporated both terms directly, resulting in wall models formulated as partial differential equations (PDEs) solved on three-dimensional wall-model grids. Alternatively, Hickel *et al.* [54] modeled both terms to preserve the computational simplicity of ODEs solved on one-dimensional grids. Yang *et al.* [164] proposed an integral wall model that is local and algebraic rather than differential in space. Notably, only PDE-based models avoid the restrictive assumption that the wall shear stress vector aligns with the LES velocity vector at the matching location.

Machine learning approaches have recently emerged as promising alternatives for turbulence modeling [32, 31]. Most studies have focused on RANS closures and SGS stress modeling using supervised learning techniques, including random forests,

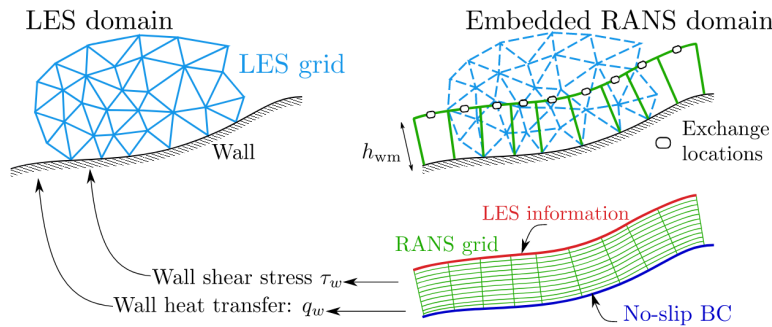


Figure 1.3. Implementation of wall-stress-modeled LES where the wall-model is decoupled from the LES grid in complex geometries. The wall-modeled thickness h_{wm} is specified independently from the LES grid, which necessitates interpolation from the LES grid to the top of the wall-modeled layer. Reproduced from Larsson *et al.* [75].

support vector machines, and neural networks. Yang *et al.* [163] were among the first to apply machine learning to wall modeling in LES, training a neural network on DNS data from fully developed channel flow to predict wall shear stress from flow field inputs. Subsequent research [14, 57, 80, 90, 166, 170] has extended this approach. Notably, Lozano-Durán and Bae [80, 81] achieved preliminary success in simulating the NASA juncture flow by first classifying flow regions before predicting wall shear stress. However, Zhou *et al.* [170] observed significant discrepancies in *a posteriori* tests of periodic hill flow, despite excellent *a priori* performance. These discrepancies arise from systematic differences between the WRLES training environment and the WMLES testing environment, particularly differences between explicit filtering during training and implicit filtering during testing [70]. This highlights the importance of accounting for both SGS and numerical errors in wall model development [75].

DRL [130, 36] offers potential solutions to these challenges. In DRL, an agent learns an optimal policy by interacting with an environment: the agent observes states, selects actions, and receives rewards, with the objective of maximizing cumulative reward over episodes. While DRL has been extensively applied to flow control and aerodynamic optimization [17, 41, 144], recent work has explored its application to SGS closures [71] and wall modeling [7] for LES. A key advantage of DRL is that training occurs within WMLES runs rather than using DNS/WRLES data, thereby eliminating systematic training-testing discrepancies and naturally accounting for numerical and SGS errors. Furthermore, DRL training may be computationally more efficient than supervised learning for high-Reynolds-number complex flows. This DRL-based approach shares conceptual similarities with the optimal control framework for wall modeling introduced by Nicoud *et al.* [91]. The present study aims to explore the application of DRL to wall model development for LES.

1.5 Summary of work

This thesis presents four main contributions to the field of large-eddy simulation of wall-bounded turbulence: improved understanding of rotating wall-bounded turbulence through DNS, development of a GPU-accelerated LES solver, a CFD-solver-agnostic deep reinforcement learning framework, and application of deep reinforcement learning to wall modeling.

1. **Direct numerical simulation of rotating wall-bounded turbulence.** We perform DNS of turbulent pipe flow up to $Re_\tau \approx 3000$ with steady axial rotation to quantify drag reduction effects and elucidate the underlying physical mechanisms.
2. **CaLES – A GPU-accelerated solver for large-eddy simulation of wall-bounded flows.** We develop a high-performance GPU-accelerated LES solver incorporating advanced subgrid-scale models and numerical methods. The solver is validated through simulations of decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow.
3. **SmartFlow – A CFD-solver-agnostic deep reinforcement learning framework.** We introduce a versatile DRL framework for computational fluid dynamics on HPC platforms. The framework’s solver-agnostic architecture is demonstrated through three diverse applications:
 - (a) Single-agent synthetic-jet control in a cylinder wake using the high-order FLEXI solver
 - (b) Multi-agent wake control with the GPU-accelerated SOD2D spectral-element code
 - (c) Multi-agent wall-model learning in LES with the finite-difference CaLES solver
4. **Reinforcement learning for wall modeling in large-eddy simulation.** We apply deep reinforcement learning to develop wall models for large-eddy simulation of wall-bounded turbulence, with emphasis on the robustness and generalizability of the trained model.

Chapter 2

Direct numerical simulation of wall-bounded turbulence

We perform DNS of turbulent pipe flow up to $Re_\tau \approx 3000$ with steady axial rotation to investigate drag reduction effects and advance the physical understanding of rotating wall-bounded turbulence. The objectives are to quantify drag reduction, elucidate the underlying physical mechanisms, and characterize the modifications to turbulent structures and turbulent energy induced by rotation.

2.1 Navier–Stokes equations in cylindrical coordinates

The incompressible Navier–Stokes and continuity equations are rewritten with the quantities $q_\theta = v_\theta$, $q_r = r \cdot v_r$, and $q_z = v_z$. In this way the continuity equation assumes a form similar to that in Cartesian coordinates:

$$\frac{\partial q_r}{\partial r} + \frac{\partial q_\theta}{\partial \theta} + r \frac{\partial q_z}{\partial z} = 0. \quad (2.1)$$

In terms of the q_i the momentum equations in conservative form become

$$\frac{Dq_\theta}{Dt} = -\frac{1}{r} \frac{\partial p}{\partial \theta} + \frac{1}{\text{Re}} \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial q_\theta}{\partial r} \right) - \frac{q_\theta}{r^2} + \frac{1}{r^2} \frac{\partial^2 q_\theta}{\partial \theta^2} + \frac{\partial^2 q_\theta}{\partial z^2} + \frac{2}{r^3} \frac{\partial q_r}{\partial \theta} \right], \quad (2.2)$$

$$\frac{Dq_r}{Dt} = -r \frac{\partial p}{\partial r} + \frac{1}{\text{Re}} \left[r \frac{\partial}{\partial r} \left(\frac{1}{r} \frac{\partial q_r}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 q_r}{\partial \theta^2} + \frac{\partial^2 q_r}{\partial z^2} - \frac{2}{r} \frac{\partial q_\theta}{\partial \theta} \right], \quad (2.3)$$

$$\frac{Dq_z}{Dt} = -\frac{\partial p}{\partial z} + \frac{1}{\text{Re}} \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial q_z}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 q_z}{\partial \theta^2} + \frac{\partial^2 q_z}{\partial z^2} \right] \quad (2.4)$$

with

$$\frac{Dq_\theta}{Dt} \equiv \frac{\partial q_\theta}{\partial t} + \frac{1}{r^2} \frac{\partial r q_\theta q_r}{\partial r} + \frac{1}{r} \frac{\partial q_\theta^2}{\partial \theta} + \frac{\partial q_\theta q_z}{\partial z}, \quad (2.5)$$

$$\frac{Dq_r}{Dt} \equiv \frac{\partial q_r}{\partial t} + \frac{\partial}{\partial r} \left(\frac{q_r^2}{r} \right) + \frac{\partial}{\partial \theta} \left(\frac{q_\theta q_r}{r} \right) + \frac{\partial q_r q_z}{\partial z} - q_\theta^2, \quad (2.6)$$

$$\frac{Dq_z}{Dt} \equiv \frac{\partial q_z}{\partial t} + \frac{1}{r} \frac{\partial q_r q_z}{\partial r} + \frac{1}{r} \frac{\partial q_\theta q_z}{\partial \theta} + \frac{\partial q_z^2}{\partial z}, \quad (2.7)$$

where in the q_θ equation the following identities have been used:

$$\frac{1}{r} \left(\frac{\partial}{\partial r} r \frac{\partial q_\theta}{\partial r} \right) - \frac{q_\theta}{r^2} = \frac{\partial}{\partial r} \left(\frac{1}{r} \frac{\partial r q_\theta}{\partial r} \right), \quad (2.8)$$

$$\frac{1}{r^2} \frac{\partial r q_\theta q_r}{\partial r} = \frac{1}{r} \frac{\partial q_\theta q_r}{\partial r} + \frac{q_\theta q_r}{r^2}. \quad (2.9)$$

The dimensionless equations are obtained by taking the pipe radius R as reference length, the laminar Poiseuille velocity u_p , twice the bulk velocity u_b , as velocity scale. These quantities together with the kinematic viscosity ν give $Re = u_b D / \nu = u_p R / \nu$ and the rotation number $N = \Omega R / u_b$. In the z direction both periodic and wall-bounded flows can be considered with only minor changes in the numerical procedure. Due to the use of trigonometric expansions in the pressure correction solver, the method at the moment is limited to uniform grids in θ and z while a nonuniform grid can be used in the radial direction. More details on the pipe flow solver can be found in [141, 108].

2.2 The numerical dataset

A second-order finite-difference solver is used to solve the incompressible Navier-Stokes equations in cylindrical coordinates [99, 108]. The current DNS pertain to a fully developed turbulent flow, with periodic conditions along the axial direction. The DNS are carried out in a frame of reference rotating with the pipe, which has the advantage of allowing larger time steps compared to the inertial frame. Coriolis forces are then added as $-2\Omega u_r$ and $2\Omega u_\theta$ to the azimuthal and radial momentum equations, where u_θ and u_r are the velocity components in the azimuthal and radial directions. The simulations were conducted under conditions of constant mass flow rate. From now on, the subscript 0 indicates non-rotating cases, and the superscript $*$ is used to denote normalization with wall units. Normalization with wall units based on the non-rotating case is also used to better highlight the effects of pipe rotation, which we indicate with the $+$ superscript. We performed DNS for various bulk Reynolds numbers, namely $Re_b = 5300, 17000, 44000, 82500$, and 133000 , corresponding to friction Reynolds numbers of $Re_{\tau,0} = 180, 495, 1137, 1979$ and 3020 , in the absence of rotation. For each Reynolds number, we consider various rotation numbers, namely $N = 0, 0.25, 0.5, 1.0, 2.0$ and 4.0 . A list including the flow parameters for all simulations is provided in Table 2.1. The pipe length is set to $L = 15R$, and the mesh resolution for each Reynolds number is determined based on the non-rotating cases, which exhibit the highest wall friction. Specifically, the grid spacing is $\Delta z^+ \approx 10$, and $R^+ \Delta \theta \approx 4$, along the axial and azimuthal directions, respectively. In the radial direction, the grid spacing is uniform up to $y^+ \approx 5$, and then proportional to the local Kolmogorov length scale ($\eta^+ \approx 0.8(y^+)^{1/4}$) in the outer layer. About thirty grid points are allocated for $y^+ \leq 40$, with the first grid point located at $y^+ \approx 0.05$. Additional details can be found in previous publications [106]. The sensitivity of the results to pipe length and mesh resolution is analyzed in Section 2.4. Hereafter, capital letters will be used to denote flow properties averaged in the homogeneous spatial directions and in time, brackets to denote the averaging operator, and lower-case letters to denote fluctuations from the mean.

Table 2.1. Flow parameters for DNS. $Re_b = 2Ru_b/\nu$ is the bulk Reynolds number, $Re_\tau = u_\tau R/\nu$ is the friction Reynolds number, $Re_{\tau,0}$ is the friction Reynolds number for the non-rotating case, $N = \Omega R/u_b$ is the rotation number, N_θ , N_r and N_z are the number of grid points along the azimuthal, radial and axial directions, respectively, $N^+ = \Omega R/u_{\tau,0}$ is the inner-scaled rotational speed; λ is the friction factor, DR is the percentage of drag reduction and #ETT is the time-averaging interval, expressed in terms of the eddy turnover time, R/u_τ . The symbol * denotes DNS with pipe length $L = 30R$, and ** denotes DNS with doubled resolution in the radial direction. The standard uncertainty of the friction factor is estimated using a modified batch means method [118], and the uncertainty of drag reduction is evaluated by propagating the standard uncertainties of the friction factors.

Re_b	$Re_{\tau,0}$	N	Mesh ($N_\theta \times N_r \times N_z$)	Re_τ	N^+	$\lambda \times 10^3$	DR (%)	#ETT
5300	180.10	0	$256 \times 55 \times 256$	180.10	0.0	$37.469 \pm 0.24\%$	$0.00 \pm 0.00\%$	74.76
5300	180.10	0.25	$256 \times 55 \times 256$	174.69	3.7	$34.748 \pm 0.35\%$	$7.26 \pm 5.42\%$	72.51
5300	180.10	0.5	$256 \times 55 \times 256$	168.27	7.4	$32.256 \pm 0.52\%$	$13.80 \pm 3.54\%$	10.64
5300	180.10	1.0	$256 \times 55 \times 256$	168.62	14.7	$32.388 \pm 0.45\%$	$12.32 \pm 3.25\%$	70.60
5300	180.10	2.0	$256 \times 55 \times 256$	168.43	29.4	$32.317 \pm 0.58\%$	$12.52 \pm 3.94\%$	70.02
5300	180.10	4.0	$256 \times 55 \times 256$	167.26	58.9	$31.869 \pm 0.76\%$	$13.73 \pm 4.54\%$	70.34
17000	495.29	0	$768 \times 96 \times 768$	495.29	0.0	$27.160 \pm 0.09\%$	$0.00 \pm 0.00\%$	35.00
17000	495.29	0.25	$768 \times 96 \times 768$	481.12	4.3	$25.624 \pm 0.10\%$	$5.80 \pm 2.24\%$	42.51
17000	495.29	0.5	$768 \times 96 \times 768$	452.65	8.6	$22.681 \pm 0.16\%$	$16.62 \pm 0.93\%$	26.68
17000	495.29	1.0	$768 \times 96 \times 768$	411.53	17.2	$18.747 \pm 0.29\%$	$31.08 \pm 0.68\%$	24.26
17000	495.29	2.0	$768 \times 96 \times 768$	396.91	34.3	$17.439 \pm 0.44\%$	$35.89 \pm 0.81\%$	23.39
17000	495.29	4.0	$768 \times 96 \times 768$	376.59	68.6	$15.700 \pm 0.37\%$	$42.28 \pm 0.52\%$	22.20
44000	1136.59	0	$1792 \times 270 \times 1792$	1136.59	0.0	$21.352 \pm 0.10\%$	$0.00 \pm 0.00\%$	25.86
44000	1136.59	0.25	$1792 \times 164 \times 1792$	1087.91	4.8	$19.562 \pm 0.14\%$	$8.38 \pm 1.88\%$	24.78
44000	1136.59	0.5	$1792 \times 164 \times 1792$	1013.63	9.7	$16.982 \pm 0.08\%$	$20.47 \pm 0.50\%$	34.44
44000	1136.59	1.0	$1792 \times 164 \times 1792$	892.25	19.4	$13.158 \pm 0.14\%$	$38.37 \pm 0.28\%$	24.35
44000	1136.59	2.0	$1792 \times 164 \times 1792$	808.56	38.7	$10.806 \pm 0.49\%$	$49.39 \pm 0.51\%$	13.51
44000	1136.59	4.0	$1792 \times 164 \times 1792$	755.78	77.4	$9.441 \pm 0.54\%$	$55.78 \pm 0.44\%$	29.90
44000*	1136.59	4.0	$1792 \times 164 \times 3584$	767.70	77.4	$9.741 \pm 0.26\%$	$54.38 \pm 0.23\%$	25.49
44000**	1136.59	4.0	$1792 \times 328 \times 1792$	742.94	77.4	$9.123 \pm 0.00\%$	$54.38 \pm 0.00\%$	50.60
82500	1979.32	0	$3072 \times 399 \times 3072$	1979.32	0.0	$18.420 \pm 0.18\%$	$0.00 \pm 0.00\%$	18.62
82500	1979.32	0.25	$3072 \times 243 \times 3072$	1861.11	5.2	$16.285 \pm 0.14\%$	$8.38 \pm 1.74\%$	24.45
82500	1979.32	0.5	$3072 \times 243 \times 3072$	1723.57	10.4	$13.967 \pm 0.25\%$	$20.47 \pm 0.97\%$	23.67
82500	1979.32	1.0	$3072 \times 243 \times 3072$	1498.29	20.8	$10.554 \pm 0.42\%$	$38.37 \pm 0.61\%$	28.33
82500	1979.32	2.0	$3072 \times 243 \times 3072$	1316.85	41.7	$8.153 \pm 1.10\%$	$55.74 \pm 0.89\%$	19.32
82500	1979.32	4.0	$3072 \times 243 \times 3072$	1221.82	83.4	$7.019 \pm 1.42\%$	$55.78 \pm 0.88\%$	9.39
133000	3020.16	0	$4608 \times 327 \times 4608$	3020.16	0.0	$16.501 \pm 0.26\%$	$0.00 \pm 0.00\%$	6.97
133000	3020.16	0.25	$4608 \times 327 \times 4608$	2820.43	5.5	$14.390 \pm 0.37\%$	$13.44 \pm 3.08\%$	7.12
133000	3020.16	0.5	$4608 \times 327 \times 4608$	2586.30	11.0	$12.100 \pm 0.27\%$	$27.21 \pm 1.03\%$	13.51
133000	3020.16	1.0	$4608 \times 327 \times 4608$	2275.20	22.0	$9.364 \pm 0.54\%$	$43.67 \pm 0.79\%$	5.97
133000	3020.16	2.0	$4608 \times 327 \times 4608$	1882.89	44.0	$6.413 \pm 0.30\%$	$61.42 \pm 0.25\%$	6.46
133000	3020.16	4.0	$4608 \times 327 \times 4608$	1699.39	88.1	$5.224 \pm 0.73\%$	$68.57 \pm 0.36\%$	5.66

2.3 Results

2.3.1 Drag reduction

Before delving into the analysis of drag reduction, we compare our results with previous studies. Fig. 2.1 displays the friction factor results. For the non-rotating cases, the DNS results exhibit minor deviations from Prandtl's friction law [108]. For the rotating cases, the DNS results for $Re_{\tau,0} = 180$ and $Re_{\tau,0} = 495$ agree well with the DNS data of Davis *et al.* [24]. However, all DNS results exhibit large discrepancies from the experimental data of Kikuyama *et al.* [65] at high rotation number, even accounting for differences in the Reynolds number ($Re_{\tau,0} = 240$ and 570 for those authors). Orlandi and Fatica [99] attributed discrepancies to the influence of the entrance conditions, and possible difficulties in achieving a constant pressure gradient in the experimental setup. We note that differences could also be related to limited pipe length, which might prevent the achievement of a fully developed state in spatially developing flows. Indeed, as shown in Section 2.4 (see Fig. 2.11), the time interval needed to achieve a fully developed state is proportional to N , for given Re_b . The drag reduction coefficient DR is here defined as

$$DR = 1 - \frac{C_f}{C_{f,0}} = 1 - \frac{\lambda}{\lambda_0}, \quad (2.10)$$

where $\lambda = 4C_f$ is the friction factor, and again the subscript 0 indicates non-rotating cases. The average power expenditure to sustain wall rotation is

$$P = 2\pi\Omega LR^2 \tau_{r\theta}|_{r=R} = 2\pi\mu\Omega LR^3 \left. \frac{d}{dr} \left(\frac{V_\theta}{r} \right) \right|_{r=R}, \quad (2.11)$$

where $\tau_{r\theta}$ is the tangential viscous shear stress, and $V_\theta = U_\theta + \Omega r$ is the mean azimuthal velocity in the inertial frame of reference. As formally shown below, mean momentum balance under the assumption of statistically steady flow implies that $d(V_\theta/r)/dr$ is zero at the wall, i.e., there is no mean azimuthal friction. Hence, whereas energy must be spent to maintain the mass flow rate and to rotate the pipe during the initial transient, no energy must be spent to sustain wall rotation once statistically steady conditions are achieved, let alone mechanical losses in the actuation system. As a consequence, the drag reduction coefficient (2.10) is identical to the net power saving, which establishes an important difference and advantage from other types of wall manipulation requiring additional actuation energy [116].

Fig. 2.1(c) shows the variation of the drag reduction coefficient with the rotation number at constant values of Re_b (or $Re_{\tau,0}$). Pipe rotation consistently leads to drag reduction, and monotonic increase with both N and $Re_{\tau,0}$ is observed, in agreement with previous studies [24]. At $N = 4.0$ and $Re_{\tau,0} = 3020$, drag reduction is as high as 69%, which clearly highlights the potential of rotation for curtailing energy consumption in high- Re internal flows, such as fluid transportation in large-diameter pipelines, whose typical Reynolds number is $Re_{\tau,0} = 10^5 - 10^6$ [58]. This is another important difference from other passive and active drag reduction strategies, for which the drag reduction effect typically decreases with the Reynolds number [42, 116]. We note that full relaminarization would result in a drag reduction of about 97% at $Re_{\tau,0} = 3020$, so at even $N = 4.0$, the flow remains distant from the laminar state.

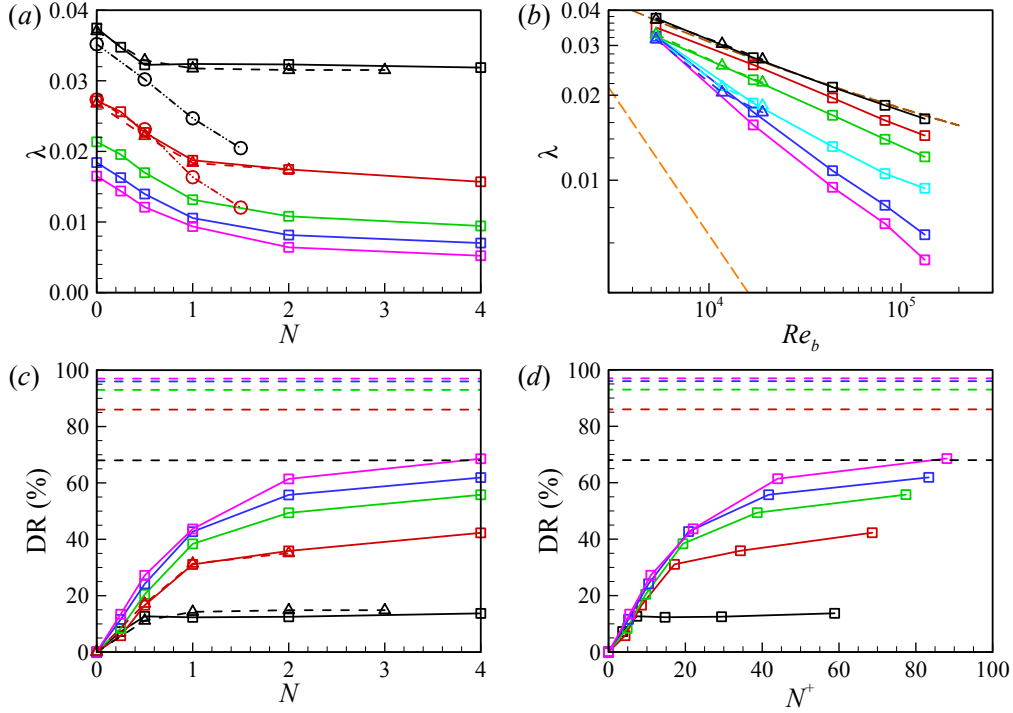


Figure 2.1. Global flow properties: (a) friction factor as a function of N ; (b) friction factor as a function of Re_b ; (c) drag reduction coefficient as a function of N ; (d) drag reduction coefficient as a function of N^+ . In (a), (c) and (d), we show data for $Re_{\tau,0} = 180$ (black), 495 (red), 1137 (green), 1979 (blue) and 3020 (pink); in (b) we show $N = 0$ (black), 0.25 (red), 0.5 (green), 1.0 (cyan), 2.0 (blue) and 4.0 (pink). In (b), the brown dashed line denotes Prandtl's friction law [108], and the orange dashed line denotes the laminar friction law $\lambda = 64/Re_b$. In (c) and (d), the horizontal dashed lines denote the laminar limits of drag reduction for various Reynolds numbers. The triangle symbols denote DNS data at $Re_{\tau,0} \approx 180$ and 540 [24], and the circles denote experiments at $Re_{\tau,0} \approx 240$ and 570 [65].

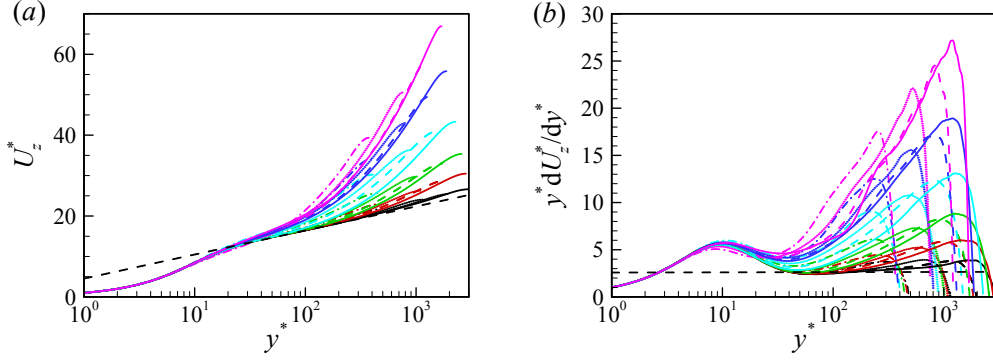


Figure 2.2. Inner-scaled mean axial velocity profiles (a), and corresponding logarithmic diagnostic functions (b). The color codes correspond to the same values of the rotation number: $N = 0$ (black), $N = 0.25$ (red), $N = 0.5$ (green), $N = 1$ (cyan), $N = 2$ (blue), $N = 4$ (pink). The line patterns correspond to the same values of the nominal Reynolds number: $Re_{\tau,0} = 495$ (dash-dotted), $Re_{\tau,0} = 1137$ (dotted), $Re_{\tau,0} = 1979$ (dashed), $Re_{\tau,0} = 3020$ (solid). The dashed black lines in panel (a) denote the expected logarithmic distribution given in Eq. (2.12), and in panel (b) the inverse of the Karman constant ($1/\kappa$.)

Fig. 2.1(d) displays the drag reduction coefficient as a function of the inner-scaled wall rotational speed based on the non-rotating friction velocity, $N^+ = \Omega R/u_{\tau,0}$, which again confirms increased drag reduction with both N^+ and $Re_{\tau,0}$. Notably, the drag reduction profiles tend to be much more universal when expressed as a function of N^+ , with deviations from a common trend occurring at higher and higher N^+ as $Re_{\tau,0}$ increases. This suggests that the proper parameter to quantify drag reduction effects could be the ratio of the pipe peripheral velocity to the friction velocity, which is the typical scale for wall turbulence, in line with what is found in drag reduction studies based on the use of oscillating walls and traveling waves [111, 132].

To analyze the flow modifications yielding drag reduction in greater detail, in Fig. 2.2(a) we display the inner-scaled mean axial velocity profiles at various rotation and Reynolds numbers. At zero and small rotation numbers, the mean axial velocity follows with good accuracy the logarithmic distribution observed in pipe flow [108],

$$U_z^* = \frac{1}{\kappa} \log y^* + B, \quad (2.12)$$

with $\kappa = 0.387$ and $B = 4.53$. Deviations from such universal behaviour occur farther and farther from the wall as the Reynolds number increases. However, deviations tend to occur earlier as N increases, and at the same time, the wake is found to grow stronger. At $N \gtrsim 1$, the whole logarithmic layer is eventually disrupted, consistent with the findings of Orlandi and Fatica [99], Davis *et al.* [24]. This process is clearer when the logarithmic diagnostic function, namely $y^* dU_z^*/dy^*$, is considered, as shown in Fig. 2.2(b). Indeed, no plateau of this indicator is found at $N \gtrsim 1$, although one can speculate that a logarithmic behaviour is recovered at higher Reynolds numbers than we consider here.

The mean axial velocity profiles are then shown in defect form in Fig. 2.3. As known from previous studies [108], normalization by the friction velocity (panel

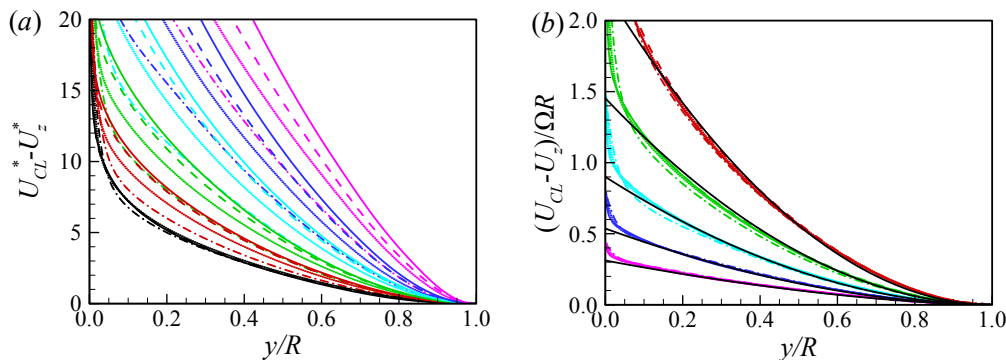


Figure 2.3. Mean axial velocity defect profiles (U_{CL} is the pipe centreline velocity), normalized by the friction velocity, u_τ (a), and by the pipe rotational velocity, ΩR (b). The colour codes correspond to the same values of rotation number: $N = 0$ (black), $N = 0.25$ (red), $N = 0.5$ (green), $N = 1$ (cyan), $N = 2$ (green), $N = 4$ (pink). The line patterns correspond to the same values of the nominal Reynolds number: $Re_{\tau,0} = 495$ (dash-dotted), $Re_{\tau,0} = 1137$ (dotted), $Re_{\tau,0} = 1979$ (dashed), $Re_{\tau,0} = 3020$ (solid). The solid black lines in panel (b) indicate the power-law fitting function (2.13), with coefficients given in Eq. (2.14).

(a)) yields excellent universality of the profiles in the absence of rotation ($N = 0$). However, when rotation is introduced, a large scatter is observed when either the rotation number or the Reynolds number varies. Indeed, as noticed by Oberlack [96], the pipe rotational speed is an independent velocity scale, and it is a better candidate to achieve universality of the mean axial velocity profiles in the presence of pipe rotation. Hence, in panel (b) we show the velocity profiles normalized by the peripheral rotational speed. Excellent universality of the axial velocity profiles is then observed for any given (non-zero) rotation number, however with obvious dependence on N . Of course, this kind of normalization cannot apply to the non-rotating case as $\Omega = 0$. Based on these empirical observations, we then assume the following form for the defect velocity profiles,

$$\frac{U_{CL} - U_z}{\Omega R} = \frac{1}{N} \varphi(N) \left(\frac{r}{R} \right)^{\alpha(N)}, \quad (2.13)$$

where fitting of the DNS data in the range $0 \leq r/R \leq 0.9$ yields

$$\varphi(N) = 0.90 + 0.25 \log N, \quad \alpha(N) = 2.0 - 0.071 N^{1.2}. \quad (2.14)$$

It should be noted that previous experiments [65] and DNS [99] also suggested values of the power-law exponent $\alpha \approx 2$, however, based on a much more limited set of data.

The previous observations can be leveraged to obtain predictions for the friction coefficient. Indeed, integration of the defect profile (2.13) yields the following relationship between the centreline and bulk velocity

$$U_{CL}^* = u_b^* \left(1 + \frac{2}{2 + \alpha} \varphi \right). \quad (2.15)$$

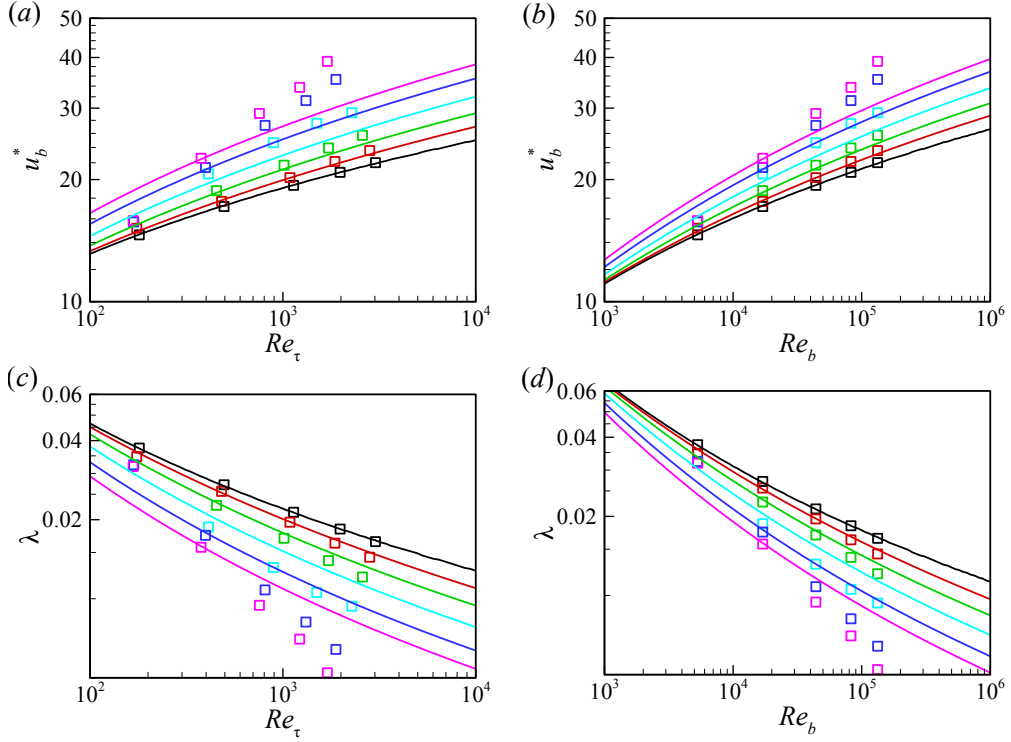


Figure 2.4. Inner-scaled bulk velocity (a,b) and friction factor (c,d), as a function of the friction Reynolds number (a,c) and bulk Reynolds number (b,d). The colour codes correspond to the same values of rotation number: $N = 0$ (black), $N = 0.25$ (red), $N = 0.5$ (green), $N = 1$ (cyan), $N = 2$ (blue), $N = 4$ (pink). Symbols denote the DNS data, and lines the corresponding predictions of Eq. (2.18). The solid black line denotes the predictions of Prandtl's friction law for a non-rotating pipe [108].

