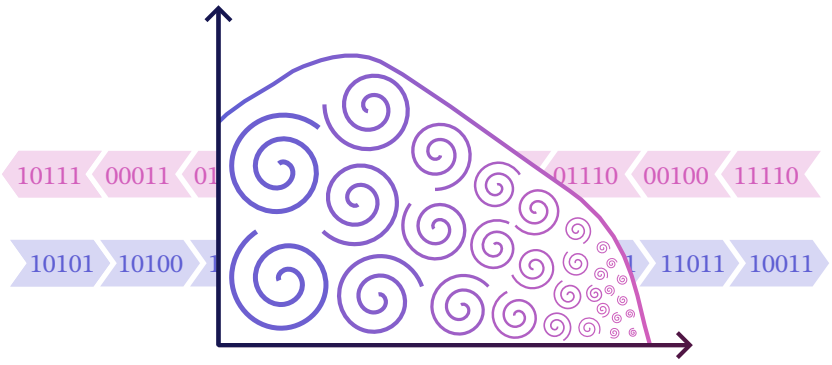
DATA-DRIVEN DISCRETE CLOSURE MODELS FOR LARGE-EDDY SIMULATION OF INCOMPRESSIBLE TURBULENCE



SYVER DØVING AGDESTEIN

April 2026

The research presented in this thesis was conducted at Centrum Wiskunde & Informatica (CWI) in Amsterdam. It is supported by the project "Discretize first, reduce next" (with project number VI.Vidi.193.105) of the research programme NWO Talent Programme Vidi.

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available at:

https://bitbucket.org/amiede/classicthesis/

Generative AI:
During the preparation of this thesis and the articles it contains, the author used GitHub Copilot to propose wordings and Anthropic Claude Code for improving language and readability. After using these tools/services, the author reviewed and edited the content as needed and take full responsibility for the content.

Cover: Bits and bytes, whirls and curls, scales in a spectrum.

Image credits:
Boeing 777-200 photograph by Jules Meulemans, licensed under the GNU Free Documentation License, version 1.2 or later.
Polynesian flying fox photograph by Charles J. Sharp, licensed under CC BY-SA 4.0.

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-6703-4.

Printed by: Gildeprint

Data-driven discrete closure models for large-eddy simulation of incompressible turbulence

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr. S.K. Lenaerts, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 28 mei 2026 om 13:30 uur

door

Syver Døving Agdestein

geboren te Bærum, Noorwegen

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	prof.dr.ir. B.F. van Dongen
1e promotor:	dr.ir. B. Sanderse
2e promotor:	prof.dr. R.W.C.P. Verstappen (Rijksuniversiteit Groningen)
leden:	prof.dr.-ing. A. Beck (Universität Stuttgart)
	prof.dr.ir. B.J. Geurts
	prof.dr.ir. A.W. Vreman
	dr. A. Lozano-Durán (Massachusetts Institute of Technology)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Til Kari, Einar, Unni og Reidar.

Big whirls have little whirls,
That feed on their velocity;
And little whirls have lesser whirls,
And so on to viscosity.

— Lewis Fry Richardson [149]

PREFACE

This thesis is the result of my research at CWI in Amsterdam, funded through the NWO Vidi grant "Discretize first, reduce next" obtained by my supervisor Benjamin Sanderse. The research has evolved in ways I did not anticipate: initial ideas sometimes turned out not to work as intended, but they naturally led to new insights and unexpected research directions.

The Greek letter λ—used in both mathematics and informatics—decorates CWI in the form of sculptures and paintings.

The initial idea was to start from a discretized system of equations and derive a system of equations for large-eddy simulation (LES) from there. I first explored discrete filtering, reconstruction, and operator inference for non-uniform filter widths, which resulted in a conference paper [1] forming chapter 2 of this thesis. I then turned to machine learning approaches for learning LES closure models from data. At the time, there seemed to be a consensus in the community that machine learning based LES models were difficult to get right and could quickly lead to instabilities. This motivated the development of a differentiable fluid solver for training neural networks in an a-posteriori setting, thus encouraging the model to achieve stability. During this process, I realized that if the underlying filter was designed carefully, and if the target data was obtained using discretization-consistent expressions, a-priori training would also yield stable models. These results were published in a journal article [2] and form chapter 3 of this thesis.

The next step was to consider symmetry-preserving neural network architectures for LES modeling, such as tensor-basis neural networks. During the implementation of these models, I wanted to repeat the procedure of the previous article—using a-priori rather than a-posteriori training—since a-posteriori training was becoming increasingly computationally expensive for large three-dimensional simulations. The seemingly small difference was that I now needed discretization-consistent target data for the unresolved stress *tensor*, rather than the divergence of the stress tensor. This turned out to be far from straightforward.

Interestingly, the nature of the unresolved stress tensor in LES appearing after discretization seemed not to have been thoroughly investigated in the literature. Most works assumed a continuous setting and derived all models accordingly. When a discretization was introduced, it was typically rewritten in the continuous setting using a discretization-induced filter. I began to ask whether we should not do it the other way around: write the LES equations in the fully discrete setting. This led to a new expression for the unresolved stress tensor in LES. What started as a small implementation detail—a byproduct of the code for computing target data in the work on symmetry-preserving closure models—grew into a 30-page article on LES, the finite volume method, implicit and explicit closure modeling, filtering,

discretization, and incompressibility [5]. This article forms chapter 4 of this thesis.

The work on symmetry-preserving neural networks for LES modeling [4] forms chapter 5 of this thesis. I find it remarkable how a machine learning model can be enhanced with seemingly straightforward physical knowledge: a fluid flow should not depend on the angle from which it is observed, yet this is not automatically the case for neural networks.

Many ideas have also not been explored in full depth or have been abandoned along the way, as is the nature of research. These include work on discrete deconvolution, Fourier neural operators for learning spectral relations in LES closure modeling, incorporating history into the LES models through the Mori-Zwanzig formalism, modeling the unresolved energy through additional latent variables, and stability analysis of a-priori and a-posteriori training. A number of test cases have also been implemented and considered without making it into the published work, including natural convection (Rayleigh-Bénard), turbulent channel flow, and closure modeling for unresolved terms arising due to boundary conditions. Implementation details for some of these cases, along with a presentation of the software suite that I developed during this thesis, are provided in chapter 6.

During the course of this thesis, I have had the opportunity to travel and present my work at conferences and meetings in the Netherlands, France, Norway, Portugal, Italy, and the United States. These trips have been inspiring and have allowed me to meet and exchange ideas with people working on similar topics. Traveling with fellow group members has been equally valuable, as it allows one to get to know each other beyond the office.

At CWI, I have also participated in activities beyond my primary research. In the Scientific Computing group, we organized two *group challenges*: one with Deltares on estimating dike failure probabilities for the flood defenses of the Netherlands, and another with the Royal Dutch Meteorological Institute (KNMI) on interpolating crowd-sourced weather data of mixed fidelity to generate a weather map for the European continent. It was encouraging to work on something more applied and practical, where the impact is immediately visible—in contrast to fundamental research, which takes time to bear fruit. These challenges also offered a welcome opportunity to collaborate closely with group members, as my PhD research has been more individual in nature. I have additionally participated in the CWI works council, where employees take part in overseeing the well-functioning of the organization as a whole. This has been a valuable experience, offering a perspective on the organization quite different from that of individual research.

The GPU fan of the "Turbulator" spinning during a direct numerical simulation.

Other activities at CWI and Amsterdam Science Park, in no particular order: throwing basketballs on sunny days, attending and giving seminars, supervising Master students, playing piano sonatas after work in the activity room, ping-pong, bouldering, eating excellent leftover food from the events of other research groups, building a GPU workstation from scratch by or-

dering parts online and buying a screwdriver from the local hardware store, partaking in the monthly *praethuys* event at CWI, going for walks in the parks of Amsterdam with colleagues, and singing with the CWI choir in the festive season.

Amsterdam, April 2026

Syver Døving Agdestein

CONTENTS

1

INTRODUCTION

Turbulent flows are present in many natural and industrial systems, from the atmosphere, rivers, and oceans to the flows around aircraft, cars, ships, and wind turbines, and the combustion processes inside engines. The ability to accurately predict and control the behavior of these systems is therefore of great importance. Turbulence is inherently unpredictable, yet its mean dynamics can be computed with considerable accuracy. In many applications, this level of information is sufficient, as the mean behavior can be advantageous under specific conditions—for instance, in processes that rely on enhanced mixing, or in wind farms where turbulence restores the wind field behind a turbine.

Turbulent plume from a snow canon near the author's family home.

1.1 THE SCALES OF TURBULENCE

Many phenomena in nature are characterized by a multitude of scales. This is particularly true for non-linear systems, such as turbulent flows. A useful concept is that of *nondimensionalization*, which expresses the governing equations in dimensionless form, thereby revealing the essential parameters that characterize the system. Two systems that share the same dimensionless parameters are said to be *dynamically similar*—they exhibit the same behavior regardless of the specific units used to measure them [30].

Consider the viscous Burgers' equation [31] for a velocity field $u(x,t)$:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) - \nu\frac{\partial^2 u}{\partial x^2} = 0, \tag{1.1}$$

where $\nu > 0$ is the viscosity. This equation describes the combined effects of non-linear convection and linear diffusion. It depends on four quantities—time t, position x, velocity u, and viscosity ν—each with their own units. However, nondimensionalization reveals that the equation is governed by a single dimensionless parameter. Define characteristic velocity and length scales U and L. The characteristic time scale follows as $T \coloneqq L/U$. We can then define the dimensionless variables $u^* \coloneqq u/U$, $t^* \coloneqq t/T$, $x^* \coloneqq x/L$, and $\mathrm{Re} \coloneqq UL/\nu$. The nondimensional form of eq. (1.1) is then

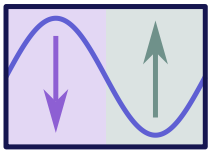
$$\frac{\partial u^*}{\partial t^*} + \frac{\partial}{\partial x}\left(\frac{u^{*2}}{2}\right) - \frac{1}{\mathrm{Re}}\frac{\partial^2 u^*}{\partial x^{*2}} = 0. \tag{1.2}$$

The dimensionless number Re is called the Reynolds number. It characterizes the relative importance of the non-linear convective term $\partial(u^2/2)/\partial x$

versus the diffusive term $\nu\partial^2 u/\partial x^2$. Two systems governed by the Burgers' equation with the same Reynolds number are dynamically similar: they have the same dimensionless solution u^*, regardless of what units we use to measure the system.

The one-dimensional Burgers' equation (1.1) can already tell us something about the appearance of turbulence. Turbulence is characterized by the presence of many different scales, which are best described in terms of Fourier modes. Consider a sine wave $u(x) \coloneqq \mathrm{e}^{\mathrm{i}kx}$ for some wavenumber k, where i is the imaginary unit. The diffusive term is then

$$\frac{\partial^2 u}{\partial x^2}(x) = (\mathrm{i}k)^2 \mathrm{e}^{\mathrm{i}kx} = -k^2 u(x). \tag{1.3}$$



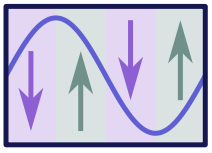
The diffusive force dampens Fourier modes.

This means that in a diffusion equation $\partial u/\partial t = \nu\partial^2 u/\partial x^2$, an initial Fourier mode $u(x,0) \coloneqq \mathrm{e}^{\mathrm{i}kx}$ decays exponentially as $u(x,t) = u(x,0)\mathrm{e}^{-\nu k^2 t}$. The rate of decay increases quadratically with the wavenumber k. The diffusive term therefore dampens out high-wavenumber modes (small scales) much faster than low-wavenumber modes (large scales). The presence of diffusivity limits the range of scales that can coexist in a system over prolonged periods of time, and more diffusivity in a system therefore lowers the chance of turbulence arising.

On the other hand, the non-linear convective term in eq. (1.1) for a Fourier mode is

$$\frac{\partial}{\partial x}\left(\frac{u^2}{2}\right)(x) = \frac{\partial}{\partial x}\left(\frac{\mathrm{e}^{2\mathrm{i}kx}}{2}\right) = \mathrm{i}k\mathrm{e}^{\mathrm{i}(2k)x}. \tag{1.4}$$

This means that a Fourier mode of wavenumber k has a time-derivative containing a new mode of wavenumber $2k$. Over time, the non-linear term in the Burgers' equation therefore creates more and more small scales (characterized by high wavenumbers), since the mode at $2k$ will in turn create a mode at $4k$, and so on. The Reynolds number Re characterizes the competition between the non-linear term, which creates small scales, and the diffusive term, which destroys small scales. Higher Reynolds numbers lead to a larger range of scales coexisting, which is a prerequisite for turbulence.



The convective force creates a Fourier mode at twice the original wavenumber.

Dynamic similarity implies that the turbulent nature of a flow is fully determined by its Reynolds number, regardless of the physical dimensions of the system. Two flows at the same Reynolds number—whether in a wind tunnel or around a full-scale aircraft—exhibit the same dimensionless behavior. This principle is the foundation of experimental fluid mechanics, where scale models are used to study flows that are too large or too expensive to investigate at full scale.

In addition to the range of scales present, another important characterization of turbulence is how energy is distributed across these scales. Let

$e \coloneqq u^2/2$ denote the kinetic energy density. If we multiply eq. (1.1) by u, we get the energy equation

$$\frac{\partial e}{\partial t} + u\frac{\partial}{\partial x}\left(e - \nu\frac{\partial u}{\partial x}\right) = 0. \tag{1.5}$$

Using the product rule for the spatial derivatives, we get

$$\frac{\partial e}{\partial t} + \frac{\partial}{\partial x}\underbrace{\left(\frac{2}{3}ue - \nu\frac{\partial e}{\partial x}\right)}_{A} + \nu\underbrace{\frac{\partial u}{\partial x}\frac{\partial u}{\partial x}}_{B} = 0. \tag{1.6}$$

The term A is an energy flux, and it therefore cannot create nor destroy energy, only redistribute it. The term B is always positive, which means that it can only dissipate energy (since it is in the left-hand side). The energy equation thus illustrates two key mechanisms of turbulence: the convective redistribution of energy across scales and the viscous dissipation of energy at small scales.

However, turbulence is not only characterized by the presence of multiple scales and their energy interactions. Another aspect of turbulence is its chaotic nature, which is not present in the 1D Burgers' equation (whose solutions eventually form shocks and decay monotonically due to the absence of a sustained forcing mechanism). Chaos can be described as sensitivity to small perturbations. Let $u_a(x,0) = u_0(x)$ and $u_b(x,0) \coloneqq u_0(x) + \epsilon(x)$ denote two slightly different initial conditions to an equation for some noise field ϵ with small amplitude. If the solutions $u_a(x,t)$ and $u_b(x,t)$ diverge over time, then the system is chaotic.

A Boeing 777-200 airplane (top) and a Polynesian flying fox (bottom). Both are immersed in the same fluid, governed by the same equation, but at different scales, and thus at different Reynolds numbers.

To capture both the multi-scale and chaotic aspects of turbulence, we need to move beyond the 1D Burgers' equation. The Burgers' equation can be seen as a simplified version of the 3D Navier-Stokes equations. In the incompressible case, these equations read

$$\nabla \cdot u = 0, \quad \frac{\partial u}{\partial t} + \nabla \cdot (u \otimes u - 2\nu S + p\delta) = f, \tag{1.7}$$

where δ is the identity tensor, p is the pressure field, u is the velocity, $S \coloneqq (\nabla u + \nabla u^T)/2$ is the symmetric rate-of-strain tensor, $\nu > 0$ is the kinematic viscosity, and f is a body force (for example gravity or an electromagnetic Lorentz force). As in the Burgers' equation, the turbulent nature of the Navier-Stokes equations arises for sufficiently high Reynolds numbers $\mathrm{Re} \coloneqq UL/\nu$, and the convective term perfectly conserves kinetic energy while redistributing it across scales. The balance between convective redistribution and viscous dissipation determines the energy content at each scale of motion, which can be quantified through the energy spectrum.

Define the energy density as $e \coloneqq \|u\|^2/2$. The energy spectrum in a wavenumber interval $[k_a, k_b]$ is defined as

$$\sum_{k_a \le \|k\| < k_b} \hat{e}(k), \tag{1.8}$$

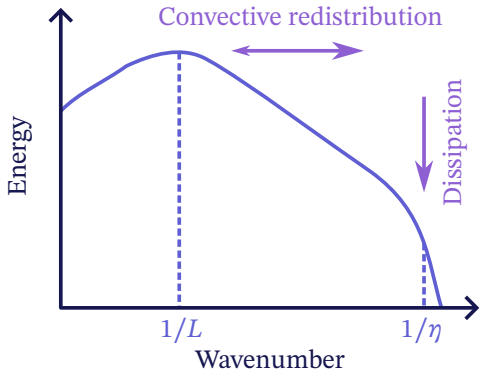


Figure 1.1: Model spectrum for statistically stationary turbulent isotropic flows. Both the abscissa and ordinate are in logarithmic scale.

where $\widehat{(\cdot)}$ denotes the Fourier transform. $\hat{e}$ can also be computed as $\hat{e} = \|\hat{u}\|^2/2$ with Parseval's theorem.

A model spectrum for statistically stationary turbulent isotropic flows is shown in fig. 1.1. The spectrum is divided into three ranges of wavenumbers k [144]:

1. The first range is the integral (or energy-containing) range, containing the largest scales in the flow. This range is characterized by a characteristic integral length scale L. In this range, the convective term dominates the equation.

2. The second range is the inertial range, characterized by a $-5/3$ slope in the spectrum. In this range, energy is "on average" transferred from large scales to small scales through the convective term (although some "back-scatter" from small to large scales also occurs). The newly formed energy is also partially dissipated through the diffusive term.