Matching the wall-normal gradients of the inner profile (2.12) with the defect profile (2.13) yields the condition

$$\frac{dU_z^*}{d\eta} = \frac{1}{\kappa\eta} = \alpha\varphi u_b^* (1 - \eta)^{\alpha-1}, \quad (2.16)$$

where $\eta = y/R$. This condition can be numerically solved to determine the outer-scaled matching location η_0 . Under the assumption $\eta_0 \ll 1$, the approximate solution holds

$$\eta_0 \approx \frac{1}{\kappa\alpha\varphi u_b^*}. \quad (2.17)$$

Finally, matching the pointwise values of Eqs. (2.12) and (2.13) at $\eta = \eta_0$ yields

$$\frac{1}{\kappa} \log \eta_0 + \frac{1}{\kappa} \log Re_\tau + B = u_b^* \left(1 + \frac{2\varphi}{2 + \alpha} - \varphi (1 - \eta_0)^\alpha \right). \quad (2.18)$$

Equation (2.18) can be numerically solved to obtain u_b^* for a given Re_b (as $Re_\tau = Re_b/(2u_b^*)$) and N , and in turn obtain the friction factor, $\lambda = 8/u_b^{*2}$.

The charts of u_b^* and f as a function of the Reynolds number are shown in Fig. 2.4, where symbols denote the DNS data, and lines represent the corresponding

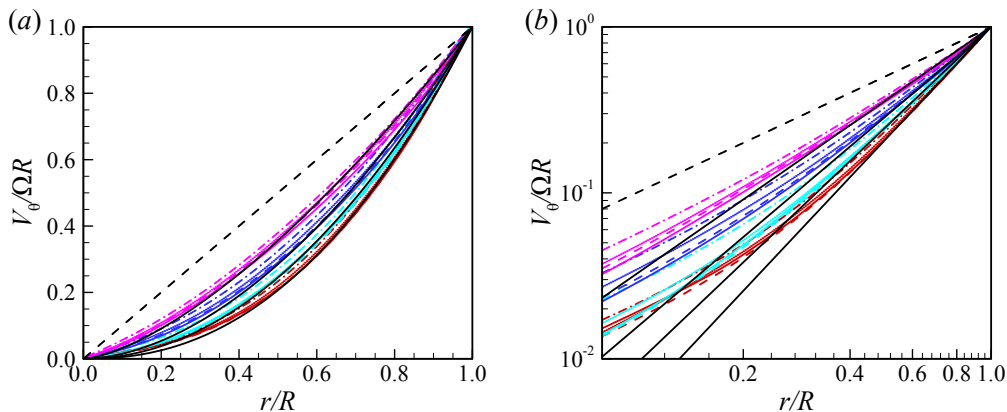


Figure 2.5. Radial profiles of mean azimuthal velocity in linear (a) and logarithmic (b) scale. The colour codes correspond to the same values of rotation number: $N = 0.25$ (red), 1.0 (cyan), 2.0 (blue) and 4.0 (pink). The line patterns correspond to the same values of the nominal Reynolds number: $Re_{\tau,0} = 495$ (dash-dotted), $Re_{\tau,0} = 1137$ (dotted), $Re_{\tau,0} = 1979$ (dashed), $Re_{\tau,0} = 3020$ (solid). The solid black lines denote Eq. (2.21), with the coefficient given in Eq. (2.22). The black dashed lines correspond to the case of solid body rotation.

predictions of Eq. (2.18). The latter formula indeed captures the correct Reynolds number trends including the progressive departure of the friction curves from the non-rotating case, and it yields good quantitative predictions at low-to-moderate rotation numbers. However, the drag reduction effect becomes noticeably underestimated at $N \gtrsim 1$, the reason being the previously noted breakdown of the logarithmic profile (2.12) at high rotation numbers, which we used for theoretical inference.

In Fig. 2.5, we further present the distributions of the absolute mean velocity in the azimuthal direction. Before analyzing its scaling laws, it is informative to consider the momentum balance equation projected along the θ direction,

$$\nu \frac{d}{dr} \left(\frac{1}{r} \frac{d}{dr} (r V_\theta) \right) = \frac{1}{r^2} \frac{d}{dr} \left(r^2 \langle u_r u_\theta \rangle \right). \quad (2.19)$$

Multiplying Eq. (3.2) by r^2 , then integrating from 0 to r yields

$$\nu r \frac{d}{dr} \left(\frac{V_\theta}{r} \right) = \langle u_r u_\theta \rangle, \quad (2.20)$$

which highlights the balance between the viscous and the turbulent shear stress. At the wall, $\langle u_r u_\theta \rangle = 0$, resulting in $V_\theta \overset{r \rightarrow R}{\sim} \Omega r$. As one moves away from the wall, the presence of a nonlinear distribution of V_θ becomes evident, due to positive values of $\langle u_r u_\theta \rangle$. Oberlack [96] suggested that, for $r/R \gtrsim 0.2$, the mean azimuthal velocity should follow a power-law variation, namely

$$\frac{V_\theta}{\Omega R} = \left(\frac{r}{R} \right)^\psi, \quad (2.21)$$

with $\psi = 2.0$, as inferred from the set of data available at that time, which according to Eq. (2.20) would result in linear variation of the turbulent shear stress $\langle u_r u_\theta \rangle$,

with slope proportional to N . However, Orlandi and Fatica [99] reported that the slope of $\langle u_r u_\theta \rangle$ is not strictly proportional to N , even in the core region, thereby preventing perfect collapse of the profiles in Fig. 2.5. In our simulations, $V_\theta/\Omega R$ is found to gradually increase with N , exhibiting a similar trend as other DNS [99, 24, 33] and experimental studies [65]. We have checked the influence of pipe length on the azimuthal velocity profile, and confirmed that using $L = 30R$ yields almost the same results as using $L = 15R$ (see Fig. 2.10(b)). It is worth mentioning that the experimental data of Reich and Beer [115] for $Re_b = 5000$ with $N = 1.0$ and 5.0 are in good agreement with the quadratic law, however, they are likely affected by entrance effects, as pointed out by Orlandi and Fatica [99]. Instead, our DNS data show that the exponent ψ is clearly dependent on the rotation number, and also has weaker dependence on the bulk Reynolds number (panel (a)). As the rotation number increases, the azimuthal velocity profiles become closer to the case of solid body rotation, corresponding to smaller values of ψ . This tendency is regarded to be reasonable, considering that as the rotation number increases, turbulent motions become increasingly suppressed due to the stronger centrifugal instability. The weak influence of the nominal bulk Reynolds number includes slightly higher values of ψ as Re_b increases. This tendency is partly explained in terms of diminished viscous effects near the wall, which shortens the layer with approximately linear variation. Fitting the DNS data in the range $0.2 \leq r/R \leq 0.9$, we get

$$\psi(N) = 2.48 - 0.45N^{0.57}, \quad (2.22)$$

which is quite accurate as shown in Fig. 2.5. At $N \approx 1$, the formula yields $\psi \approx 2.0$, corresponding to a quadratic power law. We would like to note that for $r/R \lesssim 0.2$, the azimuthal velocity profiles observed in the DNS exhibit deviations from the quadratic law and become closer to a linear distribution (see panel (b)), thus indicating that the flow near the pipe axis is close to a solid-body rotation state.

2.3.2 Organisation of turbulence

A perspective view of the instantaneous axial velocity field is provided in Fig. 2.6 for two selected Reynolds numbers, including for reference the case of pipe flow in the absence of rotation [108]. As the Reynolds number increases, finer scales are observed, but the flow in the cross-stream plane is always dominated by a limited number of bulges distributed along the azimuthal direction, which correspond to alternating intrusions of high-speed fluid from the pipe core and ejections of low-speed fluid from the wall. Streaks are clear in the near-wall cylindrical shell, whose pattern has a close association with the turbulence organisation in the cross-stream plane. The R -sized low-speed streaks are linked to the large-scale ejections, and R -sized high-speed streaks are associated with the large-scale inrush from the core flow. Simultaneously, smaller streaks scaled in wall units prevail very close to the wall, which is correlated with buffer-layer ejections and sweeps. Hence, the organisation of the flow at two characteristic length scales is apparent, whose separation increases with the Reynolds number.

When rotation is introduced, a distinctive axial coherence becomes apparent in the core region [154], reminiscent of the columnar structures observed in rotating

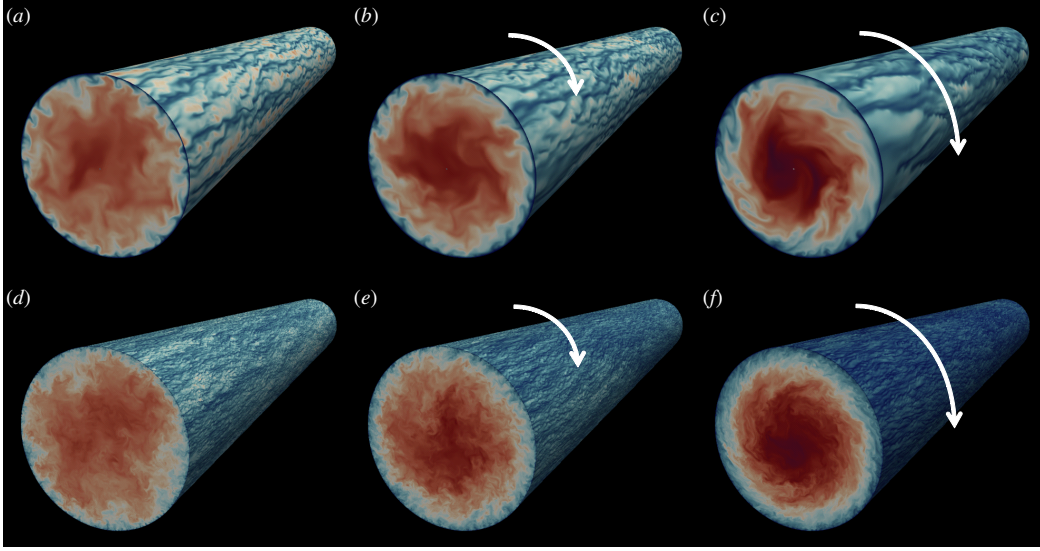


Figure 2.6. Instantaneous fields of axial velocity on a near-wall cylindrical shell and cross-stream plane. The cylindrical shell is located at a wall distance $y^+ \approx 15$. The colour scale is from blue (low speed) to red (high speed). Rotation numbers $N = 0, 0.5$ and 2.0 are reported in panels (a,d), (b,e), (c,f), respectively, for $Re_{\tau,0} = 495$ (a-c) and $Re_{\tau,0} = 3020$ (d-f). The arrows indicate the direction of rotation.

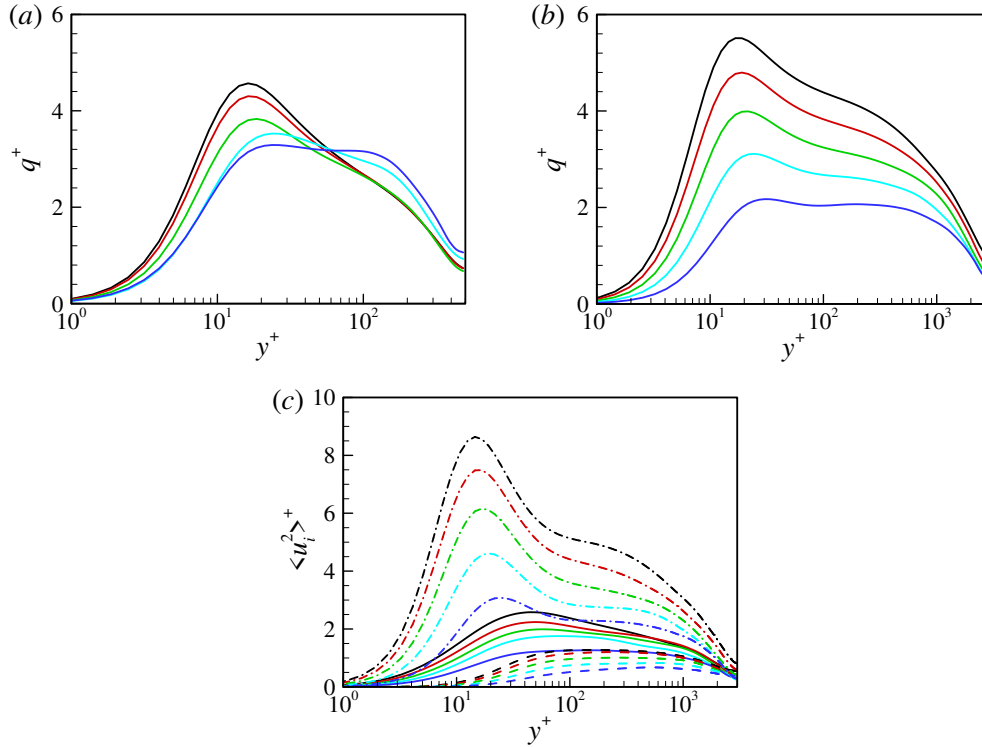


Figure 2.7. Distributions of: (a) turbulence kinetic energy for $Re_{\tau,0} = 495$; (b) turbulence kinetic energy for $Re_{\tau,0} = 3020$; (c) individual velocity variances for $Re_{\tau,0} = 3020$. In (c), lines correspond to: $\langle u_\theta^2 \rangle$ (solid); $\langle u_r^2 \rangle$ (dashed); $\langle u_z^2 \rangle$ (dash-dot). Color codes: $N = 0$ (black), 0.25 (red), 0.5 (green), 1.0 (cyan) and 2.0 (blue).

homogeneous turbulence [56, 46]. Meanwhile, the R -sized structures in the cross-stream plane undergo deformation due to the decreasing angular velocity towards the core. The bulges gradually tend to lag behind the azimuthal motion of the wall as one moves away from it. For the low- Re case, the turbulent fluctuations are intensified in the core region for $N \gtrsim 1.0$, as highlighted by increased turbulence kinetic energy, $q = \langle u_r^2 + u_\theta^2 + u_z^2 \rangle / 2$, in Fig. 2.7(a). Orlandi and Fatica [99] conjectured that such increase is due to a low- Re effect, whereby the fluctuations in the central part of the pipe are amplified by the near-wall enlarged vortical structures. This conjecture is corroborated herein as the phenomenon is much less evident in the high- Re case, as substantiated by decreased turbulence kinetic energy shown in Fig. 2.7(b). It is nevertheless important to highlight that at sufficiently low rotation numbers the fluctuations in the core region become consistently suppressed regardless of the Reynolds number. Turning to the near-wall region, a noticeable reduction in both the number and magnitude of the small-scale streaks is evident at both Reynolds numbers, which points to attenuation of sweeps and ejections in the buffer layer. This causes a distinct decrease in the axial velocity fluctuations compared with the other two components (see Fig. 2.7(c)), which is not unexpected as the production of q is associated with the axial velocity component. Moreover, the Taylor-Proudman theorem [47] implies that the vorticity component normal to the axis of rotation should be suppressed to reduce the axial velocity gradients. Noteworthy is the lack of discernible azimuthal tilting of the small-scale streaks. However, tilting of the footprints of the large-scale structures towards the direction of rotation is evident on the cylindrical shell in the low- Re case.

The modifications in the characteristic length scales can be quantified in terms of the pre-multiplied azimuthal energy spectra of the axial velocity, which we report in Fig. 2.8. In the non-rotating cases, the spectra exhibit a prominent ridge along $\lambda_\theta \sim y$, which should be interpreted as a hierarchy of wall-attached eddies as suggested by Townsend [134]. Here, λ_θ is the wavelength in the azimuthal direction. The inner peak, associated with the near-wall turbulence regeneration cycle [61], and the outer site, linked with R -sized large-scale motions [60], exhibit a more pronounced separation in the high- Re case. Upon imposition of rotation, the most remarkable modification is the progressive attenuation of the amplitude of the inner spectra. Concurrently, an outer-layer peak of $\lambda_\theta/R = 1.08$ emerges at $y/R = 0.22$ for $N = 2.0$ in the low- Re case, which we believe to be linked with the R -sized distorted structures observed in Fig. 2.6(c). In contrast, the outer-layer peak of $\lambda_\theta/R = 1.26$ at $y/R = 0.19$ in the high- Re , non-rotating case vanishes, and a subdominant peak is found instead at $\lambda_\theta/R = \pi$, at a wall distance $y^+ \lesssim 200$. At both Reynolds numbers, the inner peak undergoes a top-right shift, from about $y^+ = 13$ to 19, with a concurrent increase in the typical streaks spacing from about $\lambda_\theta^+ = 120$ to about 200. Consequently, the spectral ridge becomes steeper, taking the form of a power law, with the exponent changing with Re and N .

2.3.3 Contributions to frictional drag

The Fukagata-Iwamoto-Kasagi (FIK) identity [38] is herein used to isolate the contributions of molecular viscosity and turbulence to the overall wall friction, and to elucidate mechanisms contributing to the observed drag reduction. It can be

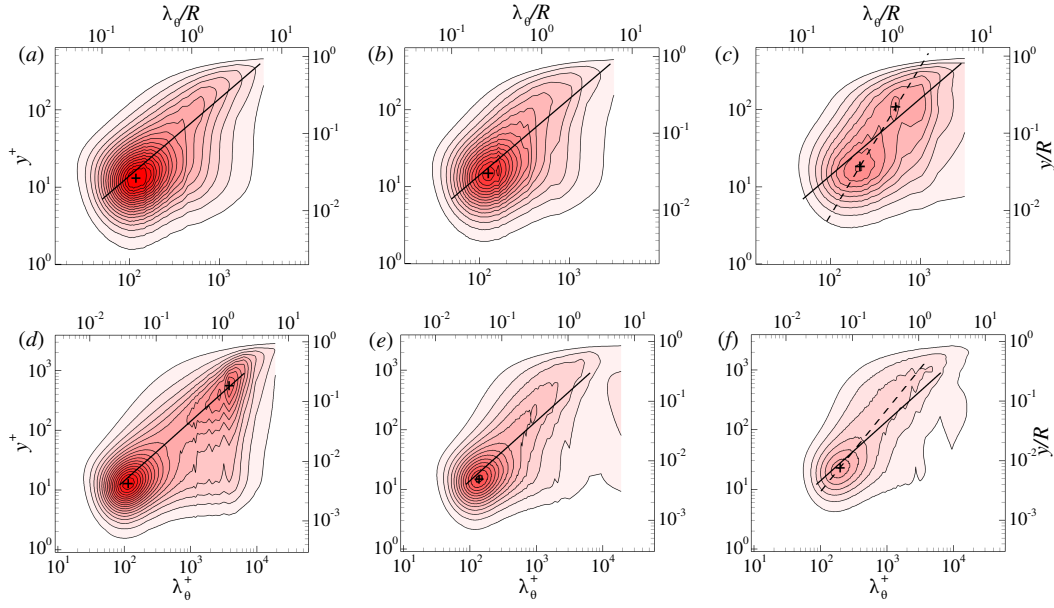


Figure 2.8. Pre-multiplied azimuthal energy spectra of axial velocity $(\kappa_\theta E_{uu})^+$ for $N = 0$ (a, d), $N = 0.5$ (b, e), $N = 2.0$ (c, f), at $Re_{\tau,0} = 495$ (a-c), $Re_{\tau,0} = 3020$ (d-f). The solid lines denote the trend $y = 0.14\lambda_\theta$, and the dashed lines denote the trend $\lambda_\theta/R = 2.35(y/R)^{0.52}$ in panel (c), and $\lambda_\theta/R = 2.20(y/R)^{0.72}$ in panel (f). The crosses denote spectral peaks.

shown that the form of the FIK identity for a rotating pipe is identical to the case of a non-rotating pipe as reported by Fukagata *et al.* [38], because its derivation originates from the momentum equation in the z direction, which does not depend on the imposition of axial rotation. This results in the following expression for the relative drag reduction,

$$DR = 1 - \frac{C_f}{C_{f,0}} = \int_0^{\log Re_{\tau,0}} \left(\frac{y}{R}\right) \left(c_{f_0}^T(y^+) - c_f^T(y^+)\right) d \log y^+, \quad (2.23)$$

where

$$c_f^T(y^+) = 4 \left(\frac{r}{R}\right)^2 \langle u_r u_z \rangle^+, \quad (2.24)$$

denotes the local contribution of turbulence at a given wall distance to the overall friction coefficient. It is noteworthy that the viscous contribution to wall friction ($16/Re_b$) is the same at the various rotation numbers as the DNS were carried out at constant mass flow rate, hence it cancels out from Eq. (2.23). We have made sure that the relative errors between friction coefficients obtained from the direct calculation and the FIK identity are well below 1%.

Fig. 2.9(a) clearly shows that the reduction of the turbulent shear stress is greater at higher Reynolds numbers. Whereas the maximum shear stress slowly increases asymptotically to unity as Re_b increases, the peak decreases in the case of a rotating pipe, despite amplification of turbulence kinetic energy in the core region at high rotation numbers in the low- Re cases (see Fig. 2.7(a)). This can be interpreted better by noticing that the inner-scaled wall rotational speed, $N^* = N/2(Re_b/Re_\tau)$, increases with Re_b , with likely increased suppression of the near-wall turbulence.

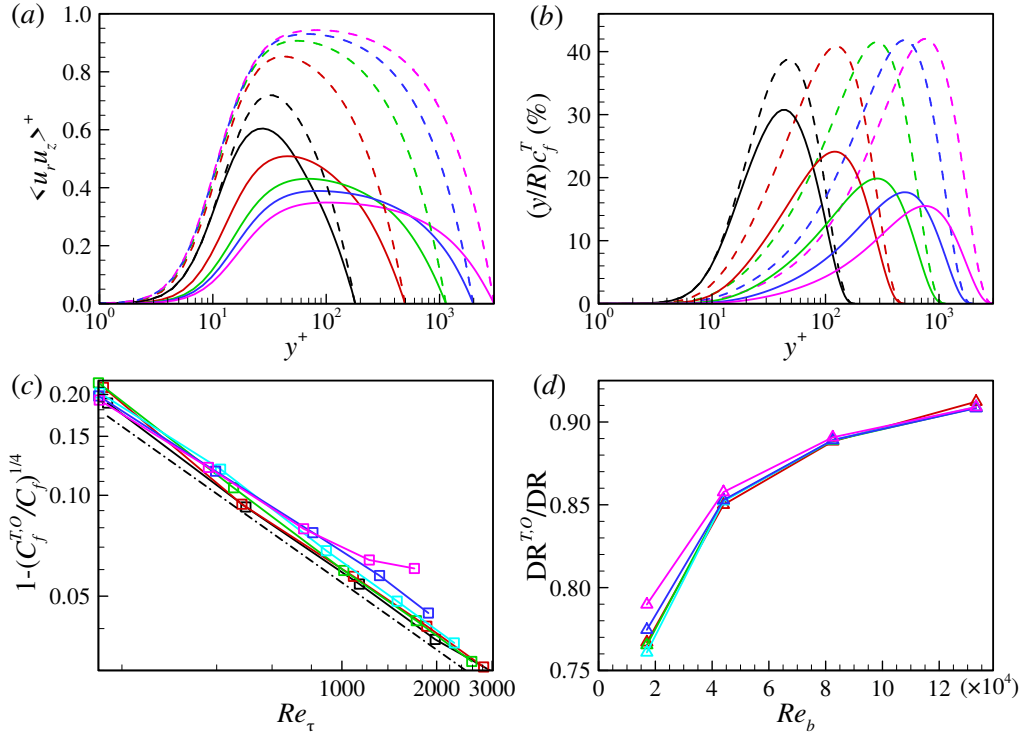


Figure 2.9. (a) Turbulent shear stress for $N = 0$ (dashed lines) and $N = 2$ (solid lines); (b) local contributions to turbulent friction (as from Eq. (2.24)), for $N = 0$ (dashed lines) and $N = 2$ (solid lines); (c) turbulent contribution to friction as a function of Re_{τ} ; (d) fraction of drag reduction due to turbulence as a function of Re_b . In (a) and (b), $Re_{\tau,0} = 180$ (black), 495 (red), 1137 (green), 1979 (blue) and 3020 (pink). In (c) and (d), $N = 0$ (black), 0.25 (red), 0.5 (green), 1.0 (cyan), 2.0 (blue) and 4.0 (pink). In (c), the dash-dotted line indicates the prediction of Eq. (2.26).

Fig. 2.9(b) reports the wall-normal distribution of the local turbulence contribution to the overall friction. Note that the plot is reported in a semi-log scale and in pre-multiplied form, to have a correct perception of the integrated contributions. The figure shows that, at low Reynolds numbers, suppression of the turbulent contribution to friction mainly takes place in the core part of the flow, with a reduction of the absolute peak. At higher Reynolds numbers, a strong decrease of the peak value is still present and increasing, but substantial drag reduction is also coming from the near-wall layer. To have a quantification of the effects of the inner and outer layers, we then split the turbulent wall friction into integrated contributions from the inner layer (say, $C_f^{T,I}$) and from the outer layer (say, $C_f^{T,O}$), based on the position of the peak turbulent shear stress (say, y_m), as suggested by Hurst *et al.* [59]. Accordingly, we define

$$\frac{C_f^{T,O}}{C_{f,0}} = \int_{\log y_m^+}^{\log Re_{\tau,0}} \left(\frac{y}{R} \right) c_f^T(y^+) d \log y^+. \quad (2.25)$$

In non-rotating pipe flow, the turbulent shear stress attains a peak at $y_m^+ \simeq c(Re_\tau/\kappa)^n$ (Chen *et al.* [19] recommended $n = 1/3$ for $Re_\tau \leq 3000$), and farther from the wall it decays linearly, hence $\langle u_r u_z \rangle^+ \sim r/R$. Under these assumptions, Eq. (2.25) yields

$$1 - \left(\frac{C_f^{T,O}}{C_f} \right)^{1/4} \approx 4.0(\kappa Re_\tau^2)^{-1/3}. \quad (2.26)$$

Fig. 2.9(c) shows that Eq. (2.26) also yields a satisfactory approximation in the case of rotating pipes, as long as $N \leq 2$. The formula also well underscores that $C_f^{T,O}$ asymptotically approaches C_f with increasing Re_τ , as a result of the increasing influence of the very large turbulent structures that populate the outer layer and modulate the small near-wall structures [60]. In Eq. (2.26), the splitting location for the rotating cases is based on the respective profiles of $\langle u_r u_z \rangle$, but the asymptotic behaviour remains valid if y_m is determined based on the non-rotating cases, since $y_{m,0} \lesssim y_m$, as inferred from Fig. 2.9(a). In Fig. 2.9(d), we present the ratio of the drag reduction associated with the outer layer to the total drag reduction. This indicator is observed to exceed 90% for $N = 4.0$ and $Re_{\tau,0} = 3020$, tending asymptotically to unity, consistent with theoretical expectations. We also emphasize that it becomes almost independent from N at high enough Reynolds numbers. It is important to note that the FIK identity per se does not imply any direct causality link between its right- and left-hand sides, meaning that the observed reduction of the turbulent shear stress in the outer layer could as well be a consequence of the reduced wall friction, rather than the opposite.

2.4 Numerical issues

The DNS herein reported assume a pipe length $L = 15R$. Fig. 2.10 compares the profiles of axial velocity and azimuthal velocity obtained using $L = 15R$ and $30R$ for the case of $Re_{\tau,0} = 1137$ and $N = 4.0$. The longer configuration predicts a more turbulent mean axial velocity profile. The friction factors for the two lengths have a relative difference of only about 3%, which is much less than the drag reduction amount of about 55% for this case (see Table 2.1). Note that the discrepancies

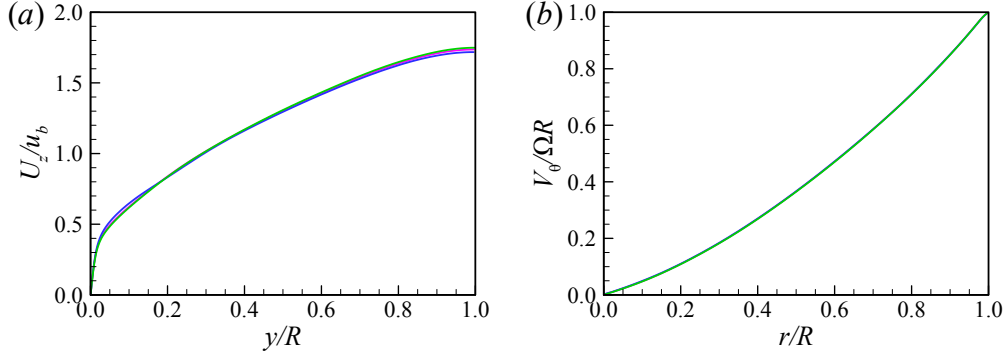


Figure 2.10. Radial profiles for $Re_{\tau,0} = 1137$ and $N = 4.0$: (a) axial velocity; (b) azimuthal velocity. Pink: $L = 15R$ and $N_r = 164$; blue: $L = 30R$ and $N_r = 164$; green: $L = 15R$ and $N_r = 328$.

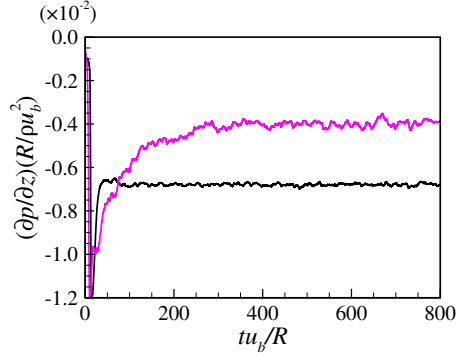


Figure 2.11. Time history of axial pressure gradient for $Re_{\tau,0} = 495$. Black: $N = 0$; pink: $N = 4.0$.

would be smaller with decreasing N due to the weakened long columnar structures in the core region. Fig. 2.10 also displays the results obtained using a grid with a higher resolution in the radial direction. The refined resolution results in a slightly less turbulent profile of the mean axial velocity, and it brings out a difference in the friction factor of only about 3%, which is again much lower than the drag reduction amount. Hence, we conclude that the current numerical settings could be considered enough especially when drag reduction is the primary focus. In our time-evolving simulations, the time interval needed to achieve a fully developed state becomes significantly longer with increasing N at a fixed Reynolds number. Fig. 2.11 shows the time history of the axial pressure gradient for $Re_{\tau,0} = 495$ at $N = 0$ and 4.0, showing that it takes a time interval of about $100R/u_b$ for the non-rotating case to become fully developed, whereas the rotating case needs about $400R/u_b$.

2.5 Conclusions

Direct numerical simulations have been performed to study axially rotating pipe flow up to nominal friction Reynolds number $Re_{\tau,0} = 3020$. The drag reduction rate, which we show to be equivalent to net power saving in the fully-developed scenario

assuming no mechanical losses, increases with either N and $Re_{\tau,0}$, becoming a sole function of the inner-scaled wall rotational speed at high enough Reynolds numbers. The drag reduction is as high as about 70% at $N = 4.0$ for $Re_{\tau,0} = 3020$, although the flow is still far from full relaminarisation. We have developed a theoretical analysis informed with DNS data which allows for quantitative prediction of this effect, based on the observation that the rotational speed becomes the relevant velocity scale in the core flow. The analysis yields a predictive formula for the friction coefficient (2.18), which yields accurate approximation of the DNS data up to $N \approx 1$, regardless of the bulk Reynolds number, which however breaks down once the logarithmic layer in the mean axial velocity profile is disrupted.

Analysis of the instantaneous velocity fields reveals the role of rotation in weakening the near-wall sweeps and ejections and in the elongation and broadening of the streaks. In the core region, rotation leads to turbulence suppression at moderate rotation numbers and high Reynolds numbers, resulting in the disappearance of the outer-layer peak in the pre-multiplied spectra. Turbulence is instead intensified at moderate rotation numbers and low Reynolds numbers, and a distinct outer-layer peak or plateau, emerges in the pre-multiplied spectra. Nevertheless, a consistent decrease of the turbulent shear stress is observed. Use of the FIK identity reveals that the turbulent drag reduction originating from the outer layer asymptotically approaches the total turbulent drag reduction with increasing $Re_{\tau,0}$, consistent with a theoretically derived formula, and conveys that both the inner and outer layers increasingly contribute to drag reduction as N increases.

It is finally important to acknowledge opportunities and challenges in applying wall rotation in practical contexts. On the positive side, it is clear from this paper that large drag reduction is possible without reverting to complicated wall actuation rules. In particular, we note that at a statistically steady state, the work required to sustain rotation is zero (leaving mechanical losses aside), and drag reduction effects increase with the Reynolds number, unlike in conventional wall actuation techniques. On the negative side, setting a full pipeline into rotation may not be an easy task. In addition, we find that the achievement of a fully developed state requires a longer distance than for non-rotating flows (typically, a few hundred diameters), and energy must be spent during this transient state to put the whole system into rotation. This energy expenditure should be accounted for in the evaluation of axial rotation for realistic applicative scenarios.

Chapter 3

GPU-accelerated large-eddy simulation solver

We introduce CaLES, a GPU-accelerated solver for large-eddy simulation of incompressible wall-bounded flows. We first present the governing equations, subgrid-scale models, and numerical methods (Section 3.1), followed by the implementation details and performance analysis (Section 3.2). The LES capabilities of the solver are validated through simulations of decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow (Section 3.3). Finally, we summarize the key findings (Section 3.4).