3. The third range is the dissipative range, where the viscous term dominates. The energy therefore sharply drops. This range is characterized by the Kolmogorov length scale $\eta := (\nu^3/\epsilon)^{1/4}$, where $\epsilon := 2\nu\langle S, S\rangle$ is the viscous dissipation rate, and $\langle S, S\rangle := \int_\Omega S_{ij}S_{ij}\mathrm{d}V$ (repeated indices imply summation).

Although the Navier-Stokes equations are deterministic, their chaotic nature at high Reynolds numbers means that the solution is extremely sensitive to initial and boundary conditions. In practice, the exact state of a turbulent flow can never be known with sufficient precision to predict its future evolution. It is therefore natural to adopt a statistical description of turbulence, where the velocity field u is treated as a random variable [144]. Statistical quantities derived from u, such as the mean velocity, the variance, or the

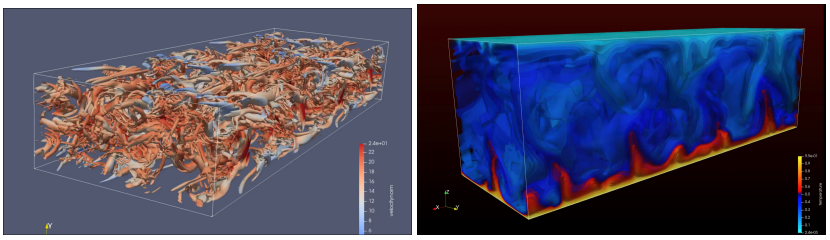


Figure 1.2: Left: DNS of a turbulent channel flow. Right: Temperature field from a DNS of Rayleigh-Bénard convection. The code is explained in chapter 6.

energy spectrum, are deterministic even though the individual realizations of u are not reproducible. The energy spectrum can be seen as a distribution of kinetic energy in wavenumber space: it tells us how much energy resides at each scale, regardless of the particular realization.

This statistical perspective has direct consequences for how we evaluate numerical simulations of turbulent flows. Since individual realizations are inherently unpredictable, there is no single "correct" solution to compare against. A simulation is therefore considered successful if it reproduces the correct statistical properties of the flow, such as the energy spectrum or mean velocity profiles. Another common requirement is that the simulation behaves correctly in limiting cases, such as for laminar flows that have not yet developed turbulence, or in the boundary layers close to walls.

Beyond statistical accuracy, many argue [141, 164, 191] that a simulation should also respect the fundamental structural properties of the Navier-Stokes equations. These include conservation of mass, momentum, and kinetic energy, as well as secondary conserved quantities such as enstrophy and helicity. Furthermore, the Navier-Stokes equations possess *symmetry properties*: they are invariant under certain transformations such as rotations, reflections, and translations. A simulation that preserves these properties is more likely to produce physically consistent results, even at coarse resolutions. We will return to these properties in later chapters (in particular: mass in chapter 3, momentum in chapter 4, and symmetry properties in chapter 5).

1.2 NUMERICAL SIMULATION OF TURBULENT FLOWS

The most direct way to compute a turbulent flow is to discretize and solve the Navier-Stokes equations (1.7) without any simplification. This approach is known as direct numerical simulation (DNS). In DNS, all scales of motion are resolved, from the largest energy-containing eddies down to the Kolmogorov length scale η (a common criterion is that the grid spacing h satisfies $h \leq 2.1\eta$ [209]). Two examples of DNS are shown in fig. 1.2.

We discuss the computational aspects of DNS in chapter 6.

DNS requires no modeling assumptions beyond the Navier-Stokes equations themselves, making it the most accurate approach. However, the computational cost of DNS is prohibitive for most practical applications. The number of grid points needed to resolve all scales in three dimensions grows as $\mathrm{Re}^{9/4}$, and the total number of floating point operations required to solve a 3D simulation up to a fixed time scales cubically with the Reynolds number [144]. For an aircraft at cruising speed, $\mathrm{Re} \approx 10^8$, making DNS infeasible with current or foreseeable computing resources.

Since DNS cannot be used for most applications of engineering interest, methods have been developed that reduce the computational cost by not resolving all scales of motion. For statistically steady flows, the Reynolds-averaged Navier-Stokes (RANS) equations average over different turbulent realizations of the velocity field, yielding equations for the mean flow alone. Since the averaging removes all turbulent fluctuations, the computational cost is independent of the Reynolds number. However, RANS requires modeling the effect of all turbulent scales on the mean flow, which limits its accuracy. For unsteady flows where the large-scale dynamics matter, large-eddy simulation (LES) offers a middle ground: only the large scales are resolved, while the effect of the small scales is modeled [152]. LES is computationally much cheaper than DNS, while retaining more of the turbulent dynamics than RANS. LES has already proven its value in a range of engineering applications, including combustion in gas turbines [142], atmospheric boundary layer flows [127], and aeroacoustics of aircraft [27]. LES is the main topic of this thesis.

All three approaches—DNS, RANS, and LES—require a spatial and temporal discretization of the governing equations. The choice of discretization depends on the geometry of the domain, the required accuracy, and the available computational resources. The finite volume method (FVM) [54] is widely used for fluid flows in complex geometries. The domain is divided into control volumes, and the conservation laws are written in integral form over each volume. A central challenge in FVM is the reconstruction of *fluxes* at the faces of each control volume, for which many schemes have been proposed [54, 99]. For simpler geometries, the finite difference method offers high-order accuracy, though at the cost of wider stencils. *Compact* finite difference schemes achieve high-order accuracy with narrower stencils by making the operators implicit (requiring the solution of tri-diagonal or penta-diagonal systems) [97]. For rectangular domains with periodic or no-slip boundary conditions, pseudo-spectral methods based on discrete Fourier, cosine, or Chebyshev transforms achieve very high accuracy [136, 151], but cannot be applied to complex geometries. Spectral element methods combine the geometric flexibility of finite elements with the high-order accuracy of spectral methods [55, 73].

For time integration, explicit methods are generally preferred for turbulent flows. In wall-bounded flows, the small grid spacing near the walls can

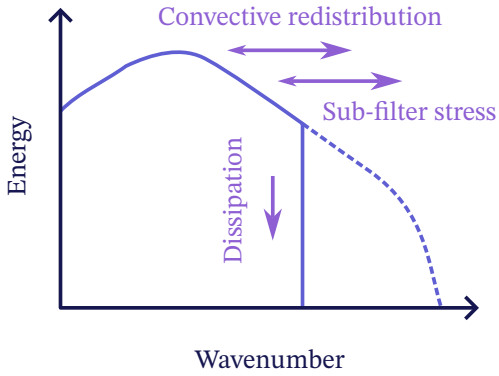


Figure 1.3: Model spectrum for LES of statistically stationary turbulent isotropic flows. The filter is a spectral cutoff filter. Both the abscissa and ordinate are in logarithmic scale.

impose a severe time-step restriction on the viscous term, motivating implicit treatment of the viscous term [154].

1.3 LARGE-EDDY SIMULATION

In LES, the computational cost is reduced by only resolving the large scales of the flow, while modeling the effect of the unresolved small scales. Formally, this is done by applying a low-pass filter $\overline{(\cdot)}$ to the Navier-Stokes equations (1.7) [98], yielding the filtered equations

$$\nabla \cdot \bar{u} = 0, \quad \frac{\partial \bar{u}}{\partial t} + \nabla \cdot \left(\overline{u \otimes u} - 2\nu \bar{S} + \bar{p}\delta \right) = \bar{f}. \tag{1.9}$$

Here we assumed that filtering commutes with differentiation, which does not hold in general at boundaries. Due to the non-linearity of the convective term, the equations contain an unresolved stress $\overline{u \otimes u}$ which cannot be computed from the filtered solution $\bar{u}$ alone. This stress is modeled as $\overline{u \otimes u} \approx \bar{u} \otimes \bar{u} + m(\bar{u})$, where m is a closure model for the unresolved stress.

The need for a closure model m can be understood from the energy balance. Setting $m = 0$ would ignore the interaction between resolved and unresolved scales, leading to an energy pile-up at the smallest resolved scales. This is illustrated in fig. 1.3: in a full DNS (fig. 1.1), energy is transferred from large to small scales through convective redistribution and ultimately dissipated at the Kolmogorov scale. On a discrete grid, the convective term $\bar{u} \otimes \bar{u}$ cannot transfer energy beyond the maximum resolved wavenumber, as higher modes get aliased back to the resolved range. Without a model, energy accumulates at the grid scale. The closure model m must therefore account for the net energy transfer to the unresolved scales.

Closure models are typically classified as either *functional* or *structural*. Functional models aim to correctly account for the energy transfer of the unresolved stress. Structural models, on the other hand, aim to reconstruct the unresolved stress tensor itself. This sub-filter stress (SFS) tensor can be defined as

$$\tau \coloneqq \overline{u \otimes u} - \bar{u} \otimes \bar{u}, \tag{1.10}$$

while the dissipation it produces is given by the dissipation coefficient, which follows from the contraction $\tau_{ij}\bar{S}_{ij}$ (repeated indices imply summation).

Eddy-viscosity models are an example of functional models, where the model is $m(\bar{u}) \coloneqq -2\nu_t\bar{S}$ and $\nu_t > 0$ is a so-called eddy viscosity. This family of models gives strictly dissipative behavior, since the resulting dissipation coefficient $-2\nu_t\bar{S}_{ij}\bar{S}_{ij} \leq 0$ is negative. They therefore cannot account for back-scatter, unless negative values for ν_t are possible. One of the first eddy-viscosity models was the Smagorinsky model $\nu_t \coloneqq C_S^2\Delta^2\sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$, where C_S is a constant and Δ is the filter width [168]. Other improved eddy-viscosity models have been proposed since then, notably to better deal with edge cases such as near-wall behavior and laminar flows [130, 131, 181, 192, 195].

An example of a structural model is the gradient model of Clark [41]. By performing a Taylor expansion of the filtered variable in the filter width Δ, the SFS tensor is

$$\tau = -\frac{\Delta^2}{12}\nabla\bar{u}\nabla\bar{u}^T + \mathcal{O}(\Delta^4), \tag{1.11}$$

We test the Clark model in chapter 5.

assuming that the filter kernel is Gaussian or top-hat. Truncating the higher-order terms gives the gradient model $m \coloneqq -(\Delta^2/12)\nabla\bar{u}\nabla\bar{u}^T$. When used by itself, this model is known to be unstable [199], notably due to the amount of back-scatter it produces. It is therefore often used with ad-hoc clipping procedures or in combination with stabilizing eddy-viscosity models.

Meneveau and Katz argue that in addition to the space and time axes, LES also positions itself along a third axis: the axis of *scale* [124]. For homogeneous isotropic turbulence, this axis can be thought of as the wavenumber amplitude axis in fig. 1.3. Unlike RANS, an LES with a spectral cutoff filter applied in the inertial range still resolves some of the turbulent length scales. This insight motivates the use of *scale-similarity* models, which rely on the assumption that the structure of τ does not depend on where Δ is within the inertial range of length scales. By introducing a second *test filter* $\widetilde{(\cdot)}$ (typically of width 2Δ), the structure of τ can be estimated from the resolved LES solution $\bar{u}$ alone. An example of a structural scale-similarity model is the model of Bardina et al. [10], where

$$m \coloneqq \widetilde{\bar{u} \otimes \bar{u}} - \widetilde{\bar{u}} \otimes \widetilde{\bar{u}} \tag{1.12}$$

is obtained by applying the test filter to the resolved LES solution. Scale-similarity types of models have also been used to dynamically estimate the

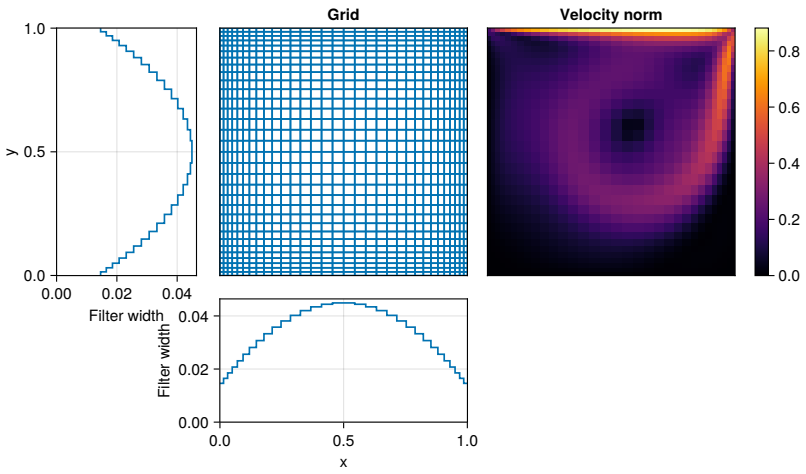


Figure 1.4: Non-uniform filter widths (in the x and y directions) derived from a non-uniform grid. The velocity field is induced by moving the top lid at constant speed to the right.

coefficients of functional models, as proposed by Germano et al. for the Smagorinsky model [61]. Stallcup and Dahm successfully used the dynamic procedure to estimate the coefficients of the full SFS tensor in a complete tensor basis [175].

The LES modeling presented above assumes a continuous setting, where filtering commutes with differentiation and the sub-filter stress only arises due to the non-linearity. However, LES is typically performed in complex geometries with non-uniform meshes and various boundary conditions. Both the discretization and the boundary conditions lead to additional unresolved stresses that need to be modeled. Two general modeling approaches to this problem have been proposed [62, 65, 158]:

1. **Continuous filter.** Modify the filter in the continuous LES formulation to account for the discretization.
2. **Discretize first.** Reformulate LES as a discrete problem.

We consider the second approach in chapters 2 and 3. In chapter 4, we unify both approaches into a common framework.

The first approach models the discretization and boundary effects without departing from the continuous framework. This is achieved by considering the discretization itself as a filter, where the filter width is related to the grid spacing. In particular, non-uniform grid spacings, non-structured grid volumes, and boundaries are treated as modifications to the filter (for example by introducing a non-uniform filter width). Ghosal and co-workers were able to restore the commutation between filtering and differentiation for non-uniform filters up to a certain order by using filters with cancelling moments [64, 128, 186, 188]. To illustrate this, fig. 1.4 shows a lid-driven cavity

We test LES with non-uniform filters in chapter 2.

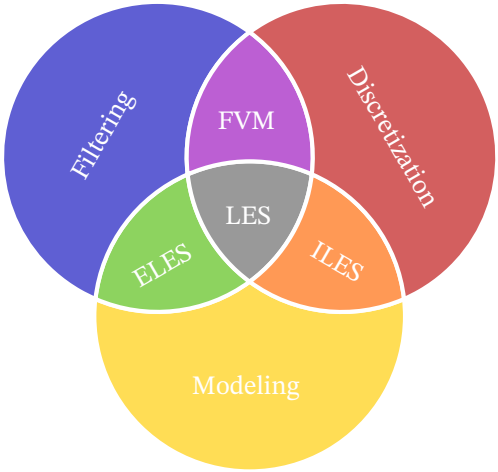


Figure 1.5: Our view on LES as a synthesis between filtering, modeling, and discretization, based on ideas of Verstappen [193].

In chapter 4, we show how the discrete divergence can be written as the continuous divergence composed with a grid-induced filter.

solved with the finite volume method on a coarse non-uniform hyperbolic-tangent grid [182]. Even though no explicit LES filter is applied, the discretization itself acts as an implicit filter: the grid can only represent scales larger than the local grid spacing. The resulting effective filter widths in the x and y directions, determined by the grid spacing, are shown in the figure. On a non-uniform grid, the filter width varies in space, which means that the commutation error between filtering and differentiation is non-zero.

The second approach takes the opposite perspective: rather than adapting the continuous filter to account for the discretization, it starts from the discrete equations and defines the closure problem directly in the discrete setting. Veldman introduced the philosophy "discretize-first" in the context of fluid simulation [189]. The idea is that the equations should be discretized before they are manipulated into a suitable form through algebraic manipulations. Verstappen further argues that LES should be seen as a synthesis of filtering, modeling, and discretization [193]. We visualize this idea in fig. 1.5, where LES is the intersection of the three concepts. The FVM is a discretization method that is obtained by applying a grid-filter, but it does not involve any physical modeling beyond flux reconstruction. In *implicit LES* (ILES), the discretization itself serves as the closure model: numerical dissipation built into the discretization scheme accounts for the missing sub-grid dissipation [22]. This avoids the need for an explicit closure model entirely, but requires careful calibration of the numerical dissipation. No explicit LES filter is applied. In explicit LES (ELES), an LES filter is explicitly applied, and the resulting unresolved stresses are explicitly modeled.

This modeling is typically done in the continuous setting, so it misses the discretization aspect.

In this *discretize-first* philosophy, the numerical discretization is not an afterthought but fundamentally shapes the closure model and its properties. The discrete operators (divergence, gradient, Laplacian) have specific structural properties—such as symmetry and conservation—that differ from their continuous counterparts. A closure model designed in the continuous setting may lose these properties upon discretization, while a model designed directly in the discrete setting can be constructed to preserve them.

Despite these advances, current physics-driven closure models face fundamental trade-offs that limit their applicability. Eddy-viscosity models are simple and robust, but strictly dissipative, preventing them from capturing back-scatter. Structural models like the gradient model can represent the sub-filter stress tensor more faithfully, but tend to be unstable without additional regularization. Scale-similarity models adapt to the local flow structure, but rely on assumptions about the inertial range that may not hold near walls or in transitional flows. The *discretize-first* approach offers structural consistency, but requires closure models to be redesigned for each discretization. The core difficulty is that no single physics-driven model is simultaneously accurate, stable, and generalizable to flow configurations beyond those for which it was calibrated. These limitations motivate the use of data-driven methods, which can learn closure models directly from high-fidelity simulation data.

1.4 MACHINE LEARNING: LEARNING MODELS FROM DATA

Machine learning offers a fundamentally different approach to the LES closure problem: rather than deriving models from physical assumptions, closure models can be *learned* from high-fidelity simulation data [28, 122, 157]. The basic building block is the neural network, composed of layers $\ell : \mathbb{R}^n \to \mathbb{R}^m$ defined as

We discuss the hardware and software aspects of machine learning coupled with fluid solvers in chapter 6.

$$\ell(x) \coloneqq \sigma(Wx + b), \tag{1.13}$$

where $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are trainable parameters (weights and biases), and σ is a non-linear activation function. The *universal approximation theorem* guarantees that neural networks can approximate any continuous function to arbitrary accuracy [45, 76], making them a flexible tool for representing unknown closure models.

We employ deep neural networks to learn discretization-informed LES closure models in chapters 3 and 5.

Deep learning refers to neural networks with many layers, trained on large datasets in high-dimensional parameter spaces. Deep learning has been remarkably successful in image recognition, natural language processing, and generative modeling [66]. For scientific computing on structured grids, *convolutional neural networks* (CNNs) [96] are particularly well-suited,

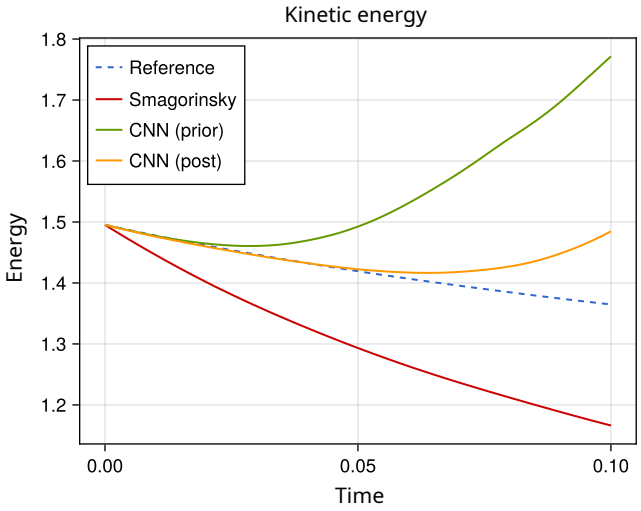


Figure 1.6: Kinetic energy profile for an underresolved decaying turbulence simulation. A neural-network based LES closure model is trained with a-priori and a-posteriori training. The a-priori trained model is unstable, while the a-posteriori trained model remains stable for longer. More details are given in chapter 3.

as they exploit the spatial structure of the data—much like stencil-based discretization operators.

Applying machine learning to LES closure modeling introduces several challenges. A first challenge is the training data: experimental data is difficult to obtain, so DNS is the preferred data source, but DNS data is discrete, requiring care to avoid discretization errors when computing training targets. A second challenge is computational efficiency: the learned model must be fast enough to provide actual speedups over DNS. The most fundamental challenge, however, is ensuring that the learned model is accurate, stable, and generalizable to flow configurations beyond the training data. A model that fits the training data well (in an a-priori sense) may still produce unstable or unphysical results when deployed in an actual LES (see fig. 1.6 for an example). This stability gap between training and deployment is a central obstacle for data-driven LES, and *addressing it is the primary focus of this thesis*.

1.5 RESEARCH GOAL AND APPROACH

The central difficulty with data-driven LES closure models is that models trained in the standard way—by fitting the model output to exact closure terms computed from DNS—often produce unstable simulations when de-

ployed in LES, as shown in fig. 1.6. This *model-data inconsistency* arises because the training targets are defined in the continuous or DNS setting, while the model is deployed in a coarse discrete setting with different numerical properties.

The main goal of this thesis is to develop data-driven LES closure models that are consistent with the numerical discretization and respect the structural properties of the Navier-Stokes equations. This is what we mean by *data-driven discrete closure models*—closure models that are learned from data, but whose formulation is rooted in the discrete equations rather than the continuous ones, and whose architecture encodes known physical structure. This goal can be decomposed into three sub-goals:

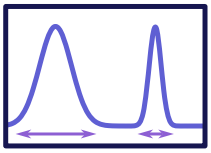
1. **Discretization-consistent closure formulations.** How should the closure problem be formulated so that the training data is consistent with the discrete LES setting in which the model is deployed? Can we derive exact expressions for the discretization-induced unresolved stresses, so that the training targets for structural models are correct?

2. **Incorporating physical structure.** Can known properties of the Navier-Stokes equations—such as symmetries—be built into the neural network architecture to improve generalization and reduce the need for training data?

3. **Software for data-driven closure modeling.** How can existing CFD software be adapted to support data-driven closure models? What are the computational and software engineering challenges of coupling neural networks with numerical solvers?

1.6 THESIS CONTRIBUTIONS AND OUTLINE

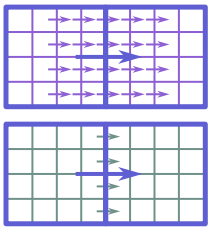
The thesis is structured as follows. Chapters 2 to 6 are adapted from research articles and are self-contained. They address the three sub-goals stated above, and are summarized below. Chapter 7 summarizes the main findings.



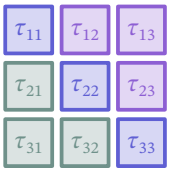
Does LES work for non-uniform filters?

Chapter 2 addresses sub-goal 1 (discretization-consistent closure formulations) in a 1D setting with a non-uniform filter. We study the discretize-first approach to filtering in the context of the linear convection equation. We compare intrusive and non-intrusive methods for inferring the discretely filtered convection operator, and show that the discretize-first approach avoids commutation errors that arise in the classical filter-first framework.

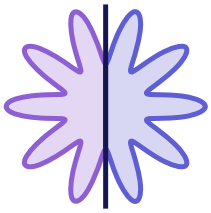
Chapter 3 also addresses sub-goal 1, now for the non-linear Navier-Stokes equations. We compare two methods for training data-driven closure models: a-priori and a-posteriori training. In a-priori training, the model output (predicted closure term) is fitted to the target (exact closure term computed from DNS). In a-posteriori training, the model is embedded into an LES simulation, which is then solved in time to produce a final LES solution.



Can a-priori training be stable?



Are we modeling the wrong stress in LES?



Should a discrete LES model possess the same symmetries as the continuous Navier-Stokes equations?

How can software, hardware, and neural networks be woven together to resolve turbulence?

The result of this LES simulation is fitted to the filtered DNS solution at the same final time. A-posteriori training has the advantage that the only data required is the filtered DNS solution, while a-priori training requires the exact definition of the closure term that also accounts for the discretization. However, a-posteriori training is more complicated to implement, since gradient descent optimization of the neural network parameters requires a differentiable LES solver. The main result is that if the discretization is accounted for correctly, a-priori training is sufficient for achieving accurate and stable neural-network based LES. If the discretization is not accounted for, then a-posteriori training is required to achieve these goals.

Chapter 4 addresses sub-goal 1 and 2. We derive exact expressions for the discretization-induced unresolved stresses arising from the finite volume method. We allow for arbitrary LES filters, whereas in chapter 3, we only considered FVM filters. The exact closure term is written in conservative form, which gives correct expressions for the target data of structural closure models. The discrete unresolved stress tensor is shown to have different physical structure than its continuous counterpart (it is non-symmetric and non-local).

Chapter 5 addresses sub-goal 2 (incorporating physical structure). We investigate how known symmetry properties of the Navier-Stokes equations can be built into neural network closure models. The universal approximation theorem guarantees that neural networks can represent any closure model, but this generality comes at a cost: the network must learn all structure from data. By building symmetries directly into the network architecture, we reduce the space of functions the network explores. We show that symmetry-preserving neural networks require less training data to achieve good generalization.

Chapter 6 addresses sub-goal 3 (software for data-driven closure modeling). We present `IncompressibleNavierStokes.jl`, an open-source Julia package for DNS and LES on staggered Cartesian grids. The solver uses energy-conserving finite volume discretizations and is fully differentiable through hand-written adjoint kernels, enabling a-posteriori training of neural network closure models. Hardware-agnostic kernels allow the same code to run on multi-threaded CPUs and GPUs. We discuss memory optimization techniques that enable double-precision DNS with up to 512^3 degrees of freedom on a single consumer GPU, as well as the tension between pure functional programming (needed for differentiability) and in-place array mutation (needed for memory efficiency). The chapter also covers the software development practices employed—version control, automated testing, continuous integration, documentation generation, and release management—and examines the implications of floating point arithmetic and reduced-precision GPU hardware for scientific computing.

In chapter 7, we summarize the main findings, revisit the research goals stated above, and present recommendations for future research.

The appendices contains additional material for each of the chapters 3 to 6. Chapter 2 is based on a conference paper and does not have an appendix. Chapter A contains additional results for the a-priori and a-posteriori training methods used in chapter 3. Chapter B includes derivations for LES-FVM used in chapter 4. Chapter C contains supplementary material for for chapter 5. Chapter D provides details about the kernels for the operators and adjoints in chapter 6.

2

LEARNING FILTERED DISCRETIZATION OPERATORS: NON-INTRUSIVE VERSUS INTRUSIVE APPROACHES

This chapter is based on the following article: Syver Døving Agdestein and Benjamin Sanderse. "Discretize First, Filter next – a New Closure Model Approach." In: 8th European Congress on Computational Methods in Applied Sciences and Engineering. *CIMNE, 2022.* DOI: *10.23967/eccomas.2022.094.*

The thesis introduction presented the LES closure problem and the discretize-first philosophy in general terms. This chapter studies these concepts in a simplified setting—the linear convection equation—where we can isolate and examine commutator errors without the complications of nonlinearity.

Simulating multi-scale phenomena such as turbulent fluid flows is typically computationally expensive. Filtering the smaller scales enables the use of coarse discretizations; however, this requires closure models to account for the effects of the unresolved scales on the resolved scales. The common approach is to filter the continuous equations, but this introduces commutator errors due to nonlinear terms, non-uniform filters, or boundary conditions.

We propose a new approach to filtering, where the equations are discretized first and then filtered. For a non-uniform filter applied to the linear convection equation, we show that the discretely filtered convection operator can be inferred using three methods: intrusive ('explicit reconstruction') or non-intrusive operator inference, either via 'derivative fitting' or 'trajectory fitting' (embedded learning). We show that explicit reconstruction and derivative fitting identify a similar operator and produce small errors, but that trajectory fitting requires considerable training effort to achieve comparable performance. Moreover, the explicit reconstruction approach is more prone to instabilities.

Core idea: discretize first, then filter—avoiding continuous commutator errors entirely.

2.1 INTRODUCTION

In LES, low-pass filters are applied to the governing equations to obtain equations for the large-scale components. These filters typically do not commute with the differential operators, nonlinear terms, or boundary conditions [186]. Furthermore, requiring that they do commute imposes severe restrictions on the choice of filters, e.g. by requiring vanishing moments or spatial uniformity [115, 186, 188]. This often conflicts with other restrictions, such as requiring the filter to be linked to or directly deduced from the spatial discretization (*implicit* LES) [22, 23]. More recently, several authors have proposed to learn the closure model *for a given filter and discretization* using deep neural networks [9, 13], thereby closing the equations after discretization.

In this chapter, we pursue this discrete approach, aiming to find discrete representations of filtered operators constructed from high-fidelity simu-

lations. We compare three different approaches: (i) explicit construction of the filtered operator using quadrature and reconstruction operators; (ii) least-squares reconstruction of the filtered operator, analogous to operator inference; (iii) reconstruction of the embedded filtered operator, analogous to "solver in the loop" methods [89, 107, 185].

We focus specifically on commutator errors arising from non-uniform filtering of linear operators (in which case filtering and differentiation do not commute), so that the filtered operators to be constructed remain linear, and we do not need to employ complex function approximators such as neural networks. This allows us to obtain a clear picture of how these three data-driven methods perform in constructing a filtered operator. In section 2.2, we present the discrete problem. In section 2.3, we present the three approaches for building the discrete filtered operator. In section 2.4, we show the results for the filtered convection equation, comparing the three approaches for two different filters.

2.2 PROBLEM STATEMENT

In this chapter, we consider the one-dimensional linear partial differential equation

$$\frac{\partial u}{\partial t}(x,t) = \mathcal{A}u(x,t), \quad x \in \Omega, \quad t > 0, \tag{2.1}$$

where $\Omega = [0,1]$ is a periodic domain, $\mathcal{A}$ is a linear differential operator with respect to the spatial variable x, and $u(x,0) = u_0(x)$ are the initial conditions.

A uniform periodic discretization of Ω is given by $\mathbf{x}^{(N)} = (x_n^{(N)})_{n=1}^N \in \mathbb{R}^N$, where N is the number of grid points, $x_n^{(N)} = n\Delta x^{(N)}$, and $\Delta x^{(N)} = 1/N$. A semi-discrete approximation to eq. (2.1) is then given by

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t}(t) = \mathbf{A}\mathbf{u}(t), \quad t > 0, \tag{2.2}$$

where $\mathbf{u}(t) \in \mathbb{R}^N$ is a semi-discrete solution and $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a discrete approximation of the differential operator $\mathcal{A}$, taking into account the periodicity. The exact semi-discrete solution is given by $\mathbf{u}(t) = \mathrm{e}^{t\mathbf{A}}\mathbf{u}_0$.

We are interested in the case where the spectral content of u is too large to be represented by the discrete solution. A low-pass filter is thus applied to obtain a new set of equations that can be resolved with a reasonably small N. Here, we consider a general linear continuous kernel filter $\mathcal{F}$ defined by

$$\bar{u}(x) = \mathcal{F}(u)(x) = \int_{\mathbb{R}} G(x,\xi)u(\xi)\,\mathrm{d}\xi \quad \forall x \in \Omega, \tag{2.3}$$

where $\bar{u}$ is the filtered solution and $G : \Omega \times \mathbb{R} \to \mathbb{R}$ is the kernel. Note that the function u is extended by periodicity to $\mathbb{R}$, i.e. $u(x) = u(x \mod |\Omega|)$ for

$x \notin \Omega$, thus allowing for infinitely long filter kernels such as a Gaussian kernel. The filter is non-uniform in space, so it is in general not of convolution type (in which case one would have $G(x,\xi) = G(\xi - x)\ \forall (x,\xi)$).

In many closure modelling approaches, the equations are filtered on the continuous level:

$$\frac{\partial \bar{u}}{\partial t} = \overline{(\mathcal{A}u)} = \mathcal{A}\bar{u} + \mathcal{C}(u,\bar{u}), \tag{2.4}$$

where $\mathcal{C}(u,\bar{u}) = \overline{(\mathcal{A}u)} - \mathcal{A}\bar{u}$ is the commutator error, typically caused by nonlinear terms. In the current setting where $\mathcal{A}$ is linear, a commutator error still arises due to the fact that non-uniform filtering and differentiation do not commute. The above equations are *unclosed* in the sense that $\mathcal{C}$ depends on u, and typically has to be modelled as a function of $\bar{u}$ only, i.e. $\mathcal{C}(u,\bar{u}) \approx \mathcal{M}(\bar{u})$, where $\mathcal{M}$ is a *closure model*, for example obtained by truncating the continuous Taylor series expansion of u in the integral (2.3) [64]. The filtered equations are then discretized on a coarse grid $\mathbf{x}^{(M)}, M \leq N$:

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t}(t) = \mathbf{A}^{(M)}\bar{\mathbf{u}}(t) + \mathbf{M}(\bar{\mathbf{u}}(t)), \quad t > 0, \tag{2.5}$$

where $\bar{\mathbf{u}}(t) \in \mathbb{R}^M$ is an approximation to $\bar{u}(\mathbf{x}^{(M)}, t)$, and $\mathbf{A}^{(M)} \in \mathbb{R}^{M\times M}$ and $\mathbf{M} : \mathbb{R}^M \to \mathbb{R}^M$ are coarse-grid approximations of the operators $\mathcal{A}$ and $\mathcal{M}$ respectively. In this approach, one thus first incurs a continuous closure model error, and then a coarse-scale discretization error.

We propose to obtain the discrete equations for the filtered solution in a different way, by *discretizing first*, and *then filtering*. We employ a discrete approximation of the filter, represented by a matrix $\mathbf{W}$ built using a *quadrature rule*:

$$\bar{\mathbf{u}} = \mathbf{W}\mathbf{u}. \tag{2.6}$$

Note that the filtered and unfiltered discrete solutions $\bar{\mathbf{u}}$ and $\mathbf{u}$ are represented on different grids $\mathbf{x}^{(M)}$ and $\mathbf{x}^{(N)}$, with $\bar{\mathbf{u}}(t) \in \mathbb{R}^M$ and $\mathbf{u}(t) \in \mathbb{R}^N$, $M \leq N$. An accurate discrete filter should be such that $\sum_n W_{mn} u(x_n^{(N)}, t) \approx \bar{u}(x_m^{(M)}, t)$, $m \in \{1, \dots, M\}$. It is assumed that all the relevant frequencies of u are resolved on $\mathbf{x}^{(N)}$, but not necessarily on $\mathbf{x}^{(M)}$. To close the filtered equations, the 'discretize first' approach requires an approximation of the inverse filtering operator by using a reconstruction rule, represented by a matrix $\mathbf{R} \in \mathbb{R}^{N\times M}$, such that $\mathbf{u} \approx \mathbf{R}\bar{\mathbf{u}}$. The discretely filtered solution $\bar{\mathbf{u}}$ is then obtained by solving the equations

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t}(t) = \bar{\mathbf{A}}\bar{\mathbf{u}}(t), \quad t > 0, \tag{2.7}$$

where $\bar{\mathbf{A}} = \mathbf{WAR}$ is the discretely filtered differential operator and the filtered initial conditions are given by $\bar{\mathbf{u}}(0) = \mathbf{W}\mathbf{u}_0$. Since the unfiltered

solution is reconstructed directly, eq. (2.7) is closed. In this approach, one thus incurs a fine-grid discretization error, a quadrature error, and a reconstruction error. The choice of $\bar{\mathbf{A}}$ may be seen as an effective closure model, and the resulting coarse-grid commutator error $\bar{\mathbf{A}} - \mathbf{A}^{(M)}$ can be computed, although it is not explicitly modelled (nor is it needed).