3.1 Methodology

3.1.1 Governing equations and subgrid-scale models

The filtered incompressible Navier-Stokes (NS) equations read as

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0, \quad (3.1)$$

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial \bar{u}_i \bar{u}_j}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{u}_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}}{\partial x_j}, \quad (3.2)$$

where $\bar{u}_i$ represents the filtered velocity, ρ is the fluid density, $\bar{p}$ is the filtered pressure, and ν denotes the kinematic viscosity. The term $\tau_{ij} = \overline{u_i u_j} - \bar{u}_i \bar{u}_j$ is the subgrid-scale stress tensor, which encapsulates the effects of unresolved scales and requires a suitable closure model. The isotropic component of the SGS stress is typically absorbed into the pressure, resulting in a modified pressure field, $\bar{p} \leftarrow \bar{p} + \frac{1}{3} \rho \tau_{kk}$. The remaining deviatoric part is modeled, according to the Boussinesq assumption, as

$$\tau_{ij} - \frac{1}{3} \tau_{kk} \delta_{ij} = -2\nu_t \bar{S}_{ij}, \quad (3.3)$$

where δ_{ij} denotes the Kronecker delta, and $\bar{S}_{ij}$ is the filtered strain-rate tensor. We implemented two representative closure models: the classical Smagorinsky model [126] and its dynamic version [43, 78]. The baseline Smagorinsky model reads as [126]

$$\nu_t = (C_s \Delta D(y))^2 \bar{S}, \quad (3.4)$$

where C_s is a model constant, Δ is the filter width, $D(y)$ is the near-wall damping function, and $\bar{S}$ is the rate-of-strain, i.e., $\bar{S} = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$. The van Driest damping function [136] is commonly used in the presence of no-slip walls, i.e., $D(y) = 1 - \exp(-y^*/25)$, where “*” denotes the wall distance non-dimensionalized by the wall units. A drawback of the standard Smagorinsky model is its inaccurate prediction of the eddy dissipation in laminar and transitional flows, leading to erroneous wall shear stresses and delayed transition to turbulence. Moreover, the optimal value of the model constant C_s depends significantly on the flow features. Lilly [79] showed that for isotropic turbulence, with spatial resolution within the inertial subrange, $C_s \approx 0.17$, whereas Deardorff [26] suggested $C_s \approx 0.1$ for wall-bounded turbulent shear flows. The Smagorinsky model can be improved using a dynamic procedure, whereby the model coefficient is evaluated dynamically by comparing the eddy dissipation at two filter levels. The dynamic Smagorinsky model with Lilly’s modification [43, 78] is expressed as:

$$\nu_t = c_s \Delta^2 \bar{S}, \quad (3.5)$$

where

$$c_s = \frac{\langle M_{ij} L_{ij} \rangle}{\langle M_{ij} M_{ij} \rangle}, \quad (3.6)$$

$$L_{ij} = \widetilde{\widetilde{u_i u_j}} - \widetilde{\widetilde{u_i}} \widetilde{\widetilde{u_j}}, \quad (3.7)$$

$$M_{ij} = 2\Delta^2 \widetilde{\widetilde{S_{ij}}} - 2(\alpha\Delta)^2 \widetilde{\widetilde{S_{ij}}}. \quad (3.8)$$

Here, the bar denotes filtering with filter width Δ , the tilde indicates test filtering with filter width $\alpha\Delta$, and the brackets $\langle \rangle$ represent an averaging operation. The ratio of the two filter widths is commonly set to $\alpha = 2.0$. The test-filtered strain rate is computed as

$$\widetilde{\widetilde{S_{ij}}} = \frac{1}{2} \left(\frac{\partial \widetilde{\widetilde{u_i}}}{\partial x_j} + \frac{\partial \widetilde{\widetilde{u_j}}}{\partial x_i} \right). \quad (3.9)$$

The dynamic Smagorinsky model provides reasonable subgrid-scale dissipation and automatically switches off in laminar flows. However, it demands more memory and computational resources than the static version and often requires averaging or clipping to ensure numerical stability [78].

Wall models are required when the near-wall mesh resolution is too coarse to resolve the inner-layer turbulent scales up to the inertial subrange. These models are typically classified into near-wall RANS models [93] and wall-stress models [63]. In classical wall-stress models, the wall-parallel velocity at a specific wall distance is used as input and the wall shear stress is estimated as output, which replaces the no-slip boundary condition. The simplest wall-stress model is the logarithmic wall law:

$$\frac{U_{wm}}{u_\tau} = \frac{1}{\kappa} \ln \left(\frac{u_\tau h_{wm}}{\nu} \right) + B, \quad (3.10)$$

where $\kappa = 0.41$, $B = 5.2$, u_τ is the friction velocity, h_{wm} is the wall-modeled layer thickness, and U_{wm} is the wall-parallel velocity magnitude at the top of the wall-modeled layer. At first sight, the logarithmic wall model may appear to be accurate

only for equilibrium flows, but both experience and basic near-wall turbulence physics suggest that this is not generally the case [75]. A high-quality WMLES will resolve the dynamically important eddies in the outer layer, and should therefore accurately capture non-equilibrium effects there. The turbulence time scale decreases towards the wall in the inner layer, and thus the turbulence there may well exist in some type of quasi-equilibrium even in a non-equilibrium outer flow. The literature contains some evidence supporting this, especially when focusing on studies in which the WMLES grid and wall-model exchange location were carefully designed to limit those errors. For example, Bermejo-Moreno *et al.* [12] conducted WMLES using an equilibrium wall model to study interactions between oblique shock waves and turbulent boundary layers in a nearly square duct. Despite having secondary corner flows and shock-induced three-dimensional separation bubbles, their results showed favorable agreement with experimental data.

3.1.2 Numerical methods

The filtered incompressible Navier-Stokes equations are solved using a fractional-step method [66]. Time integration is performed with a low-storage, three-step Runge-Kutta scheme [156], while spatial discretization is handled using a second-order finite-difference method on staggered grids [52]. The continuity equation (3.1) and the momentum equation (3.2) are coupled through a pressure-correction method [4]. This section provides an overview of the numerical schemes, with a focus on key aspects of the implementation of SGS and wall models. For additional details, refer to the descriptions of CaNS [22].

At each sub-step, the flow field is updated as

$$u_i^* = u_i^k + \Delta t \left[\alpha_{k+1} \left(H_i^k + \nu L_{jj} u_i^k \right) + \beta_{k+1} \left(H_i^{k-1} + \nu L_{jj} u_i^{k-1} \right) - \gamma_{k+1} \partial_i p^{k-1/2} \right], \quad (3.11)$$

$$L_{jj} \phi = \frac{\partial_i u_i^*}{\gamma_{k+1} \Delta t}, \quad (3.12)$$

$$u_i^{k+1} = u_i^* - \gamma_{k+1} \Delta t \partial_i \phi, \quad (3.13)$$

$$p^{k+1/2} = p^{k-1/2} + \phi. \quad (3.14)$$

Here, $k = 0$ corresponds to physical time level n , and $k + 1 = 3$ corresponds to time level $n + 1$. The symbol u_i^* denotes the predicted velocity, and ϕ represents the pressure correction. The Runge-Kutta coefficients are $\alpha_{k+1} = (8/15, 5/12, 3/4)$, $\beta_{k+1} = (0, -17/60, -5/12)$, and $\gamma_{k+1} = \alpha_{k+1} + \beta_{k+1}$. In Eq. (3.11), H_i includes convective and SGS stress terms,

$$H_i = \frac{\partial u_i u_j}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j}, \quad (3.15)$$

and the Laplace operator in the diffusive term is defined as

$$L_{jj} = \frac{\partial^2}{\partial x_j \partial x_j}. \quad (3.16)$$

For low-Reynolds-number flows or very fine grids, it may be desirable to use implicit temporal discretization for the diffusion terms. With Crank-Nicolson time integration, this results in

$$u_i^{**} = u_i^k + \Delta t \left[\alpha_{k+1} H_i^k + \beta_{k+1} H_i^{k-1} + \gamma_{k+1} \left(\nu L_{jj} u_i^k - \partial_i p^{k-1/2} \right) \right], \quad (3.17)$$

$$u_i^* - \gamma_{k+1} \frac{\nu \Delta t}{2} L_{jj} (u_i^*) = u_i^{**} - \gamma_{k+1} \frac{\nu \Delta t}{2} L_{jj} u_i^k, \quad (3.18)$$

$$p^{k+1/2} = p^{k-1/2} + \phi - \gamma_{k+1} \frac{\nu \Delta t}{2} L_{jj} \phi. \quad (3.19)$$

Eqs. (3.17) and (3.18) are intentionally not combined to illustrate that u_i^{**} provides a better approximation of u_i^{k+1} than the sum of the terms on the right-hand side of Eq. (3.18) [23]. Alternatively, implicit treatment of the viscous terms can be performed only in the y direction, resulting in

$$\begin{aligned} u_i^{**} = u_i^k + \Delta t \Big\{ & \alpha_{k+1} \left[H_i^k + \nu (L_{11} + L_{33}) u_i^k \right] \\ & + \beta_{k+1} \left[H_i^{k-1} + \nu (L_{11} + L_{33}) u_i^{k-1} \right] \\ & + \gamma_{k+1} \left(\nu L_{22} u_i^k - \partial_i p^{k-1/2} \right) \Big\}, \end{aligned} \quad (3.20)$$

$$\left(1 - \gamma_{k+1} \frac{\nu \Delta t}{2} L_{22} \right) u_i^* = u_i^{**} - \gamma_{k+1} \frac{\nu \Delta t}{2} L_{22} u_i^k, \quad (3.21)$$

$$p^{k+1/2} = p^{k-1/2} + \phi - \gamma_{k+1} \frac{\nu \Delta t}{2} L_{22} \phi. \quad (3.22)$$

In the solver, z is designated as the coordinate direction along which non-uniform grid spacing can be applied, typically corresponding to the wall-normal direction in a plane channel. However, to maintain consistency with the conventions in the turbulence community, we refer to the wall-normal direction as y throughout this paper. The Poisson equation (3.12), once discretized, is solved using an eigenfunction expansion method [124], which allows for an efficient solution using the Thomas algorithm along the y direction. When implicit treatment of the viscous diffusive terms is applied, the resulting three modified Helmholtz equations (3.18) are solved with the same direct solver used for the Poisson equation.

The SGS stress terms are always handled explicitly, regardless of the time integration scheme used for the viscous terms. This allows the SGS stress terms to be grouped with the convective terms, as shown in Eq. (3.15). The SGS model is evaluated at cell centers, where both the SGS viscosity and the strain-rate tensor are stored. The diagonal components of the strain-rate tensor are directly computed at the cell centers, while the non-diagonal components are first calculated at the cell edge midpoints and then averaged to the cell centers. In the dynamic procedure, two-dimensional (2D) or three-dimensional (3D) box filters can be applied. The implementation of the filters assumes uniform grid spacing for simplicity. The 3D box filter cannot be used in the first off-wall layer, hence a 2D box filter is applied through linear extrapolation of the wall-parallel velocity to the ghost points, followed by the application of a 3D filter. In this layer, we set $\alpha = 4^{1/3}$ in Eq. (3.8), which

mathematically corresponds to a 2D box filter. Our WMLES tests on channel flows indicate that using this value in this first layer yields more accurate results than $\alpha = 8^{1/3}$. The averaging operation in Eq. (3.6) is performed in the homogeneous directions, and the averaged coefficient is clipped to zero when it is negative. The velocity components are averaged to the cell centers before applying the test filter to evaluate L_{ij} in Eq. (3.7).

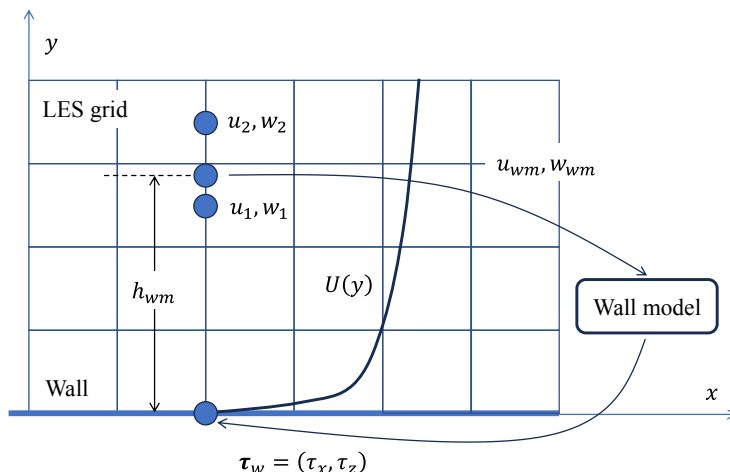


Figure 3.1. Schematic of the wall-model implementation.

For the wall model, the Newton–Raphson iterative method is used to determine the wall shear stress from Eq. (3.10), typically requiring 3 to 7 iterations to achieve convergence in wall shear stress within a relative tolerance of 0.01%. Fig. 3.1 illustrates how the wall model is coupled to the LES solution. The input velocity at the top of the wall-modeled layer is the magnitude of the instantaneous wall-parallel velocity, which is evaluated as $U_{wm} = (u_{wm}^2 + w_{wm}^2)^{1/2}$, where u_{wm} and w_{wm} are the x - and z -direction velocities, respectively. The two components are obtained via linear interpolation in the wall-normal direction between locations 1 and 2. The resulting wall shear stress is then used as the boundary condition for the wall-parallel velocity vector. The impermeability condition is enforced for the wall-normal velocity component. When a staggered grid is used, the two wall-parallel velocity components (e.g., u and w) are stored at different cell face centers. Consequently, when the wall model is used to enforce the boundary condition for u , as illustrated in Fig. 3.1, interpolation of w to the location of u is required to evaluate the wall-parallel velocity vector. Similarly, for w , interpolation of u to the location of w is performed. This approach minimizes the number of interpolations, which is desirable when computing either SGS viscosity or wall models on staggered grids. When utilizing a wall model, the wall-normal derivatives of the wall-parallel velocity components are determined from first-order one-sided finite-difference scheme in the first off-wall layer, thus avoiding crossing the under-resolved layer between the wall and the first off-wall wall-parallel velocity location [63]. In CaLES, one-sided finite differencing is achieved through linear extrapolation of the wall-parallel velocity to the ghost points, followed by second-order central differencing. This procedure is crucial for accurate evaluation of the strain-rate tensor required to evaluate the SGS viscosity. The

viscous terms in the filtered Navier-Stokes equations are evaluated as usual in the first off-wall layer, as the wall shear stress is directly provided by the wall model, and the layer between the first and second off-wall locations of the wall-parallel velocity can be regarded as properly resolved. When the viscous terms are handled implicitly in all three directions, the wall model can only be applied in the y -direction, as homogeneous boundary conditions are required in the other two directions where Fourier transforms are applied. However, this limitation may be irrelevant, as explicit time integration is commonly used for wall-modeled LES due to the large thickness of the first off-wall layer of cells. This restriction does not apply when the viscous terms are handled implicitly only in the y -direction.

3.2 Implementation and performance

3.2.1 Overall implementation strategy

CaLES is developed using CaNS-2.3.0 as its baseline solver. Algorithm 1 outlines the overall solution procedure for explicit time integration. When the viscous terms are handled implicitly, u_i^* is computed using Eqs. (3.17) and (3.18), and $p^{k+1/2}$ is updated from Eq. (3.19). When the viscous terms are handled implicitly only in the y -direction, u_i^* is determined by Eqs. (3.20) and (3.21), while $p^{k+1/2}$ is updated from Eq. (3.22). To ensure consistency with boundary conditions, ghost cells are updated immediately after any variable is updated. Algorithm 2 details the procedure for applying boundary conditions. The calculation of wall shear stress using the wall model must be performed after all other boundary conditions have been applied, as the wall model computation relies on the updated ghost-point values at boundary points.

Algorithm 1 Overall solution procedure.

- 1: Initialize velocity u_i and pressure p .
 - 2: Compute eddy viscosity ν_t .
 - 3: Set iteration counter $n = 0$.
 - 4: **while** $n \leq n_{\max}$ **do**
 - 5: Increment iteration counter: $n \leftarrow n + 1$.
 - 6: Determine time step Δt .
 - 7: **for** $k = 0$ to 2 **do**
 - 8: Compute intermediate velocity u_i^* using Eq. (3.11);
 - 9: Compute the right-hand side of the Poisson equation and solve for ϕ in Eq. (3.12);
 - 10: Update u_i^{k+1} using the correction procedure of Eq. (3.13);
 - 11: Update $p^{k+1/2}$ using Eq. (3.14);
 - 12: Compute ν_t^{k+1} .
 - 13: **end for**
 - 14: **end while**
 - 15: Terminate simulation.
-

The GPU porting was implemented using OpenACC directives. A key factor to achieving high acceleration is to minimize data transfer between GPU and CPU

Algorithm 2 Boundary condition treatment.

- 1: Perform ghost-cell exchange between blocks.
 - 2: Update all boundary conditions except wall-model boundary conditions.
 - 3: Compute wall shear stress using Eq. (3.10).
 - 4: Update wall-model boundary conditions.
-

memory. In the solver, large arrays are transferred only during the initialization or the first time step using unstructured data lifetimes. From the second iteration onward, only scalars or small arrays are transferred between CPU and GPU memory, except when flow-field output is required. According to the OpenACC Programming and Best Practices Guide [1], parallel regions are defined using either the “kernels” construct or the “parallel” construct. Specifically, the “kernels” construct allows the compiler to automatically exploit parallelism in a region of code, while the “parallel” construct, often used in conjunction with the “loop” construct and the “collapse” clause, is applied to accelerate key loops for optimal performance. We use the “kernels” construct for simple array assignments, and the “parallel” construct to speed up loops for the computation of advective, viscous and SGS stress terms, as well as for time advancement. Additionally, the “async” clause is applied to the kernels, parallel, update, and data directives (both structured and unstructured), enabling the CPU to continue with other tasks while the accelerator performs operations, without waiting for their completion. The FFTs required by the Poisson/Helmholtz solver are carried out using the cuFFT library from the CUDA Toolkit. For additional details, refer to [23].

In line with CaNS-2.3.0, parallelization of the code is achieved through MPI, with each rank allocated to one GPU. The structured grid block is partitioned into subdomains using a 2D pencil-like decomposition. The pencil axis is recommended to be aligned with the x -direction for optimal efficiency, except when viscous terms are handled implicitly in the y -direction, in which case alignment with the y -axis is preferable. The cuDecomp library [117] manages the transpose operations required for FFT-based transforms and ghost-cell exchanges. The library not only optimizes the pencil domain decomposition layout, but also finds the most efficient communication backends for transposes and ghost-cell exchanges. This process involves runtime testing of different grid decomposition layouts and communication backends to identify the best-performing combination. Notably, transpose operations and ghost-cell exchanges can utilize different communication backends. Supported communication methods include CUDA-aware MPI point-to-point, MPI all-to-all, NVIDIA Collective Communication Library (NCCL), and NVIDIA Shared Memory (NVSHMEM), with different staging strategies. Given that the optimal setup is highly system-dependent, the hardware-adaptive decomposition provided by cuDecomp is crucial for efficient resource utilization [117].

3.2.2 Performance analysis

Assessment of the code performance was conducted on the Booster partition of the Leonardo cluster at CINECA, Italy. Each node of the cluster is equipped with an Intel Xeon Platinum 8358 CPU (2.60 GHz, 32 cores) and four NVIDIA A100 GPUs

(64 GB HBM2e). The intra-node communication bandwidth is 200 GB/s, supported by NVLink 3.0. The inter-node bandwidth is 25 GB/s, facilitated by the DragonFly+ network architecture using NVIDIA Mellanox InfiniBand HDR, giving each GPU an effective communication rate of approximately 6.25 GB/s. The performance comparison is made between a single GPU card and a CPU node. The test case under consideration is flow in a plane channel with two walls in the y direction, and periodic boundary conditions in the other two directions. The grid size is $(N_x, N_y, N_z) = (512, 384, 1440)$, which is approximately the maximum grid size that can fit into GPU memory when the Smagorinsky model with the van Driest damping function is activated and the viscous terms are handled implicitly along the y direction. It is noteworthy that, during runs on a single GPU, data sharing is also performed when handling periodic boundary conditions in the two decomposed directions, with the pencil-axis as the non-decomposed direction. Fig. 3.2 compares the wall-clock time for different methods when the viscous terms are handled explicitly or implicitly in the y direction. The inclusion of the wall model results in negligible computational overhead. However, applying the static Smagorinsky model with the van Driest damping function increases the computational cost by approximately 0.3 ns per step per grid point. When the viscous terms are handled explicitly, the speed-up factors relative to the CPU node are 13 $\times$ for DNS and 19 $\times$ for wall-resolved LES (WRLES). If the viscous terms are handled implicitly in the y direction, the speed-up factors are 12 $\times$ for DNS and 17 $\times$ for WRLES. The greater speed-up of WRLES is due to the increased computational intensity introduced by the subgrid-scale model.

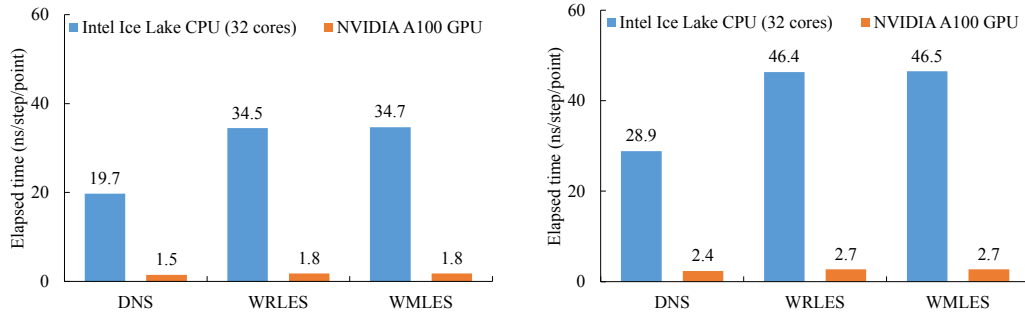


Figure 3.2. Elapsed wall-clock time per time step per grid point for different methods when the viscous terms are handled explicitly (a), or implicitly in the y direction (b).

Fig. 3.3 reports the wall-clock time breakdown for different computational components. The most time is consumed by the Poisson solver and the computation of the right-hand side of the momentum equation. The calculation of eddy viscosity using the Smagorinsky model with the van Driest damping function is the third most time-consuming part. When the viscous terms are handled implicitly in the y direction, the Helmholtz solver is the third most time-consuming component, followed by the eddy viscosity calculation. Table 3.1 presents the estimated memory footprint per grid point for different methods. The incorporation of the Smagorinsky model with the van Driest damping function increases memory usage per grid point by 32 bytes, or approximately 20% of the DNS memory footprint. In contrast, the dynamic Smagorinsky model requires an additional 216 bytes, or about 150% of the

DNS memory footprint. Note that memory requirements for WMLES simulations exactly match those of the corresponding WRLES cases and are therefore omitted from the table.

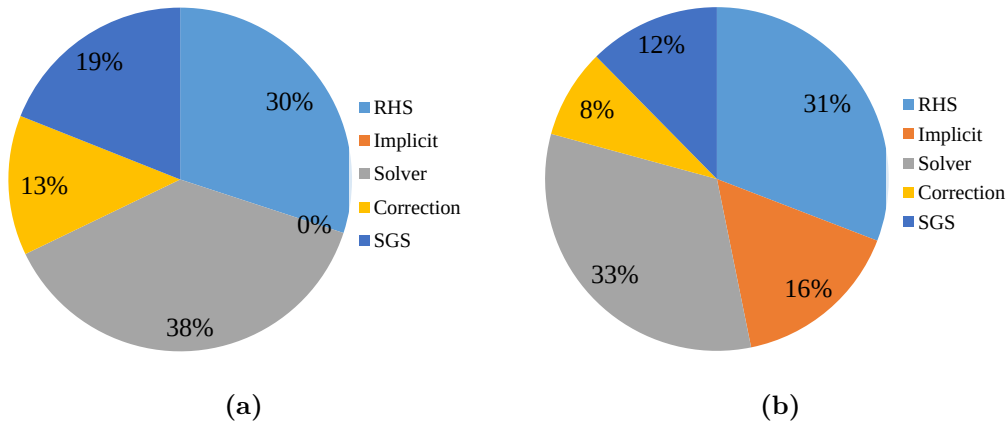


Figure 3.3. Comparison of wall-clock time breakdown for different computational components, when the viscous terms are handled explicitly (a), or implicitly in the y direction (b). “RHS” denotes the time spent calculating the right-hand side of the momentum equation, given the eddy viscosity; “Implicit” indicates the time required to solve the Helmholtz equations when the viscous terms are handled implicitly in the y direction; “Solver” corresponds to the time spent solving the Poisson equation; “Correction” denotes the time allocated for the correction procedure; and “SGS” represents the time required to evaluate the eddy viscosity using the classical Smagorinsky model with the van Driest damping function.

Table 3.1. Estimated memory footprint per grid point for different methods. “SM” denotes the Smagorinsky model with the van Driest damping function, and “DSM” is the dynamic Smagorinsky model.

Method	DNS	WRLES (SM)	WRLES (DSM)
Explicit	136 bytes	168 bytes	352 bytes
Implicit- y	160 bytes	192 bytes	376 bytes

We now proceed to analyze the strong and weak scalability performance of the solver. For the strong scaling experiments, a fixed grid size of $(N_x, N_y, N_z) = (512, 384, 1440 \times 4 \times 2)$ is used. This grid is generated by refining the previous single GPU grid by a factor of 8 along the spanwise z direction, resulting in a size close to the maximum allowable grid on two nodes when the viscous terms are handled implicitly in the y direction with the Smagorinsky model and the van Driest damping function activated. For the weak scaling tests, the grid size was set to $(512, 384, 1440 \times 4 \times N_{nodes})$. We begin the scaling tests from two nodes instead of one, to avoid performance degradation caused by transitioning from fast intra-node communication to slower inter-node communication. The wall-clock time for two nodes, T_2 , is used as the reference for reporting the code scaling performance. The domain decomposition is aligned with the x -axis when the viscous terms are handled explicitly, and with the y -axis when the viscous terms are handled implicitly in the y direction. Fig. 3.4 presents the results for both strong and weak scaling.

In GPU-resident, distributed-memory simulations of turbulent flows, weak scaling is the most critical performance metric. Maximizing GPU occupancy is always desirable, making it essential that the code maintains high performance for a fixed problem size per computational subdomain or MPI task, and consequently per GPU. Fig. 3.4(b) demonstrates that the weak scaling performance across the different computational configurations has small variations. The computational time increases by approximately 80% as the number of nodes increases from 2 to 64. When the viscous terms are handled implicitly in the y direction, there are no data points for $N_{nodes} = 64$ due to GPU memory limitations. The fully implicit scheme is not analyzed here due to its low computational efficiency when Fourier transforms are applied to solve the momentum equations. In such cases, more efficient techniques, such as the alternating-direction-implicit (ADI) scheme [66], may be employed due to its $O(N)$ computational complexity for a single dimension and its additional options for parallel implementation. By contrast, FFT methods have $O(N \log N)$ complexity and typically require extensive data transpositions. However, it should be noted that ADI can still be computationally expensive in a distributed-memory setting, as it requires either transposing the domain or solving a sequence of three tri-diagonal systems in all three directions.

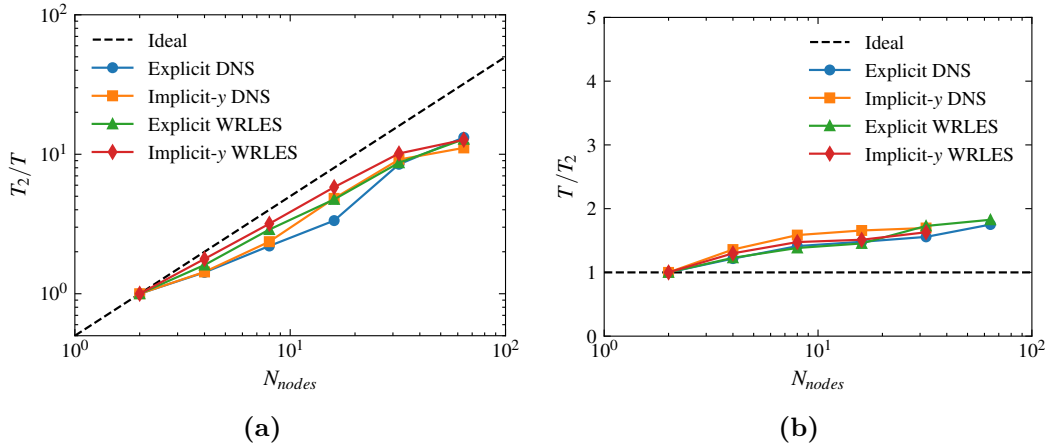


Figure 3.4. Strong (a) and weak (b) code scaling performance. Here, N_{nodes} denotes the number of nodes, T_2 represents the wall-clock time for $N_{nodes} = 2$, and T is the wall-clock time for a given N_{nodes} .

3.3 Validation

The solver is validated using three representative cases: homogeneous decaying isotropic turbulence (DIT), turbulent channel, and turbulent duct flow. The DIT case is employed to validate the SGS models, while the channel case at $Re_b = 20,000$ is to validate the wall-resolved LES capability. The channel flow at $Re_b = 250,000$ and the duct flow are instead employed to validate the wall-modeled LES capability. In these simulations, the viscous terms are handled explicitly, except for the wall-resolved LES of the channel flow, where they are handled implicitly in the y direction. For all simulations, the time step is dynamically adjusted to its maximum allowable

value for numerical stability, multiplied by a safety factor of 0.95.

3.3.1 Decaying isotropic turbulence

We first validate the LES capability using freely decaying isotropic turbulence. The physical experiment was performed by Comte-Bellot and Corrsin [21], with decaying turbulence generated behind a mesh with size $M = 5.08$ cm, and freestream velocity $U_0 = 10$ m/s. The Taylor microscale Reynolds number ($Re_\lambda = u_{rms}\lambda/\nu$, where u_{rms} is the root-mean-square of a fluctuating velocity component, λ is the Taylor microscale, and ν is the kinematic viscosity) is 71.6 at $tU_0/M = 42$, decreasing to 60.6 at $tU_0/M = 171$. In a reference frame moving with the average flow velocity, the problem is modeled as freely decaying isotropic turbulence. We simulate this by considering the flow inside a cubic domain with periodic boundary conditions, where the box edge length is $9 \times 2\pi$ cm ($\approx 11M$). Two grid resolutions are employed, with 32 and 64 cells in each direction, respectively. The corresponding grid spacings are $\Delta/\eta = 60$ and 30, where η is the Kolmogorov length scale at $tU_0/M = 42$. The computations are initialized with a synthetic turbulent field whose energy spectrum matches the filtered experimental spectrum at the initial time $tU_0/M = 42$. The filtering is done either with a spectral cutoff filter or a physical box filter. When the spectral cutoff filter is applied, the initial field is directly generated on the LES grids using an open-source tool [120]. For the physical box filter, a field is first generated on a grid of 256^3 , then it is filtered onto the LES grids.

Fig. 3.5 compares the computed energy spectra with experimental results at three time instants, namely $tU_0/M = 42, 98$ and 171. The classical Smagorinsky model (SM) with $C_s = 0.18$ accurately predicts the results on both meshes, regardless of the filter used. The only exception is the coarse grid when box filter is applied, in which case the model does not yield sufficient subgrid-scale dissipation. Our tests show that a larger model constant, $C_s = 0.22$, yields good agreement with the filtered experimental spectra (not shown in Fig. 3.5). The dependence of the model constant on the grid resolution is a common drawback of static SGS models [20, 85]. In contrast, the dynamic Smagorinsky model (DSM) does not rely on model constants and reasonably agrees with experimental data. Notably, the computed velocity spectra exhibit near-perfect agreement with experimental results when the box filter is applied, which aligns with the implementation of DSM, where the test filter is a box filter. Larger deviations are generally observed at the scales close to the Nyquist limits, likely due to the numerical errors associated with finite-differencing at high wavenumbers [92].

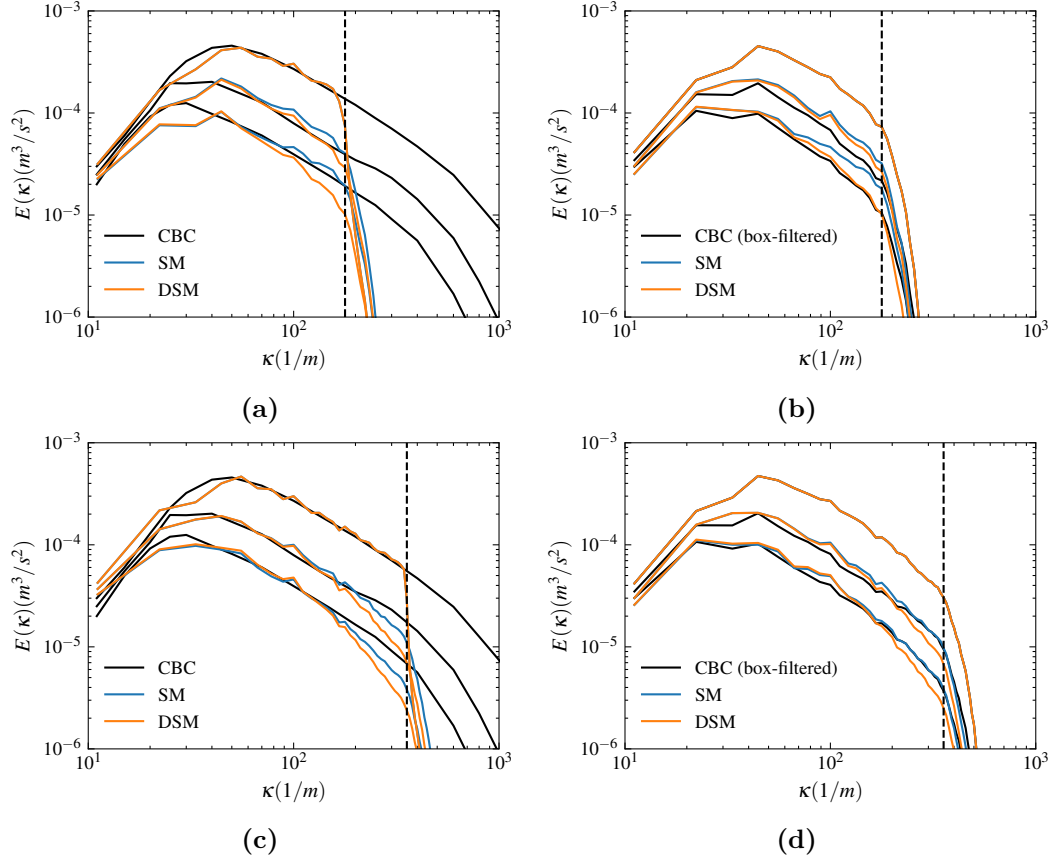


Figure 3.5. Decaying isotropic turbulence: computed velocity spectra with 32^3 (a,b) and 64^3 (c,d) grid points. The experimental spectra [21] are shown unfiltered (a, c) and box-filtered (b, d). In each plot, the results for three time instants $tU_0/M = 42, 98$, and 171 are displayed from top to bottom. The vertical dashed lines denote the Nyquist limits.

3.3.2 Wall-resolved turbulent plane channel flow

Fully developed channel flow is simulated in a domain of $(L_x, L_y, L_z) = (12.8h, 2.0h, 4.8h)$, with h the channel half-height. No-slip boundary conditions are imposed in the y direction and periodic boundary conditions are applied to the x and z directions. The bulk velocity is maintained constant in time using a time-varying, spatially uniform body force. The bulk Reynolds number is defined as $Re_b = 2u_b h/\nu$, with u_b the bulk velocity and ν the kinematic viscosity. The friction Reynolds number is defined as $Re_\tau = u_\tau h/\nu$, where $u_\tau = (\tau_w/\rho)^{1/2}$ is the friction velocity, and the friction coefficient is defined as $C_f = \tau_w/(\rho u_b^2/2)$. The bulk Reynolds number is 20,000 and reference DNS data [76] have $Re_{\tau,DNS} = 543.5$ and $C_{f,DNS} = 0.00591$. Table 3.2 reports the computational parameters for the wall-resolved LES cases, where N_x , N_y , and N_z denote the number of grid points in each direction, and Δx , Δy , and Δz represent the corresponding grid spacings. Specifically, Δy_c denotes the grid spacing at the channel centerline, and Δy_w is the height of the first off-wall layer. The superscript “+” indicates normalization by DNS wall units, whereas superscript “*” used below indicates normalization by the wall units of the simulation. The symbols “SM” and “DSM” denote use of the classical Smagorinsky model with the van Driest damping function and of the dynamic Smagorinsky model, respectively. #ETT denotes the time-averaging interval, expressed in terms of the eddy turnover time $h/u_{\tau,DNS}$. Four sets of grids are used to assess grid convergence, with Δx^+ decreasing from approximately 40 to 10 while maintaining an aspect ratio of $AR = \Delta x/\Delta z = 1.8$. At the channel centerline, the wall-normal grid spacing is approximately equal to the spanwise spacing. Table 3.2 also includes the skin friction coefficient, with its standard uncertainty estimated using a modified batch means method [119]. Its relative error is determined as

$$\epsilon_f = \frac{C_f - C_{f,DNS}}{C_{f,DNS}}. \quad (3.23)$$

The results show that the SM model becomes increasingly accurate as the grid is refined, although it shows approximately 10% error on the coarsest grid. In contrast, the DSM model consistently exhibits errors of less than approximately 3% across all the grid resolutions.

Table 3.2. Computational parameters for WRLES of channel flow.

Mesh ($N_x \times N_y \times N_z$)	Δx^+	Δz^+	Δy_c^+	Δy_w^+	AR	SGS	Re_τ	C_f	ϵ_f	#ETT
$192 \times 128 \times 128$	36.2	20.4	21.5	0.59	1.8	SM	568.4	$0.00646 \pm 0.14\%$	9.38%	32.6
$288 \times 192 \times 192$	24.2	13.6	14.3	0.39	1.8	SM	567.3	$0.00644 \pm 0.17\%$	8.95%	32.6
$384 \times 256 \times 256$	18.1	10.2	10.8	0.29	1.8	SM	556.8	$0.00620 \pm 0.16\%$	4.95%	32.6
$576 \times 384 \times 384$	12.1	6.8	7.2	0.19	1.8	SM	544.8	$0.00594 \pm 0.10\%$	0.50%	32.6
$192 \times 128 \times 128$	36.2	20.4	21.5	0.59	1.8	DSM	537.3	$0.00577 \pm 0.13\%$	-2.25%	32.6
$288 \times 192 \times 192$	24.2	13.6	14.3	0.39	1.8	DSM	541.6	$0.00587 \pm 0.13\%$	-0.69%	32.6
$384 \times 256 \times 256$	18.1	10.2	10.8	0.29	1.8	DSM	539.4	$0.00582 \pm 0.16\%$	-1.49%	32.6
$576 \times 384 \times 384$	12.1	6.8	7.2	0.19	1.8	DSM	534.7	$0.00572 \pm 0.17\%$	-3.22%	32.6

Fig. 3.6(a) shows a visualization of the flow computed on the finest grid, where the near-wall low- and high-speed streaks are clearly observed. To verify the LES implementation, Fig. 3.6(b) presents the profiles of the various contributions to the total shear stress for the grid with $\Delta z^+ = 20.4$. Achievement of a linear

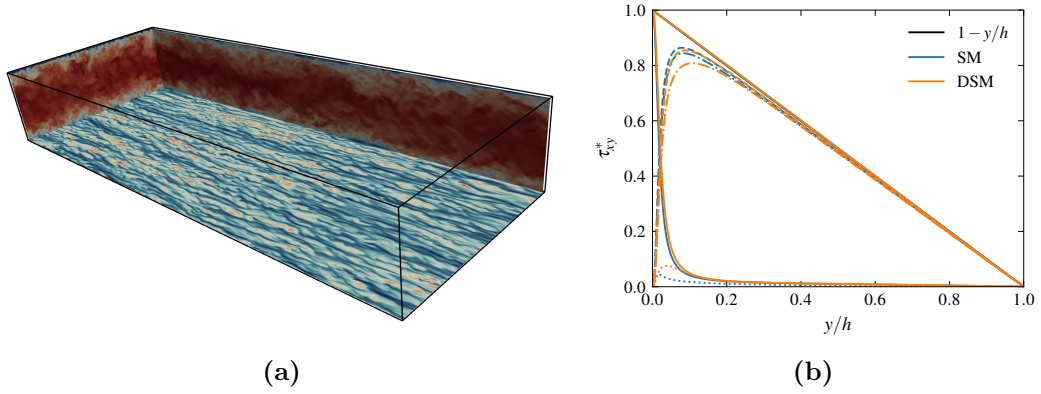


Figure 3.6. Turbulent channel flow: visualization of streamwise velocity obtained with WRLES using the SM model on a mesh with $\Delta_z^+ = 6.8$ (a) and profiles of mean total (solid), resolved and modeled turbulent (dashed), resolved turbulent (dash-dotted), modeled turbulent (dotted), and viscous (solid) shear stress obtained with WRLES using the SM and DSM models on a grid with $\Delta_z^+ = 20.4$ (b). In (a), the wall-parallel plane is located at $y^+ \approx 15$. In (b), normalization is based on the wall units of each simulation.

distribution of the total shear stress provides evidence of general reliability of the LES implementation. Overall, the DSM model generates higher levels of modeled stress along with a lower peak of the resolved shear stress, as compared to the SM model with the van Driest damping function, because the DSM yields higher eddy viscosity levels.

Fig. 3.7 presents the mean streamwise velocity profiles obtained with the SM and DSM models. The velocity profiles normalized by the wall units of each simulation exhibit sensitivity to grid refinement, particularly with the SM model. However, although not shown, when normalized by the wall units of DNS, the velocity profiles show little dependence on grid refinement, with all four grid resolutions closely matching the DNS data. Consequently, the grid sensitivity of the velocity profiles, when normalized by the wall units of each simulation, primarily stems from the grid dependence of the skin friction, particularly when the SM model is applied (see Table 3.2). Fig. 3.8 shows the turbulent normal and shear stresses, normalized by the DNS wall units, disregarding the modeled stress. When the SM model is applied, the resolved shear stress is over-predicted, whereas it is under-predicted with the DSM model. This behavior aligns with the corresponding over- and under-estimation of the friction coefficient by the two models. As for the normal stress, the peak of the streamwise component is over-estimated, as also observed in other studies [147]. Overall, grid refinement leads to convergence of all the profiles towards the DNS data.

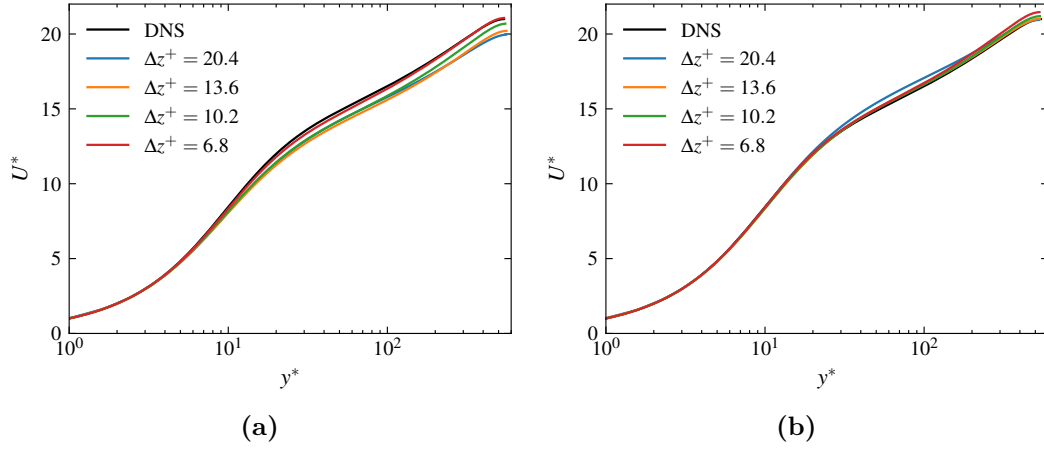


Figure 3.7. Turbulent channel flow: profiles of mean streamwise velocity obtained with WRLES using the SM (a) and DSM (b) models. Normalization is based on wall units of each simulation. The DNS data is from [76].

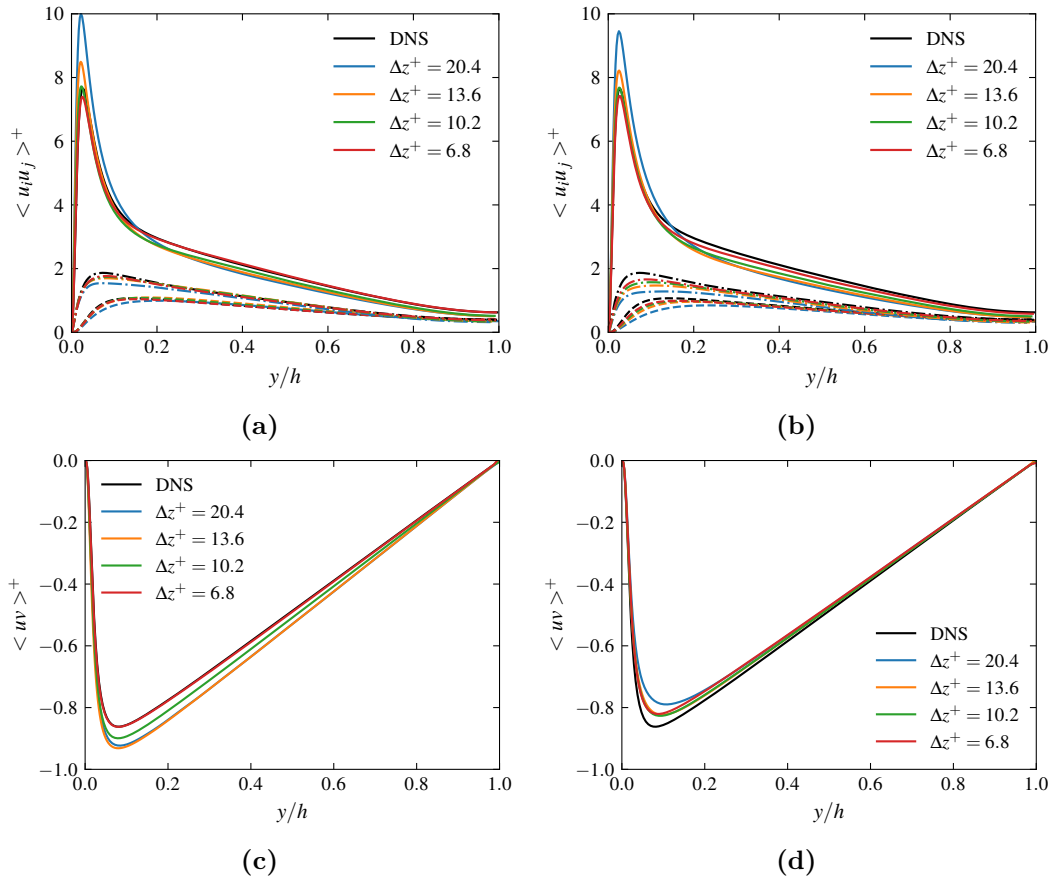


Figure 3.8. Turbulent channel flow: profiles of resolved turbulent normal stress (a,b) and shear stress (c,d) obtained with WRLES using the SM (a,c) and DSM (b,d) models. Normalization is based on the DNS wall units. The DNS data is from [76]. Line codes: $\langle uu \rangle$ (solid), $\langle vv \rangle$ (dashed), $\langle uv \rangle$ (dash-dotted).

3.3.3 Wall-modeled turbulent plane channel flow

The computational setup is identical to that of the previous WRLES case, except that the bulk Reynolds number is 250,000 and that wall-model boundary conditions are imposed on the walls. The reference data [76] have $Re_{\tau,DNS} = 5185.9$ and $C_{f,DNS} = 0.00344$. Table 3.3 provides the computational parameters for the WMLES cases. The parameter definitions are the same as in Table 3.2, but the cell spacings are given in terms of the half-channel height. We use aspect ratios of $AR = 1.0$ and $AR = 2.0$, with thirteen grids generated through uniform refinement in all three directions for the grid convergence study. The finest grid has approximately $\Delta z^+ = 10$, which qualifies it as a WRLES case. For all the cases, the wall-modeled layer thickness is $h_{wm} = 0.1h$. Fig. 3.9(a) visualizes the turbulent channel flow obtained with WMLES using the SM model on a mesh with $\Delta_z/h = 0.012$. The low- and high-speed streaks are clearly visible on the plane at $y/h = 0.1$. Fig. 3.9(b) presents the profiles of various contributions to the total shear stress for the grid with $\Delta_z/h = 0.050$. The linear distribution of total shear stress is predicted accurately, demonstrating the correct implementation of WMLES. Overall, DSM yields higher levels of modeled stress, with a lower peak in the resolved shear stress, compared to the SM model. This aligns with the WRLES results (Fig. 3.6(b)).

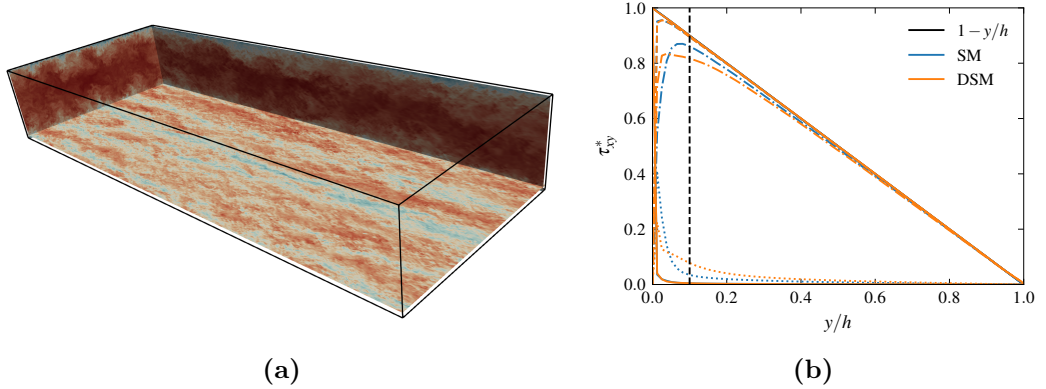


Figure 3.9. Turbulent channel flow: visualization of streamwise velocity obtained with WMLES using the SM model on a mesh with $\Delta_z/h = 0.012$ (a) and profiles of mean total (solid), resolved and modeled turbulent (dashed), resolved turbulent (dash-dotted), modeled turbulent (dotted) and viscous (solid) shear stress computed with WMLES using the SM and DSM models, for the grid with $\Delta_z/h = 0.050$ and $AR = 1.0$ (b). In (a), the wall-parallel plane is located at $y/h = 0.1$. In (b), the dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of each simulation.

Fig. 3.10 presents the computed velocity results on the grids with $AR = 1.0$ and $AR = 2.0$ and Fig. 3.11 shows the resolved turbulent normal and shear stresses for $AR = 1.0$. Profiles on four different grids are displayed to demonstrate grid convergence. For $AR = 1.0$, the profiles agree well with the DNS data, although the resolved turbulent stress profiles show noticeable variations with grid refinement. Notably, the streamwise velocity fluctuations are over-predicted, and the resolved turbulent shear stress generally increases as the grid resolution improves. The LES-computed velocity profile in the wall-modeled layer should not be taken seriously, as it is expected to be replaced by the wall model, i.e., the logarithmic law in Eq. (3.10).

Table 3.3. Computational parameters for WMLES of turbulent channel flow.

Mesh ($N_x \times N_y \times N_z$)	$\Delta x/h$	$\Delta z/h$	$\Delta y_c/h$	$\Delta y_w/h$	AR	SGS	Re_τ	C_f	ϵ_f	#ETT
$128 \times 32 \times 48$	0.100	0.100	0.100	0.0252	1.0	SM	5165.0	$0.00341 \pm 0.10\%$	-0.81%	20.7
$192 \times 48 \times 72$	0.067	0.067	0.067	0.0167	1.0	SM	5155.4	$0.00340 \pm 0.16\%$	-1.17%	20.7
$256 \times 64 \times 96$	0.050	0.050	0.050	0.0125	1.0	SM	5178.9	$0.00343 \pm 0.17\%$	-0.27%	20.7
$384 \times 96 \times 144$	0.033	0.033	0.033	0.0083	1.0	SM	5147.3	$0.00339 \pm 0.19\%$	-1.48%	20.7
$512 \times 128 \times 192$	0.025	0.025	0.025	0.0063	1.0	SM	5115.7	$0.00335 \pm 0.06\%$	-2.69%	20.7
$640 \times 160 \times 240$	0.020	0.020	0.020	0.0050	1.0	SM	5101.7	$0.00333 \pm 0.27\%$	-3.22%	20.7
$768 \times 192 \times 288$	0.017	0.017	0.017	0.0042	1.0	SM	5097.0	$0.00333 \pm 0.22\%$	-3.40%	20.7
$896 \times 224 \times 336$	0.014	0.014	0.014	0.0036	1.0	SM	5109.9	$0.00334 \pm 0.23\%$	-2.91%	20.7
$1024 \times 256 \times 384$	0.013	0.012	0.012	0.0031	1.0	SM	5121.6	$0.00336 \pm 0.24\%$	-2.46%	20.7
$1536 \times 256 \times 576$	0.008	0.008	0.012	0.0031	1.0	SM	5181.7	$0.00344 \pm 0.45\%$	-0.16%	20.7
$2048 \times 512 \times 768$	0.006	0.006	0.006	0.0016	1.0	SM	5248.3	$0.00353 \pm 0.64\%$	2.42%	20.7
$4096 \times 1024 \times 1536$	0.003	0.003	0.003	0.0008	1.0	SM	5249.3	$0.00353 \pm 0.60\%$	2.46%	20.7
$6144 \times 1536 \times 2304$	0.002	0.002	0.002	0.0005	1.0	SM	5248.4	$0.00353 \pm 0.56\%$	2.43%	20.7
$128 \times 32 \times 48$	0.100	0.100	0.100	0.0252	1.0	DSM	5229.5	$0.00350 \pm 0.27\%$	1.69%	20.7
$192 \times 48 \times 72$	0.067	0.067	0.067	0.0167	1.0	DSM	5235.3	$0.00351 \pm 0.26\%$	1.91%	20.7
$256 \times 64 \times 96$	0.050	0.050	0.050	0.0125	1.0	DSM	5243.1	$0.00352 \pm 0.08\%$	2.22%	20.7
$384 \times 96 \times 144$	0.033	0.033	0.033	0.0083	1.0	DSM	5217.6	$0.00348 \pm 0.20\%$	1.23%	20.7
$512 \times 128 \times 192$	0.025	0.025	0.025	0.0063	1.0	DSM	5181.1	$0.00344 \pm 0.09\%$	-0.19%	20.7
$640 \times 160 \times 240$	0.020	0.020	0.020	0.0050	1.0	DSM	5163.1	$0.00341 \pm 0.15\%$	-0.88%	20.7
$768 \times 192 \times 288$	0.017	0.017	0.017	0.0042	1.0	DSM	5151.3	$0.00340 \pm 0.31\%$	-1.33%	20.7
$896 \times 224 \times 336$	0.014	0.014	0.014	0.0036	1.0	DSM	5150.5	$0.00340 \pm 0.16\%$	-1.36%	20.7
$1024 \times 256 \times 384$	0.013	0.012	0.012	0.0031	1.0	DSM	5161.8	$0.00341 \pm 0.29\%$	-0.93%	20.7
$1536 \times 256 \times 576$	0.008	0.008	0.012	0.0031	1.0	DSM	5179.5	$0.00343 \pm 0.15\%$	-0.25%	20.7
$2048 \times 512 \times 768$	0.006	0.006	0.006	0.0016	1.0	DSM	5207.5	$0.00347 \pm 0.73\%$	0.83%	20.7
$4096 \times 1024 \times 1536$	0.003	0.003	0.003	0.0008	1.0	DSM	5236.4	$0.00351 \pm 0.25\%$	1.96%	20.7
$6144 \times 1536 \times 2304$	0.002	0.002	0.002	0.0005	1.0	DSM	5197.2	$0.00346 \pm 0.43\%$	0.44%	20.7
$64 \times 32 \times 48$	0.200	0.100	0.100	0.0252	2.0	SM	5227.3	$0.00350 \pm 0.22\%$	1.60%	20.7
$96 \times 48 \times 72$	0.133	0.067	0.067	0.0167	2.0	SM	5335.4	$0.00364 \pm 0.09\%$	5.85%	20.7
$128 \times 64 \times 96$	0.100	0.050	0.050	0.0125	2.0	SM	5346.2	$0.00366 \pm 0.09\%$	6.28%	20.7
$192 \times 96 \times 144$	0.067	0.033	0.033	0.0083	2.0	SM	5325.2	$0.00363 \pm 0.12\%$	5.44%	20.7
$256 \times 128 \times 192$	0.050	0.025	0.025	0.0063	2.0	SM	5264.4	$0.00355 \pm 0.09\%$	3.05%	20.7
$320 \times 160 \times 240$	0.040	0.020	0.020	0.0050	2.0	SM	5208.7	$0.00347 \pm 0.06\%$	0.88%	20.7
$384 \times 192 \times 288$	0.033	0.017	0.017	0.0042	2.0	SM	5153.5	$0.00340 \pm 0.16\%$	-1.24%	20.7
$448 \times 224 \times 336$	0.029	0.014	0.014	0.0036	2.0	SM	5113.3	$0.00335 \pm 0.18\%$	-2.78%	20.7
$512 \times 256 \times 384$	0.025	0.012	0.012	0.0031	2.0	SM	5075.8	$0.00330 \pm 0.18\%$	-4.20%	20.7
$768 \times 384 \times 576$	0.017	0.008	0.008	0.0021	2.0	SM	5057.8	$0.00327 \pm 0.37\%$	-4.88%	20.7
$1024 \times 512 \times 768$	0.013	0.006	0.006	0.0016	2.0	SM	5165.5	$0.00342 \pm 2.07\%$	-0.79%	20.7
$2048 \times 1024 \times 1536$	0.006	0.003	0.003	0.0008	2.0	SM	5196.9	$0.00346 \pm 1.12\%$	0.42%	20.7
$3072 \times 1536 \times 2304$	0.004	0.002	0.002	0.0005	2.0	SM	5192.6	$0.00345 \pm 0.53\%$	0.26%	20.7
$64 \times 32 \times 48$	0.200	0.100	0.100	0.0252	2.0	DSM	5360.1	$0.00368 \pm 0.12\%$	6.83%	20.7
$96 \times 48 \times 72$	0.133	0.067	0.067	0.0167	2.0	DSM	5356.5	$0.00367 \pm 0.08\%$	6.69%	20.7
$128 \times 64 \times 96$	0.100	0.050	0.050	0.0125	2.0	DSM	5355.5	$0.00367 \pm 0.07\%$	6.65%	20.7
$192 \times 96 \times 144$	0.067	0.033	0.033	0.0083	2.0	DSM	5338.3	$0.00365 \pm 0.08\%$	5.96%	20.7
$256 \times 128 \times 192$	0.050	0.025	0.025	0.0063	2.0	DSM	5294.1	$0.00359 \pm 0.09\%$	4.22%	20.7
$320 \times 160 \times 240$	0.040	0.020	0.020	0.0050	2.0	DSM	5244.4	$0.00352 \pm 0.04\%$	2.27%	20.7
$384 \times 192 \times 288$	0.033	0.017	0.017	0.0042	2.0	DSM	5192.8	$0.00345 \pm 0.10\%$	0.27%	20.7
$448 \times 224 \times 336$	0.029	0.014	0.014	0.0036	2.0	DSM	5142.5	$0.00339 \pm 0.13\%$	-1.67%	20.7
$512 \times 256 \times 384$	0.025	0.012	0.012	0.0031	2.0	DSM	5111.4	$0.00334 \pm 0.21\%$	-2.85%	20.7
$768 \times 384 \times 576$	0.017	0.008	0.008	0.0021	2.0	DSM	5066.6	$0.00329 \pm 0.69\%$	-4.55%	20.7
$1024 \times 512 \times 768$	0.013	0.006	0.006	0.0016	2.0	DSM	5141.1	$0.00338 \pm 0.90\%$	-1.72%	20.7
$2048 \times 1024 \times 1536$	0.006	0.003	0.003	0.0008	2.0	DSM	5205.0	$0.00347 \pm 0.72\%$	0.74%	20.7
$3072 \times 1536 \times 2304$	0.004	0.002	0.002	0.0005	2.0	DSM	5219.5	$0.00349 \pm 0.53\%$	1.30%	20.7

In contrast, the computed velocity profiles for $AR = 2.0$ become more sensitive to grid refinement, although the results on the fine grids exhibit good accuracy.

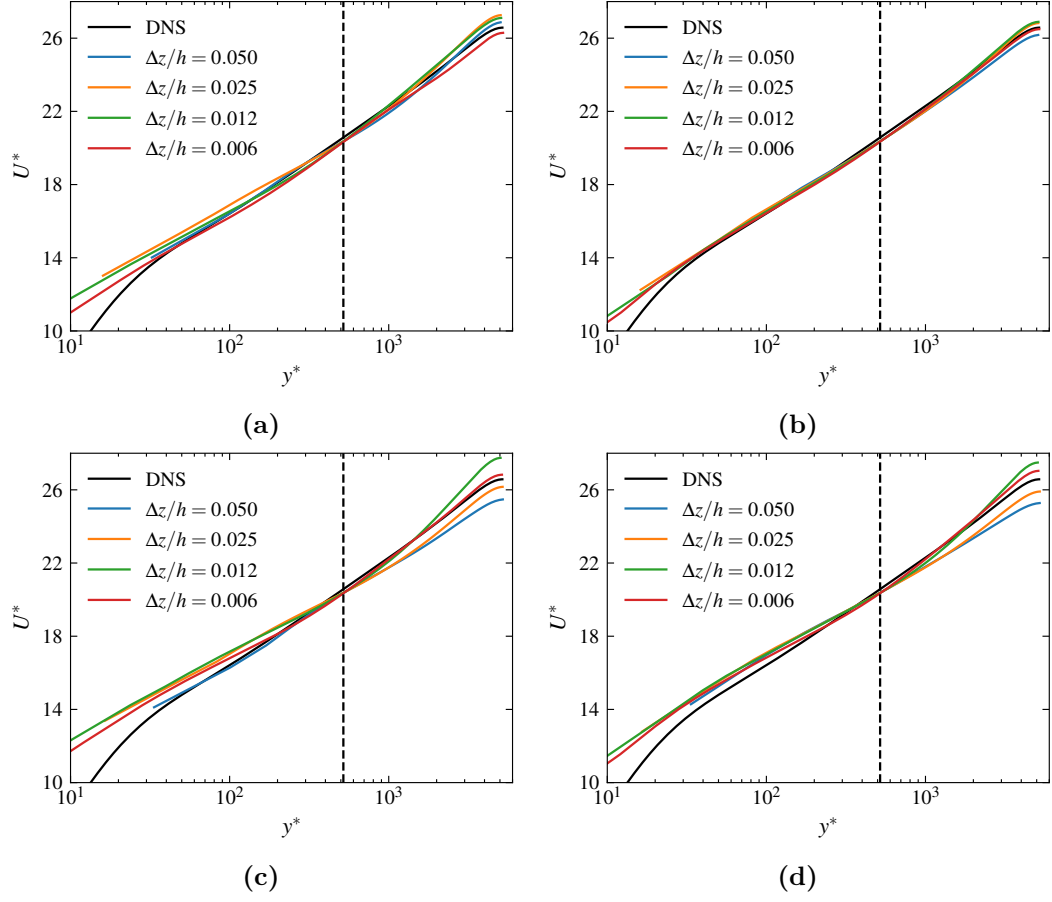


Figure 3.10. Turbulent channel flow: profiles of mean streamwise velocity obtained with WMLES using the SM (a,c) and DSM (b,d) models on the grids with $AR = 1.0$ (a,b) and $AR = 2.0$ (c,d). The dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of each simulation. The DNS data is from [76].

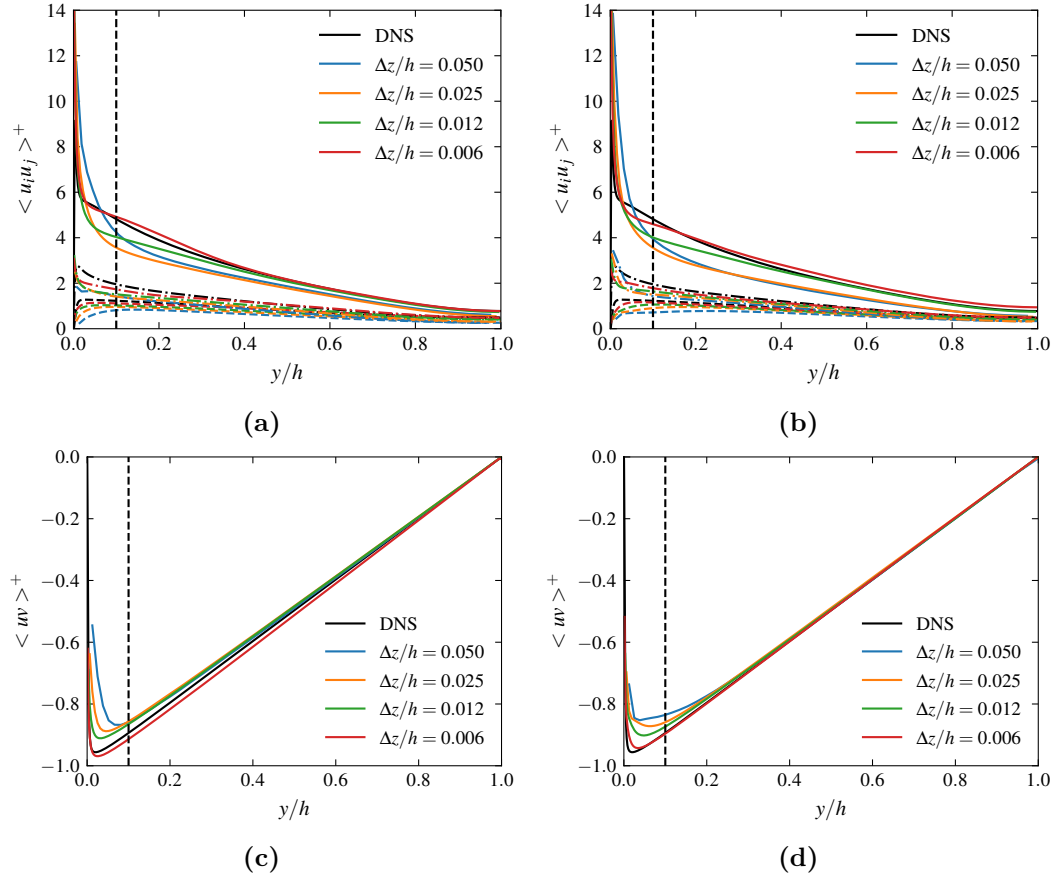


Figure 3.11. Turbulent channel flow: profiles of resolved turbulent normal stress (a,b) and shear stress (c,d) obtained with WMLES using the SM (a,c) and DSM (b,d) models on the grids with $AR = 1.0$. The dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of the DNS (a,b) and the wall units of each simulation (c,d). Normalization is based on the DNS wall units. The DNS data is from [76]. Line codes: $\langle uu \rangle$ (solid), $\langle vv \rangle$ (dashed), $\langle ww \rangle$ (dash-dotted).

For grid convergence study, Fig. 3.12 plots the variations of prediction errors with $\Delta z/h$ for different quantities, including C_f , U , $\langle uu \rangle$, $\langle vv \rangle$, $\langle ww \rangle$, and $\langle uv \rangle$. For C_f , the relative error is defined in the same way as in Eq. (3.23), and for the other quantities, the relative error is defined as

$$\epsilon_\phi = \frac{\left[\int_{y/h=0.1}^{y/h=1.0} (\phi - \phi_{ref})^2 d(y/h) \right]^{1/2}}{\left| \int_{y/h=0.1}^{y/h=1.0} \phi_{ref} d(y/h) \right|}, \quad (3.24)$$

where ϕ denotes outer-scaled quantities. Notably, the integration is performed from $y = 0.1h$ to $y = h$, excluding the wall-modeled layer, since this layer is contaminated by significant numerical errors and is expected to be replaced by the velocity profile yielded from the wall model. Fig. 3.12 pinpoints non-monotonic grid convergence, for both the SGS models on grids with the two aspect ratios, as also noted by Meyers and Sagaut [86] for WRLES. In our results, the streamwise component of the turbulent normal stress exhibits the most marked non-monotonic convergence. Non-monotonic convergence is also observed in the skin friction and turbulent shear stress. On the finest grid with $AR = 1.0$, the error in wall friction is 2.43% for the SM model and 0.44% for the DSM model. The errors for the finest grid with $AR = 2.0$ are 0.26% and 1.30%, respectively. Notably, the sign of the error on the finest grids is always positive. We have also conducted WMLES for cases with $Re_b = 20,000$. Although the results are not shown here, the signs of the errors for very fine grids are also positive. This is consistent with the fact that mean wall shear stress obtained with WMLES is biased toward larger values when the instantaneous velocity is used as input for the wall model. Additionally, we note that grids with $AR = 1.0$ exhibit better grid convergence compared to those with $AR = 2.0$, as variations with grid refinement become smaller in the errors of wall shear stress and mean velocity profile, and therefore the non-monotonic grid convergence behavior becomes less pronounced.