The problem that we address in this chapter is the following: given a fine-scale discretization $\mathbf{A}$ and a quadrature rule $\mathbf{W}$, can we infer $\bar{\mathbf{A}}$? To answer this question, we consider two different approaches, outlined in the next section.

2.3 DATA-DRIVEN OPERATOR INFERENCE

The two data-driven approaches to infer the matrix $\bar{\mathbf{A}}$ considered in this chapter are:

- *Indirectly* building $\bar{\mathbf{A}}$ by first explicitly building $\mathbf{W}$, $\mathbf{A}$, and $\mathbf{R}$ and then computing $\bar{\mathbf{A}} = \mathbf{WAR}$, see section 2.3.1. This approach is *intrusive* in the sense that $\mathbf{A} \in \mathbb{R}^{N\times N}$ is required.
- *Directly* building the reduced order operator $\bar{\mathbf{A}} \in \mathbb{R}^{M\times M}$ (see section 2.3.2), using gradient-free derivative fitting (section 2.3.2.1) and gradient-based embedded (section 2.3.2.2) approaches. This approach is *non-intrusive* since access to $\mathbf{A}$ is not required.

In the following, we will employ the snapshot matrices $\mathbf{U} \in \mathbb{R}^{N\times d}$, $\dot{\mathbf{U}} \in \mathbb{R}^{N\times d}$, $\bar{\mathbf{U}} \in \mathbb{R}^{M\times d}$, and $\dot{\bar{\mathbf{U}}} \in \mathbb{R}^{M\times d}$, containing unfiltered and filtered discrete solutions and their time derivatives at different time instances. Here, $d = n_{\mathrm{IC}} n_t$ is the number of snapshots, n_{IC} is the number of initial conditions, and n_t is the number of time instances at which the solutions are evaluated.

2.3.1 *Indirect approach: building the reconstruction matrix*

Using the midpoint rule [70], the quadrature weights $\mathbf{W}$ used for approximating the weighted integral (2.3) are constructed as follows:

$$W_{mn} = \frac{\tilde{G}(x_m, \xi_n)}{\sum_{i=1}^{N} \tilde{G}(x_m, \xi_i)}, \tag{2.8}$$

where $\tilde{G}(x, \xi) = \sum_{z\in\mathbb{Z}} G(x, \xi + z|\Omega|)$ accounts for the periodicity of Ω and potentially infinitely long kernel G. The sum is truncated to $z \in \{-1, 0, 1\}$ in the case of a sufficiently local filter support (e.g. for the Gaussian kernel). The normalization factor ensures that constant functions are preserved upon filtering. Note that higher order quadrature rules may be built by requiring that certain classes of functions are filtered exactly, but the simple rule (2.8)

is sufficient for our purposes (since N is assumed to be large) and has the advantage that it leads to positive weights if the kernel is positive.

Finding the reconstruction matrix $\mathbf{R}$ is not straightforward, as the matrix $\mathbf{W}$ is in general not invertible ($M \neq N$). $\mathbf{R}$ is therefore built by minimizing the expectation of the reconstruction error for a certain class of functions $\mathcal{U}$:

$$\min_{\mathbf{R}\in\mathbb{R}^{N\times M}} \mathbb{E}_{u\sim\mathcal{U}} L^{\text{fit}}(\mathbf{R}, \bar{u}(\mathbf{x}^{(M)}), u(\mathbf{x}^{(N)})) + \lambda L^{\text{prior}}(\mathbf{R}), \tag{2.9}$$

where L^{fit} measures the reconstruction error, L^{prior} penalizes deviation from prior assumptions on the form of $\mathbf{R}$, and λ is a regularization parameter. Setting the accuracy metric as the mean squared error

$$L^{\text{fit}}(\mathbf{R}, \bar{\mathbf{u}}, \mathbf{u}) = \|\mathbf{R}\bar{\mathbf{u}} - \mathbf{u}\|^2 \tag{2.10}$$

leads to a class of least squares problems. We consider the case where $L^{\text{prior}}(\mathbf{R}) = \|\mathbf{R}\|_F^2 = \sum_{nm} R_{nm}^2$, and after approximating the expectation value by a mean over the d snapshots $\mathbf{U}$ and $\bar{\mathbf{U}}$, we get

$$\mathbf{R} = \underset{\mathbf{R}\in\mathbb{R}^{N\times M}}{\text{argmin}} \frac{1}{d}\|\mathbf{R}\bar{\mathbf{U}} - \mathbf{U}\|_F^2 + \lambda\|\mathbf{R}\|_F^2 = \mathbf{U}\bar{\mathbf{U}}^\top(\bar{\mathbf{U}}\bar{\mathbf{U}}^\top + d\lambda\mathbf{I})^{-1}. \tag{2.11}$$

The resulting operator $\bar{\mathbf{A}}^{\text{int}} = \mathbf{WAR}$ is labelled as *intrusive* since it requires access to the full order model operator $\mathbf{A}$.

Intrusive approach: explicitly build $\bar{\mathbf{A}} = \mathbf{WAR}$ via data-driven reconstruction of $\mathbf{R}$.

2.3.2 *Direct approach: building the filtered operator*

The operator $\bar{\mathbf{A}}$ can also be constructed non-intrusively from filtered data. The solver used for eq. (2.2) is then considered to be a "black box", which returns data samples of the solution and its time derivative. Filtered samples are obtained by applying the discrete filter $\mathbf{W}$. The operator $\bar{\mathbf{A}}$ then follows from a minimization problem of the form

$$\min_{\bar{\mathbf{A}}\in\mathbb{R}^{M\times M}} \mathbb{E}_{\substack{u\sim\mathcal{U}\\ t\sim\mathcal{T}}} L^{\text{fit}}(\bar{\mathbf{A}}, \bar{u}(\mathbf{x}^{(M)}, \cdot), t) + \lambda_{\text{prior}} L^{\text{prior}}(\bar{\mathbf{A}}) + \lambda_{\text{stab}} L^{\text{stab}}(\bar{\mathbf{A}}), \tag{2.12}$$

where $\mathcal{U}$ is the space of solutions to eq. (2.1), $\mathcal{T}$ is a distribution of expected time instances, for example a uniform distribution on $[0, T]$ for some time horizon T, L^{prior} penalizes deviation from prior expected properties, and L^{stab} penalizes numerical instabilities that arise from the non-uniform filter (namely anti-diffusion caused by the non-uniformity of the filter). In the following, we set $L^{\text{stab}}(\bar{\mathbf{A}}) = \|\bar{\mathbf{A}} - \mathbf{D}^{(M)}\|_F^2$, where $\mathbf{D}^{(M)}$ is a discretization of the diffusion operator $\partial^2/\partial x^2$ on $\mathbf{x}^{(M)}$. This limits anti-diffusion due to eigenvalues with positive real part. In addition, we set $L^{\text{prior}}(\bar{\mathbf{A}}) = \|\bar{\mathbf{A}} - \mathbf{A}^{(M)}\|_F^2$, motivated by the assumption that the behavior of the filtered solution should be similar to that of the unfiltered solution (which is the motivation for LES

in the first place), resulting in a commutator error that is small compared to the exact right-hand side, i.e. $\mathcal{O}\left(\frac{\|\bar{\mathbf{A}}-\mathbf{A}^{(M)}\|_F}{\|\mathbf{A}^{(M)}\|_F}\right) = \mathcal{O}\left(\frac{\|\mathcal{C}\|}{\|\mathcal{A}\|}\right) \ll 1$.

L^{fit} is an instantaneous metric evaluating the performance of the operator $\bar{\mathbf{A}}$ with respect to a discrete filtered solution $\bar{\mathbf{u}}$, for which two different choices will be proposed, as explained in the next two sections.

2.3.2.1 *Gradient-free operator inference (derivative fitting)*

The first choice for L^{fit} is similar to the operator inference framework presented in [140], where the authors propose to fit the operators describing an ODE to data samples of the time derivative, by using performance metrics of the form

$$L^{\text{fit}}(\bar{\mathbf{A}}, \bar{\mathbf{u}}, t) = \left\| \bar{\mathbf{A}}\bar{\mathbf{u}} - \frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} \right\|^2 (t). \tag{2.13}$$

We call this approach 'derivative fitting'. Note that with this performance metric, neither the filtered solution $\bar{\mathbf{u}}$ nor its time derivative $\mathrm{d}\bar{\mathbf{u}}/\mathrm{d}t$ depend on $\bar{\mathbf{A}}$, and that the dependency on $\bar{\mathbf{A}}$ is quadratic in the loss function. As in eq. (2.11), approximating the expectation value by a sum over the d snapshots results in the following closed form expression for the operator:

$$\begin{aligned}\bar{\mathbf{A}}^{\text{DF}} &= \underset{\bar{\mathbf{A}}\in\mathbb{R}^{M\times M}}{\operatorname{argmin}} \frac{1}{d}\|\bar{\mathbf{A}}\bar{\mathbf{U}} - \dot{\bar{\mathbf{U}}}\|_F^2 + \lambda_{\text{prior}}\|\bar{\mathbf{A}} - \mathbf{A}^{(M)}\|_F^2 + \lambda_{\text{stab}}\|\bar{\mathbf{A}} - \mathbf{D}^{(M)}\|_F^2 \\ &= \left(\frac{1}{d}\dot{\bar{\mathbf{U}}}\bar{\mathbf{U}}^\top + \lambda_{\text{prior}}\mathbf{A}^{(M)} + \lambda_{\text{stab}}\mathbf{D}^{(M)}\right)\left(\frac{1}{d}\bar{\mathbf{U}}\bar{\mathbf{U}}^\top + (\lambda_{\text{prior}} + \lambda_{\text{stab}})\mathbf{I}\right)^{-1},\end{aligned} \tag{2.14}$$

Derivative fitting admits a closed-form solution—no iterative optimization needed.

where DF denotes derivative fitting.

2.3.2.2 *Embedded operator inference (solver-in-the-loop)*

The second choice for L^{fit} is the instantaneous performance metric

$$L^{\text{fit}}(\bar{\mathbf{A}}, \bar{\mathbf{u}}, t) = \|\mathbf{S}(\bar{\mathbf{A}}, \bar{\mathbf{u}}(0), t) - \bar{\mathbf{u}}(t)\|^2. \tag{2.15}$$

In this expression we introduced the notion of the ODE solver $\mathbf{S} : \mathbb{R}^{M\times M} \times \mathbb{R}^M \times \mathbb{R} \to \mathbb{R}^M$, where $\mathbf{S}(\bar{\mathbf{A}}, \bar{\mathbf{u}}_0, t)$ approximates the solution to eq. (2.7) for the given operator $\bar{\mathbf{A}}$ and filtered initial conditions $\bar{\mathbf{u}}_0$. In the previous section, the operator $\bar{\mathbf{A}}^{\text{DF}}$ was obtained by fitting known filtered solution trajectories $\bar{\mathbf{u}}$ to their time derivatives $\mathrm{d}\bar{\mathbf{u}}/\mathrm{d}t$ using least squares. In contrast, in this section the operator $\bar{\mathbf{A}}$ is inferred by fitting the predicted solution trajectories $\mathbf{S}(\bar{\mathbf{A}}, \bar{\mathbf{u}}(0), \cdot)$ to the known filtered solution trajectories $\bar{\mathbf{u}}$. The operator $\bar{\mathbf{A}}$ is thus *embedded* in the solver $\mathbf{S}$ [89, 107, 185].

Assuming the program $\mathbf{S}$ is differentiable, i.e. we have access to the quantity $(\partial \mathbf{S}/\partial \bar{\mathbf{A}})(\bar{\mathbf{A}}, \bar{\mathbf{u}}_0, t) \in \mathbb{R}^{M\times M\times M}$, we will use a gradient-descent based optimization algorithm to identify the correct $\bar{\mathbf{A}}$, using adjoint-mode differentiation with checkpointing to compute the vector-Jacobian products [34, 147].

Note that each evaluation of the loss function requires solving the ODE (2.7) of size M for the given operator $\bar{\mathbf{A}}$ (solver-in-the-loop). For more computationally expensive problems, e.g. three-dimensional non-linear equations such as the Navier-Stokes equations, evaluating L^{fit} at all initial conditions and time steps in the training dataset can quickly get expensive. We will use *stochastic* gradient descent [25] to reduce this computational cost, by evaluating the loss and its gradient for only a few initial conditions and time instances at a time. In the following, the operator fitted while embedded in the solver will be denoted by $\bar{\mathbf{A}}^{\text{emb}}$.

Embedded approach: solver-in-the-loop optimization fits predicted trajectories to reference data.

2.4 NUMERICAL RESULTS: CONVECTION EQUATION

The source code for the simulations and figures is available at https://github.com/agdestein/DiscreteFiltering.jl. All the simulations were run on a recent laptop computer.

We consider the convection equation with unit velocity, obtained by setting $\mathcal{A} = -\partial/\partial x$ in eq. (2.1). To test the effectiveness of the three approaches to infer the filtered operator $\bar{\mathbf{A}}$, we consider two filters: the top-hat filter, for which

$$G(x,\xi) = \begin{cases} \frac{1}{2h(x)}, & |\xi - x| \le h(x), \\ 0, & \text{otherwise}, \end{cases} \tag{2.16}$$

and the Gaussian filter, for which

$$G(x,\xi) = \sqrt{\frac{3}{2\pi h^2(x)}} \mathrm{e}^{-\frac{3(\xi-x)^2}{2h^2(x)}}, \tag{2.17}$$

where $h(x)$ is a variable filter radius. For both filters, it is chosen to be

$$h(x) = \left(1 + \frac{1}{3}\sin(2\pi x)\right) h_0 \tag{2.18}$$

with $h_0 = 1/50$. At $x = 1/4$, the filter radius is thus twice as large as at $x = 3/4$, and filtered waves moving from $x = 1/4$ to $x = 3/4$ are subject to *anti-diffusion*, where high-frequency components grow. Small numerical errors get amplified on this interval, and finding a stable $\bar{\mathbf{A}}$ is thus not straightforward. Note also that the Gaussian standard deviation $h/\sqrt{3}$ is chosen to give the same Taylor series expansion for the local attenuation factor as a function of kh in spectral space as for the top-hat filter, where k is the frequency.

Non-uniform filtering introduces anti-diffusion, making stability a key challenge for inferred operators.

Given initial conditions, closed form expressions are available for the unfiltered and filtered solutions and their time derivatives. However, we use

DNS to generate data samples by solving eq. (2.2), as this is the approach one would take for equations that do not admit closed-form solutions. For this purpose, the continuous convection operator $\mathcal{A}$ is discretized using a sixth order central difference stencil on the fine grid $\mathbf{x}^{(N)}$:

$$\mathbf{A} = \frac{1}{60\Delta x^{(N)}} \operatorname{circ}_N(1, -9, 45, 0, -45, 9, -1), \tag{2.19}$$

and similarly for $\mathbf{A}^{(M)}$, where $\operatorname{circ}_\eta(s_{-\alpha}, \dots, s_\alpha) \in \mathbb{R}^{\eta\times\eta}$ denotes a circulant matrix where the i-th periodically extended superdiagonal is filled with the stencil entry s_i for $i \in \{-\alpha, \dots, \alpha\}$. Similarly, we build $\mathbf{D}^{(M)} = \operatorname{circ}_M(2, -27, 270, -490, 270, -27, 2)/180 \left(\Delta x^{(M)}\right)^2$ (of order six).

The unfiltered DNS equations (2.2) are discretized with $N = 1000$ points, and solved using an adaptive fourth order Runge-Kutta time integrator [184] with absolute and relative tolerances $\epsilon_{\text{abs}} = 10^{-10}$ and $\epsilon_{\text{rel}} = 10^{-8}$. The resulting solution is saved at n_t time points. The time derivative snapshots are obtained by evaluating the DNS right hand side $\mathbf{Au}$. The filters are discretized using the periodically extended quadrature rule (2.8). The resulting $\mathbf{W}$ is then used to generate the reference solutions $\mathbf{Wu}$ and time derivatives $\mathbf{WAu}$ for a given M.

Four datasets $\mathcal{D} \in \{\mathcal{D}^{\text{train}}, \mathcal{D}^{\text{valid}}, \mathcal{D}^{\text{test}}, \mathcal{D}^{\text{long}}\}$ are created. They consist of waves with a maximum frequency of $K = 250$ and $n_{\text{IC}}^{\mathcal{D}}$ different initial conditions $(u_0^{\mathcal{D},i})_{i\in\{1,\dots,n_{\text{IC}}^{\mathcal{D}}\}}$ given by

$$u_0^{\mathcal{D},i}(x) = \sum_{k=0}^{K} \frac{1 + \epsilon^{\mathcal{D},i}}{(5+k)^2} \cos(2\pi k x + \theta^{\mathcal{D},i}), \quad i \in \{1, \dots, n_{\text{IC}}^{\mathcal{D}}\}, \tag{2.20}$$

where $\epsilon^{\mathcal{D},i} \sim \mathcal{N}(0, \frac{1}{5})$ and $\theta^{\mathcal{D},i} \sim \mathcal{U}([0, 2\pi])$ are sampled independently for each $\mathcal{D}$ and i. Note that the fine grid $\mathbf{x}^{(N)}$ resolves all the K frequencies. This choice of coefficients creates a profile where the "turbulent" features (high frequencies) are small compared to the large features, but are still visually different from their filtered counterparts. One of the resulting initial conditions and its coefficients are shown for both filters in fig. 2.1, along with the common filter radius h.

The parameters for the datasets are given in table 2.1. $\mathcal{D}^{\text{train}}$ is used to fit the operators. $\mathcal{D}^{\text{valid}}$ is used to choose the hyperparameters λ. $\mathcal{D}^{\text{test}}$ is used to measure the generalization capacity of the inferred models. $\mathcal{D}^{\text{long}}$ is used to measure the long term stability of the operators.