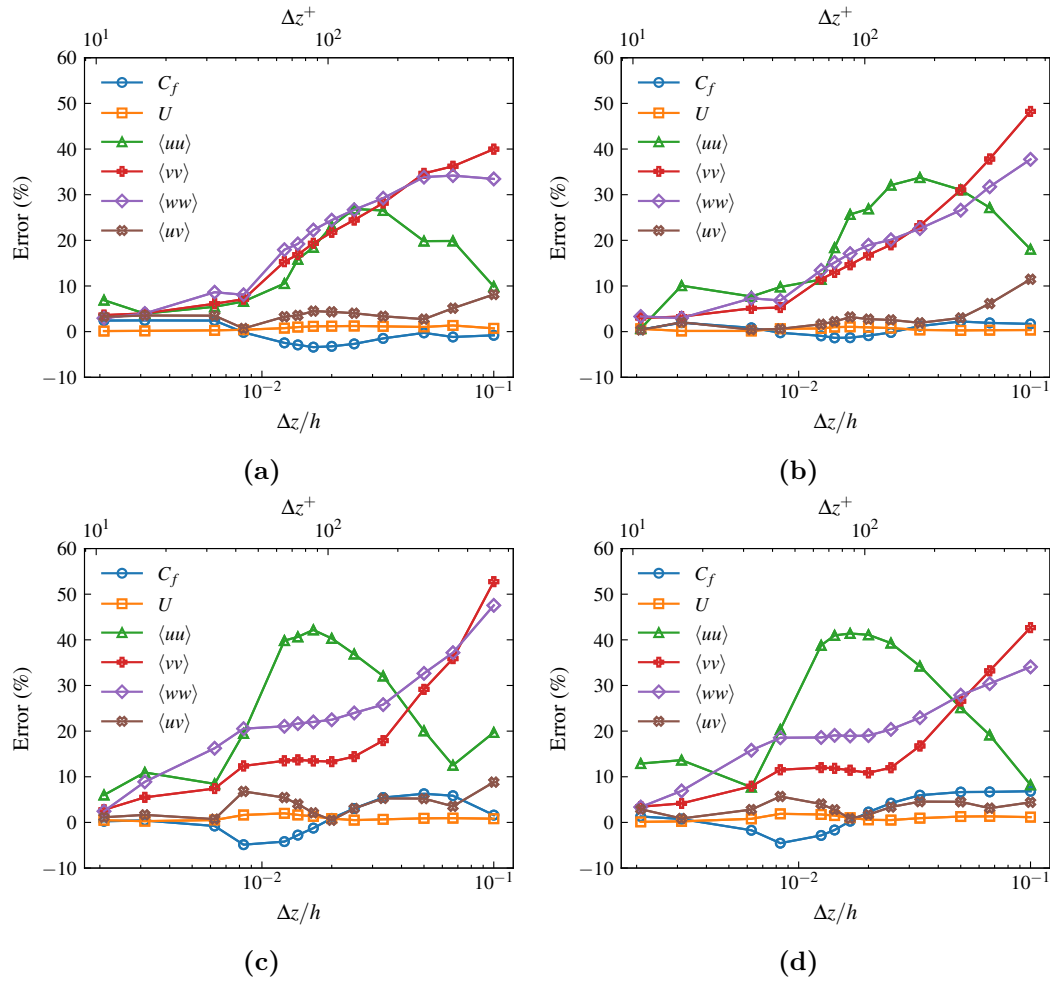


Figure 3.12. Turbulent channel flow: prediction error as a function of $\Delta z/h$ or Δz^+ for various statistical properties, as obtained from WMLES with SM (a,c) and DSM (b,d) models for grids with $AR = 1.0$ (a,b) and $AR = 2.0$ (c,d).

3.3.4 Wall-modeled turbulent square duct flow

Fully developed turbulent duct flow is simulated using WMLES, following the setup in Fig. 3.13(a). The computational domain is $(L_x, L_y, L_z) = (12.8h, 2.0h, 2.0h)$, where h is half of the duct side length. Wall-modeled boundary conditions are applied in the y and z directions, whereas periodic boundary conditions are enforced in the streamwise x direction. The simulation maintains the bulk velocity constant by applying a time-varying, spatially uniform body force. The bulk Reynolds number ($Re_b = 2u_b h/\nu$) is 40,000. DNS at the same Reynolds number was conducted in [107], which yielded $Re_{\tau, DNS} = 1055$, $C_{f, DNS} = 0.00557$. In our simulations, the wall-modeled layer thickness is set to $h_{wm} = 0.1h$. Table 3.4 provides the computational parameters, where $Re_{\tau} = u_{\tau} h/\nu$ represents the perimeter-averaged friction Reynolds number, and $C_f = \tau_w/(\rho u_b^2/2)$ is the perimeter-averaged skin friction coefficient. Six meshes with uniform grid spacings in all three directions are used for the grid convergence study, with the finest having $\Delta_z^+ = 6.3$. Fig. 3.13(b) visualizes the turbulent flow on the finest grid obtained with WMLES using the SM model with the van Driest damping function. This visualization clearly captures the low- and high-speed streaks, as well as the coherent structures responsible for the secondary flow. Table 3.4 lists the relative error in the skin friction coefficient, with absolute values within approximately 3% for all the cases. This supports the general applicability of the logarithmic law as a wall model to predict friction in flows with moderate geometrical complexity. We also report the secondary-flow kinetic energy, defined as [143]

$$K = \int_{y/h=-1}^0 \int_{z/h=-1}^0 \frac{1}{2} \left(\left(\frac{V}{u_b} \right)^2 + \left(\frac{W}{u_b} \right)^2 \right) d(y/h) d(z/h), \quad (3.25)$$

where V and W are the mean velocity components in the y and z directions, respectively. The DNS result from [107] gives a reference value of 3.26×10^{-5} . The relative errors on the finest grid are approximately 22% for the WMLES simulation with the SM model and 25% for the DSM model. Notably, the convergence behavior of both the skin friction coefficient and the cross-flow kinetic energy is non-monotonic for both SGS models.

Table 3.4. Parameters for the WMLES of turbulent square duct flow.

Mesh ($N_x \times N_y \times N_z$)	$\Delta x/h$	$\Delta y/h$	$\Delta z/h$	AR	SGS	Re_{τ}	C_f	ϵ_f	#ETT	K	ϵ_K
$128 \times 80 \times 80$	0.100	0.025	0.025	4.0	SM	1065.9	$0.00568 \pm 0.08\%$	2.08%	52.8	2.17×10^{-5}	-33.63%
$256 \times 80 \times 80$	0.050	0.025	0.025	2.0	SM	1044.5	$0.00546 \pm 0.09\%$	-1.97%	52.8	2.85×10^{-5}	-12.77%
$512 \times 80 \times 80$	0.025	0.025	0.025	1.0	SM	1038.6	$0.00539 \pm 0.11\%$	-3.08%	52.8	4.05×10^{-5}	24.05%
$768 \times 120 \times 120$	0.017	0.017	0.017	1.0	SM	1041.4	$0.00542 \pm 0.17\%$	-2.55%	52.8	4.03×10^{-5}	23.58%
$1024 \times 160 \times 160$	0.013	0.013	0.013	1.0	SM	1040.9	$0.00542 \pm 0.11\%$	-2.66%	52.8	3.67×10^{-5}	12.41%
$2048 \times 320 \times 320$	0.006	0.006	0.006	1.0	SM	1044.3	$0.00545 \pm 0.17\%$	-2.03%	52.8	3.99×10^{-5}	22.21%
$128 \times 80 \times 80$	0.100	0.025	0.025	4.0	DSM	1066.3	$0.00569 \pm 0.10\%$	2.16%	52.8	2.35×10^{-5}	-27.83%
$256 \times 80 \times 80$	0.050	0.025	0.025	2.0	DSM	1051.0	$0.00552 \pm 0.10\%$	-0.76%	52.8	3.15×10^{-5}	-3.53%
$512 \times 80 \times 80$	0.025	0.025	0.025	1.0	DSM	1046.3	$0.00547 \pm 0.10\%$	-1.65%	52.8	3.54×10^{-5}	8.37%
$768 \times 120 \times 120$	0.017	0.017	0.017	1.0	DSM	1047.0	$0.00548 \pm 0.11\%$	-1.51%	52.8	3.69×10^{-5}	13.08%
$1024 \times 160 \times 160$	0.013	0.013	0.013	1.0	DSM	1045.7	$0.00547 \pm 0.09\%$	-1.76%	52.8	3.64×10^{-5}	11.41%
$2048 \times 320 \times 320$	0.006	0.006	0.006	1.0	DSM	1044.9	$0.00546 \pm 0.13\%$	-1.91%	52.8	4.09×10^{-5}	25.29%

Fig. 3.14 presents contours obtained with the SM model for the mean velocity components (U^+ , V^+), turbulent normal stresses ($\langle uu \rangle^+$, $\langle vv \rangle^+$), and turbulent shear stress ($\langle uv \rangle^+$). Only one quarter of the full domain is presented. The contours of

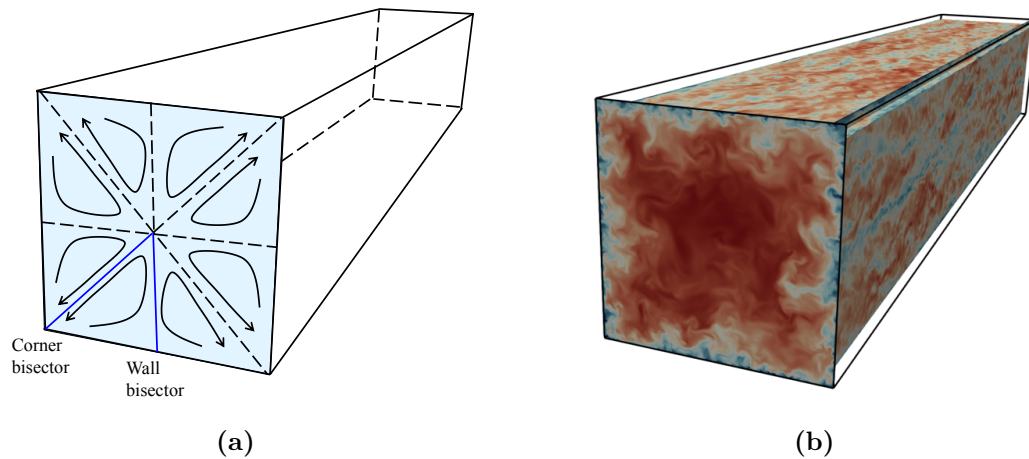


Figure 3.13. Computational setup for the square duct flow (a) and visualization of turbulent flow obtained with WMLES using the SM model on a mesh with $\Delta_z/h = 0.006$ (b). The planes display contours of the streamwise velocity, with the wall-parallel plane located at $y/h = 0.1$.

W^+ , $\langle uw \rangle^+$, and $\langle uv \rangle^+$ are not displayed, as they are symmetric with respect to the diagonal, to those of V^+ , $\langle vv \rangle^+$, and $\langle uv \rangle^+$, respectively. The distributions obtained for various grids show little variations in the core region and agree reasonably well with the reference DNS data. The primary benefit of grid refinement is an increase in the overall accuracy in the near-wall region.

We further extract the profiles of these quantities along the wall and corner bisectors (see Fig. 3.13(a)). Fig. 3.15 displays the profiles of mean streamwise velocity along both the wall bisector and the corner bisector. Consistent with the contours in Fig. 3.14, the mean streamwise velocity profiles show minor variations with grid refinement, indicating satisfactory grid convergence properties. In contrast, the most significant variations are observed in the turbulent normal stress $\langle uu \rangle$ (Fig. 3.16). Unlike the results for channel flow, the predicted peak in $\langle uu \rangle$ along the wall bisector increases with grid refinement, leading to improved accuracy. Small oscillations are observed near the turbulent stress peak for the coarser grids, which we attribute to the absence of grid clustering near the wall; these oscillations gradually disappear with grid refinement. Notably, the variations in shear stress exhibit a clearly non-monotonic pattern as the grid is refined.

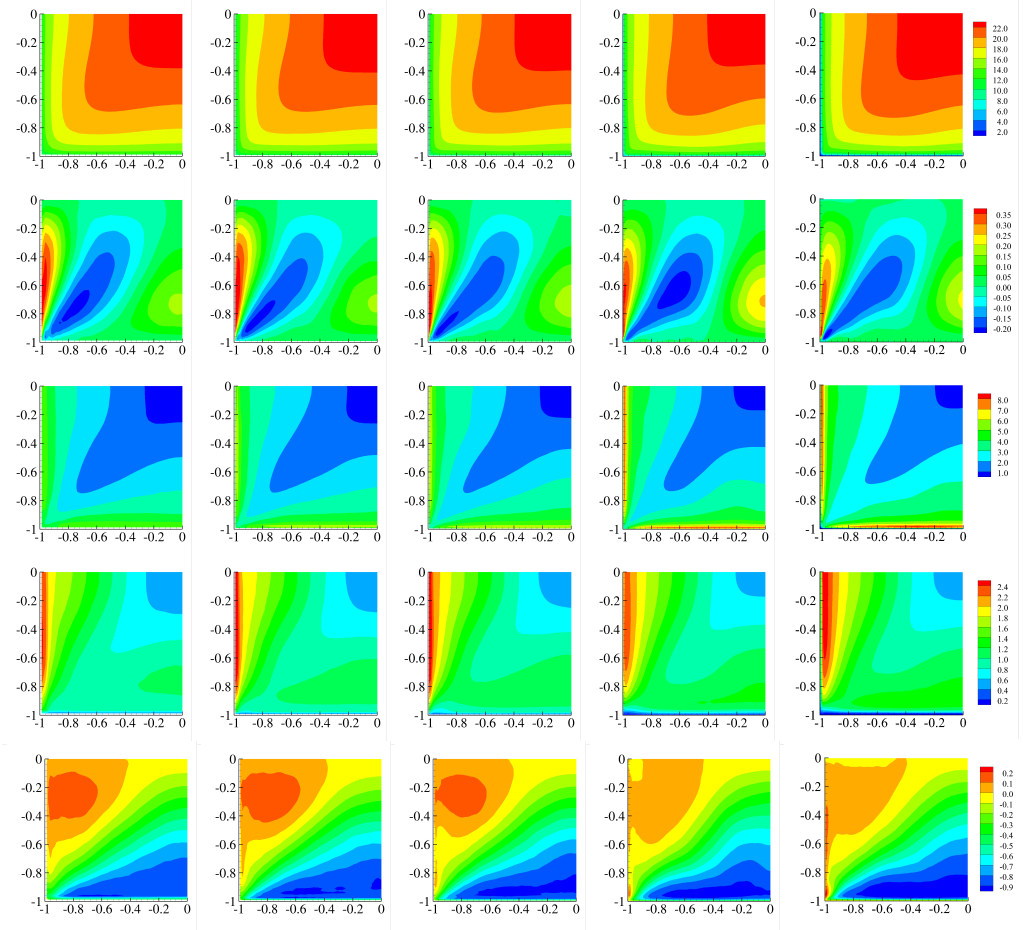


Figure 3.14. Turbulent square duct flow: cross-stream contours of mean velocity U^+ (row 1) and V^+ (row 2), turbulent normal stresses $\langle uu \rangle^+$ (row 3) and $\langle vv \rangle^+$ (row 4), and shear stress $\langle uv \rangle^+$ (row 5), computed on different grids using WMLES with the SM model. Normalization is based on the DNS wall units. Only a quarter of the full domain is presented. The horizontal axis is z and the vertical axis is y . $\Delta_z/h = 0.025$ (column 1), $\Delta_z/h = 0.017$ (column 2), $\Delta_z/h = 0.013$ (column 3), $\Delta_z/h = 0.006$ (column 4) and DNS [107] (column 5).

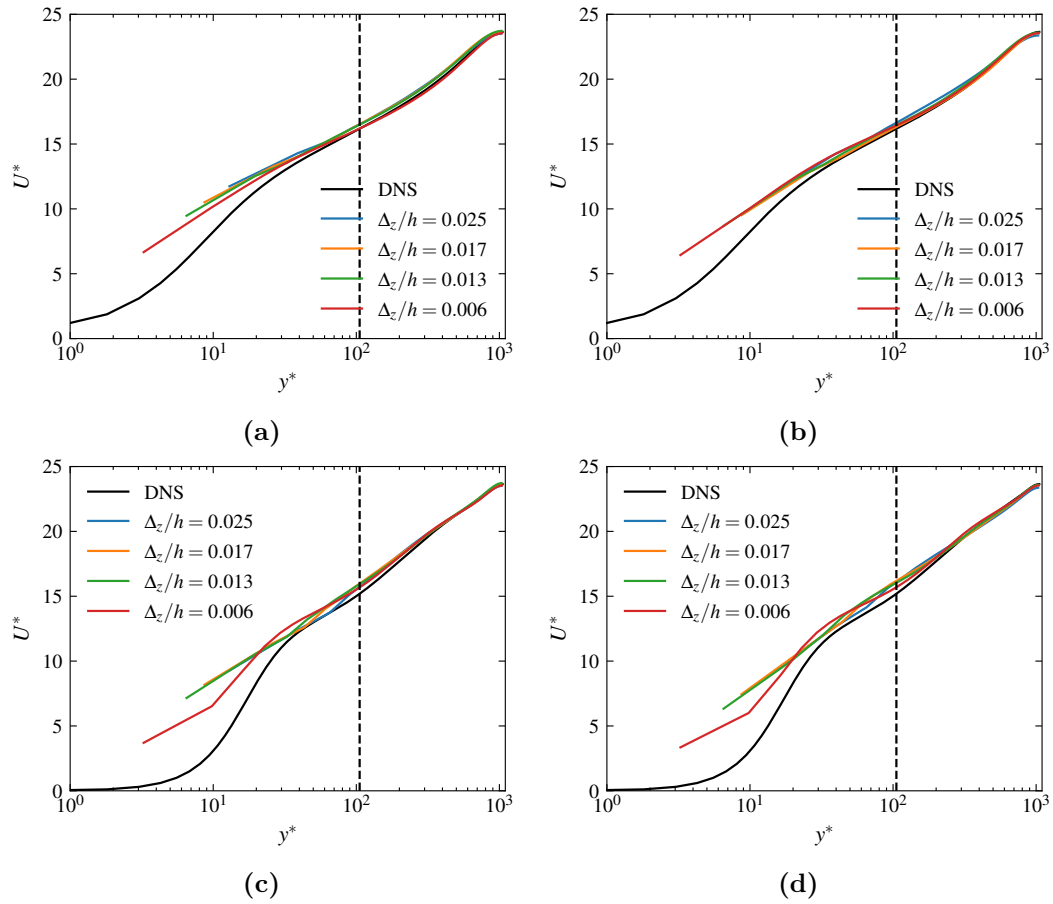


Figure 3.15. Turbulent square duct flow: profiles of mean streamwise velocity along the wall bisector (a,b) and corner bisector (c,d) obtained from WMLES with the SM (a,c) and DSM (b,d) models. The dashed line denotes $y/h = 0.1$. Normalization is based on the wall units of each simulation. The DNS data is from [107].

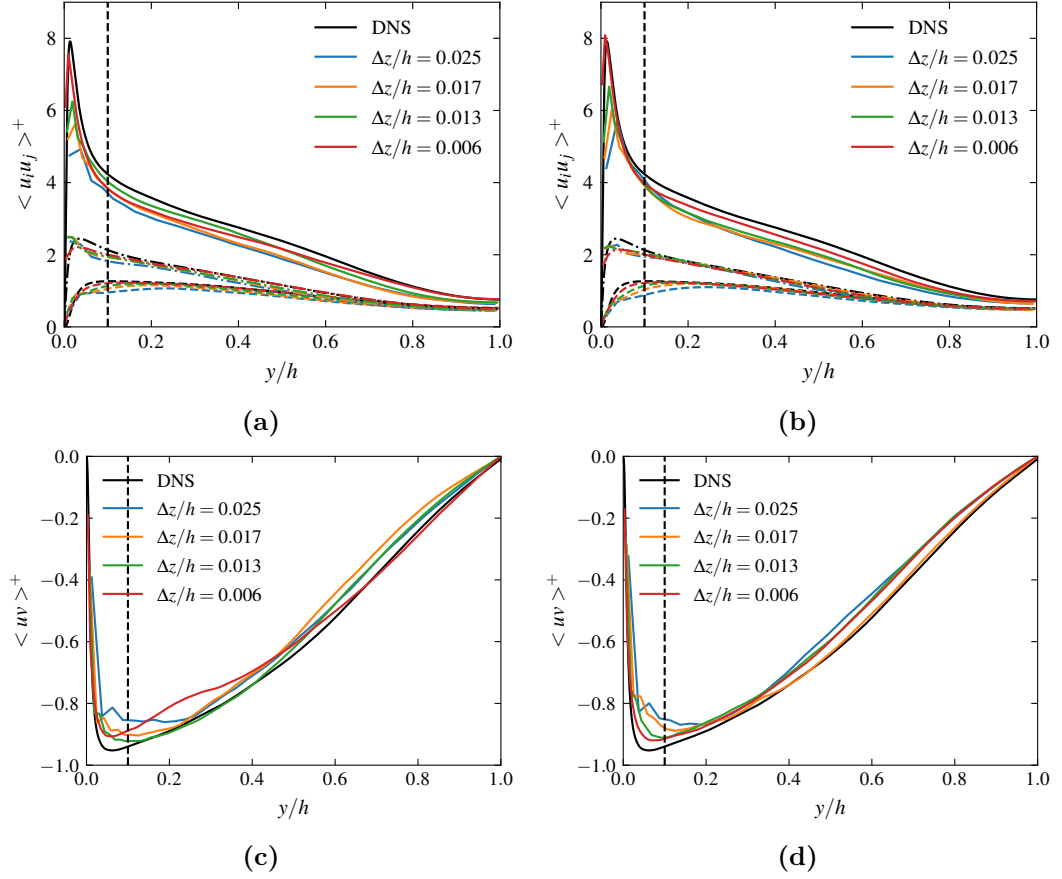


Figure 3.16. Turbulent square duct flow: profiles of resolved turbulent normal stress (a,b) and shear stress (c,d) along the wall bisector as obtained from WMLES with the SM (a,c) and DSM (b,d) models. The dashed line denotes $y/h = 0.1$. Normalization is based on the DNS wall units. The DNS data is from [107]. Line codes: $\langle uu \rangle$ (solid), $\langle vv \rangle$ (dashed), $\langle ww \rangle$ (dash-dotted).

3.4 Conclusion

We have introduced CaLES, a GPU-accelerated solver designed for large-eddy simulation of incompressible wall-bounded flows. Based on the GPU-accelerated DNS solver CaNS, CaLES demonstrates significant computational efficiency improvements through GPU parallelization using a combination of CUDA Fortran and OpenACC directives, and good scalability on massively parallel architectures. The incorporation of SGS models, including static and dynamic Smagorinsky models, along with a wall model, extends the solver’s applicability to wall-resolved and wall-modeled LES.

Performance evaluation conducted on state-of-the-art high-performance computing clusters, shows that CaLES achieves substantial speed-ups using GPU acceleration compared to its CPU-only counterparts. The solver efficiently scales across multiple GPUs, achieving approximately 15× speed-up on a single GPU compared to a full CPU node, making it highly effective for high-fidelity simulations on large computational grids. Validation cases—such as decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow—confirm the solver’s accuracy for LES, with the dynamic Smagorinsky model exhibiting somewhat superior performance in grid convergence and turbulence prediction.

CaLES has also been applied for wall-modeled LES of turbulent channel and duct flows. Its high computational efficiency enables WMLES for channels with $Re_\tau = 5200$ on progressively refined grids, meeting the grid requirements for wall-resolved LES. A key observation is the non-monotonic grid convergence in WMLES for channel and duct flows, particularly in wall friction, streamwise velocity fluctuations, and turbulent shear stresses. These results underscore the complexities inherent in WMLES and emphasize the need for continued research into SGS and wall models.

In summary, CaLES offers the computational efficiency and flexibility needed to investigate turbulent flows with moderate complexity at high Reynolds numbers. Its open-source availability makes it a valuable tool for fast simulations, which is particularly important in the application of machine learning to develop robust SGS and wall models, including data-driven approaches and reinforcement learning.

Finally, we should emphasize that the purpose of this work is not to carry out a physical investigation of the performance of SGS models and wall models. Instead, our aim is to incorporate well-established models into a baseline solver that can easily incorporate other models, such as the Vreman subgrid-scale model [147] and the ordinary differential equation-based wall model [63].

Chapter 4

CFD-solver-agnostic deep reinforcement learning framework for computational fluid dynamics

We present SmartFlow, a CFD-solver-agnostic framework for deep reinforcement learning in computational fluid dynamics. We begin by introducing the general DRL workflow (Section 4.1), then detail the SmartFlow architecture and implementation (Section 4.2). Three demonstration cases showcase the solver-agnostic versatility of SmartFlow (Section 4.3): single-agent synthetic-jet control in a cylinder wake using the high-order FLEXI solver, multi-agent wake control with the GPU-accelerated SOD2D spectral-element code, and multi-task multi-agent wall-model learning in LES with the finite-difference CaLES solver. Finally, we provide concluding remarks (Section 4.4).

4.1 Reinforcement learning

In this section, the core elements of the classic reinforcement learning paradigm are reviewed. This review is not exhaustive and focuses only on the concepts needed to explain our workflow. For a comprehensive introduction, the reader is referred to [130]. In reinforcement learning, an agent learns by interacting with an environment over discrete time steps. The interaction is formalized as a Markov decision process (MDP). At each action time step t , the environment occupies a state s_t , and the agent selects an action $a_t \sim \pi_\theta(\cdot | s_t)$, where π_θ is the agent policy. This policy maps states to a probability distribution over possible actions, specifying the likelihood of selecting each action given the current state. The selected action a_t is then applied to the environment, which responds by transitioning to a new state s_{t+1} according to

$$s_{t+1} = T(s_t, a_t), \quad (4.1)$$

where T encodes the underlying system dynamics (e.g., Navier–Stokes equations). The agent simultaneously receives a scalar reward

$$r_{t+1} = R(s_t, a_t, s_{t+1}), \quad (4.2)$$

which quantifies performance according to a user-defined metric. This cycle of observe–act–reward continues until the environment reaches a terminal state s_T , producing a trajectory

$$\tau = \{(s_0, a_0, r_1), (s_1, a_1, r_2), \dots, (s_{T-1}, a_{T-1}, r_T)\}. \quad (4.3)$$

The agent’s goal is to learn a policy π_θ , which maximizes the expected discounted return, defined from time t as

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}, \quad (4.4)$$

where the discount factor γ balances short- and long-term rewards. This quantity forms the basis of the state value function

$$V^\pi(s_t) = \mathbb{E}_\pi [G_t \mid s_t], \quad (4.5)$$

and state-action value function:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi [G_t \mid s_t, a_t]. \quad (4.6)$$

The SmartFlow training loop adheres to the classic reinforcement-learning paradigm of environment–agent interaction. Fig. 4.1 illustrates this workflow using two example environments, a cylinder flow environment for multi-agent synthetic-jet control for drag reduction and a turbulent channel flow environment for multi-agent reinforcement learning of LES wall models. In addition to the enforcing of invariants in the control law using MARL, training is accelerated by spawning several CFD simulations in parallel to collect more episodes within a given wall-clock time. At each decision time step t , each agent samples a low-dimensional observation s_t , such as probe pressure values in the cylinder flow control case, and velocity information at a fixed wall-normal height in the wall-modeling case. Based on the observation, the agent (i.e., thanks to MARL, a shared neural-network policy) computes an action a_t , such as jet-blowing/suction rate and wall-stress adjustment, and applies the action to the CFD environment. Then, it receives a scalar reward r_t that reflects instantaneous performance metrics such as skin-friction coefficient or drag reduction. During each episode, each environment–agent pair collects a trajectory $\{(s_t, a_t, r_{t+1})\}_{t=0}^{T-1}$. Once the preset horizon T is reached, experiences from all parallel instances are gathered to perform a policy update via a reinforcement learning algorithm. Over successive episodes, the shared policy steadily refines its mapping from flow observations to control actions, eventually converging on a strategy that maximizes the expected return as defined by Eq. (4.4).

SmartFlow employs two DRL techniques to accelerate training from a computational point of view. The first is to simulate multiple environments in parallel, also known as multi-environment DRL, so that the agent collects multiple experiences

in parallel[112]. The second, orthogonal to the multi-environment approach, is a multi-agent reinforcement-learning method. First introduced by Belus *et al.* [10] to fluid mechanics and later named as such in the context of flow control by Vignon *et al.* [142], MARL exploits the spatial invariance of the environment domain to reduce the dimensionality of the action space. For example, consider the channel-flow environment for wall modeling in the top panel of Fig. 4.1. Because the flow is statistically invariant in the streamwise and spanwise directions, each wall-surface cell can be treated as its own subdomain (or MARL pseudo-environment). Instead of predicting multiple actions simultaneously, individual agents, one per pseudo-environment, each predict a single local action. All agents share the same policy so that each individual learning experience is shared among all agents. This approach reduces the effective action-space dimensionality and avoids the curse of dimensionality. As reported by Suárez *et al.* [128] and Vignon *et al.* [142], using MARL enables the agent to effectively learn the dynamics of the system and a positive control strategy. If we define the number of parallel environments as N_e and the number of pseudo-environments per environment as N_{pe} , then $N_e N_{pe}$ episodes can be sampled in parallel. Consequently, SmartFlow dramatically shortens the data collection time, thus training time. Finally, it is important to highlight the definition of state, action, and reward in the multi-agent setup within the SmartFlow framework. The state is typically a local state vector, and the action is a single control vector for each pseudo-environment. However, the reward is flexible: it may be defined locally, globally, or as a combination of both.

The SmartFlow framework relies on model-free reinforcement learning, in which the policy is optimized directly through trial-and-error interactions, organized in episodes, rather than model-based reinforcement learning that relies on learning an explicit model of the dynamics of the environment. Model-free methods differ in how they gather experience and update the policy, and can be grouped into three broad classes: policy-gradient, value-function, and actor-critic algorithms. Policy-gradient algorithms parameterize the policy $\pi_\theta(a | s)$ and optimize θ to maximize expected return, while value-function algorithms estimate the value functions $V^\pi(s_t)$ and $Q^\pi(s_t, a_t)$. Actor-critic algorithms combine both approaches by learning a policy (actor) and a value function (critic) in tandem. Model-free algorithms can also be on-policy, using only data collected under the current policy, or off-policy, which reuse past experiences from a replay buffer. While off-policy methods often achieve higher sample efficiency, on-policy techniques tend to be simpler and more stable. In this work, we employ the clipped proximal policy optimization (PPO) algorithm [122], a model-free, on-policy actor-critic method.

4.2 Software architecture

SmartFlow is a single- and multi-agent deep reinforcement learning framework designed for a wide array of fluid-mechanics tasks, ranging from active flow control and shape optimization to autonomous navigation in complex flow fields and turbulence modeling. At its core lies the classic RL loop of environment-agent interaction, where agents observe states, take actions, and receive rewards. Built upon Relaxi [70] and SmartSOD2D [35], SmartFlow offers new features and enhancements to improve

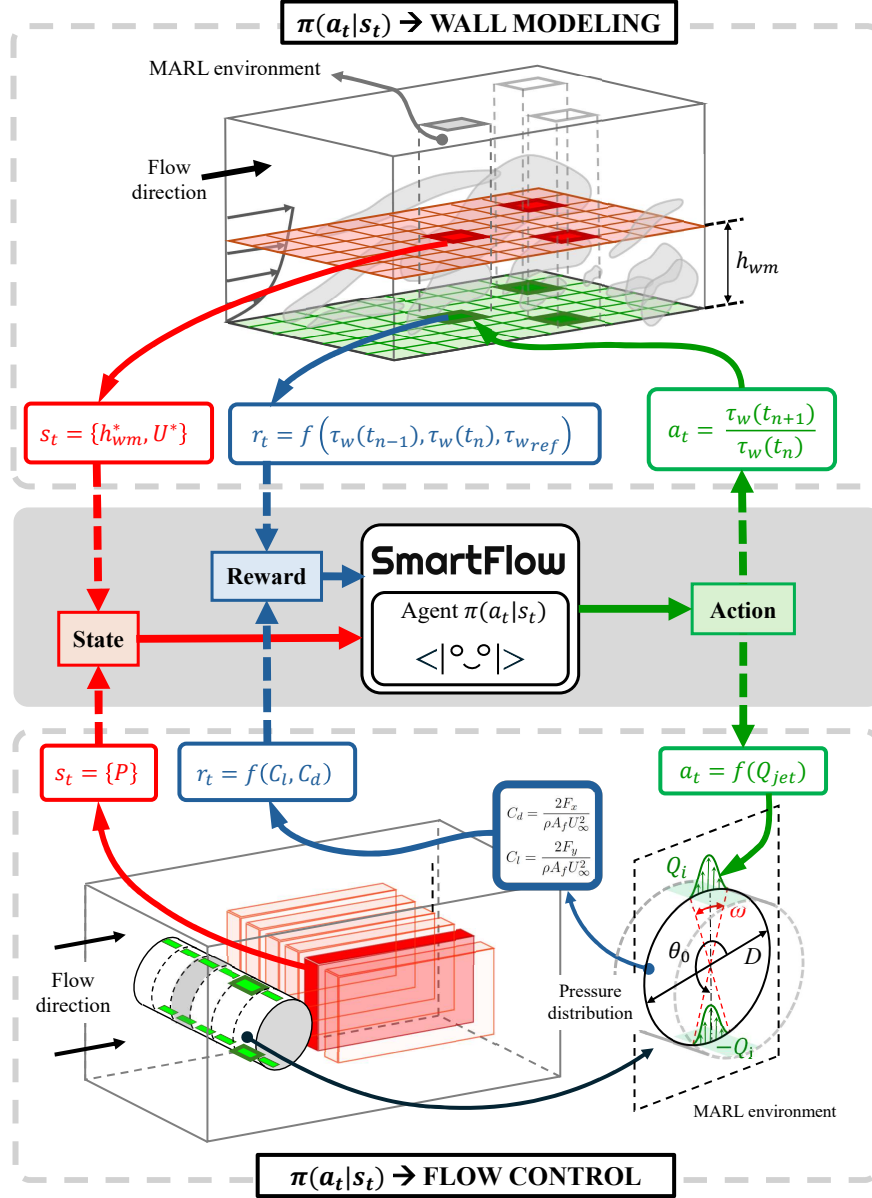


Figure 4.1. Schematic of the interaction between the SmartFlow framework (middle box) and two distinct CFD environments (top and bottom boxes) within a multi-agent deep reinforcement learning (MARL) setup. Top: In the wall-modeling case, each agent receives a local state $s_t^{(i)} = \{h_{wm}^*, U^*\}$. The shared policy $\pi(a_t | s_t)$ selects actions $a_t^{(i)} = \tau_w(t_{n+1})/\tau_w(t_n)$, and rewards are computed as $r_t^{(i)} = f(\tau_w(t_{n+1}), \tau_w(t_n), \tau_{w,ref})$. Bottom: In the flow-control case, agents positioned around a cylinder observe local pressure states $s_t^{(i)} = \{P\}$ and output control actions $a_t^{(i)} = f(Q_{jet})$, modulating the jet mass-flow rate to perturb the wake and reduce drag. Rewards are defined from aerodynamic coefficients, $r_t^{(i)} = f(C_l, C_d)$, where C_l and C_d are lift and drag coefficients. This illustration is to highlight that the middle-box SmartFlow framework can be easily connected to distinct environments easily. The illustration is inspired by Figure 1 in Suárez *et al.* [128].

modularity and performance. Table 4.1 shows the evolution of the SmartFlow framework from Relexi through SmartSOD2D. Its nearly CFD-solver-agnostic design relies on the SmartRedis-MPI library to bridge Fortran-based CFD codes and Python-based RL agents: any CFD solver (e.g., CaLES, FLEXI, SOD2D) can be easily integrated to the DRL framework. For example, the CaLES solver is integrated into the framework by adding only five lines of code. All reinforcement learning algorithms run in PyTorch via Stable-Baselines3, which provides a cleaner, more flexible API compared to TensorFlow’s TF-Agents. Both agents and CFD solvers can fully leverage GPU acceleration on modern HPC clusters. Defining custom environments in Python requires minimal boilerplate. Training metrics, including rewards, losses, and hyperparameters, are logged automatically to Weights & Biases or TensorBoard for end-to-end experiment tracking.

Figure 4.2 outlines the architecture of SmartFlow as a modular RL framework that implements an RL training loop for fluid-dynamics applications. Coupling HPC application codes with modern machine-learning libraries is often tedious. This is especially true for RL, where the interaction between the HPC application and the RL algorithm is intricate. A considerable part of these difficulties arises from different programming-language preferences and hardware requirements. Many existing approaches attempt to rewrite machine learning capabilities in HPC languages or re-implement HPC solvers in Python, but this requires rewriting large codebases and does not keep up with ML advances. The parameter update of the policy requires access to the collected trajectories to compute the gradients through backpropagation. At the same time, running CFD simulations requires high MPI parallel efficiency and high GPU speedup rates, in particular, for high-fidelity simulation of real engineering problems, thus compromising either side by code-rewriting is inadvisable. SmartFlow bridges this “two-language gap” by employing the SmartSim infrastructure library to couple a state-of-the-art RL library with modern HPC solvers. Its modular design allows changing RL algorithms and simulation environments with minimal changes to the underlying code. SmartFlow is designed for HPC systems and comprises the following building blocks:

1. SmartSim [102]: the in-memory orchestration and communication library,
2. SmartRedis-MPI: the data communication library developed based on SmartRedis,
3. CFD solver: any CFD solver for CFD environment simulation,
4. Stable-Baselines3 [114]: the RL algorithm library.

4.2.1 SmartSim

The SmartSim library [102] simplifies the coupling of HPC workloads with machine learning workflows. It comprises two core components: the SmartSim infrastructure library (IL) and SmartRedis. The IL handles job orchestration—launching, managing, and distributing MPI-parallel CFD simulations within a single batch allocation—and can submit jobs automatically to the scheduler. In SmartFlow, we leverage the IL to

Table 4.1. Evolution of the SmartFlow framework from Relexi through SmartSOD2D.

Framework	New features / enhancements	Key modifications
Relexi (Kurz <i>et al.</i> [70])	• Original scalable reinforcement learning framework	• SmartSim IL for orchestration on HPC • SmartRedis client for in-memory data communication • TF-Agents as the DRL backend • Coupled to the FLEXI discontinuous Galerkin solver • MPMD supported with OpenMPI via automatically generated rankfiles
SmartSOD2D (Font <i>et al.</i> [35])	• GPU-accelerated SOD2D solver support • Environment customization based on a generic CFD-environment base class • Reduced environment-startup overhead via parallel environment startup	• Fork of Relexi • Replace FLEXI with GPU-accelerated SOD2D solver • Customized environment class inherited from a general CFD environment • MPI's MPMD supported by automated allocation of resources on SLURM environments
SmartFlow (present)	• Improved modularity and usability • Nearly CFD-solver-agnostic • Cleaner and more flexible API for reinforcement learning • Enhanced support for GPU-accelerated CFD solver • Support for multi-task learning	• Fork of SmartSOD2D • CFD-solver-agnostic architecture by introducing the SmartRedis-MPI library • Switched from TF-Agents to Stable-Baselines3 • Enhanced compatibility with GPU-accelerated CFD solvers • Refined CFDEnv interface for easy environment customization • Logging via Weights & Biases

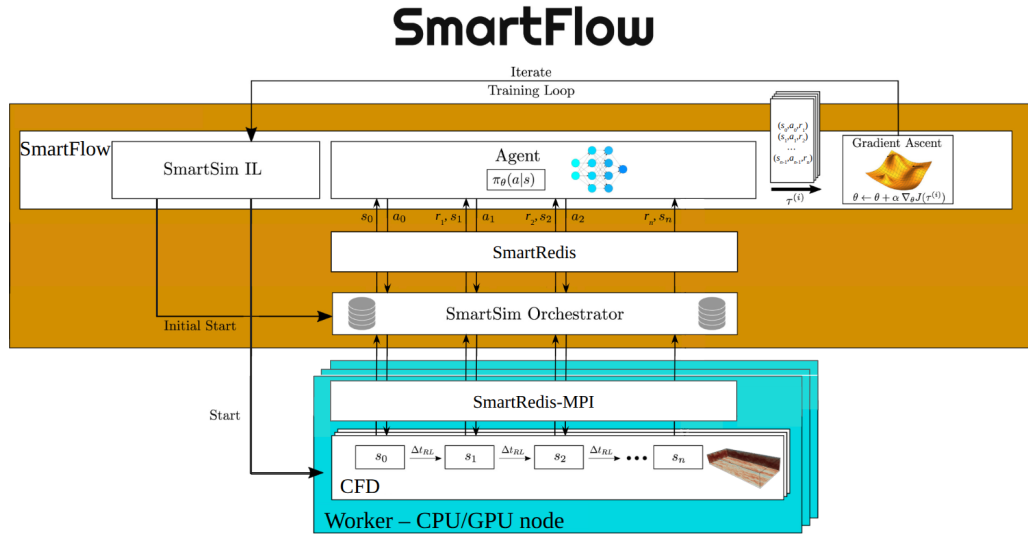


Figure 4.2. SmartFlow architecture. Before entering the training loop, the SmartSim IL is used to launch the SmartSim orchestrator, which provides an in-memory database for data exchange between SmartFlow and the HPC workload. At the start of each iteration, the SmartSim IL launches a batch of CFD instances and distributes them to the worker nodes. Each CFD instance could be initialized from a random state and then advanced in time. For every RL update interval Δt_{RL} , each CFD environment sends its current states to the agents and receives new actions. We apply a synchronous RL algorithm, which means that SmartFlow waits until all simulations have terminated, then aggregates the collected episodes $\tau^{(i)}$ and performs an update on the policy weights θ . Here, $\nabla_\theta J(\tau^{(i)})$ denotes the gradient of the expected return evaluated on the current batch of experience. The loop of collect–update is then repeated with the new policy until the policy converges. The figure is adapted from Kurz *et al.* [70].

launch the exact number of CFD environments needed for each rollout, individually or in parallel using MPI’s multi-program multi-data (MPMD) mode. While our current implementation opts for individual launches to maximize robustness, future releases will add MPMD support to further reduce CFD-environment-startup latency.

4.2.2 SmartRedis-MPI

We leverage another key feature of the SmartSim IL: its ability to configure and launch an in-memory, Redis-based database, known as the Orchestrator database, directly on the head node. SmartSim also supports the distribution of the Orchestrator across multiple nodes, but a single instance on the head node is commonly sufficient for our applications. All simulation processes communicate with this Orchestrator via SmartRedis clients (available in Python, Fortran, C, and C++), which support both data exchange and remote procedure calls (e.g., triggering preloaded ML inference routines). The SmartRedis-MPI library is a lightweight library that streamlines data exchange between the parallel CFD solver and Redis database by providing a small set of MPI-aware subroutines written on the basis of SmartRedis. With just a few lines of code, it enables any MPI-parallel CFD application to push and pull data from a Redis database. The SmartRedis-MPI library provides APIs tailored for reinforcement learning tasks. The core APIs in the library are listed in Table 4.2.

Table 4.2. Key API functions of the SmartRedis-MPI library.

Function	Description
<code>init_smartredis_mpi</code>	Initialize the MPI environment and creates a SmartRedis client connection on the master process
<code>put_state</code>	Collect the states for all agents across MPI ranks and send them to the Orchestrator datastore
<code>get_action</code>	Retrieve the control actions computed by the RL agent from the Orchestrator datastore and distribute them across MPI ranks
<code>put_reward</code>	Collect the rewards for all agents across MPI ranks and send them to the Orchestrator datastore
<code>finalize_smartredis_mpi</code>	Cleanly destroy the SmartRedis client connection on the master process

During training, each CFD instance uses SmartRedis-MPI to push flow states into the in-memory Orchestrator. The head-node SmartFlow process then pulls these states via the Python SmartRedis client, computes the next control actions, and writes them back into the Orchestrator. Finally, each CFD instance retrieves its assigned actions through SmartRedis-MPI. The SmartRedis-MPI library is the key to SmartFlow’s nearly CFD-solver-agnostic design.

4.2.3 CFD solver

A CFD solver is typically parallelized via MPI and may use finite-difference, finite-volume, spectral-element, or flux-reconstruction discretizations. By linking in the SmartRedis-MPI library, often with just a few extra lines of code, each rank can efficiently push its local states and rewards to, and pull control actions from, the shared Orchestrator datastore.

4.2.4 Stable-Baselines3

SmartFlow builds on the PyTorch-based Stable-Baselines3 library for RL algorithms. Custom training environments are created by subclassing the `VecEnv` API—overriding only `reset()`, `step_async()`, `step_wait()`, and a few utility methods—so that all of the CFD coupling details remain encapsulated within the environment class. This design lets any algorithm provided in Stable-Baselines3 (e.g. PPO, SAC, TD3) interact with the CFD solver. To further streamline development, SmartFlow provides a `CFDEnv` base class (itself a `VecEnv`), which implements a generic, asynchronous, vectorized CFD environment; users can simply subclass `CFDEnv` to implement their own specific CFD environments.

With the four building blocks in place, the SmartFlow workflow can be summarized as follows. Figure 4.2 depicts its modular architecture, and Algorithm 3 details the multi-agent RL loop. Before training, SmartSim IL launches the Orchestrator, an in-memory Redis (or KeyDB) datastore, to exchange data between SmartFlow and the HPC workload. At each iteration, the IL launches the required CFD instances, each linked via SmartRedis-MPI. Every instance pushes its initial states for all its agents to the Orchestrator, which SmartFlow then polls and reads. The current policy computes actions from these states and writes them back to the datastore. CFD instances poll for their actions, apply them, advance one control interval, and push the next resulting states and rewards. SmartFlow continues to poll, read, and act until the horizon of the episode is reached. Once a batch of episodes is collected, SmartFlow updates the policy and begins the next collect–update cycle. This process repeats until the policy converges. Crucially, a single batch of CFD instances can encompass multiple tasks, which might include distinct flow conditions or numerical setups, enabling multi-task RL; moreover, each task may include several parallel CFD instances within the same batch.

To further demonstrate the nearly CFD-solver-agnostic nature of SmartFlow and help users easily integrate their own solvers, we provide guidelines for coupling a MPI-parallel CFD solver with the SmartFlow framework. In most cases, only several lines of code are required. As an example, we present the integration of the CaLES solver [158] with SmartFlow to enable multi-agent deep reinforcement learning of wall models in large-eddy simulation. Listings 4.1 and 4.2 show a portion of the main program and the complete DRL-based wall model subroutine (`wallmodel_DRL`), respectively. Only five additional lines of code, highlighted in blue, are required for the integration. The 1st and 5th lines—`init_smartredis_mpi` and `finalize_smartredis_mpi`—are inserted into the main program to initialize and finalize the SmartRedis client. The 2nd, 3rd, and 4th lines—`put_state`, `put_reward`, and `get_action`—are added to the `wallmodel_DRL` subroutine to communicate with

Algorithm 3 SmartFlow multi-agent RL training workflow

```
1: Input:  $K$  (number of CFD instances),  $N$  (agents per CFD instance),  $S_{\max}$  (total
   environment steps),  $T$  (rollout horizon)
2:  $S \leftarrow 0$ 
3: Initialization
4: Launch SmartSim Orchestrator
5: while  $S < S_{\max}$  do
6:   Clear trajectory buffer  $\mathcal{B}$ 
7:   for  $k = 1$  to  $K$  do
8:     Start CFD instance  $k$ 
9:   end for
10:  for  $k = 1$  to  $K$  do
11:    Receive initial states  $\{s_0^{(k,i)}\}_{i=1}^N$  from CFD instance  $k$ 
12:  end for
13:  for  $t = 0$  to  $T - 1$  do
14:    for  $k = 1$  to  $K$  do
15:      Compute actions  $\{a_t^{(k,i)}\}_{i=1}^N$  via policy  $\pi_\theta$  from states  $\{s_t^{(k,i)}\}$ 
16:      Put  $\{a_t^{(k,i)}\}_{i=1}^N$  to the Orchestrator database
17:    end for
18:    Advance all CFD instances by one control time interval  $\Delta t_{RL}$ 
19:    for  $k = 1$  to  $K$  do
20:      Poll and get next states  $\{s_{t+1}^{(k,i)}\}_{i=1}^N$  and rewards  $\{r_{t+1}^{(k,i)}\}_{i=1}^N$  from the
      Orchestrator database
21:    end for
22:     $S \leftarrow S + K \times N$ 
23:  end for
24:  Update policy  $\pi_\theta$  on buffer  $\mathcal{B}$  for a fixed number of epochs
25: end while
26: Shutdown SmartSim Orchestrator
27: Finalization
```

the database. For clarity, GPU-specific code is omitted from Listings 4.1 and 4.2. The integration of the GPU-accelerated CaLES to SmartFlow is available in the CaLES GitHub repository: <https://github.com/CaNS-World/CaLES>.

In SmartFlow, a Redis database is used by default to host the Orchestrator. To improve the Orchestrator’s memory efficiency and throughput, the multi-threaded Redis fork KeyDB can be substituted as the underlying datastore—this has been supported since recent versions of SmartSim and was leveraged in the Relexi framework to improve parallel performance [70]. A potential bottleneck arises from the overhead of sequentially starting a large number of parallel environments across thousands of MPI ranks. In some scenarios (e.g. fast GPU-accelerated CFD simulations), this startup time can rival or even exceed the simulation runtime. This issue can be addressed by leveraging MPI’s multiple-program multiple-data (MPMD) mode, supported by SmartSim, to launch all simulations (each with its own command-line arguments) in a single MPI call. It also helps to copy required simulation files (parameter and restart files) to each node’s local RAM disk, which could reduce access times compared to a parallel file system like Lustre. In SmartFlow, we dropped mandatory MPMD support to preserve solver-agnosticism: requiring MPMD would force users to modify their CFD solver to support MPMD before integration. For simplicity, input files remain on the parallel file system. Future SmartFlow releases will offer optional support for these optimizations in performance-critical applications.

```

1 call MPI_INIT(ierr)
2 call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
3 call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
4 call init_smartredis_mpi(db_clustered, MPI_COMM_WORLD) ! Line 1 of 5
   lines of code
5
6 ...
7
8 call finalize_smartredis_mpi ! Line 5 of 5 lines of code
9 call MPI_FINALIZE(ierr)

```

Listing 4.1. Main program

```

1 subroutine wallmodel_DRL(visc, flattened_state, flattened_stress,
   flattened_metric)
2   implicit none
3   real(rp), intent(in) :: visc
4   type(FlattenedState), intent(in) :: flattened_state
5   type(FlattenedStress), intent(inout) :: flattened_stress
6   type(FlattenedMetric), intent(in), optional :: flattened_metric
7   real(rp) :: u1, u2, upar, tauw_tot, tauw1, tauw2, this_action
8   integer :: n_points, i
9   logical, save :: is_first = .true.
10  real(rp), allocatable, dimension(:,,:), save :: drl_state
11  real(rp), allocatable, dimension(:,,:), save :: drl_action
12  real(rp), allocatable, dimension(:,,:), save :: drl_reward
13  n_points = size(flattened_state%vel_x1)
14  if (is_first) then
15     allocate(drl_state(2, n_points))
16     allocate(drl_reward(2, n_points))
17     allocate(drl_action(1, n_points))
18  end if

```

```

19  drl_state(1, :) = flattened_state%hwm_plus
20  drl_state(2, :) = flattened_state%velh_plus
21  call put_state(trim(adjustl(tag))//".state", shape(drl_state),
    drl_state) ! Line 2 of 5 lines of code
22
23  if (is_first) then
24      is_first = .false.
25  else
26      drl_reward(1, :) = flattened_metric%tauw1
27      drl_reward(2, :) = flattened_metric%tauw1_prev
28      call put_reward(trim(adjustl(tag))//".reward", shape(drl_reward),
    drl_reward) ! Line 3 of 5 lines of code
29  end if
30
31  call get_action(trim(adjustl(tag))//".action", shape(drl_action),
    drl_action) ! Line 4 of 5 lines of code
32  flattened_stress%action(:, 1) = drl_action(1, :)
33  do i = 1, n_points
34      this_action = flattened_stress%action(i, 1)
35      tauw_tot = this_action * flattened_stress%tauw(i)
36      u1 = flattened_state%vel_x1(i)
37      u2 = flattened_state%vel_x2(i)
38      upar = sqrt(u1**2 + u2**2)
39      tauw1 = tauw_tot * u1 / (upar + eps)
40      tauw2 = tauw_tot * u2 / (upar + eps)
41      flattened_stress%tauw1(i) = tauw1
42      flattened_stress%tauw2(i) = tauw2
43  end do
44  end subroutine wallmodel_DRL

```

Listing 4.2. DRL-based wall model subroutine

SmartFlow has been tested across a range of computing environments, from local laptops and workstations to several HPC clusters. These systems include both CPU- and GPU-accelerated configurations, demonstrating the flexibility and portability of the framework. For instance, deployments have been carried out on GPU-enabled partitions such as Booster at CINECA (Italy), N32EA14P at BSCC (China), Alvis (Sweden) and TACC Stampede3 (USA), as well as on CPU-only systems like DCGP at CINECA and BSCC-A at BSCC.

To facilitate the use of the SmartFlow framework for DRL applications in fluid mechanics, we provide a set of usage examples that cover a range of computing architectures. Table 4.4 provides an overview of the example configurations. All examples focus on the same multi-agent DRL problem for LES wall modeling. Examples #1 and #2 are designed to run on a single laptop; Examples #3 and #4 on a standalone local server; Examples #5, #6, and #7 on a single node of an HPC cluster; and Examples #8 and #9 across multiple nodes (using two nodes in these examples). For each architecture, both CPU-based and GPU-accelerated solvers are considered. All examples use the $Re_\tau = 5200$ turbulent channel flow for training, except for the multi-task example #7 on a single node, where multiple Reynolds numbers are used: $Re_\tau = u_\tau h / \nu = 10^3, 10^5, 10^7$, and 10^{10} .

Table 4.3. Computing systems on which SmartFlow has been tested. NIC refers to the network interface card.

Cluster	Partition	Command Launcher	NIC	GPU acceleration	
Local laptop/server	/	local / mpirun	lo	Yes / No	
Single compute node	/	local / mpirun	lo	Yes / No	
CINECA (Italy)	Booster	srun	Slurm	lo / ib0	Yes
CINECA (Italy)	DCGP	srun	Slurm	lo / ib0	No
BSCC (China)	N32EA14P	srun	Slurm	lo / ib0	Yes
BSCC (China)	BSCC-A	srun	Slurm	lo / ib0	No
Alvis (Sweden)	/	srun	Slurm	lo	Yes
TACC Stampede3 (USA)	skx-dev	srun	Slurm	lo / ib0	Yes

4.3 Illustrative applications

We illustrate SmartFlow’s versatility through three case studies, each coupling a different CFD solver to an RL task in fluid mechanics. Two examples address active flow control: (1) single-agent drag reduction in 2D cylinder flow via zero-net-mass-flux synthetic jets, with the environments simulated via the high-order FLEXI solver (discontinuous-Galerkin spectral element), and (2) multi-agent drag reduction in 3D cylinder flow using the GPU-accelerated SOD2D solver (high-order spectral element). The third example tackles turbulence modeling, using multi-agent RL to train a wall model for large-eddy simulation with the GPU-accelerated CaLES solver (second-order finite-difference). In all studies, we employ proximal policy optimization (PPO) with a multilayer-perceptron (MLP) policy network. These demonstrations underscore SmartFlow’s support for both single- and multi-agent RL across a variety of CFD backends.

4.3.1 Single-agent RL for flow control

This demonstration case, following the approach of Rabault *et al.* [113], focuses on reducing both the drag and the amplitude of the unsteady lift oscillations acting on a cylinder in laminar flow. Both the drag and the lift fluctuations are significantly increased by vortex shedding behind the cylinder. Therefore, the main control objective is to suppress the vortex shedding. Given the well-characterized periodic nature of this phenomenon, the case represents a relatively simple control problem, making it a well-suited benchmark for RL-based active flow control.

Based on the “2D-2” benchmark case proposed by Schäfer *et al.* [121], the two-dimensional simulation environment features a circular cylinder of diameter D , placed within a channel of size $L \times H = 22D \times 4.1D$, where L and H are the sizes in the streamwise and wall-normal directions, respectively. The cylinder axis is vertically offset by $0.05D$ from the channel centerline to promote vortex shedding. A schematic of this configuration is shown in Fig. 4.3. At a Reynolds number $Re_D = U_b D / \nu = 100$ and a Mach number of $Ma = U_b / \sqrt{\gamma R T} = 0.2$, both defined

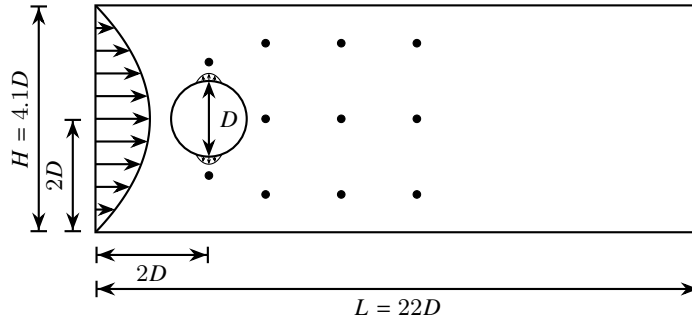
Table 4.4. SmartFlow usage examples provided in the SmartFlow GitHub repository. NIC refers to the network interface card, N_{cases} denotes the number of parallel cases (tasks) for training, N_{ctas} is the number of CFD simulations per case, and N_{procs} is the number of MPI processes used per CFD simulation.

ID	Example	System	Task type	Hardware	Command	Launcher	NIC	$N_{cases} \times N_{ctas} \times N_{procs}$
1	laptop_single_task_cpu	Laptop	Single-task	CPU	mpirun	local	lo	$1 \times 1 \times 1$
2	laptop_single_task_gpu	Laptop	Single-task	CPU+GPU	mpirun	local	lo	$1 \times 1 \times 1$
3	local_server_single_task_cpu	Local server	Single-task	CPU	mpirun	local	lo	$1 \times 4 \times 4$
4	local_server_single_task_gpu	Local server	Single-task	CPU+GPU	mpirun	local	lo	$1 \times 4 \times 1$
5	single_node_single_task_cpu	Single node	Single-task	CPU	mpirun	local	lo	$1 \times 4 \times 4$
6	single_node_single_task_gpu	Single node	Single-task	CPU+GPU	mpirun	local	lo	$1 \times 4 \times 1$
7	single_node_multi_task_gpu	Single node	Multi-task	CPU+GPU	mpirun	local	lo	$4 \times 1 \times 1$
8	multi_nodes_single_task_cpu	Multiple nodes	Single-task	CPU	srun	slurm	ib0	$1 \times 4 \times 4$
9	multi_nodes_single_task_gpu	Multiple nodes	Single-task	CPU+GPU	srun	slurm	ib0	$1 \times 4 \times 1$

Table 4.5. Illustrative applications of SmartFlow integrated with different CFD solvers

ID	Purpose	Number of agents	CFD solver	Details
1	Active flow control	Single	FLEXI	2D cylinder flow control through zero-net-mass-flux synthetic jets
2	Active flow control	Multiple	GPU-accelerated SOD2D	3D cylinder flow control through zero-net-mass-flux synthetic jets
3	Turbulence modeling	Multiple	GPU-accelerated CaLES	Wall model development for large-eddy simulation

with respect to D and the bulk velocity U_b , the flow behaves quasi-incompressible. This ensures consistency with the reference results of Rabault *et al.* [113], despite the use of the compressible solver FLEXI [68].

**Figure 4.3.** Simulation setup for the flow around a two-dimensional cylinder. Pressure probe locations are indicated by black circles.

The control employs two synthetic jets, each following a cosinusoidal velocity distribution along the cylinder's angular coordinate φ . The profile is symmetrically confined to an angular width ω centered at $\varphi_i = \pm\pi/2$, such that

$$u_{\perp,i}(\varphi) = \begin{cases} Q_i \frac{\pi}{2\omega D^2} \cos\left(\frac{\pi}{\omega}(\varphi - \varphi_i)\right), & |\varphi - \varphi_i| \leq \frac{\omega}{2} \\ 0, & |\varphi - \varphi_i| > \frac{\omega}{2} \end{cases}. \quad (4.7)$$

Zero total mass flow rate is enforced by requiring $\sum_i^n Q_i = 0$, where Q_i denotes the mass flow rate of the i -th of n jets. For the present $n = 2$ case, this constraint reduces the control input to a single scalar quantity $Q = Q_1 = -Q_2$. The nondimensional control input is then defined as $Q^* = Q/Q_D$, where Q_D is the reference mass flow rate, computed as the integral of the inflow mass flux over the cylinder's projection

onto the inflow boundary. A comprehensive description of the simulation setup is provided in Kurz *et al.* [73], with additional technical details discussed by Rabault *et al.* [113].

In this simulation environment, we search for an optimal control strategy using single-agent reinforcement learning. Both the policy and the value function are represented by MLPs with two hidden layers of 64 neurons each, employing a hyperbolic tangent activation function. Since MLPs inherently rely on the input ordering, they do not require explicit positional encoding in the state representation. The state consists of relative pressure values measured at 11 probes distributed near the jets and within the cylinder wake, as illustrated in Fig. 4.3. Based on this state, a scalar action is sampled from a Gaussian distribution parameterized by the policy network, reflecting the control input Q^* . The sampling is restricted to a bounded action space of $|Q^*| \leq 0.068$ to prevent non-physical control inputs and numerical instabilities near the actuation region. For training, a modified version of the reward function of Rabault *et al.* [113] is used, which is based on the ratio of the achieved to the expected maximum drag reduction, rather than the drag itself. This normalizes the drag reduction values to be within approximately $[0, 1]$, and also increases the sensitivity of the reward signal to drag changes, thereby facilitating the convergence of training. Building on the approach introduced by Cavallazzi *et al.* [18], the reward is defined as

$$r_t = \frac{\langle c_D \rangle^{UC} - \langle c_D \rangle}{\langle c_D \rangle^{UC} - \langle c_D \rangle^{EC}} - \alpha |\langle c_L \rangle|, \quad (4.8)$$

where c_D and c_L are the drag and lift coefficients, respectively, with respect to the reference length D and the bulk velocity U_b . The operator $\langle \cdot \rangle$ represents the exponential temporal moving average, emphasizing the most recent values. Superscripts UC and EC refer to the uncontrolled reference case and the expected minimum under effective control, respectively.

The training was performed using a full batch comprising 26 CFD environments in parallel, each generating a trajectory of 80 steps per episode. The simulations were initialized from randomly selected initial fields drawn from a predefined set of four, and then proceeded for 20 non-dimensional time units, $t^* = tU_b/D$. The complete training consisted of 400 episodes and required approximately 4.6 hours on 52 AMD EPYC 7543 cores. After that, the learned policy reached sufficient convergence for demonstration purposes.

The evolution of the averaged return is shown in Fig. 4.4, which demonstrates fast convergence. The pronounced fluctuations in the average return indicate that the policy has not yet fully converged. The fluctuation may also be influenced by the elevated learning rate, which is set five times higher than in prior work of Kurz *et al.* [73]. Despite different convergence rates, the final average returns are comparable. These differences in convergence speed are attributed to variations in the PPO implementations and parameter setups.

To evaluate the long-term behavior of the learned policy, a greedy evaluation run was conducted for a time period up to $t^* = 100$. The results, shown in Fig. 4.5, demonstrate clear improvements over the uncontrolled case in both objectives: drag reduction and mitigation of lift oscillations. However, the reference policy exhibits a notably different behavior. While our MLP generally surpasses the reference in drag reduction, it induces a pronounced shift toward negative lift coefficients. This could

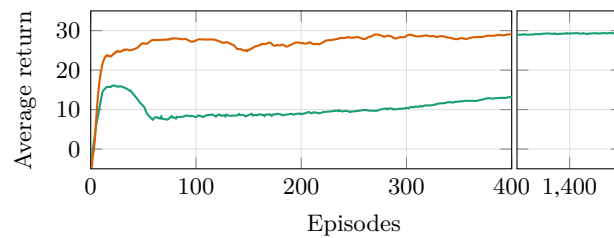


Figure 4.4. Evolution of the average training return for the MLP policies of Kurz *et al.* [73] (—) and the present study (—). The x -axis is truncated to skip intermediate episodes. The plot resumes at episode 1350 to show the converged values of the reference.

likely be reduced, if desirable, by increasing the lift penalization coefficient α in the reward function, as highlighted by Rabault *et al.* [113].

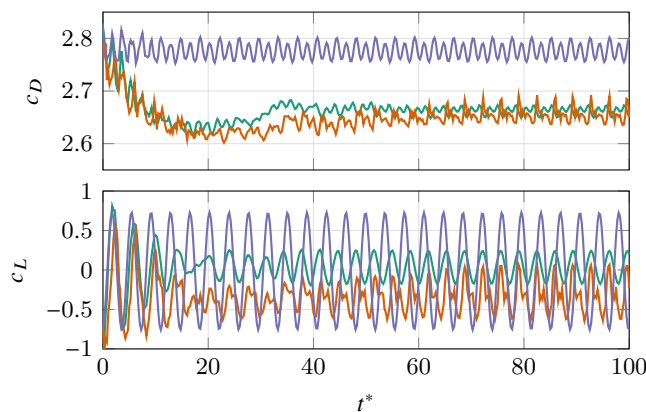


Figure 4.5. Long-term evolution of the drag (c_D) and lift (c_L) coefficients, comparing the uncontrolled case (—) to the MLP results from Kurz *et al.* [73] (—) and the present study (—).

4.3.2 Multi-agent RL for flow control of bluff body

This case illustrates the flow-control application of SmartFlow integrated with the spectral-element SOD2D solver. The target is to use DRL for active flow control (AFC) to reduce drag in bluff bodies, which is of great interest in many engineering applications. While single-agent DRL has shown great potential in simple AFC applications, especially for 2D flows [142], it faces limitations due to the curse of dimensionality when dealing with 3D flows exhibiting complex dynamics [10]. The multi-agent reinforcement learning mitigates this challenge by dividing a domain into smaller subdomains (i.e., pseudo-environments), making high-dimensional AFC problems more tractable. This setup enables policies to exploit spatial invariance and optimize local rewards, thereby enhancing exploration efficiency.

We consider a 3D cylinder at $Re = 100$ ($Re = U_\infty D / \nu$, where U_∞ is the inflow velocity, D is the cylinder diameter, and ν is the kinematic viscosity). This setup has been investigated by Suárez *et al.* [129] and Montalà *et al.* [88]. The flow is governed

by the incompressible Navier–Stokes equations. The computational domain spans $L_x = 30D$, $L_y = 15D$, and $L_z = 4D$ in the streamwise, cross-stream, and spanwise directions, respectively. The cylinder is placed at $(x, y) = (7.5D, 7.5D)$ and extends the full length L_z in the spanwise direction. At the inlet, a constant free-stream velocity U_∞ is prescribed, while zero-gradient conditions with constant pressure are applied at the outlet. Slip conditions are enforced on the top and bottom surfaces, and periodic boundary conditions are applied in the spanwise direction.

Figure 4.1 illustrates the MARL configuration applied to the cylinder. The AFC is achieved through ten pairs of wall-mounted actuators ($n_{\text{jet}} = 10$) which are uniformly distributed along the spanwise direction of the cylinder. Each pair consists of one actuator on the top surface ($\theta_0 = 90^\circ$) and one on the bottom ($\theta_0 = 270^\circ$). The actuators have a spanwise width of $0.4D$ and an angular width of $\omega = 10^\circ$ in the $x - y$ plane. To ensure instantaneous mass conservation, each actuator pair operates under a zero-net-mass-flux (ZNMF) condition by imposing opposite mass flow rates: $Q_{\text{top}} = -Q_{\text{bot}}$. Each actuator pair defines one pseudo-environment, resulting in ten pseudo-environments in total. The action of the DRL agent is the mass flow rate Q , which determines the imposed velocity at the wall via the following Dirichlet boundary condition:

$$||U_{\text{jet}}(Q, \theta)|| = Q \frac{\pi}{\rho D \omega} \cos\left(\frac{\pi}{\omega}(\theta - \theta_0)\right), \quad (4.9)$$

where $Q = \dot{m}/L_z$ and $|\theta - \theta_0| \in [-\omega/2, \omega/2]$, $\dot{m}$ is the mass flow rate. The absolute value of the jet velocity is projected into the x and y axes, since $||U_{\text{jet}}||$ corresponds to the radial cylinder direction. To balance the learning efficiency and numerical stability, $|Q| \leq 0.176$ was selected based on the prior study [129].

Each pseudo-environment includes 85 witness points located around the cylinder wall at its local mid-span (z -middle) position. Due to the periodic boundary condition in the spanwise direction, the state of each pseudo-environment also includes its two neighboring ones, resulting in $3 \times 85 = 255$ points per state vector.