The operators $\mathbf{R}$ and $\bar{\mathbf{A}}^{\text{DF}}$ are both fitted to $\mathcal{D}^{\text{train}}$ using least squares (eq. (2.11) and eq. (2.14), respectively), while $\bar{\mathbf{A}}^{\text{emb}}$ is fitted using 10^4 iterations of stochastic gradient descent with the ADAM optimizer [87] with step size 0.001. A grid search over the regularization parameter $\lambda \in \{10^{-12}, 10^{-11}, \dots, 10^0\}$

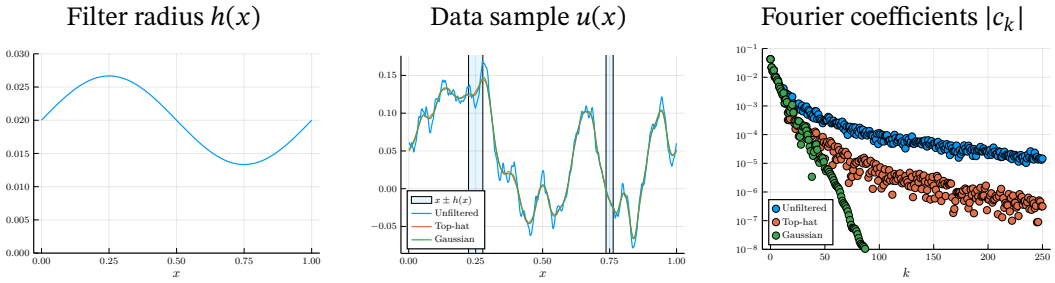


Figure 2.1: Left: Filter radius h used in the simulations. Middle and right: Unfiltered and filtered initial conditions and their Fourier coefficients for one of the training samples.

$\mathcal{D}$	n_{IC}	n_t	T
train	1000	50	0.1
valid	20	10	1.0
test	100	20	1.0
long	50	500	100.0

Table 2.1: Parameters used to build the different datasets $\mathcal{D}$: number of initial conditions ($n_{\text{IC}}^{\mathcal{D}}$), number of time points ($n_t^{\mathcal{D}}$), and convection time ($T^{\mathcal{D}}$).

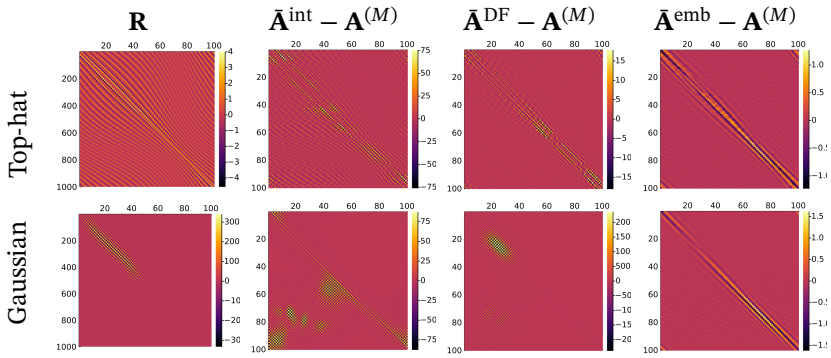


Figure 2.2: Different operators for $M = 100$. Note that $\mathbf{R} \in \mathbb{R}^{1000\times100}$ is non-square, despite the appearance. Here, $\mathbf{A}^{(100)} = \text{circ}_{100}(5/3, -15, 75, 0, -75, 15, -5/3)$.

is performed to obtain the smallest time averaged validation error $\frac{1}{n_t}\sum_{i=1}^{n_t^{\text{valid}}} e_{\bar{\mathbf{A}}}^{\text{valid}}(t_i^{\text{valid}})$, where the error for a given dataset $\mathcal{D}$ at a time t is given by

$$e_{\bar{\mathbf{A}}}^{\mathcal{D}}(t) = \frac{1}{n_{\text{IC}}^{\mathcal{D}}} \sum_{\bar{\mathbf{u}}\in\mathcal{D}} \frac{\|\mathbf{S}(\bar{\mathbf{A}}, \bar{\mathbf{u}}(0), t) - \bar{\mathbf{u}}(t)\|}{\|\bar{\mathbf{u}}(t)\|}. \tag{2.21}$$

The resulting operators are shown in fig. 2.2 for both filters (2.16) and (2.17) with $M = 100$. Note that this grid is 10× coarser than the DNS grid and cannot represent all frequencies. While the top-hat reconstructor and resulting filtered operator are dense, with large off-diagonal elements, the corresponding Gaussian operators admit a sparser structure. One can clearly recognize the filter radius profile h in the different operators, with more blurred and stretched out features on the left half of the domain. Note also that the embedded operator $\bar{\mathbf{A}}^{\text{emb}}$ is closer to the unfiltered operator $\mathbf{A}^{(M)}$ than $\bar{\mathbf{A}}^{\text{int}}$ and $\bar{\mathbf{A}}^{\text{DF}}$, possibly due to a local minimum near the initial guess $\mathbf{A}^{(M)}$.

In fig. 2.3, the time averaged relative error $\frac{1}{n_t^{\text{test}}}\sum_{i=1}^{n_t^{\text{test}}} e_{\bar{\mathbf{A}}}^{\text{test}}(t_i^{\text{test}})$ is shown for both filters as a function of M. For the unfiltered operator $\mathbf{A}^{(M)}$ the error initially decreases with M, but stabilizes at a point between 3% and 5% after $M = 100$. This is due to the true commutator error $\mathcal{C}$, confirming that a closure model is indeed necessary to predict the filtered dynamics. For $M < 100$ the discretization error is more important than the commutator error, but this depends on the test case. The operators $\bar{\mathbf{A}}^{\text{int}}$ and $\bar{\mathbf{A}}^{\text{DF}}$ have very similar error profiles. In fact, they only differ by the regularization terms, as the data fitting terms are identical (up to a linear transformation). For large M, they both admit errors that are orders of magnitude smaller than $\mathbf{A}^{(M)}$. Note that for small M, the algorithm gives a high weight to the prior

Key finding: intrusive and derivative fitting identify essentially the same operator—non-intrusive is sufficient.

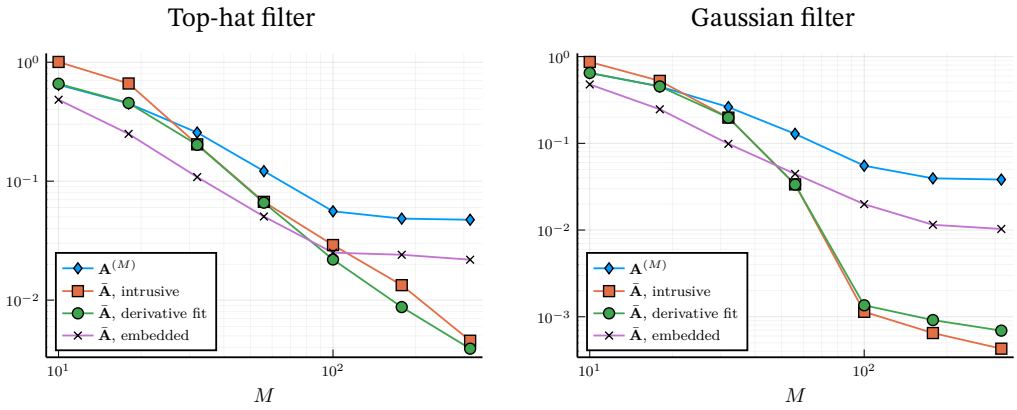


Figure 2.3: Average relative error on test dataset $\mathcal{D}^{\text{test}}$ for increasing coarse grid precision M for the top-hat filter (left) and Gaussian filter (right). The error measures generalization capacity of the ODE-model for new initial conditions and time points.

for $\bar{\mathbf{A}}^{\text{DF}}$, resulting in the initial overlap with $\mathbf{A}^{(M)}$. The embedded operator $\bar{\mathbf{A}}^{\text{emb}}$ performs the best for small M, but the errors appear to stagnate like those of its initial guess $\mathbf{A}^{(M)}$ for large M. A possible explanation is that the gradients quickly become small during optimization, potentially due to vanishing gradients in back-propagation through the solver. The operator may also have converged to a local minimum near the initial guess. For all the operators, the errors for the Gaussian filters are smaller than for the top-hat filter, as the higher frequencies are filtered more strongly.

To further investigate the long-term stability of the inferred operators, the evolution of the mean relative error $e^{\text{long}}_{\bar{\mathbf{A}}}$ is computed on the long-term testing data $\mathcal{D}^{\text{long}}$ for $M = 100$. The results are shown in fig. 2.4. At $t = 0$, all operators have an error of zero, since the initial conditions are given by the filtered DNS solution. The unfiltered coarse-scale operator $\mathbf{A}^{(M)}$ initially has the highest errors, as it does not account for the filter. Moreover, the error profile is not fully periodic but increases at a linear rate. This indicates that the filtered solutions contain frequencies not fully resolved by the sixth order central difference convection stencil on the coarse grid. The operators $\bar{\mathbf{A}}^{\text{int}}$ and $\bar{\mathbf{A}}^{\text{DF}}$ initially perform the best and show similar error profiles. However, the intrusive operator is not regularized for stability, and the errors start increasing rapidly after a few convection periods, eventually surpassing the ones of the unfiltered operator. This is likely due to instabilities from anti-diffusion on the parts of the domain where the filter radius h is decreasing (on $[1/4, 3/4]$). For $\bar{\mathbf{A}}^{\text{DF}}$ the errors seem to be increasing at the same rate as $\mathbf{A}^{(M)}$. The errors for the embedded operator $\bar{\mathbf{A}}^{\text{emb}}$ stay about one order of magnitude below those of $\mathbf{A}^{(M)}$, and increase at the same rate.

The intrusive approach lacks stability regularization, causing long-term error growth from anti-diffusion.

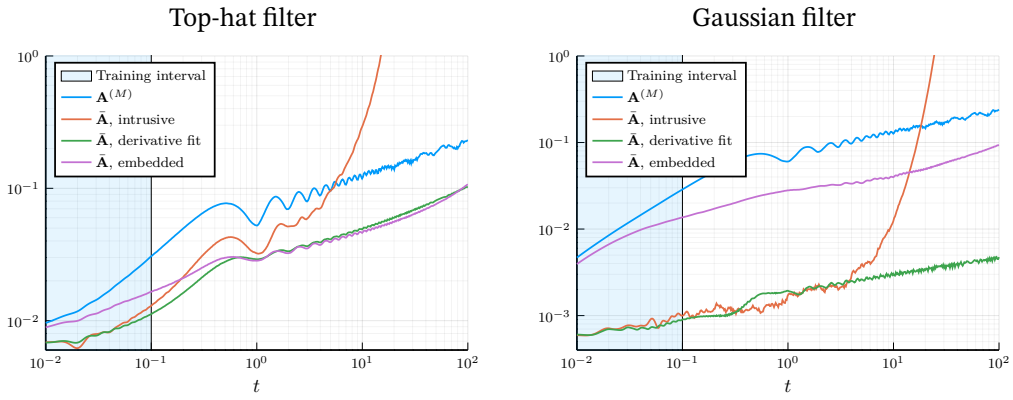


Figure 2.4: Evolution of the relative error on $\mathcal{D}^{\text{long}}$ for $M = 100$ using the top-hat filter (left) and Gaussian filter (right). The abscissa show the time elapsed starting from the filtered initial conditions to one hundred convection periods on Ω.

Note that putting the continuously filtered initial conditions $\bar{u}_0$ into the unfiltered continuous equations (2.1) would lead to errors, as the commutator error $\mathcal{C}$ would not be accounted for. However, the error profile would still be periodic, since the exact solution to eq. (2.1) is periodic for all initial conditions. In particular, the error would be exactly zero when $t = n$ for all $n \in \mathbb{N}$. This does not appear to be the case for the unfiltered reference operator $\mathbf{A}^{(M)}$. The inferred operators thus learn not only to account for the explicit filter $\mathcal{F}$, but also for the under-resolved discretization $\mathbf{x}^{(M)}$.

Inferred operators compensate for both the filter and *coarse-grid discretization errors—a key advantage of the discretize-first approach.*

2.5 CONCLUSION

In this chapter, we have considered the effect of applying discrete non-uniform filters to a linear partial differential equation. Three discrete closure models were compared, each using filtered data samples to infer operators. First, we built an explicit reconstruction model that uses the DNS equations to evolve the system in time. This approach yielded satisfactory short-term errors but was shown to lack long-term stability. It also requires access to the full-order model operator $\mathbf{A}$ on the fine discretization. Second, we built the discrete filtered operator directly, using gradient-free and embedded approaches. The former produced error profiles similar to those of the intrusive approach, while the latter yielded less significant improvements for the cases considered. When building the filtered operator directly, one learns not only the correction for the filter, but also potentially improves the finite difference approximation of the unfiltered operator itself.

For the convection equation, the least-squares time-derivative fit on the coarse grid was found to perform as well as the intrusive approach involving the DNS grid, and was sufficient for inferring operators with good extrapolation and generalization properties. In contrast, the embedded model did not achieve the same level of accuracy. It requires the selection of additional hyperparameters, including the initial guess, number of iterations, step size, regularization, and sensitivity algorithm for computing the gradients. Care must be taken to avoid local minima.

In conclusion, this study demonstrates that constructing a discrete closure model with an embedded strategy, even for a linear model problem, leads to a challenging optimization problem with numerous design choices (such as hyperparameters) and does not readily yield more accurate or stable results than the more straightforward 'derivative fitting' approach. This suggests that more advanced parameterizations, such as neural networks, will also require careful consideration to be effective in an embedded set-up.

3

DISCRETIZE FIRST, FILTER NEXT: LEARNING DIVERGENCE-CONSISTENT CLOSURE MODELS FOR LARGE-EDDY SIMULATION

This chapter is based on the following article:
Syver Døving Agdestein and Benjamin Sanderse. "Discretize First, Filter next: Learning Divergence-Consistent Closure Models for Large-Eddy Simulation." In: Journal of Computational Physics *522 (Feb. 2025), p. 113577.* DOI: *10.1016/j.jcp.2024.113577*.

CRediT author statement
***Syver Døving Agdestein**: Conceptualization, Formal analysis, Methodology, Software, Validation, Visualization, Writing – Original Draft*
***Benjamin Sanderse**: Funding acquisition, Project administration, Supervision, Writing – Review & Editing*

The previous chapter studied the discretize-first approach for linear operators, where commutator errors could be examined without neural networks. This chapter extends the approach to the nonlinear Navier-Stokes equations, where neural networks are used as closure models and the interplay between discretization consistency and training methodology becomes critical.

We propose a new neural-network-based large eddy simulation framework for the incompressible Navier-Stokes equations based on the paradigm "discretize first, filter and close next". This leads to full model-data consistency and allows neural closure models to be employed in the same environment in which they were trained. Since the LES discretization error is included in the learning process, the closure models can learn to account for the discretization.

Furthermore, we employ a divergence-consistent discrete filter defined through face-averaging and provide novel theoretical and numerical filter analysis. This filter preserves the discrete divergence-free constraint by construction, unlike general discrete filters such as volume-averaging filters. We show that using a divergence-consistent LES formulation coupled with a convolutional neural closure model produces stable and accurate results for both a-priori and a-posteriori training, while a general (divergence-inconsistent) LES model requires a-posteriori training or other stability-enforcing measures.

3.1 INTRODUCTION

In LES, the contributions of the unresolved scales are grouped into a commutator error term $c(u)$, which is approximated by a closure model $m(\bar{u}, \theta) \approx c(u)$ that depends only on the resolved scales. The filter can be either explicitly known or implicit, where the discretization itself acts as a filter [14].

Recently, machine learning has been used to learn closure models, focusing mostly on implicit LES [12–14, 53, 75, 91, 92, 107, 119, 165–167]. The idea is to represent the closure model $m(\bar{u}, \theta)$ by an artificial neural network (ANN). ANNs are in principle well-suited candidates, as they are universal function approximators [11, 35]. However, using ANNs as closure models typically suffers from stability issues, which have been attributed to a so-called model-data inconsistency: the environment in which the neural network is trained differs from the one in which it is deployed [14]. Several

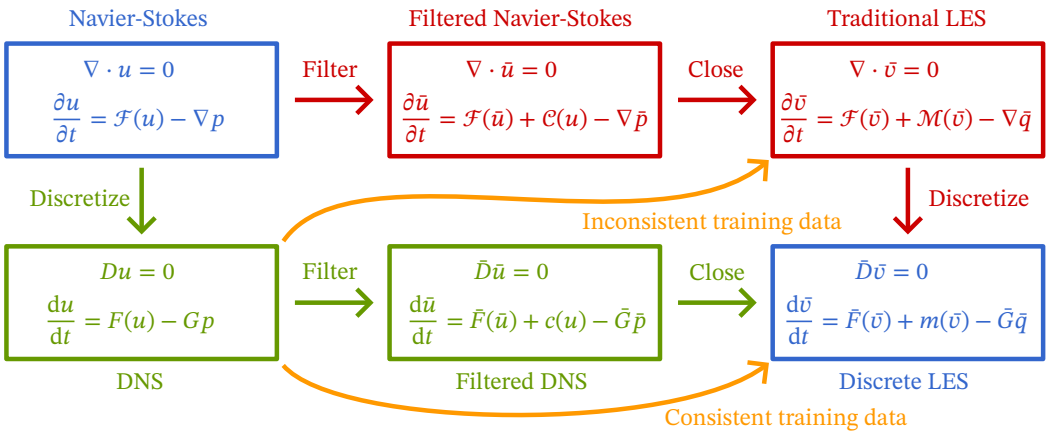


Figure 3.1: Proposed route (in green) to a discrete LES model, based on "discretize first, then filter" instead of "filter first, then discretize" (in red). The term $\mathcal{F}(u)$ contains the convective and diffusive terms.

approaches like backscatter clipping, a-posteriori training and projection onto an eddy-viscosity basis have been used to enforce stability [12, 89, 107, 138] – for an overview, see [157]. Our view is that *one of the problems at the root of the model-data inconsistency is a discrepancy between the LES equations (obtained by first filtering the continuous Navier-Stokes equations, then discretizing) and the training data (obtained by discretizing the Navier-Stokes equations and then applying a discrete filter).*

Key insight: discretize first, then filter. This avoids model-data inconsistency by generating training data in the same discrete environment where the closure model is deployed.