The lift and drag coefficients are defined as $C_l = l/(1/2\rho U_\infty^2 S)$ and $C_d = d/(1/2\rho U_\infty^2 S)$, respectively, where l and d denote the lift and drag forces, respectively. The reference area S is defined based on the cylinder diameter D and the spanwise length of the pseudo-environment. Taking C_l and C_d into account, the reward of a pseudo-environment R_i is defined as

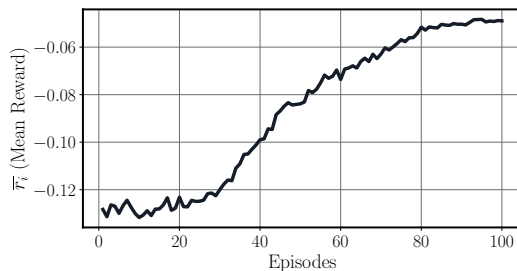
$$r_i = (C_{d,\text{ref}} - C_d) - \alpha|C_l| \quad (4.10)$$

$$R_i = (1 - \beta)r_i + \frac{\beta}{n_{\text{jets}}} \sum_{j=1}^{n_{\text{jets}}} r_j, \quad (4.11)$$

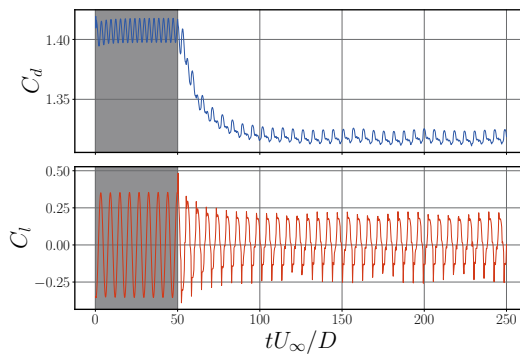
where $C_{d,\text{ref}}$ denotes the reference C_d , and α and β are weighting parameters. In particular, α weights the penalty of C_l , while β balances the contributions of local and global rewards. We use $\alpha = 0.3$ and $\beta = 0.2$ in the present study.

The PPO algorithm is used for training the DRL agent, with both actor and critic modeled as multi-layer perceptrons comprising two hidden layers of 512 neurons each. Each episode lasts $T_{\text{episode}} \approx 34.88 D/U_\infty$, equivalent to approximately six vortex shedding cycles, during which 120 actions are applied ($T_{\text{action}} = T_{\text{episode}}/120$). We trained the agent for 100 episodes. To improve sampling efficiency, four CFD

simulations are performed in parallel. Since each simulation contains 10 pseudo-environments, 40 trajectories are collected in total per policy update, resulting in a batch size of 40. Hyperparameters are chosen based on [129, 88] to reproduce the results. However, those studies used TF-Agents or Tensorforce, while the present work adopts Stable-Baselines3. Since PPO implementations can differ across libraries, some variation in performance can be expected. However, detailed hyperparameter tuning is beyond the scope of this demonstration, whose primary goal is to illustrate the functionality of the SmartFlow framework.



(a) Training curve of policy network during exploration.



(b) Temporal evolution of C_l and C_d before and after activating DRL control in deterministic mode.

Figure 4.6. (a) Training curve of the policy networks for drag-reduction control using SOD2D. Note that the $\bar{r}_i$ denotes the mean reward of all trajectories for a single episode. (b) Temporal evolution of the drag (C_d) and lift coefficient (C_l) before and after applying DRL control in deterministic mode, where the gray regions denote the uncontrolled evolution of C_d and C_l . Note that tU_∞/D is the dimensionless time unit.

The mean reward history is shown in Figure 4.6a. It shows that the curve converges to approximately -0.04 in the final 20 episodes. This value is slightly lower than that in Montalà *et al.* [88], likely due to differences in PPO implementations. The training was performed on a single node with four NVIDIA A100 GPUs and required approximately 40 hours for 100 episodes.

The learned policy is evaluated in deterministic mode. Figure 4.6b presents the time evolution of C_d and C_l before and after DRL activation. Initially, an uncontrolled simulation is run for $50D/U_\infty$ to establish baseline values (gray regions). Once the control is activated, significant reductions in both C_d and C_l are observed. The control is maintained for an additional $200D/U_\infty$ to ensure convergence. Relative to the baseline, the mean drag coefficient is reduced by 6.0%, and the lift coefficient

fluctuation (i.e., standard deviation) by 46.6%, demonstrating the effectiveness of MARL for 3D AFC applications.

4.4 Conclusions

Deep reinforcement learning is rapidly growing into a powerful general-purpose toolbox for fluid dynamics, from active flow control and autonomous navigation to shape optimization, turbulence modeling and the discovery of new numerical algorithms. In this work, we introduced SmartFlow, a single- and multi-agent RL framework designed to operate seamlessly on modern HPC clusters and to interface with a wide variety of CFD solvers with minimal boilerplate. Built on top of Relexi and SmartSOD2D, SmartFlow leverages the SmartSim infrastructure library to orchestrate CFD simulations and the SmartRedis client for in-memory data exchange via the Orchestrator datastore. Key features include an in-memory SmartRedis-MPI data-transfer layer between the CFD solver and the Orchestrator datastore, full support for GPU-accelerated CFD solvers, and a modular Gym-style environment interface that can be subclassed for environment customization in Python. Crucially, SmartFlow is effectively CFD-solver-agnostic, making it straightforward to integrate with any CPU- or GPU-accelerated CFD code.

To demonstrate the solver-agnostic design and versatility of SmartFlow, we integrated three distinct CFD backends and trained RL agents on three fluid-mechanics problems. First, we applied a single-agent policy to zero-net-mass-flux synthetic-jet control in cylinder flow using the high-order FLEXI solver. Second, we extended this setup to a 3D-flow multi-agent case with the GPU-accelerated SOD2D spectral-element solver. Third, we developed a multi-agent wall model for large-eddy simulation by coupling SmartFlow to the finite-difference CaLES solver to learn accurate wall-shear predictions across a wide range of Reynolds numbers. In each case, SmartFlow’s in-memory orchestration and asynchronous data exchange incurred minimal overhead.

SmartFlow thus offers a nearly solver-agnostic framework for deploying both single- and multi-agent RL in fluid mechanics on modern GPU-based HPC infrastructure, lowering the barrier to entry for researchers seeking to leverage reinforcement learning as a tool. In future work, we will further enhance SmartFlow and the SmartRedis-MPI library to improve data-exchange efficiency in large-scale RL training.

Chapter 5

Reinforcement learning for wall modeling in large-eddy simulation

We explore the application of deep reinforcement learning to develop wall models for large-eddy simulation. Specifically, we employ multi-task multi-agent reinforcement learning (MARL) to train a wall model using the SmartFlow framework, which is integrated with the GPU-accelerated CaLES solver. The training leverages multi-task learning across turbulent channel flows at multiple Reynolds numbers to enhance the generalizability of the learned model. The robustness of the resulting wall model is systematically evaluated by testing across a wide range of flow conditions and varying the wall-model matching heights.

5.1 Training setup

The wall model is trained in turbulent channel flows at $Re_\tau = u_\tau h / \nu = 10^3, 10^5, 10^7$ and 10^{10} , where h is the channel half-height, u_τ the friction velocity, and ν the kinematic viscosity. The domain size is $L_x = 6.4h$, $L_y = 2.0h$, $L_z = 2.4h$ with $\Delta x = \Delta z = 0.05h$, and the wall-normal mesh stretches from $\Delta y_w = 0.0125h$ at the wall to $\Delta y_c = 0.050h$ at the center. Rather than a single Reynolds number, we include four Reynolds numbers during training, hoping the learned model can generalize across a wide range of Re_τ . At each rollout, four CFD runs, one per Reynolds number, are executed in parallel (additional instances per Re_τ can be launched if desired). This is the so-called multi-task learning.

Each wall-modeled LES run is initialized with the velocity field obtained from the same Reynolds-number WMLES simulation with the equilibrium log-law wall model, with its initial wall-shear stress randomized between 80% and 120% of the true value. The simulations are driven by constant pressure gradients, so the mean wall shear stress is known *a priori*. For each CFD environment, the matching location h_{wm} is randomly selected between $0.075h$ and $0.150h$ to ensure training over a smooth range of inner-scaled matching locations within the log layer. The velocity is then interpolated to the chosen wall-normal location h_{wm} to form the state vector. Agents are arranged with spacings $\Delta x_{RL} = 4\Delta x$ and $\Delta z_{RL} = 4\Delta z$, so there are 768 agents

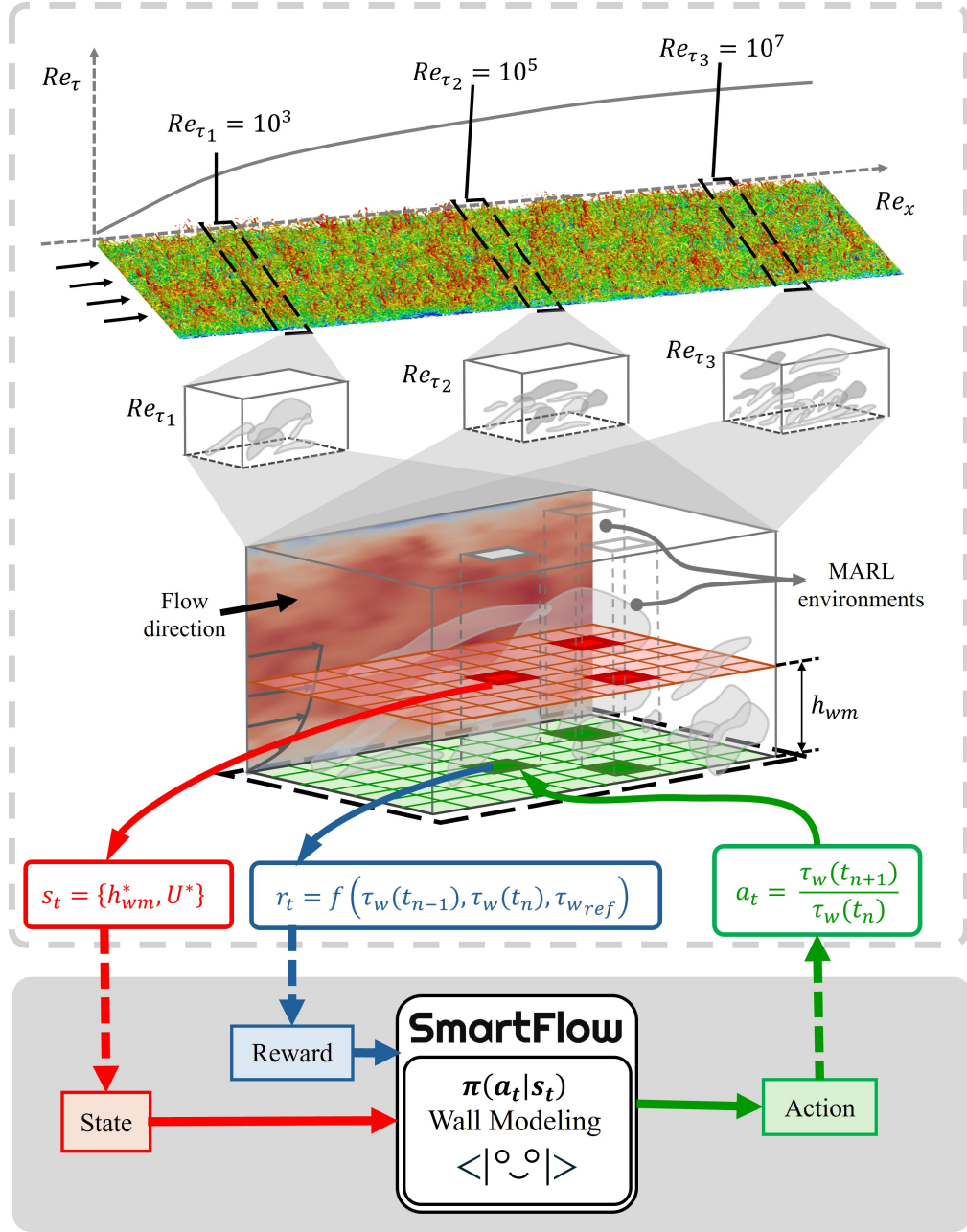


Figure 5.1. Schematic of the multi-task multi-agent reinforcement learning (MT-MARL) setup. Top: Zero-pressure-gradient turbulent boundary layer. The curve shows the variation of Re_τ with Re_x , i.e., $Re_\tau \sim Re_x^{0.7}$. The turbulent boundary layer is used to demonstrate the variation of Re_τ in the streamwise direction. The true range of Re_τ values at the marked locations do not match the marked values. Middle: Turbulent channel flows at different Re_τ values, as marked in the top turbulent boundary layer at different streamwise locations. Bottom: The interaction between the agents and the environments. The agents share a single policy neural network. The agents receive local states $s_t^{(i)} = \{h_{wm}^*, U_{wm}^*\}$. The shared policy $\pi(a_t | s_t)$ selects actions $a_t^{(i)} = \tau_w(t_{n+1})/\tau_w(t_n)$, and rewards are computed as $r_{t+1}^{(i)} = f(\tau_w(t_{n+1}), \tau_w(t_n), \tau_{w,ref})$. In the MT-MARL setup, the training includes multiple tasks, i.e., channel flows at multiple Re_τ values. For each time step, the states and rewards are obtained from a range of turbulent channel flows, and the shared policy samples actions, which are imposed on the corresponding channel environments. The goal is to train a single policy that is valid for different flow environments.

per CFD environment. Each training iteration advances the CFD simulations for 120 action time steps ($120\Delta t_{RL}$, with $\Delta t_{RL}u_b/h \approx 0.4$). Prior to agent interaction, each CFD run undergoes a warm-up of $\Delta t u_b/h = 96$ with fixed wall-shear stress (initialized wall-stress values) to remove the initial transients.

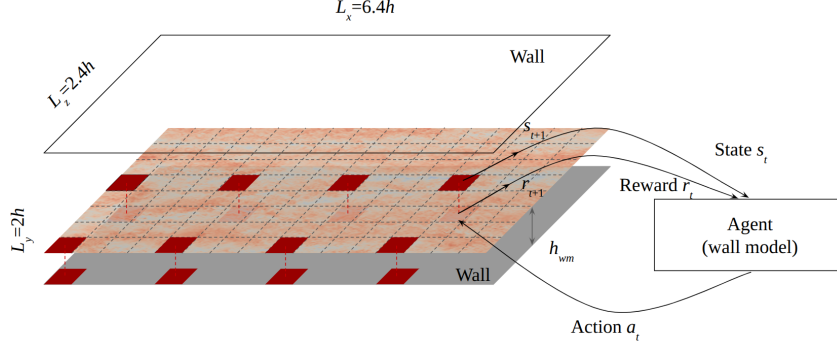


Figure 5.2. Channel environment setup. Agents are distributed evenly along the wall; each agent obtains state information at a wall-normal height h_{wm} , computes the reward at the wall, and provides the state to the policy π_θ , which then outputs the action for the next time step. Dashed lines denote grid lines. The figure is inspired by the illustration in Bae and Koumoutsakos [7].

We employ the clipped proximal policy optimization (PPO) algorithm [122], a model-free, on-policy actor-critic method. The policy is represented by a multilayer perceptron with two hidden layers of 64 units each and Tanh activations. At each time step, the network outputs the mean μ_θ and log-standard deviation $\log \sigma_\theta$ of a diagonal Gaussian, from which continuous actions are sampled. We train the network using the Adam optimizer with a fixed learning rate of 3×10^{-4} , a minibatch size of 368640 transitions (i.e. the full batch), and 10 gradient-descent epochs per policy update. Each action from the policy network is interpreted as a multiplicative factor $a_n(x, z) \in [0.9, 1.1]$ on the current wall-shear stress [7],

$$a_n(x, z) = \frac{\tau_w(x, z, t_{n+1})}{\tau_w(x, z, t_n)}. \quad (5.1)$$

The reward is defined as [7]

$$r_{n+1}(x, z) = - \left(\frac{|\tau_w^{ref} - \tau_w(x, z, t_{n+1})| - |\tau_w^{ref} - \tau_w(x, z, t_n)|}{\tau_w^{ref}} \right) + \text{Bonus}, \quad (5.2)$$

where

$$\text{Bonus} = \mathbb{I} \left(\frac{|\tau_w^{ref} - \tau_w(x, z, t_{n+1})|}{\tau_w^{ref}} < 0.01 \right), \quad (5.3)$$

where $\mathbb{I}$ is an indicator function and τ_w^{ref} is the true mean wall-shear stress. This gives a reward that is proportional to the improvement in the prediction of the wall-shear stress compared to the one obtained in the previous time step with an

additional reward if the predicted wall-shear stress is within 1% of the true value. These action and reward definitions follow [7], while the state vector definition differs from [7]: rather than using log-law slope and intercept computed from the instantaneous LES flow information at time t_n , we take the natural logarithm of the wall-normal height of the matching point $\ln(h_{wm}^*)$ and the instantaneous wall-parallel velocity magnitude $U^*(h_{wm})$, where the superscript $*$ denotes normalization by the wall units at time t_n . The state vector is

$$s_n(x, z) = (\ln(h_{wm}^*(x, z, t_n)), U_{wm}^*(x, z, t_n)). \quad (5.4)$$

The filtered incompressible Navier–Stokes equations are solved by CaLES using wall-modeled LES with a staggered, second-order finite-difference spatial discretization, a fractional-step projection method, and a third-order low-storage Runge–Kutta time integrator. Subgrid-scale stresses are closed with the classical Smagorinsky model, with model coefficient $C_s = 0.11$, augmented by a van Driest damping function near the wall. For the channel flows, periodic boundary conditions are imposed in the streamwise and spanwise directions, while wall-stress boundary conditions with no-penetration are imposed at the top and bottom walls. The wall-shear stress τ_w computed via Eq. (5.1) is first evaluated at wall-cell centers; its components $(\tau_{w,x}, \tau_{w,z})$ are then obtained by assuming the stress vector aligns with the wall-parallel velocity at the matching point. These components are interpolated to the midpoints of wall-cell edges and enforced as Neumann boundary conditions on the LES domain:

$$\tau_{w,x} = \nu \frac{\partial u}{\partial y}, \tau_{w,z} = \nu \frac{\partial w}{\partial y}, \quad (5.5)$$

where the subscript w denotes evaluation at the wall. Further details on CaLES’s wall-modeled LES implementation and validation are provided in [158].

5.2 Results

The training comprises 800 CFD runs, yielding a total of 614,400 agent episodes (768 episodes per CFD instance). All simulations were carried out on a single node of the CINECA Leonardo Booster partition, where each node features an Intel Xeon Platinum 8358 CPU (2.60 GHz, 32 cores) and four NVIDIA A100 GPUs (64 GB HBM2e) interconnected by NVLink 3.0 (200 GB/s intra-node bandwidth). Each batch of four concurrent CFD simulations completed in approximately 12 seconds, and the full training required only 40 minutes, thanks to the GPU-accelerated CaLES solver.

Figure 5.3 plots the agent’s episode reward averaged over each batch of four simulations. Convergence is effectively reached after about 153600 agent episodes (i.e. 200 CFD runs), although we completed all 800 runs to guarantee full convergence. We then evaluated the final wall model (trained over 800 runs). In fact, its performance is nearly identical to the model obtained after 200 runs. Figure 5.4 presents the mean velocity profiles from evaluation runs driven by a constant pressure gradient, with the wall-model matching height fixed at $h_{wm} = 0.1h$. Nine Reynolds numbers were tested, $Re_\tau = 10^3, 5200, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9$, and 10^{10} . In all cases, LES

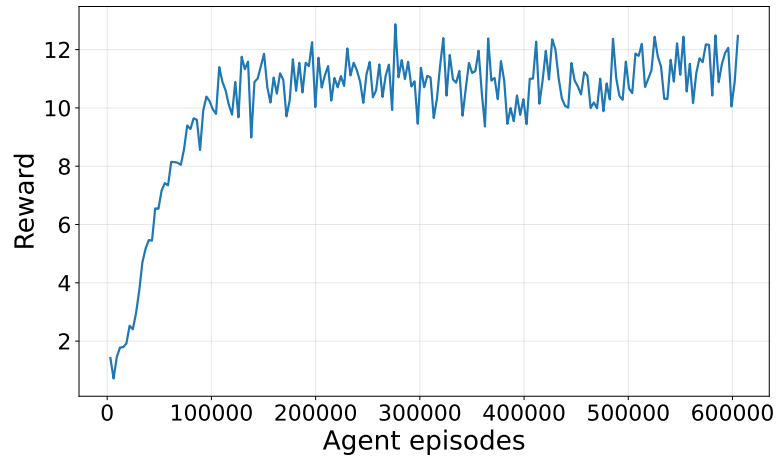


Figure 5.3. Evolution of the agent's episode reward versus agent episode number. The reward is averaged over one batch of CFD simulations; the values are raw and no moving average is applied.

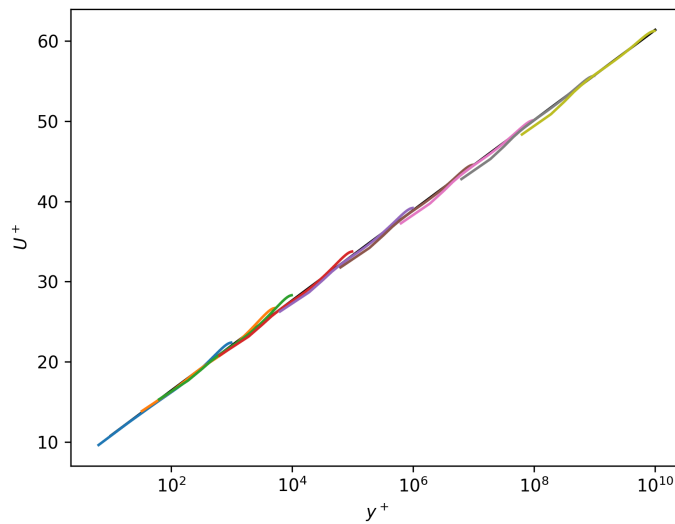


Figure 5.4. Mean streamwise velocity U^+ as a function of the wall-normal distance y^+ , where the superscript $+$ indicates normalization by mean wall units. The different colors from bottom left to top right denote $Re_\tau = 10^3, 5200, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}$. The log law (black line) corresponds to $u^+ = 1/\kappa \ln(y^+) + B$, with $\kappa = 0.41$ and $B = 5.2$.

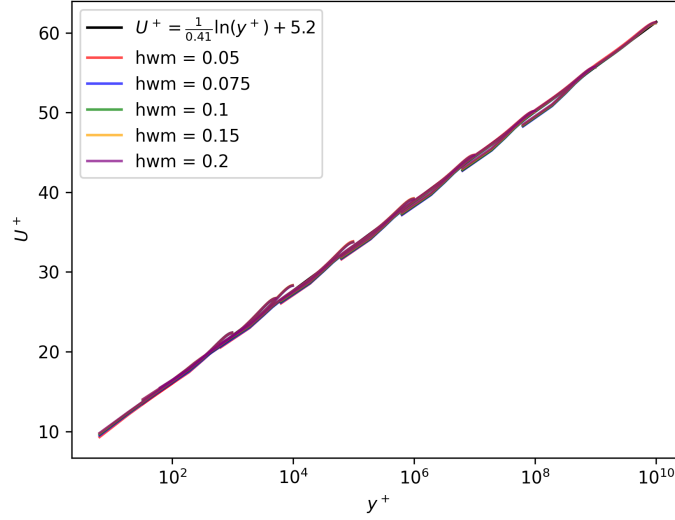


Figure 5.5. Mean streamwise velocity U^+ as a function of the wall-normal distance y^+ , where the superscript $+$ indicates normalization by mean wall units. The different colors from bottom left to top right denote $Re_\tau = 10^3, 5200, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}$. The log law (black line) corresponds to $u^+ = 1/\kappa \ln(y^+) + B$, with $\kappa = 0.41$ and $B = 5.2$. The wall-modeled layer thickness is set as $h_{wm} = 0.05h, 0.075h, 0.1h, 0.15h, 0.2h$.

coupled with the trained wall model has reliably reproduced the reference profiles up to $Re_\tau = 10^{10}$.

To assess the sensitivity of the trained wall model to the matching height, we conducted additional evaluation runs at five different wall-modeled layer thicknesses: $h_{wm} = 0.05h, 0.075h, 0.1h, 0.15h$, and $0.2h$. Figure 5.5 shows the resulting mean velocity profiles for each matching height across the same nine Reynolds numbers tested previously. The trained wall model demonstrates consistent accuracy across all matching heights, successfully reproducing the reference velocity profiles in each configuration. This robustness to variations in h_{wm} confirms the model’s applicability to LES across different layer thicknesses.

5.3 Conclusions

This chapter explored deep reinforcement learning as a methodology for developing wall models in large-eddy simulation of wall-bounded turbulent flows. By employing multi-task multi-agent reinforcement learning within the SmartFlow framework coupled with the GPU-accelerated CaLES solver, we trained a wall model across turbulent channel flows at multiple Reynolds numbers. The multi-task approach enhanced model generalizability, enabling the trained wall model to perform robustly across diverse flow conditions. Evaluations at various Reynolds numbers and wall-model matching heights confirmed the model’s robustness, demonstrating the viability of reinforcement learning for developing robust turbulence closures in LES.

Chapter 6

Conclusions

This research presents contributions to the field of large-eddy simulation of wall-bounded turbulence. First, we perform direct numerical simulation of turbulent pipe flow up to $Re_\tau \approx 3000$ with steady axial rotation to quantify drag reduction effects and elucidate the underlying physical mechanisms. Second, we develop CaLES, a high-performance GPU-accelerated LES solver incorporating advanced subgrid-scale models and numerical methods, validated through simulations of decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow. Third, we introduce SmartFlow, a versatile and nearly CFD-solver-agnostic deep reinforcement learning framework for computational fluid dynamics on HPC platforms, demonstrated through single-agent synthetic-jet control in cylinder wake, multi-agent wake control, and multi-task multi-agent wall-model learning. Fourth, we apply multi-task multi-agent deep reinforcement learning to develop wall models for large-eddy simulation of wall-bounded turbulence, with emphasis on the robustness and generalizability of the trained model.

1. Direct numerical simulations have been performed to study axially rotating pipe flow up to nominal friction Reynolds number $Re_{\tau,0} = 3020$. The drag reduction rate, which we show to be equivalent to net power saving in the fully developed scenario assuming no mechanical losses, increases with both N and $Re_{\tau,0}$, becoming a sole function of the inner-scaled wall rotational speed at high enough Reynolds numbers. The drag reduction is as high as about 70% at $N = 4.0$ for $Re_{\tau,0} = 3020$, although the flow is still far from full relaminarization. We have developed a theoretical analysis informed by DNS data which allows for quantitative prediction of this effect, based on the observation that the rotational speed becomes the relevant velocity scale in the core flow. The analysis yields a predictive formula for the friction coefficient (2.18), which yields accurate approximation of the DNS data up to $N \approx 1$, regardless of the bulk Reynolds number, which however breaks down once the logarithmic layer in the mean axial velocity profile is disrupted.

Analysis of the instantaneous velocity fields reveals the role of rotation in weakening the near-wall sweeps and ejections and in the elongation and broadening of the streaks. In the core region, rotation leads to turbulence suppression at moderate rotation numbers and high Reynolds numbers, resulting in the disappearance of the outer-layer peak in the pre-multiplied spectra. Turbulence

is instead intensified at moderate rotation numbers and low Reynolds numbers, and a distinct outer-layer peak or plateau, emerges in the pre-multiplied spectra. Nevertheless, a consistent decrease of the turbulent shear stress is observed. Use of the FIK identity reveals that the turbulent drag reduction originating from the outer layer asymptotically approaches the total turbulent drag reduction with increasing $Re_{\tau,0}$, consistent with a theoretically derived formula, and conveys that both the inner and outer layers increasingly contribute to drag reduction as N increases.

It is finally important to acknowledge opportunities and challenges in applying wall rotation in practical contexts. On the positive side, it is clear from this paper that large drag reduction is possible without reverting to complicated wall actuation rules. In particular, we note that at a statistically steady state, the work required to sustain rotation is zero (leaving mechanical losses aside), and drag reduction effects increase with the Reynolds number, unlike in conventional wall actuation techniques. On the negative side, setting a full pipeline into rotation may not be an easy task. In addition, we find that the achievement of a fully developed state requires a longer distance than for non-rotating flows (typically, a few hundred diameters), and energy must be spent during this transient state to put the whole system into rotation. This energy expenditure should be accounted for in the evaluation of axial rotation for realistic applicative scenarios.

2. We have introduced CaLES, a GPU-accelerated solver designed for large-eddy simulation of incompressible wall-bounded flows. Based on the GPU-accelerated DNS solver CaNS, CaLES demonstrates significant computational efficiency improvements through GPU parallelization using a combination of CUDA Fortran and OpenACC directives, and good scalability on massively parallel architectures. The incorporation of SGS models, including static and dynamic Smagorinsky models, along with a wall model, extends the solver's applicability to wall-resolved and wall-modeled LES.

Performance evaluation conducted on state-of-the-art high-performance computing clusters, shows that CaLES achieves substantial speed-ups using GPU acceleration compared to its CPU-only counterparts. The solver efficiently scales across multiple GPUs, achieving approximately 15× speed-up on a single GPU compared to a full CPU node, making it highly effective for high-fidelity simulations on large computational grids. Validation cases—such as decaying isotropic turbulence, turbulent channel flow, and turbulent duct flow—confirm the solver's accuracy for LES, with the dynamic Smagorinsky model exhibiting somewhat superior performance in grid convergence and turbulence prediction.

CaLES has also been applied for wall-modeled LES of turbulent channel and duct flows. Its high computational efficiency enables WMLES for channels with $Re_{\tau} = 5200$ on progressively refined grids, meeting the grid requirements for wall-resolved LES. A key observation is the non-monotonic grid convergence in WMLES for channel and duct flows, particularly in wall friction, streamwise velocity fluctuations, and turbulent shear stresses. These results underscore the complexities inherent in WMLES and emphasize the need for continued

research into SGS and wall models.

In summary, CaLES offers the computational efficiency and flexibility needed to investigate turbulent flows with moderate complexity at high Reynolds numbers. Its open-source availability makes it a valuable tool for fast simulations, which is particularly important in the application of machine learning to develop robust SGS and wall models, including data-driven approaches and reinforcement learning.

Finally, we should emphasize that the purpose of this work is not to carry out a physical investigation of the performance of SGS models and wall models. Instead, our aim is to incorporate well-established models into a baseline solver that can easily incorporate other models, such as the Vreman subgrid-scale model [147] and the ordinary differential equation-based wall model [63].

3. Deep reinforcement learning is rapidly growing into a powerful general-purpose toolbox for fluid dynamics, from active flow control and autonomous navigation to shape optimization, turbulence modeling and the discovery of new numerical algorithms. In this work, we introduced SmartFlow, a single- and multi-agent RL framework designed to operate seamlessly on modern HPC clusters and to interface with a wide variety of CFD solvers with minimal boilerplate. Built on top of Relexi and SmartSOD2D, SmartFlow leverages the SmartSim infrastructure library to orchestrate CFD simulations and the SmartRedis client for in-memory data exchange via the Orchestrator datastore. Key features include an in-memory SmartRedis-MPI data-transfer layer between the CFD solver and the Orchestrator datastore, full support for GPU-accelerated CFD solvers, and a modular Gym-style environment interface that can be subclassed for environment customization in Python. Crucially, SmartFlow is effectively CFD-solver-agnostic, making it straightforward to integrate with any CPU- or GPU-accelerated CFD code

To demonstrate the solver-agnostic design and versatility of SmartFlow, we integrated three distinct CFD backends and trained RL agents on three fluid-mechanics problems. First, we applied a single-agent policy to zero-net-mass-flux synthetic-jet control in cylinder flow using the high-order FLEXI solver. Second, we extended this setup to a 3D-flow multi-agent case with the GPU-accelerated SOD2D spectral-element solver. Third, we developed a multi-agent wall model for large-eddy simulation by coupling SmartFlow to the finite-difference CaLES solver to learn accurate wall-shear predictions across a wide range of Reynolds numbers. In each case, SmartFlow’s in-memory orchestration and asynchronous data exchange incurred minimal overhead.

SmartFlow thus offers a nearly solver-agnostic framework for deploying both single- and multi-agent RL in fluid mechanics on modern GPU-based HPC infrastructure, lowering the barrier to entry for researchers seeking to leverage reinforcement learning as a tool. In future work, we will further enhance SmartFlow and the SmartRedis-MPI library to improve data-exchange efficiency in large-scale RL training.

4. We investigated the use of deep reinforcement learning to create wall models for

large-eddy simulation of wall-bounded turbulent flows. We applied multi-task multi-agent reinforcement learning to train the wall model within the Smart-Flow framework, coupled with the GPU-accelerated CaLES solver. Training was conducted across turbulent channel flows spanning multiple Reynolds numbers to improve model generalizability. The trained wall model’s robustness was assessed through testing over diverse Reynolds numbers and different wall-model matching heights.

Bibliography

- [1] *OpenACC Programming and Best Practices Guide*, May 2021. URL https://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0.pdf. Accessed: October 14, 2024.
- [2] LESGO: A parallel pseudo-spectral large-eddy simulation code, 2024. URL <https://lesgo.me.jhu.edu>.
- [3] R. Agrawal, M. P. Whitmore, K. P. Griffin, S. T. Bose, and P. Moin. Non-Boussinesq subgrid-scale model with dynamic tensorial coefficients. *Physical Review Fluids*, 7(7):074602, 2022.
- [4] A. A. Amsden. The SMAC method: a numerical technique for calculating incompressible fluid flows. *Los Alamos Sci. Lab. Rep.*, 1970.
- [5] C. P. Arroyo, J. Dombard, F. Duchaine, L. Gicquel, B. Martin, N. Odier, and G. Staffelbach. Towards the large-eddy simulation of a full engine: Integration of a 360 azimuthal degrees fan, compressor and combustion chamber. Part I: Methodology and initialisation. *Journal of the Global Power and Propulsion Society*, (Special issue):133115, 2021.
- [6] F. Auteri, A. Baron, M. Belan, G. Campanardi, and M. Quadrio. Experimental assessment of drag reduction by traveling waves in a turbulent pipe flow. *Phys. Fluids*, 22, 2010.
- [7] H. J. Bae and P. Koumoutsakos. Scientific multi-agent reinforcement learning for wall-models of turbulent flows. *Nature Communications*, 13(1):1443, 2022.
- [8] H. J. Bae and A. Lozano-Duran. Numerical and modeling error assessment of large-eddy simulation using direct-numerical-simulation-aided large-eddy simulation. *arXiv preprint*, 2022.
- [9] A. Beck and M. Kurz. Toward discretization-consistent closure schemes for large eddy simulation using reinforcement learning. *Physics of Fluids*, 35(12), 2023.
- [10] V. Belus, J. Rabault, J. Viquerat, Z. Che, E. Hachem, and U. Reglade. Exploiting locality and translational invariance to design effective deep reinforcement learning control of the 1-dimensional unstable falling liquid film. *AIP Advances*, 9(12), 2019.

-
- [11] M. Beneitez, A. Cremades, L. Guastoni, and R. Vinuesa. Improving turbulence control through explainable deep learning, 2025. URL <https://arxiv.org/abs/2407.17032>.
 - [12] I. Bermejo-Moreno, L. Campo, J. Larsson, J. Bodart, D. Helmer, and J. K. Eaton. Confinement effects in shock wave/turbulent boundary layer interactions through wall-modelled large-eddy simulations. *Journal of Fluid Mechanics*, 758:5–62, 2014.
 - [13] M. Bernardini, D. Modesti, F. Salvatore, S. Sathyanarayana, G. Della Posta, and S. Pirozzoli. STREAmS-2.0: Supersonic turbulent accelerated Navier-Stokes solver version 2.0. *Computer Physics Communications*, 285:108644, 2023.
 - [14] R. Bhaskaran, R. Kannan, B. Barr, and S. Priebe. Science-guided machine learning for wall-modeled large eddy simulation. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1809–1816. IEEE, 2021.
 - [15] S. T. Bose and G. I. Park. Wall-modeled large-eddy simulation for complex turbulent flows. *Annual review of Fluid Mechanics*, 50(1):535–561, 2018.
 - [16] E. Bou-Zeid, C. Meneveau, and M. B. Parlange. Large-eddy simulation of neutral atmospheric boundary layer flow over heterogeneous surfaces: Blending height and effective surface roughness. *Water Resources Research*, 40(2), 2004.
 - [17] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52(1):477–508, 2020.
 - [18] G. M. Cavallazzi, L. Guastoni, R. Vinuesa, and A. Pinelli. Deep reinforcement learning for the management of the wall regeneration cycle in wall-bounded turbulent flows. *Flow, Turbulence and Combustion*, pages 1–27, 2024.
 - [19] X. Chen, F. Hussain, and Z. She. Quantifying wall turbulence via a symmetry approach. Part 2. Reynolds stresses. *J. Fluid Mech.*, 850:401–438, 2018. ISSN 0022-1120 1469-7645. doi: 10.1017/jfm.2018.405.
 - [20] R. Cocle, L. Bricteux, and G. Winckelmans. Scale dependence and asymptotic very high Reynolds number spectral behavior of multiscale subgrid models. *Physics of Fluids*, 21(8), 2009.
 - [21] G. Comte-Bellot and S. Corrsin. Simple Eulerian time correlation of full-and narrow-band velocity signals in grid-generated, ‘isotropic’ turbulence. *Journal of Fluid Mechanics*, 48(2):273–337, 1971.
 - [22] P. Costa. A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. *Computers & Mathematics with Applications*, 76(8):1853–1862, 2018.
 - [23] P. Costa, E. Phillips, L. Brandt, and M. Fatica. GPU acceleration of CaNS for massively-parallel direct numerical simulations of canonical fluid flows. *Computers & Mathematics with Applications*, 81:502–511, 2021.

-
- [24] J. Davis, S. Ganju, N. Ashton, S. Bailey, and C. Brehm. A DNS Study to Investigate Turbulence Suppression in Rotating Pipe Flows. In *AIAA Aviation 2019 Forum*, 2019.
 - [25] F. De Vanna and G. Baldan. URANOS-2.0: Improved performance, enhanced portability, and model extension towards exascale computing of high-speed engineering flows. *Computer Physics Communications*, page 109285, 2024.
 - [26] J. W. Deardorff. A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. *Journal of Fluid Mechanics*, 41(2):453–480, 1970.
 - [27] S. Deck, F. Gand, V. Brunet, and S. Ben Khelil. High-fidelity simulations of unsteady civil aircraft aerodynamics: stakes and perspectives. Application of zonal detached eddy simulation. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2022):20130325, 2014.
 - [28] P. Drazin and W. Reid. *Hydrodynamic Stability*. Cambridge University Press, 2nd edn edition, 2004.
 - [29] C. Duprat, G. Balarac, O. Métais, P. M. Congedo, and O. Brugière. A wall-layer model for large-eddy simulations of turbulent flows with/out pressure gradient. *Physics of fluids*, 23(1), 2011.
 - [30] D. Dupuy, N. Odier, and C. Lapeyre. Data-driven wall modeling for turbulent separated flows. *Journal of Computational Physics*, 487:112173, 2023.
 - [31] K. Duraisamy. Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence. *Physical Review Fluids*, 6(5):050504, 2021.
 - [32] K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51(1):357–377, 2019.
 - [33] A. Feiz, M. Ould-Rouis, and G. Lauriat. Turbulence Statistics in a Fully Developed Rotating Pipe Flow. *J. Enhanc. Heat Transf.*, 12:273–288, 2005.
 - [34] P. Fischer, S. Kaltenbach, S. Litvinov, S. Succi, and P. Koumoutsakos. Optimal Lattice Boltzmann Closures through Multi-Agent Reinforcement Learning, 2025.
 - [35] B. Font, F. Alcántara-Ávila, J. Rabault, R. Vinuesa, and O. Lehmkuhl. Deep reinforcement learning for active flow control in a turbulent separation bubble. *Nature Communications*, 16(1):1422, 2025.
 - [36] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.* An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

-
- [37] L. M. Fuchs, J. G. von Saldern, T. L. Kaiser, and K. Oberleithner. Deep reinforcement learning-augmented Spalart–Allmaras turbulence model: Application to a turbulent round jet flow. *Fluids*, 9(4):88, 2024.
 - [38] K. Fukagata, K. Iwamoto, and N. Kasagi. Contribution of Reynolds stress distribution to the skin friction in wall-bounded flows. *Phys. Fluids*, 14: L73–L76, 2002.
 - [39] M. Gamahara and Y. Hattori. Searching for turbulence models by artificial neural network. *Physical Review Fluids*, 2(5):054604, 2017.
 - [40] X. Garcia, A. Miró, P. Suárez, F. Alcántara-Ávila, J. Rabault, B. Font, O. Lehmkuhl, and R. Vinuesa. Deep-reinforcement-learning-based separation control in a two-dimensional airfoil. *International Journal of Heat and Fluid Flow*, 116:109913, 2025. ISSN 0142-727X.
 - [41] P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, 2021.
 - [42] D. Gatti and M. Quadrio. Reynolds-number dependence of turbulent skin-friction drag reduction induced by spanwise forcing. *J. Fluid Mech.*, 802: 553–582, 2016. ISSN 0022-1120 1469-7645. doi: 10.1017/jfm.2016.485.
 - [43] M. Germano, U. Piomelli, P. Moin, and W. H. Cabot. A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A: Fluid Dynamics*, 3(7):1760–1765, 1991.
 - [44] K. Goc, S. T. Bose, and P. Moin. Large eddy simulation of the NASA high-lift common research model. In *AIAA SciTech 2022 Forum*, page 1556, 2022.
 - [45] K. A. Goc, O. Lehmkuhl, G. I. Park, S. T. Bose, and P. Moin. Large eddy simulation of aircraft at affordable cost: a milestone in computational fluid dynamics. *Flow*, 1:E14, 2021.
 - [46] F. Godeferd and F. Moisy. Structure and dynamics of rotating turbulence: a review of recent experimental and numerical results. *Appl. Mech. Rev.*, 67: 030802, 2015.
 - [47] H. Greenspan. *The Theory of Rotating Fluids*. Cambridge University Press, 1968.
 - [48] K. P. Griffin and L. Fu. A new ODE-based turbulence wall model accounting for pressure gradient and Reynolds number effects. *arXiv preprint arXiv:2010.04097*, 2020.
 - [49] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, *et al.* TF-Agents: A library for reinforcement learning in TensorFlow, 2018. URL <https://github.com/tensorflow/agents>.

-
- [50] L. Guastoni, J. Rabault, P. Schlatter, H. Azizpour, and R. Vinuesa. Deep reinforcement learning for turbulent drag reduction in channel flows. *The European Physical Journal E*, 46(4):27, 2023.
 - [51] P. Gunnarson, I. Mandralis, G. Novati, P. Koumoutsakos, and J. O. Dabiri. Learning efficient navigation in vortical flow fields. *Nature communications*, 12(1):7143, 2021. ISSN 2041-1723.
 - [52] F. H. Harlow, J. E. Welch, *et al.* Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182, 1965.
 - [53] L. Heimbach, S. Kaltenbach, P. Karnakov, F. J. Alexander, and P. Koumoutsakos. Reinforcement Learning Closures for Underresolved Partial Differential Equations using Synthetic Data, 2025. URL [arXivpreprintarXiv:2505.11308](https://arxiv.org/abs/2505.11308).
 - [54] S. Hickel, E. Touber, J. Bodart, and J. Larsson. A parametrized non-equilibrium wall-model for large-eddy simulations. In *Eighth International Symposium on Turbulence and Shear Flow Phenomena*. Begel House Inc., 2013.
 - [55] G. Hoffmann and C. Benocci. Approximate wall boundary conditions for large eddy simulations. In *Advances in Turbulence V: Proceedings of the Fifth European Turbulence Conference, Siena, Italy, 5–8 July 1994*, pages 222–228. Springer, 1995.
 - [56] R. Hu, X. Li, and C. Yu. Transfers of energy and helicity in helical rotating turbulence. *J. Fluid Mech.*, 946:A19, 2022. ISSN 0022-1120 1469-7645. doi: 10.1017/jfm.2022.580.
 - [57] X. L. Huang, X. I. Yang, and R. F. Kunz. Wall-modeled large-eddy simulations of spanwise rotating turbulent channels—Comparing a physics-based approach and a data-based approach. *Physics of Fluids*, 31(12), 2019.
 - [58] M. Hultmark, M. Vallikivi, S. Bailey, and A. Smits. Turbulent pipe flow at extreme Reynolds numbers. *Phys. Rev. Lett.*, 108:094501, 2012. ISSN 1079-7114 (Electronic) 0031-9007 (Linking). doi: 10.1103/PhysRevLett.108.094501. URL <https://www.ncbi.nlm.nih.gov/pubmed/22463643>.
 - [59] E. Hurst, Q. Yang, and Y. Chung. The effect of Reynolds number on turbulent drag reduction by streamwise travelling waves. *J. Fluid Mech.*, 759:28–55, 2014. ISSN 0022-1120 1469-7645. doi: 10.1017/jfm.2014.524.
 - [60] N. Hutchins and I. Marusic. Evidence of very long meandering features in the logarithmic region of turbulent boundary layers. *J. Fluid Mech.*, 579:1–28, 2007. ISSN 0022-1120 1469-7645. doi: 10.1017/s0022112006003946.
 - [61] J. Jiménez and A. Pinelli. The autonomous cycle of near-wall turbulence. *J. Fluid Mech.*, 389:335–359, 1999. ISSN 0022-1120 1469-7645. doi: 10.1017/s0022112099005066.

-
- [62] M. Kang, Y. Jeon, and D. You. Neural-network-based mixed subgrid-scale model for turbulent flow. *Journal of Fluid Mechanics*, 962:A38, 2023.
 - [63] S. Kawai and J. Larsson. Wall-modeling in large eddy simulation: Length scales, grid resolution, and accuracy. *Physics of Fluids*, 24(1):015105, 2012.
 - [64] J. Keim, A. Schwarz, S. Chiocchetti, C. Rohde, and A. Beck. A Reinforcement Learning Based Slope Limiter for Two-Dimensional Finite Volume Schemes. In *Finite Volumes for Complex Applications X—Volume 2, Hyperbolic and Related Problems*, volume 433, pages 209–217. Springer, Cham, 2023.
 - [65] K. Kikuyama, M. Murakami, K. Nishibori, and K. Maeda. Flow in an Axially Rotating Pipe: A calculation of flow in the saturated region. *B. JSME*, 26: 506–513, 1983.
 - [66] J. Kim and P. Moin. Application of a fractional-step method to incompressible Navier-Stokes equations. *Journal of Computational Physics*, 59(2):308–323, 1985.
 - [67] C. Koh, L. Pagnier, and M. Chertkov. Physics-guided actor-critic reinforcement learning for swimming in turbulence. *Physical Review Research*, 7(1):013121, 2025. ISSN 2643-1564.
 - [68] N. Krais, A. Beck, T. Bolemann, H. Frank, D. Flad, G. Gassner, F. Hindenlang, M. Hoffmann, T. Kuhn, M. Sonntag, and C.-D. Munz. FLEXI: A high order discontinuous Galerkin framework for hyperbolic-parabolic conservation laws. *Computers & Mathematics with Applications*, 81:186–219, Jan. 2021. ISSN 08981221. URL <https://linkinghub.elsevier.com/retrieve/pii/S0898122120302029>.
 - [69] M. Kurz, P. Offenhäuser, D. Viola, M. Resch, and A. Beck. Relexi—A scalable open source reinforcement learning framework for high-performance computing. *Software Impacts*, 14:100422, 2022. ISSN 2665-9638.
 - [70] M. Kurz, P. Offenhäuser, D. Viola, O. Shcherbakov, M. Resch, and A. Beck. Deep reinforcement learning for computational fluid dynamics on HPC systems. *Journal of Computational Science*, 65:101884, 2022. ISSN 1877-7503.
 - [71] M. Kurz, P. Offenhäuser, and A. Beck. Deep reinforcement learning for turbulence modeling in large eddy simulations. *International journal of heat and fluid flow*, 99:109094, 2023.
 - [72] M. Kurz, A. Beck, and B. Sanderse. Harnessing equivariance: Modeling turbulence with graph neural networks, 2025. URL [arXivpreprintarXiv:2504.07741](https://arxiv.org/abs/2504.07741).
 - [73] M. Kurz, R. Kaushik, M. Blind, P. Kopper, A. Schwarz, F. Rodach, and A. Beck. Invariant Control Strategies for Active Flow Control using Graph Neural Networks, 2025. URL [arXivpreprintarXiv:2503.22775](https://arxiv.org/abs/2503.22775).

-
- [74] C. Lagemann, L. Paehler, J. Callahan, S. Mokbel, S. Ahnert, K. Lagemann, E. Lagemann, N. Adams, and S. Brunton. HydroGym: A Reinforcement Learning Platform for Fluid Dynamics. In *7th Annual Learning for Dynamics & Control Conference*, volume 283, pages 497–512. PMLR, 2025.
 - [75] J. Larsson, S. Kawai, J. Bodart, and I. Bermejo-Moreno. Large eddy simulation with modeled wall-stress: recent progress and future directions. *Mechanical Engineering Reviews*, 3(1):15–00418, 2016.
 - [76] 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.
 - [77] R. Li, Y. Zhang, and H. Chen. Learning the aerodynamic design of supercritical airfoils through deep reinforcement learning. *AIAA Journal*, 59(10):3988–4001, 2021. ISSN 0001-1452, 1533-385X.
 - [78] D. Lilly. A proposed modification of the Germano sugrid-scale closure method. *Physics of Fluids A: Fluid Dynamics*, 4:633–635, 1992.
 - [79] D. K. Lilly. The representation of small-scale turbulence in numerical simulation experiments. In *Proc. IBM Sci. Comput. Symp. on Environmental Science*, pages 195–210, 1967.
 - [80] A. Lozano-Durán and H. J. Bae. Self-critical machine-learning wall-modeled LES for external aerodynamics. *arXiv preprint arXiv:2012.10005*, 2020.
 - [81] A. Lozano-Durán and H. J. Bae. Machine learning building-block-flow wall model for large-eddy simulation. *Journal of Fluid Mechanics*, 963:A35, 2023.
 - [82] Y. Mao, S. Zhong, and H. Yin. DRLFluent: A distributed co-simulation framework coupling deep reinforcement learning with Ansys-Fluent on high-performance computing systems. *Journal of Computational Science*, 74:102171, 2023. ISSN 1877-7503.
 - [83] P. Mason and N. Callen. On the magnitude of the subgrid-scale eddy coefficient in large-eddy simulations of turbulent channel flow. *Journal of Fluid Mechanics*, 162:439–462, 1986.
 - [84] T. Mauery, J. Alonso, A. Cary, V. Lee, R. Malecki, D. Mavriplis, G. Medic, J. Schaefer, and J. Slotnick. A guide for aircraft certification by analysis. Technical report, 2021.
 - [85] C. Meneveau and T. S. Lund. The dynamic Smagorinsky model and scale-dependent coefficients in the viscous range of turbulence. *Physics of Fluids*, 9(12):3932–3934, 1997.
 - [86] J. Meyers and P. Sagaut. Is plane-channel flow a friendly case for the testing of large-eddy simulation subgrid-scale models? *Physics of Fluids*, 19(4):048105, 2007.

-
- [87] A. Mole, M. Weissenbacher, G. Rigas, and S. Laizet. Reinforcement Learning Increases Wind Farm Power Production by Enabling Closed-Loop Collaborative Control. *arXiv preprint arXiv:2506.20554*, 2025.
 - [88] R. Montalà, B. Font García, P. Suárez Morales, J. Rabault, O. Lehmkuhl Barba, R. Vinuesa Motilva, and I. M. Rodríguez Pérez. Towards active flow control strategies through deep reinforcement learning. In *ECCOMAS: 9th European Congress on Computational Methods in Applied Sciences and Engineering: Lisbon, Portugal, 3rd-7th June*. International Centre for Numerical Methods in Engineering (CIMNE), 2024.
 - [89] R. Montalà, B. Font, P. Suárez, J. Rabault, O. Lehmkuhl, R. Vinuesa, and I. Rodriguez. Deep Reinforcement Learning for Active Flow Control around a Three-Dimensional Flow-Separated Wing at $Re = 1,000$. In *Proceedings of the 1st International Symposium on AI and Fluid Mechanics*, Chania, Greece, 2025.
 - [90] N. Moriya, K. Fukami, Y. Nabae, M. Morimoto, T. Nakamura, and K. Fukagata. Inserting machine-learned virtual wall velocity for large-eddy simulation of turbulent channel flows. *arXiv preprint arXiv:2106.09271*, 2021.
 - [91] F. Nicoud, J. Baggett, P. Moin, and W. Cabot. Large eddy simulation wall-modeling based on suboptimal control theory and linear stochastic estimation. *Physics of fluids*, 13(10):2968–2984, 2001.
 - [92] F. Nicoud, H. B. Toda, O. Cabrit, S. Bose, and J. Lee. Using singular values to build a subgrid-scale model for large eddy simulations. *Physics of Fluids*, 23(8):085106, 2011.
 - [93] N. Nikitin, F. Nicoud, B. Wasistho, K. Squires, and P. R. Spalart. An approach to wall modeling in large-eddy simulations. *Physics of Fluids*, 12(7):1629–1632, 2000.
 - [94] G. Novati and P. Koumoutsakos. Remember and Forget for Experience Replay. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.
 - [95] G. Novati, H. L. de Laroussilhe, and P. Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1):87–96, 2021.
 - [96] M. Oberlack. Similarity in non-rotating and rotating turbulent pipe flows. *J. Fluid Mech.*, 379:1–22, 1999.
 - [97] P. Orlandi. Helicity fluctuations and turbulent energy production in rotating and non-rotating pipes. *Phys. Fluids*, 9:2045–2056, 1997. ISSN 1070-6631 1089-7666. doi: 10.1063/1.869324.
 - [98] P. Orlandi and D. Ebstein. Turbulent budgets in rotating pipes by DNS. *Int. J. Heat Fluid Fl.*, 21:499–505, 2000.

-
- [99] P. Orlandi and M. Fatica. Direct simulations of turbulent flow in a pipe rotating about its axis. *J. Fluid Mech.*, 343:43–72, 1997. ISSN 0022-1120 1469-7645. doi: 10.1017/s0022112097005715.
 - [100] D. E. Ozan, A. Nóvoa, G. Rigas, and L. Magri. Data-assimilated model-informed reinforcement learning, 2025. URL [arXivpreprintarXiv:2506.01755](https://arxiv.org/abs/2506.01755).
 - [101] G. I. Park and P. Moin. An improved dynamic non-equilibrium wall-model for large eddy simulation. *Physics of Fluids*, 26(1), 2014.
 - [102] S. Partee, M. Ellis, A. Rigazzi, A. E. Shao, S. Bachman, G. Marques, and B. Robbins. Using machine learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling. *Journal of Computational Science*, 62:101707, 2022. ISSN 1877-7503.
 - [103] U. Piomelli. Wall-layer models for large-eddy simulations. *Progress in aerospace sciences*, 44(6):437–446, 2008.
 - [104] U. Piomelli and E. Balaras. Wall-layer models for large-eddy simulations. *Annual review of fluid mechanics*, 34(1):349–374, 2002.
 - [105] U. Piomelli, J. Ferziger, P. Moin, and J. Kim. New approximate boundary conditions for large eddy simulations of wall-bounded flows. *Physics of Fluids A: Fluid Dynamics*, 1(6):1061–1068, 1989.
 - [106] S. Pirozzoli and P. Orlandi. Natural grid stretching for DNS of wall-bounded flows. *J. Comput. Phys.*, 439:110408, 2021. ISSN 00219991. doi: 10.1016/j.jcp.2021.110408.
 - [107] S. Pirozzoli, D. Modesti, P. Orlandi, and F. Grasso. Turbulence and secondary motions in square duct flow. *Journal of Fluid Mechanics*, 840:631–655, 2018.
 - [108] S. Pirozzoli, J. Romero, M. Fatica, R. Verzicco, and P. Orlandi. One-point statistics for turbulent pipe flow up to $Re_\tau \approx 6000$. *J. Fluid Mech.*, 926:A28, 2021. ISSN 0022-1120 1469-7645. doi: 10.1017/jfm.2021.727.
 - [109] K. Qin, Z. Zou, L. Zhu, and O. S. Pak. Reinforcement learning of a multi-link swimmer at low Reynolds numbers. *Physics of Fluids*, 35(3), 2023.
 - [110] M. Quadrio and S. Sibilla. Numerical simulation of turbulent flow in a pipe oscillating around its axis. *J. Fluid Mech.*, 424:217–241, 2000.
 - [111] M. Quadrio, P. Ricco, and C. Viotti. Streamwise-travelling waves of spanwise wall velocity for turbulent drag reduction. *J. Fluid Mech.*, 627:161–178, 2009. ISSN 0022-1120 1469-7645. doi: 10.1017/s0022112009006077.
 - [112] J. Rabault and A. Kuhnle. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 31(9), 2019.

-
- [113] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019. ISSN 0022-1120, 1469-7645.
 - [114] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
 - [115] G. Reich and H. Beer. Fluid flow and heat transfer in an axially rotating pipe—I. Effect of rotation on turbulent pipe flow. *Int. J. Heat Mass Tran.*, 32: 551–562, 1989.
 - [116] P. Ricco, M. Skote, and M. Leschziner. A review of turbulent skin-friction drag reduction by near-wall transverse forcing. *Prog. Aerosp. Sci.*, 123:100713, 2021. ISSN 03760421. doi: 10.1016/j.paerosci.2021.100713.
 - [117] J. Romero, P. Costa, and M. Fatica. Distributed-memory simulations of turbulent flows on modern GPU systems using an adaptive pencil decomposition library. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2022.
 - [118] S. Russo and P. Luchini. A fast algorithm for the estimation of statistical error in DNS (or experimental) time averages. *J. Comput. Phys.*, 347:328–340, 2017. ISSN 00219991. doi: 10.1016/j.jcp.2017.07.005.
 - [119] S. Russo and P. Luchini. A fast algorithm for the estimation of statistical error in DNS (or experimental) time averages. *Journal of Computational Physics*, 347:328–340, 2017.
 - [120] T. Saad, D. Cline, R. Stoll, and J. C. Sutherland. Scalable tools for generating synthetic isotropic turbulence with arbitrary spectra. *AIAA Journal*, 55(1): 327–331, 2017.
 - [121] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher. Benchmark computations of laminar flow around a cylinder. In *Flow simulation with high-performance computers II: DFG priority research programme results 1993–1995*, pages 547–566. Springer, 1996.
 - [122] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.
 - [123] U. Schumann. Subgrid scale model for finite difference simulations of turbulent flows in plane channels and annuli. *Journal of computational physics*, 18(4): 376–404, 1975.
 - [124] U. Schumann and R. A. Sweet. Fast Fourier transforms for direct solution of Poisson’s equation with staggered boundary conditions. *Journal of Computational Physics*, 75(1):123–137, 1988.
 - [125] M. Shams and A. H. Elsheikh. Gym-preCICE: Reinforcement learning environments for active flow control. *SoftwareX*, 23:101446, 2023. ISSN 2352-7110.

-
- [126] J. Smagorinsky. General circulation experiments with the primitive equations: I. The basic experiment. *Monthly Weather Review*, 91(3):99–164, 1963.
 - [127] P. R. Spalart. Strategies for turbulence modelling and simulations. *International journal of heat and fluid flow*, 21(3):252–263, 2000.
 - [128] P. Suárez, F. Alcántara-Ávila, A. Miró, J. Rabault, B. Font, O. Lehmkuhl, and R. Vinuesa. Active Flow Control for Drag Reduction Through Multi-agent Reinforcement Learning on a Turbulent Cylinder at $Re_D = 3900$. *Flow, Turbulence and Combustion*, 115(1):3–27, 2025.
 - [129] P. Suárez, F. Alcántara-Ávila, J. Rabault, A. Miró, B. Font, O. Lehmkuhl, and R. Vinuesa. Flow control of three-dimensional cylinders transitioning to turbulence via multi-agent reinforcement learning. *Communications Engineering*, 4(1):1–13, 2025.
 - [130] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, Cambridge, MA, 1998. ISBN 978-0262193986.
 - [131] F. Tonti, J. Rabault, and R. Vinuesa. Navigation in a simplified urban flow through deep reinforcement learning. *Journal of Computational Physics*, 538: 114194, 2025. ISSN 0021-9991.
 - [132] E. Toubert and M. Leschziner. Near-wall streak modification by spanwise oscillatory wall motion and drag-reduction mechanisms. *J. Fluid Mech.*, 693: 150–200, 2012. ISSN 0022-1120 1469-7645. doi: 10.1017/jfm.2011.507.
 - [133] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments, 2024. URL <https://arxiv.org/abs/2407.17032>.
 - [134] A. Townsend. *The Structure of Turbulent Shear Flow*. Cambridge University Press, 2nd edn edition, 1976.
 - [135] A. Vadrot, X. I. Yang, H. J. Bae, and M. Abkar. Log-law recovery through reinforcement-learning wall model for large eddy simulation. *Physics of Fluids*, 35(5), 2023.
 - [136] E. R. Van Driest. On turbulent flow near a wall. *Journal of the Aeronautical Sciences*, 23(11):1007–1011, 1956.
 - [137] P. Varela, P. Suárez, F. Alcántara-Ávila, A. Miró, J. Rabault, B. Font, L. M. García-Cuevas, O. Lehmkuhl, and R. Vinuesa. Deep reinforcement learning for flow control exploits different physics for increasing Reynolds number regimes. *Actuators*, 11(12):359, 2022.
 - [138] J. Vasanth, J. Rabault, F. Alcántara-Ávila, M. Mortensen, and R. Vinuesa. Multi-agent reinforcement learning for the control of three-dimensional Rayleigh–Bénard convection. *Flow, Turbulence and Combustion*, pages 1–37, 2024.

- [139] S. Verma, G. Novati, and P. Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018. ISSN 0027-8424, 1091-6490.
- [140] R. Verstappen and A. Veldman. Symmetry-preserving discretization of turbulent flow. *Journal of Computational Physics*, 187(1):343–368, 2003.
- [141] R. Verzicco and P. Orlandi. A Finite-Difference Scheme for Three-Dimensional Incompressible Flows in Cylindrical Coordinates. *Journal of Computational Physics*, 123:402–414, 1996.
- [142] C. Vignon, J. Rabault, J. Vasanth, F. Alcántara-Ávila, M. Mortensen, and R. Vinuesa. Effective control of two-dimensional Rayleigh–Bénard convection: Invariant multi-agent reinforcement learning is all you need. *Physics of Fluids*, 35(6), 2023.
- [143] R. Vinuesa, P. Schlatter, and H. Nagib. Secondary flow in turbulent ducts with increasing aspect ratio. *Physical Review Fluids*, 3(5):054606, 2018.
- [144] R. Vinuesa, O. Lehmkuhl, A. Lozano-Durán, and J. Rabault. Flow control in wings and discovery of novel approaches via deep reinforcement learning. *Fluids*, 7(2):62, 2022.
- [145] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem. Direct shape optimization through deep reinforcement learning. *Journal of Computational Physics*, 428:110080, 2021.
- [146] J.-P. von Bassewitz, S. Kaltenbach, and P. Koumoutsakos. Closure Discovery for Coarse-Grained Partial Differential Equations Using Grid-based Reinforcement Learning, 2024.
- [147] A. Vreman. An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications. *Physics of Fluids*, 16(10):3670–3681, 2004.
- [148] H.-C. Wang, M. Yu, and H. Xiao. First-principle-like reinforcement learning of nonlinear numerical schemes for conservation laws, 2023.
- [149] L. Wang, W. K. Anderson, E. J. Nielsen, P. Balakumar, M. A. Park, J.-R. Carlson, P. S. Iyer, and B. Diskin. Wall-modeled large-eddy simulations for high-lift configurations using FUN3D. In *AIAA SciTech 2022 Forum*, page 1555, 2022.
- [150] M. Wang. LES with wall models for trailing-edge aeroacoustics. *Annual research briefs*, pages 355–364, 1999.
- [151] M. Wang and P. Moin. Dynamic wall modeling for large-eddy simulation of complex turbulent flows. *Physics of Fluids*, 14(7):2043–2051, 2002.
- [152] Q. Wang, L. Yan, G. Hu, C. Li, Y. Xiao, H. Xiong, J. Rabault, and B. R. Noack. DRLinFluids: An open-source Python platform of coupling deep reinforcement

-
- learning and OpenFOAM. *Physics of Fluids*, 34(8), 2022. ISSN 1070-6631, 1089-7666.
- [153] H. Werner and H. Wengle. Large-eddy simulation of turbulent flow over and around a cube in a plate channel. In *Turbulent Shear Flows 8: Selected Papers from the Eighth International Symposium on Turbulent Shear Flows, Munich, Germany, September 9–11, 1991*, pages 155–168. Springer, 1993.
 - [154] A. White. Flow of a fluid in an axially rotating pipe. *J. Mech. Eng. Sci.*, 6: 47–52, 1963.
 - [155] D. C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, 2006.
 - [156] A. A. Wray. Minimal-storage time advancement schemes for spectral methods. Technical Report MS 202, NASA Ames Research Center, 1990.
 - [157] M. Xiao, Y. Zhang, and F. Zhou. Numerical investigation of the unsteady flow past an iced multi-element airfoil. *AIAA Journal*, 58(9):3848–3862, 2020.
 - [158] M. Xiao, A. Ceci, P. Costa, J. Larsson, and S. Pirozzoli. CaLES: A GPU-accelerated solver for large-eddy simulation of wall-bounded flows. *Computer Physics Communications*, page 109546, 2025. ISSN 0010-4655.
 - [159] T. Xiong, Z. Liu, Y. Wang, C. J. Ong, and L. Zhu. Chemotactic navigation in robotic swimmers via reset-free hierarchical reinforcement learning. *Nature Communications*, 16(1):5441, 2025.
 - [160] L. Yan, Y. Li, G. Hu, W.-l. Chen, W. Zhong, and B. R. Noack. Stabilizing the square cylinder wake using deep reinforcement learning for different jet locations. *Physics of Fluids*, 35(11), 2023. ISSN 1070-6631, 1089-7666.
 - [161] L. Yan, Q. Wang, G. Hu, W. Chen, and B. R. Noack. Deep reinforcement cross-domain transfer learning of active flow control for three-dimensional bluff body flow. *Journal of Computational Physics*, 529:113893, 2025.
 - [162] T. Yang, Y. Yin, Q. Liu, T. Yu, Y. Wang, H. Zhou, and Z. Ren. Reinforcement Learning for Submodel Assignment in Adaptive Modeling of Turbulent Flames. *AIAA Journal*, 63(2):1–9, 2024.
 - [163] X. Yang, S. Zafar, J.-X. Wang, and H. Xiao. Predictive large-eddy-simulation wall modeling via physics-informed neural networks. *Physical Review Fluids*, 4(3):034602, 2019.
 - [164] X. I. Yang, J. Sadique, R. Mittal, and C. Meneveau. Integral wall model for large eddy simulations of wall-bounded turbulent flows. *Physics of Fluids*, 27(2), 2015.
 - [165] X. I. Yang, M. Abkar, and G. Park. Grid Convergence Properties of Wall-Modeled Large Eddy Simulations in the Asymptotic Regime. *Journal of Fluids Engineering*, 146(8), 2024.

- [166] R. Zangeneh. Data-driven model for improving wall-modeled large-eddy simulation of supersonic turbulent flows with separation. *Physics of Fluids*, 33(12), 2021.
- [167] Z. Zhang and B. Wang. DNS Study of Turbulent Flow in a Circular Pipe Subjected to Axial System Rotation. In *12th International Symposium on Turbulence and Shear Flow Phenomena (TSFP12)*, 2022.
- [168] D. Zhou and H. J. Bae. Sensitivity analysis of wall-modeled large-eddy simulation for separated turbulent flow. *Journal of Computational Physics*, 506:112948, 2024.
- [169] D. Zhou and H. J. Bae. Wall modeling of turbulent flows with varying pressure gradients using multi-agent reinforcement learning. *AIAA Journal*, 62(10): 1–15, 2024.
- [170] Z. Zhou, G. He, and X. Yang. Wall model based on neural networks for LES of turbulent flows over periodic hills. *Physical Review Fluids*, 6(5):054610, 2021.
- [171] X. Zhu, E. Phillips, V. Spandan, J. Donners, G. Ruetsch, J. Romero, R. Ostilla-Mónico, Y. Yang, D. Lohse, R. Verzicco, *et al.* AFiD-GPU: a versatile Navier–Stokes solver for wall-bounded turbulent flows on GPU clusters. *Computer Physics Communications*, 229:199–210, 2018.