Our key insight is that the LES equations can also be obtained by "discretizing first" instead of "filtering first": following the green instead of the red route in fig. 3.1. By discretizing the PDE first and then applying a *discrete* filter, the model-data inconsistency can be avoided and one can generate exact training data for the discrete LES equations. Training data obtained by filtering discrete DNS solutions is fully consistent with the environment where the discrete closure model is used. The resulting LES equations do not have a coarse grid discretization error, but an underlying fine grid discretization error and a commutator error from the discrete filter, which can be learned using a neural network. In our recent work [1], we showed the benefits of the "discretize first" approach on a 1D convection equation. With the discretize-first approach we obtain stable results without the need for the stabilizing techniques mentioned above (backscatter clipping, a-posteriori training, projection onto an eddy-viscosity basis).

In this chapter, the goal is to extend the "discretize first" approach to the full 3D incompressible Navier-Stokes equations. A major challenge that arises in incompressible Navier-Stokes is the presence of the divergence-free constraint. We show that discrete filters are in general not divergence-consistent, meaning that divergence-free DNS solutions do not remain

divergence-free upon filtering. Kochkov et al. used a face-averaging filter to achieve divergence-consistency [89]. We employ this filter and show that it leads to a different set of equations than non-divergence-consistent filters. The main result is that our divergence-consistent neural closure models lead to stable simulations.

In addition, we remark that divergence-consistent filtering is an important step towards LES closure models that satisfy an energy inequality. Such models were developed by us in [187] for one-dimensional equations with quadratic nonlinearity (Burgers, Korteweg–de Vries). When extending this approach to 3D LES, the derivation of the energy inequality hinges on having a divergence-free constraint on the filtered solution field.

This chapter is structured as follows. In section 3.2, we present the discrete DNS equations that serve as the ground truth, based on a second-order accurate finite volume discretization on a staggered grid. In section 3.3, we introduce discrete filtering, derive unclosed equations for the large scales, and show that the filtered velocity is not automatically divergence-free. We then present two discrete filters on a staggered grid, one of which is divergence-consistent. In section 3.4, we present our discrete closure modelling framework, resulting in two discrete LES formulations, and discuss their validity in terms of divergence-consistency, the choice of closure model, and how the model parameters can be learned. In section 3.5, we present the results of numerical experiments on a forced turbulence test case. Section 3.6 concludes with final remarks.

Additional details are included in the appendices. In section A.1, we explain the numerical experiments. In section A.2, we show that the face-averaging filter is divergence-consistent on both uniform and non-uniform grids. In section A.3, we show the problems that can occur when filtering before differentiating the divergence-free constraint. In section A.4, we analyze the transfer functions of a continuous face-averaging filter and a continuous volume-averaging filter. In section A.5, we analyze the difference between "discretize first" and "filter first" for an analytical solution to the incompressible Navier-Stokes equations. In section A.6, we provide results for two more test cases: a decaying turbulence test case in 2D, and a forced turbulence test case in 3D.

3.2 DIRECT NUMERICAL SIMULATION OF ALL SCALES

In this section, we present the continuous Navier-Stokes equations and define a discretization aimed at resolving all scales of motion. The resulting discrete equations serve as the ground truth for learning an equation for the large scales.

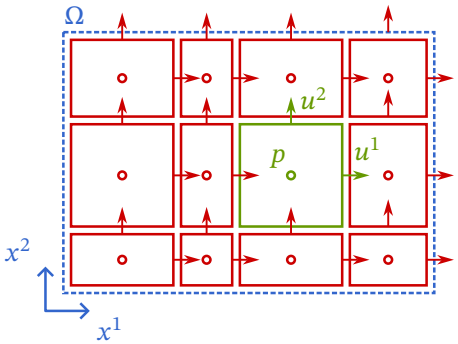


Figure 3.2: Finite volume discretization on a staggered grid. The pressure is defined in the volume center, and the velocity components on the volume faces.

3.2.1 *The Navier-Stokes equations*

The incompressible Navier-Stokes equations describe conservation of mass and conservation of momentum, which can be written as a divergence-free constraint and an evolution equation:

$$\nabla \cdot u = 0, \tag{3.1}$$

$$\frac{\partial u}{\partial t} + \nabla \cdot (uu^\mathsf{T}) = -\nabla p + \nu \nabla^2 u + f, \tag{3.2}$$

where $\Omega \subset \mathbb{R}^d$ is the domain, $d \in \{2, 3\}$ is the spatial dimension, $u = (u^1, \dots, u^d)$ is the velocity field, p is the pressure, ν is the kinematic viscosity, and $f = (f^1, \dots, f^d)$ is the body force per unit of volume. The velocity, pressure, and body force are functions of the spatial coordinate $x = (x^1, \dots, x^d)$ and time t. For the remainder of this work, we assume that Ω is a rectangular domain with periodic boundaries, and that f is constant in time.

3.2.2 *Spatial discretization*

For the discretization scheme, we use a staggered Cartesian grid as proposed by Harlow and Welch [71]. Staggered grids have excellent conservation properties [102, 141]; in particular, their exact discrete divergence-free constraint is important for this work. The finite volume discretization is explained in chapter 6. Details about the discretization can be found in section D.1.

We partition the domain Ω into N finite volumes. Let $u(t) \in \mathbb{R}^{dN}$ and $p(t) \in \mathbb{R}^N$ be vectors containing the unknown velocity and pressure components in their canonical positions as shown in fig. 3.2. These should not be confused with their space-continuous counterparts $u(x, t)$ and $p(x, t)$, which will no longer be referred to in what follows. The discrete and continuous versions of u and p have the same physical dimensions.

Equations (3.1) and (3.2) are discretized as

$$Du = 0, \tag{3.3}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} = F(u) - Gp, \tag{3.4}$$

where $D \in \mathbb{R}^{N \times dN}$ is the divergence operator, $G = -\Omega_u^{-1} D^{\mathsf{T}} \Omega_p \in \mathbb{R}^{dN \times N}$ is the gradient operator, $\Omega_u \in \mathbb{R}^{dN \times dN}$ and $\Omega_p \in \mathbb{R}^{N \times N}$ are element-wise scaling operators containing the velocity and pressure volume sizes, and $F(u) \in \mathbb{R}^{dN}$ contains the convective, diffusive, and body force terms.

3.2.3 *Pressure projection*

The two vector equations (3.3) and (3.4) form an index-2 differential-algebraic equation system [68, 69], consisting of a divergence-free constraint and an evolution equation. Given u, the pressure can be obtained by solving the discrete Poisson equation $0 = DF(u) - DGp$, which is obtained by differentiating the divergence-free constraint in time. This can equivalently be written as

$$Lp = \Omega_p DF(u), \tag{3.5}$$

where the Laplace matrix $L = \Omega_p DG = -\Omega_p D \Omega_u^{-1} D^{\mathsf{T}} \Omega_p$ is symmetric and negative semi-definite. We denote by $L^\dagger$ the solver to the scaled pressure Poisson equation (3.5). Since no boundary value for the pressure is prescribed, L is rank-1 deficient and the pressure is only determined up to a constant. We set this constant to zero by choosing

$$L^\dagger = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} L & e \\ e^{\mathsf{T}} & 0 \end{pmatrix}^{-1} \begin{pmatrix} I \\ 0 \end{pmatrix}, \tag{3.6}$$

where $e = (1, \dots, 1) \in \mathbb{R}^N$ is a vector of ones and the additional degree of freedom enforces the constraint of an average pressure of zero (i.e. $e^{\mathsf{T}} L^\dagger = 0$) [110]. In this case we still have $LL^\dagger = I$, even though $L^\dagger L \neq I$.

We now introduce the projection operator P, which plays an important role in the development of our new closure model strategy in section 3.3: $P = I - GL^\dagger \Omega_p D$. It is used to make velocity fields divergence-free [37], since $DP = D - \Omega_p^{-1} LL^\dagger \Omega_p D = 0$. It naturally follows that P is a projector, since $P^2 = P - GL^\dagger \Omega_p DP = P$.

Having defined P, it is (at least formally) possible to eliminate the pressure from equations (3.3)-(3.4) into a single "pressure-free" evolution equation for the velocity [154], given by

$$\frac{\mathrm{d}u}{\mathrm{d}t} = PF(u). \tag{3.7}$$

This way, an initially divergence-free velocity field u stays divergence-free regardless of what value the total force F takes. We note that equation (3.7) alone does not enforce the divergence-free constraint, as it also requires the initial conditions to be divergence-free.

$L^\dagger$ and P are non-local operators that are not explicitly assembled; instead, their action on vector fields is computed on demand using an appropriate linear solver. Formulation (3.7) is used as a starting point for developing a new filtering technique in section 3.3.

3.2.4 *Time discretization*

We use Wray's low storage third order Runge-Kutta method (Wray3) for the incompressible Navier-Stokes equations [154, 206]. While explicit methods may require smaller time steps (depending on the problem-specific trade-off between stability and accuracy), they are easier to differentiate with automatic differentiation tools.

Given the solution u_n at a time t_n, the next solution at a time t_{n+1} is given by

$$u_{n+1} = u_n + \Delta t_n \sum_{i=1}^{s} b_i k_i, \tag{3.8}$$

where

$$k_i = PF\left(u_n + \Delta t_n \sum_{j=1}^{i-1} a_{ij} k_j\right), \tag{3.9}$$

$\Delta t_n = t_{n+1} - t_n$, s is the number of stages, $a \in \mathbb{R}^{s\times s}$, and $b \in \mathbb{R}^s$. In practice, each of the RK steps (3.9) is performed by first computing a tentative (non-divergence-free) velocity field, subsequently solving a pressure Poisson equation, and then correcting the velocity field to be divergence-free. As the method is explicit, this is equivalent to (3.9) and does not introduce a splitting error [154]. For Wray3, we set $s = 3$, $b_1 = 1/4$, $b_2 = 0$, $b_3 = 3/4$, $a_{21} = 8/15$, $a_{31} = 1/4$, $a_{32} = 5/12$, and the other coefficients $a_{ij} = 0$.

3.3 DISCRETE FILTERING AND DIVERGENCE-CONSISTENCY

The DNS discretization presented in the previous section is generally too expensive to simulate for problems of practical interest and is only used to generate reference data for a limited number of test cases. In this section, we present discrete filtering from the fine DNS grid to a coarse grid in order to alleviate the computational burden. This means we filter the discretized equations, in contrast to many existing approaches, which filter the *continuous* Navier-Stokes equations, and then apply a discretization.

The advantages of "discretizing first" were already mentioned in section 3.1. However, one disadvantage of "discretizing first" is that the filtered velocity field is in general not divergence-free. Kochkov et al. pointed out that using a face-averaging filter preserves the divergence-free constraint for the filtered velocity [89]. We will employ this filter and compare it to a non-divergence-preserving volume-averaging filter.

3.3.1 *Filtering from fine to coarse grids*

We consider two computational grids: a fine grid of size $N \in \mathbb{N}^d$ and a coarse grid of size $\bar{N} \in \mathbb{N}^d$, with $\bar{N}_\alpha \leq N_\alpha$ for all $\alpha \in \{1, \dots, d\}$. The operators D, F, G, P, etc., are defined on the fine grid. On the coarse grid, similar operators are denoted $\bar{D}$, $\bar{F}$, $\bar{G}$, $\bar{P}$, etc.

Consider a flow problem. We assume that the flow is fully resolved on the fine grid, meaning that the grid spacing is at most half the size of the smallest significant spatial structure of the flow. The resulting fully resolved solution $u \in \mathbb{R}^{dN}$ is referred to as the *DNS solution*. In addition, we assume that the flow is not fully resolved on the coarse grid, meaning that the coarse grid spacing is larger than the smallest significant spatial structure of the flow. The aim is to solve for the large scale features of u on the coarse grid. For this purpose, we construct a discrete spatial filter $\Phi \in \mathbb{R}^{d\bar{N} \times dN}$. The resulting filtered DNS velocity field is given by

$$\bar{u} = \Phi u \in \mathbb{R}^{d\bar{N}}, \tag{3.10}$$

and is a coarse-grid quantity. We stress that $\bar{u}$ is by definition a consequence of the DNS. It is *not* obtained by solving the Navier-Stokes equations on the coarse grid. That is instead the goal in the next sections.

Since Φ is a coarse-graining filter, it does not generally commute with discrete differential operators. While the divergence-free constraint is preserved for continuous convolutional filters, this is not automatically the case for discrete filters. We examine this property in detail and investigate its impact on the resulting large-scale equations.

3.3.2 *Equation for large scales*

Directly filtering the differential-algebraic system (3.3)-(3.4) raises several challenges. These are detailed in section A.3 and summarized here:

- The filter Φ acts on inputs defined at the velocity points and is designed for filtering the momentum equation. It is not immediately clear how to filter the divergence-free constraint (which is defined at the pressure points), or whether a second filter needs to be defined for the pressure points.

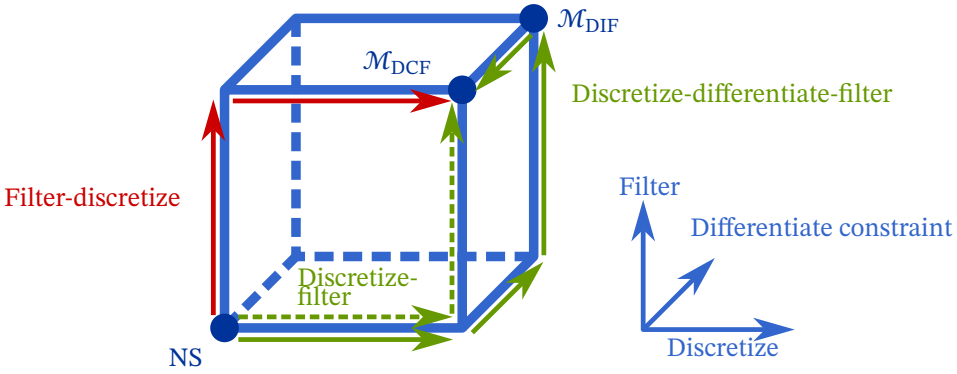


Figure 3.3: Alternative view of fig. 3.1 to highlight the effect of differentiating the constraint. The red arrows show the traditional route of filtering first and then discretizing. The solid green arrows show our proposed route of discretizing first, then differentiating the constraint, then filtering, and finally reintroducing a pressure term (if the filter is divergence-consistent). This is done to circumvent the pressure problems of the dashed green route (discretizing first, then filtering).

- The momentum equation includes a pressure term. While its gradient can be filtered with Φ, it is unclear what a filtered pressure $\bar{p}$ should be or how it should appear in the filtered momentum equation.

To circumvent these issues, we propose to differentiate the discrete divergence constraint first (to remove the pressure), and then apply the filter to the pressure-free DNS equation (3.7). This results in the sequence "discretize – differentiate constraint – filter", as shown by solid green arrows in fig. 3.3. The advantage over the route defined by dashed green arrows, "discretize – filter", is that we do not need to consider the pressure or the divergence-free constraint, and a single filter for the velocity field is sufficient. The "implied" filtered pressure will be discussed in section 3.4.1. The resulting equation for the filtered DNS-velocity $\bar{u}$ is $\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} = \Phi P F(u)$, which is rewritten as

$$\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} = \bar{P}\bar{F}(\bar{u}) + c(u), \tag{3.11}$$

with the unclosed commutator error defined by

$$c(u) = \Phi P F(u) - \bar{P}\bar{F}(\Phi u). \tag{3.12}$$

A crucial point is that when filtering from a fine grid to a coarse grid, the resulting filtered velocity field is generally *not* divergence-free ($\bar{D}\bar{u} \neq 0$), as discrete filtering and discrete differentiation do not generally commute.

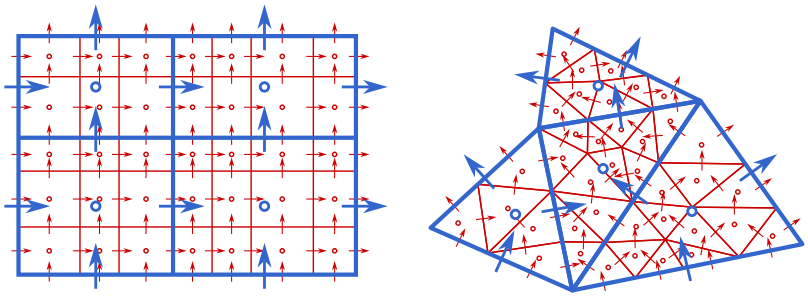


Figure 3.4: Four coarse volumes (blue) and their fine grid sub-grid volumes (red) in 2D. For each of the coarse volume faces, the discrete filter Φ^{FA} combines the DNS velocities u into one LES velocity $\bar{u}$ using averaging. The interior sub-grid velocities are not present in $\bar{u}$. The coarse grid pressure $\bar{p}$ is defined in the coarse volume centers, but is not obtained by filtering p. Instead, it is computed from $\bar{u}$. **Left:** Structured grid, used in this work. **Right:** Unstructured grid.

3.3.3 *Divergence-consistent discrete filter*

Our approach is to design the filter and coarse grid such that the divergence-free constraint is preserved. This is achieved by merging fine DNS volumes to form coarse LES volumes, such that the faces of DNS and LES volumes overlap (as shown in fig. 3.4). The extracted large scale velocities $\bar{u} = \Phi u$ are then obtained by averaging the DNS velocities that are found on the LES volume *faces* [89]. We denote this face-averaging discrete filter by Φ^{FA}. This approach to filtering also naturally generalizes to unstructured grids, as long as the coarse volume faces overlap with the fine ones. An example is shown to the right in fig. 3.4 for triangular volumes.

The face-averaging filter Φ^{FA} differs from more traditional volume-averaging filters such as the volume-averaging top-hat filter [152] (that we denote Φ^{VA}). The face-averaging filter can be thought of as a top-hat filter acting on the dimensions orthogonal to the velocity components only, while the volume-averaging filter is averaging over all dimensions. The associated transfer functions of these two types of filters are further analyzed in section A.4. In this work, we define the two discrete filters Φ^{FA} and Φ^{VA} with uniform weights and with filter width equal to the coarse grid spacing $\bar{\Delta}$. Since both filters are top-hat like, the filter width is defined as the diameter of the averaging domain in the infinity norm. Note that due to the normalization, all discrete velocity components $\bar{u}_I^\alpha$ and u_I^α share the same dimension as the continuous velocity $u^\alpha(x,t)$ (not velocity times area or velocity times volume).

The two discrete filter supports are compared in fig. 3.5. One advantage of the face-averaging filter is that it does not require modifications at non-

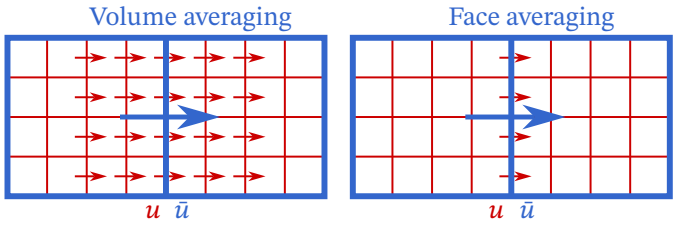


Figure 3.5: DNS velocity components u contributing to a single filtered DNS component $\bar{u}$ for two filters. Both filters have a filter width equal to the grid size. Both $u(t)$ and $\bar{u}(t)$ share the same dimension as the continuous velocity $u(x,t)$. **Left:** Volume-averaging filter Φ^{VA}. **Right:** Face-averaging filter Φ^{FA}.

periodic boundaries, whereas the volume-averaging filter does (for example by using a volume of half the size and twice the weight to avoid averaging outside solid walls). The main advantage, however, lies in the preservation of the divergence-free constraint, as we now show.

The face-averaging filter preserves the divergence-free constraint by construction through a telescoping sum argument on the sub-grid fluxes.

When $\bar{u}$ is obtained through face-averaging, the difference of fluxes entering and leaving an LES volume is equal to a telescoping sum of all sub-grid flux differences, which in turn is zero due to the fine grid divergence-free constraint. The proof is shown in section A.2. Note that this property no longer holds if the filter weights are non-uniform or if the filter width is different from the coarse grid spacing. The face-averaging filter thus preserves *by construction* the divergence-free constraint for the filtered DNS velocity:

$$\text{For } \Phi^{\text{FA}} : \boxed{\forall u,\ Du = 0 \implies \bar{D}\bar{u} = 0.} \tag{3.13}$$

This property is the primary motivation for our filter choice (it can also be thought of as a discrete equivalent of the divergence theorem $\nabla \cdot u = 0 \implies \int_{\partial\mathcal{O}} u \cdot n \, \mathrm{d}\Gamma = 0$ for all $\mathcal{O}$). It can be used to show that the face-averaging filter has the property

$$\bar{D}\Phi P = 0 \tag{3.14}$$

since, for all u, if $w = Pu$, then $Dw = 0$ by definition of P, and thus $\bar{D}\Phi P u = \bar{D}\Phi w = \bar{D}\bar{w} = 0$ from (3.13). In other words, divergence-free fine grid velocity fields stay divergence-free upon filtering. For a general volume-averaging filter, we would not be able to guarantee that all the sub-grid fluxes cancel out, and we would not get a divergence-free constraint for $\bar{u}$. This constraint is often enforced regardless, possibly introducing unforeseen errors, as pointed out by Sirignano et al. [167]. With the face-averaging filter, we do not need to worry about such errors.

With the face-averaging filter choice, the momentum commutator error is divergence-free, since

$$\bar{D}c = \bar{D}\left(\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} - \bar{P}\bar{F}(\bar{u})\right) = \frac{\mathrm{d}(\bar{D}\bar{u})}{\mathrm{d}t} - (\bar{D}\bar{P})\bar{F}(\bar{u}) = 0 \tag{3.15}$$

since $\bar{D}\bar{u} = 0$ and $\bar{D}\bar{P} = 0$ on the coarse grid just like $DP = 0$ on the fine grid. As a result, $c = \bar{P}c$, and the right hand side of the large scale equation is divergence-free. This allows us to rewrite equation (3.11) as

The commutator error c inherits the divergence-free property, enabling the projection $\bar{P}$ to act on c without loss.

$$\boxed{\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} = \bar{P}\left(\bar{F}(\bar{u}) + c(u)\right).} \tag{3.16}$$

The fact that the projection operator $\bar{P}$ now also acts on the commutator error will play an important role in learning a new LES closure model in section 3.4.

Since $Du(0) = 0$ and thus $\bar{D}\bar{u}(0) = 0$ (for the face-averaging filter), we can rewrite the filtered equation into an equivalent constrained form, similar to the unfiltered equations (3.3)-(3.4):

$$\bar{D}\bar{u} = 0, \tag{3.17}$$

$$\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} = \bar{F}(\bar{u}) + c(u) - \bar{G}\bar{p}, \tag{3.18}$$

where $\bar{p}$ is the "implied" pressure defined in the coarse volume centers. It is obtained by solving the pressure Poisson equation with the additional sub-grid forcing term c in the right hand side:

$$\bar{L}\bar{p} = \bar{\Omega}_p\bar{D}\left(\bar{F}(\bar{u}) + c(u)\right). \tag{3.19}$$

Note that the coarse pressure $\bar{p}$ is not obtained by defining a pressure filter, but arises from enforcing the coarse-grid divergence-free constraint. In other words, by filtering the pressure-free momentum equation (3.7) with a divergence-consistent filter, we discover what the (implicitly defined) filtered pressure is. An implicit volume-averaging pressure filter can still be defined, see section A.2 for further details.

3.3.4 *Other divergence-consistent filters*

We comment here on other approaches for constructing divergence-consistent discrete filters.

3.3.4.1 *Discrete differential filters*

Continuous filters can be built using differential operators, for example Germano's filter $\bar{u}(x,t) = (1 - \bar{\Delta}^2/24\nabla^2)^{-1}u$ [8, 60]. Differential filters can

also be extended to the discrete case. Trias and Verstappen propose using polynomials of the discrete diffusion operator (which we will denote D_2) as a filter [183]:

$$\Phi = I + \sum_{i=1}^{m} \gamma_i D_2^i. \tag{3.20}$$

where m is the polynomial degree and γ_i are filter coefficients. This ensures that the filter has useful properties. However, divergence-freeness is only respected approximately, the convective operator may need to be modified to preserve skew-symmetry, and there is no coarsening ($\bar{N} = N$).

3.3.4.2 *Spectral cut-off filters*

Spectral cut-off filters are divergence-consistent, but only so with respect to the spectral divergence operator $\hat{u}_k \mapsto 2\pi i k^\mathsf{T} \hat{u}_k$ (which acts element-wise in spectral space). For pseudo-spectral discretizations, spectral cut-off filters are therefore natural choices. On our staggered grid however, a spectral cut-off filter would not automatically be such that $\bar{D}\bar{u} = 0$.

3.3.4.3 *Projected filters*

By including the projection operator into the filter definition, any filter can be made divergence-consistent [183]. For example, the volume averaging filter Φ^{VA} can be replaced with $\tilde{\Phi}^{\text{VA}} = \bar{P}\Phi^{\text{VA}}$, which is a divergence-consistent filter. However, this makes the filter non-local. The projection step is also more expensive, which could be a problem when the filter is used to generate many filtered DNS training data samples for a neural closure model.

3.4 LEARNING A CLOSURE MODEL FOR THE LARGE SCALE EQUATION

In this section, we present our new closure model formulation: a discrete LES model based on the divergence-consistent face-averaging filter.

3.4.1 *Discrete large eddy simulation*

Our "discretize-differentiate-filter" framework has led to equation (3.11), which describes the exact evolution of the large scale components $\bar{u}$ for a general filter, but still contains the unclosed term $c(u)$ from equation (3.12). Solving this equation would require access to the underlying DNS solution u. We therefore replace $c(u)$ with a parameterized closure model $m(\bar{u}, \theta) \approx c(u)$, which depends on $\bar{u}$ only [17, 144, 152]. This produces a

new *approximate* large scale velocity field $\bar{v} \approx \bar{u}$. It is defined as the solution to the discrete LES model

$$\mathcal{M}_{\mathrm{DIF}}: \quad \boxed{\frac{\mathrm{d}\bar{v}}{\mathrm{d}t} = \bar{P}\bar{F}(\bar{v}) + m(\bar{v}, \theta)}. \tag{3.21}$$

Given the discretize-differentiate-filter framework, this constitutes a general LES model formulation, which does not assume yet that the filter is divergence-consistent. We therefore give it the label DIF (divergence-inconsistent formulation).

Since our face-averaging filter is divergence-consistent, we propose an alternative LES model, by replacing c with m in equation (3.16) instead of in equation (3.11). The result is a new divergence-consistent LES model:

$$\frac{\mathrm{d}\bar{v}}{\mathrm{d}t} = \bar{P}\left(\bar{F}(\bar{v}) + m(\bar{v}, \theta)\right). \tag{3.22}$$

This equation is in "pressure-free" form, which was obtained by differentiating the constraint. By reversing the process, i.e. integrating the constraint in time, the model can now be written back into a constrained form in which the pressure reappears:

$$\mathcal{M}_{\mathrm{DCF}}: \quad \boxed{\begin{aligned} &\bar{D}\bar{v} = 0, \\ &\frac{\mathrm{d}\bar{v}}{\mathrm{d}t} = \bar{F}(\bar{v}) + m(\bar{v}, \theta) - \bar{G}\bar{q}. \end{aligned}} \tag{3.23}$$

Novel DCF formulation: no explicit pressure filter needed; the filtered pressure arises as a Lagrange multiplier.

This is our proposed divergence-consistent formulation that will be denoted by "DCF". The derivation of (3.23) from (3.22) hinges on the fact that $\bar{D}\bar{v}(0) = \bar{D}\bar{u}(0) = 0$. The pressure field $\bar{q} \approx \bar{p}$ is the filtered pressure field ensuring that the LES solution $\bar{v}$ stays divergence-free. We stress that in our approach no pressure filter needs to be defined explicitly. With a divergence-consistent formulation the filtered pressure can be seen as a Lagrange multiplier.

Same form as classical LES, but with a crucial difference: full model-data consistency between training data and deployment environment.

We note that the system (3.23) seems to have the same *form* as the LES equations that are common in literature, being obtained by the classic route "filter first, then discretize" (see red route in fig. 3.3). One might argue that the divergence-consistent face-averaging filter is merely a way to ensure that the operations of differentiation and filtering commute. However, as we mentioned in section 3.1, there is an important difference: in contrast to the classic approach, in our approach we have precisely defined what the filter is, and the training data ($\bar{u}$) is fully discretization-consistent with our learning target ($\bar{v}$). This is a key ingredient in obtaining model-data consistency and hence stable closure models [52, 91].

3.4.2 *Divergence-consistency and LES*

We now compare in more detail the properties of the two models, the "divergence-inconsistent" formulation $\mathcal{M}_{\text{DIF}}$ and the "divergence-consistent" formulation $\mathcal{M}_{\text{DCF}}$, see table 3.1.

$\mathcal{M}_{\text{DIF}}$ is valid for a general (non-divergence-consistent) filter and leads to a non-divergence-free $\bar{v}$. In case a divergence-consistent filter is used, the model $\mathcal{M}_{\text{DIF}}$ is still different from $\mathcal{M}_{\text{DCF}}$, unless $\bar{P}m = m$. This is because one can have an exact commutator error c with the property $\bar{P}c = c$ (meaning $\bar{D}c = 0$) but still learn an approximate commutator error m such that $\bar{P}m \neq m$ (meaning $\bar{D}m \neq 0$).

If the $\mathcal{M}_{\text{DCF}}$ model is used with a volume-averaging filter, inconsistencies appear. For example, while the filtered DNS data is not divergence-free, the DCF model would enforce the LES solution to be (incorrectly) divergence-free.

Model	Filter	$\bar{D}\bar{u} = 0$	$\bar{D}\bar{v} = 0$
$\mathcal{M}_{\text{DIF}}$	VA	False	False
$\mathcal{M}_{\text{DIF}}$	FA	True	True if $\bar{D}m = 0$
$\mathcal{M}_{\text{DCF}}$	VA	False	True
$\mathcal{M}_{\text{DCF}}$	FA	True	True

Table 3.1: Divergence compatibility chart for LES models (rows) and filter properties (columns). The last row shows our preferred combination.

So far, we have only discussed the two LES formulations in terms of their divergence properties, leaving out other properties that may also be important. In section 3.5, we compare the four combinations from table 3.1 for a turbulent flow test case, and discuss whether they are good choices or not.

3.4.3 *Choosing the objective function*

To learn the model parameters θ, we exploit having access to exact filtered DNS data samples $\bar{u}$ and exact commutator errors $c(u)$ obtained through (explicitly) filtered DNS solutions. A straightforward and commonly used approach [12, 86, 116] is then to minimize a loss function of a-priori type, that only depends on the DNS-solution u. We use the commutator error loss

$$L^{\text{prior}}(\mathcal{B}, \theta) = \frac{1}{\#\mathcal{B}} \sum_{u \in \mathcal{B}} \frac{\|m(\bar{u}, \theta) - c(u)\|^2}{\|c(u)\|^2} \tag{3.24}$$

where $\mathcal{B}$ is a batch of $\#\mathcal{B}$ DNS snapshots. Note that (3.24) does not involve $\bar{v}$, so the effect of the closure model on the LES solution is not taken into account. We therefore call this a-priori training [157]. This approach makes training fast, since only gradients of the neural network itself are required for gradient descent.

Alternatively, one can minimize an a-posteriori loss function, that also depends on the LES solution $\bar{v}$. We use the trajectory loss

$$L^{\text{post}}(u_0, \theta) = \frac{1}{n_{\text{unroll}}} \sum_{i=1}^{n_{\text{unroll}}} \frac{\|\bar{v}_i - \bar{u}_i\|^2}{\|\bar{u}_i\|^2}, \tag{3.25}$$

where $\bar{u}_i = \Phi u_i$ is obtained by filtering the DNS solution, $u_{i+1} = \text{RK}_{\Delta t}(u_i)$ is obtained using one RK4 time step from section 3.2.4, u_0 are random initial conditions, and the LES solution $v_{i+1} = \text{RK}_{\Delta t, \mathcal{M}, m, \theta}(v_i)$ is computed using the same time stepping scheme as u_{i+1} but with LES formulation $\mathcal{M}$, closure m, and parameters θ, starting from the exact initial conditions $\bar{v}_0 = \bar{u}_0$.

The parameter n_{unroll} determines how many time steps we unroll. If we choose $n_{\text{unroll}} = 1$, L^{post} will be very similar to L^{prior}. If we choose a large n_{unroll}, we predict long trajectories, and the loss may be more sensitive to small changes in θ ("exploding" gradients). List et al. and Melchers et al. argue that the number of unrolled time steps should depend on the characteristic time scale (Lyapunov time scale) of the problem [107, 121]. For chaotic systems (including turbulent flows), L^{post} is expected to grow fast in time, and the number of unrolled time steps should be small. List et al. found good results with $n_{\text{unroll}} \in [30, 60]$ for the incompressible Navier-Stokes equations in 2D [107]. For the chaotic Kuramoto-Sivashinsky equation in 1D, Melchers et al. found $n_{\text{unroll}} = 30$ to give optimal long term results, with $n_{\text{unroll}} = 120$ performing poorly [121]. Kochkov et al. found 32 steps to be ideal for unrolling [89]. For our test case, we choose $n_{\text{unroll}} = 50$, so that the trajectory becomes long enough in time while keeping the number of steps limited. See section A.1.5 for further details.

The a-priori loss function is easy to evaluate and easy to differentiate with respect to θ, as it does not involve solving the LES ODE given by the model $\mathcal{M} \in \{\mathcal{M}_{\text{DIF}}, \mathcal{M}_{\text{DCF}}\}$. However, minimizing L^{prior} does not take into account the effect of the prediction error on the LES solution error. The a-posteriori loss does take into account this effect, but has a longer computational chain involving the solution of the LES ODE [90, 106, 107, 114, 185]. This is illustrated in fig. 3.6.

3.4.4 *Choosing the model architecture*

Traditionally, closure models are formulated in a continuous setting and replace the unclosed term $\nabla \cdot (\overline{uu} - \bar{u}\bar{u})$ by either structural or functional models [152]. In recent machine learning approaches, discrete data are

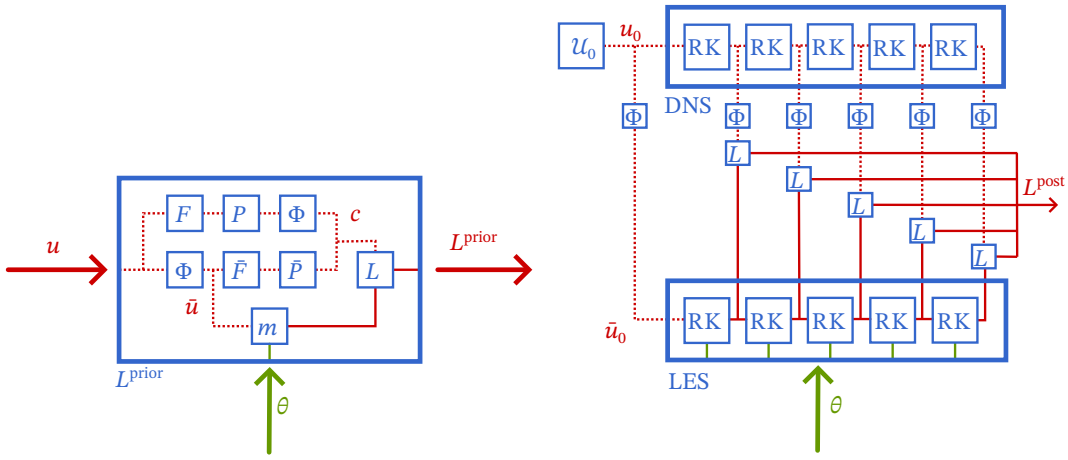


Figure 3.6: Computational chain of loss functions. Solid lines are affected by changes in θ. Dotted lines are not affected by θ, and can be precomputed before training. **Left:** A-priori loss function (3.24). The mean squared error L is computed between the closure model m and the commutator error c. **Right:** A-posteriori loss function (3.25) (here shown for five unrolled time steps). DNS initial conditions are sampled from the distribution $\mathcal{U}_0$, and filtered (Φ) to produce LES initial conditions. After every time step, the LES solution is compared to the corresponding filtered DNS solution using the mean squared error L. The parameters θ are used in each LES RK time step, but not in the DNS time steps.

inherently used for training the closure model, and the loss function can take into account both structural and functional elements [67]. In this work, we use a convolutional neural network for the closure model m [107] (see section 3.4.4.2), and compare it to a traditional eddy-viscosity model (see section 3.4.4.1).

3.4.4.1 *Eddy viscosity models*

Eddy viscosity models are functional models that consist of adding an additional diffusive term

$$m(\bar{u}, \theta) = \nabla \cdot (2\nu_t \bar{S}) \tag{3.26}$$

to the continuously filtered Navier-Stokes equations, where $\bar{S} = \frac{1}{2}\left(\nabla \bar{u} + \nabla \bar{u}^{\mathsf{T}}\right)$ is the large scale strain rate tensor and ν_t is a turbulent viscosity (parameterized by θ). This term models transfer of energy from large to unresolved scales. Note that $\bar{u}(x,t)$ and $m(\bar{u}(\cdot,t),\theta)(x)$ are here continuous quantities that subsequently need to be discretized to $\bar{u}(t)$ and $m(\bar{u}(t),\theta)$.

The Smagorinsky model [101, 168] predicts a local viscosity of the form

$$\nu_t = \theta^2 \bar{\Delta}^2 \sqrt{2\,\mathrm{tr}(\bar{S}\bar{S})}, \tag{3.27}$$

where $\bar{\Delta}$ is the filter width and $\theta \in [0,1]$ is the only model parameter (the Smagorinsky coefficient). In our experiments, this parameter is fitted to filtered DNS data, similar to [67, 162]. The model is discretized on the coarse grid, and $\bar{\Delta}$ is taken to be the LES grid size.

3.4.4.2 *Convolutional neural networks (CNNs)*

Convolutional neural networks (CNNs) are commonly used in closure models when dealing with structured data [12, 67, 107, 162], which is a natural choice given our structured Cartesian grid. A convolutional node $\mathrm{conv} : (u^n)_{n=1}^{N_{\mathrm{chan}}} \mapsto v$ in a CNN transforms N_{chan} discrete input channel functions into one discrete output function in the following non-linear way:

$$v_I = \sigma\left(b + \sum_{\substack{J \in \{-r,\dots,r\}^d \\ n \in \{1,\dots,N_{\mathrm{chan}}\}}} K_J^n u_{I+J}^n\right), \tag{3.28}$$

where σ is a non-linear activation function, $b \in \mathbb{R}$ is a bias, $K \in \mathbb{R}^{(2r+1)^d \times N_{\mathrm{chan}}}$ is the kernel, and r is the kernel radius.

In a convolutional node, it is typically assumed that all input channels are fields defined at the same grid points. Our closure model, however, is defined on a staggered grid, with inputs and outputs at the velocity points,

whose locations differ across the Cartesian directions, see fig. 3.2. Our CNN closure model is therefore defined as follows:

$$m_{\text{CNN}} = \text{decollocate} \circ \begin{pmatrix} \text{conv} \\ \vdots \\ \text{conv} \end{pmatrix} \circ \ldots \circ \begin{pmatrix} \text{conv} \\ \vdots \\ \text{conv} \end{pmatrix} \circ \text{collocate} \tag{3.29}$$

where the full vector of degrees of freedom $\bar{u}$ contains d sub-vectors $\bar{u}^\alpha$ used as input channels to the CNN. The degrees of freedom in these sub-vectors belong to their own canonical velocity points. We therefore introduce a so-called collocation function as an initialization layer to the CNN in order to produce quantities that are all defined in the pressure points. The subsequent inner layers map from pressure points to pressure points using kernels of odd diameters. Since the closure term is required in the velocity points, a "decollocation" function is introduced in the last layer to map back from pressure points to velocity points. Here, we use a linear interpolation for both collocation and decollocation functions. It is also possible to use "divergence of a stress tensor" as a decollocation function in order to mimic the structure of the continuous commutator error. However, our commutator error also includes discretization effects, where this form may not be relevant. It would also require more (de)collocation functions to produce the off-diagonal elements of the tensors (which should be the volume corners in 2D and volume edges in 3D).

3.5 NUMERICAL EXPERIMENT: FORCED TURBULENCE IN A PERIODIC BOX

The code is available at https://github.com/agdestein/DivergenceConsistency. All simulations were run on a single Nvidia H100 GPU on the Dutch national supercomputer Snellius.

We consider a unit square domain $\Omega = [0, 1]^d$ with periodic boundaries. The DNS initial conditions are sampled from a random velocity field defined through its prescribed energy spectrum $\hat{E}_k$. Similar to [117, 135, 153], we create an initial energy profile by multiplying a growing polynomial with a decaying exponential as

$$\hat{E}_k = \frac{8\pi}{3\kappa_p^5} \kappa^4 \mathrm{e}^{-2\pi\left(\frac{\kappa}{\kappa_p}\right)^2}, \tag{3.30}$$

where $k \in \mathbb{Z}^d$ is the wavenumber, $\kappa = \|k\|$, and κ_p is the peak wavenumber. The profile should grow for $\kappa < \kappa_p$, and the decay should take over for $\kappa > \kappa_p$. To further distinguish between the different setups and prevent energy decay during long simulations, we add a Kolmogorov-type body force to inject energy into the system, as in [33, 89, 106]. It is defined as a sinusoidal force at wavenumber $\kappa = 4$ as

$$f^\alpha(x^1, \ldots, x^d) = \delta_{\alpha=1} \sin(8\pi x^2). \tag{3.31}$$

For more details about the initialization procedure, see section A.1.2.

We perform all simulations in our open source package *IncompressibleNavierStokes.jl*, implemented in the Julia programming language [20]. We use the *KernelAbstractions.jl* [39] framework for implementing back-end agnostic differential operators, *Lux.jl* [137] for neural networks components, *Zygote.jl* [78] for reverse mode automatic differentiation, and *Makie.jl* [47] for visualization. All array operations for DNS, LES, and training are performed on a CUDA-compatible GPU, using *CUDA.jl* [18, 19].

3.5.1 *Filtered DNS (2D and 3D)*

Before showing results of our new divergence-consistent LES models, we perform a-priori tests to investigate some characteristics of the DNS and filtered DNS solutions. Note that "a-priori" here is used in relation to the analysis of the results (namely before the LES model is employed), while "a-priori" in section 3.4.3 was related to the training procedure.

We generate two DNS trajectories $u(t)$ (one in 2D, one in 3D), starting from the initial conditions defined above. The 2D simulation is performed with resolution $N = (4096, 4096)$, and the 3D simulation with $N = (1024, 1024, 1024)$. For both simulations, we set $\kappa_p = 20$ and solve until $t_{\text{end}} = 1$ using adaptive time stepping. The Reynolds number is $\text{Re} = 10^4$ in 2D and $\text{Re} = 6000$ in 3D. All array operations are performed on the GPU with double precision floating point numbers in 2D (to demonstrate divergence-freeness), which works fine for this study even though GPUs are not optimized for double precision, and single precision in 3D (to fit all arrays in the memory of a single H100 GPU).

For the filter, we consider the face-averaging filter Φ^{FA} and the volume-averaging filter Φ^{VA}. For the 2D setup, we use $\bar{N} = (\bar{n}, \bar{n})$, and for the 3D setup, we use $\bar{N} = (\bar{n}, \bar{n}, \bar{n})$, with $\bar{n} \in \{32, 64, 128, 256\}$.

3.5.1.1 *Energy spectra*

Figure 3.7 shows the kinetic energy spectra at the final time for the 2D and 3D simulations. The initial velocity field is smooth (containing only low wavenumbers), while the final DNS fields also contain higher wavenumbers. The theoretical slopes of the inertial regions of κ^{-3} in 2D and $\kappa^{-5/3}$ in 3D [144] are also shown (see section A.1.1). The inertial region is clearly visible in 2D, but less so in 3D since the DNS-resolution is smaller. The effect of diffusion is visible for $\kappa > 128$ in the 2D plot and for $\kappa > 32$ in the 3D plot, with an attenuation of the kinetic energy. The energy injection wavenumber is visible as a spike in the 3D plot, but not in the 2D plot. The filtered DNS spectra are also shown. A grid of size $(n, \dots, n)$ can only fully resolve wavenumbers in the range $0 \le \kappa \le n/2 - 1$, which is visible in the sudden stops of the filtered spectra at $\bar{n}/2 - 1$. The face-averaging filter and

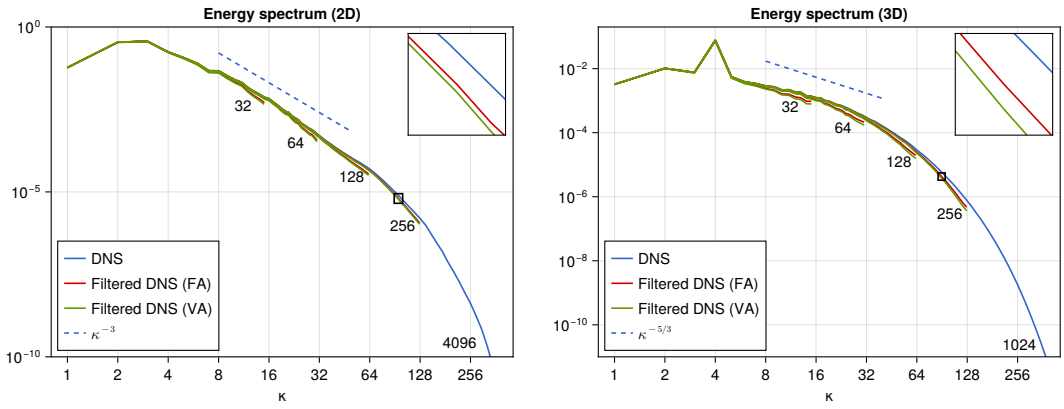


Figure 3.7: Kinetic energy spectra of DNS and filtered DNS at final time. Both filters are applied for four different filter sizes $\bar{n}$, visible in the four sudden stops at the cut-off wavenumbers $\bar{n}/2$. The numbers below the lines indicate the grid sizes $\bar{n}$ and n. **Left:** 2D simulation, with theoretical scaling κ^{-3}. **Right:** 3D simulation, with theoretical scaling $\kappa^{-5/3}$.

the volume-averaging filter have very similar energy profiles. Note that the filtered energy is slightly dampened even before the filter cut-off wavelengths. Since both Φ^{FA} and Φ^{VA} are top-hat like, their transfer functions do not perform sharp cut-offs in spectral space, but affect all wavenumbers [17, 144]. The face-averaging filter is damping slightly less than the volume-averaging filter. This is because it averages over one less dimension than the volume-averaging filter, leaving the dimension normal to the face intact. Still, the coarse-graining of the discrete filters creates a spectral cut-off effect that hides the damping of the top-hat transfer functions. This is because the filter width is very close to the coarse-graining spectral cut-off filter width. For further details about the transfer functions, see section A.4.

3.5.1.2 *Filtered fields*

Figure 3.8 shows the discrete curl $\nabla \times \varphi$ of various 2D fields φ ($u(0)$, u, $\bar{u}$, $PF(u)$, $\bar{P}\bar{F}(\bar{u})$, $c(u)$) for the face-averaging filter with $\bar{N} = 128^2$. We plot the curl since φ is a vector field. Each pixel corresponds to a pressure volume, in which the curl is interpolated for visualization. The filtered field $\bar{u}$ in the top-right corner is clearly unable to represent all the sub-grid fluctuations seen in the DNS field u, but the large eddies of u are still recognizable in $\bar{u}$. We stress again that the aim of our neural closure models is to reproduce $\bar{u}$, without having knowledge of the DNS field u. This will be shown in section 3.5.2.

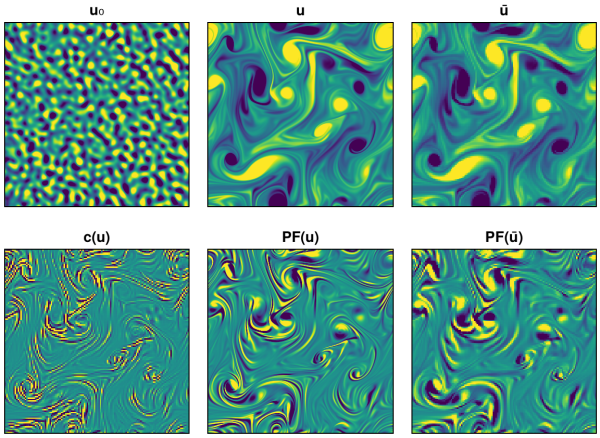


Figure 3.8: A-priori results: Discrete curl $-\delta_2\varphi^1 + \delta_1\varphi^2$ of various 2D fields $\varphi \in \{u(0), u, \bar{u}, PF(u), \bar{P}\bar{F}(\bar{u}), c(u)\}$. The filter is face-averaging, $\bar{N} = 128^2$.

The discretize-first commutator error includes coarse-grid discretization effects, allowing the CNN to learn to correct for numerical oscillations.

Note in particular that the coarse grid right hand side $\bar{P}\bar{F}(\bar{u})$ contains small oscillations which make the discrete curl look grainy. These are due to the under-resolved central-difference discretization on the coarse grid, and are subsequently also present in the discrete commutator error c. The closure model m thus has to predict the oscillations in c in order to correct for those in $\bar{P}\bar{F}(\bar{u})$, using information from the smooth field $\bar{u}$ only. If c was defined by filtering first and then discretizing, as is commonly done in LES, these oscillations would not be part of c and other means of stabilization would be needed to correct for the oscillations in $\bar{P}\bar{F}(\bar{u})$, such as explicit LES [15, 58, 111] (where the filter is applied to the non-linear convective term in the LES right hand side). This problem has also been addressed in literature; for example, Geurts et al. [62] and Beck and Kurz [14] argue that all commutator errors should be modeled, and Stoffer et al. [176] show that instabilities in the high wavenumbers can occur if the discretization is not included in the commutator error.

Figure 3.9 shows the vortex cores of the 3D simulation at initial and final time. The vortex cores are visualized as isocontours of λ_2-criterion [82]. It is defined as negative regions of $\lambda_2(S^2 + T^2)$, where λ_2 denotes the second largest eigenvalue in absolute value of the 3×3-tensor, and $S = \frac{1}{2}(\nabla u + \nabla u^\mathsf{T})$ and $T = \frac{1}{2}(\nabla u - \nabla u^\mathsf{T})$ are the symmetric and anti-symmetric parts of the velocity gradient tensor. With the prescribed initial low wavenumber energy spectrum, only large DNS vortex structures are present at the initial time. They are clearly visible in the left plot. As energy gets transferred to higher wavenumbers, smaller turbulent vortex structures are formed (middle plot).

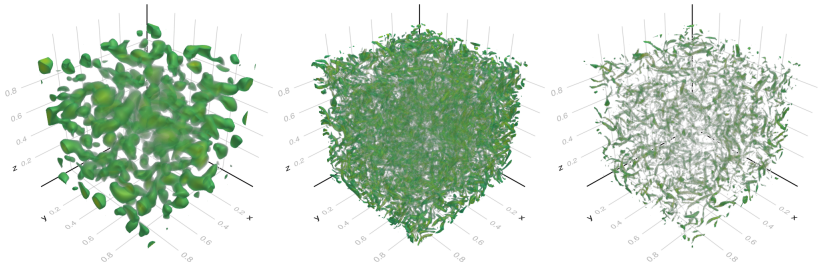


Figure 3.9: Vortex cores visualized as 10 isocontours of negative regions of $\lambda_2(S^2+T^2)$, where λ_2 denotes the second largest eigenvalue, and $S = \frac{1}{2}(\nabla u + \nabla u^\mathsf{T})$ and $T = \frac{1}{2}(\nabla u - \nabla u^\mathsf{T})$ are the symmetric and anti-symmetric parts of the velocity gradient. **Left:** DNS, initial time. **Middle:** DNS, final time. **Right:** Filtered DNS (face-averaging, $\bar{N} = 128^3$), final time.

The same DNS field still contains larger vortex structures, which become visible after filtering (right plot).

3.5.1.3 *Divergence, commutator errors, and kinetic energy*

Table 3.2 shows the magnitude of various quantities derived from the two DNS trajectories. All quantities $q(u(t))$ are averaged over time with a frequency of $s = 20$ time steps as follows:

$$\langle q \rangle_s = \frac{1}{n_t/s + 1} \sum_{i=0}^{n_t/s} q(u(t_{si})). \tag{3.32}$$

The considered quantities $q(u)$ are: Normalized divergence $\|\bar{D}\bar{u}\|/\|\bar{u}\|$, magnitude of non-divergence-free part of filtered velocity field $\|\bar{u} - \bar{P}\bar{u}\|/\|\bar{u}\|$, magnitude of non-divergence-free part of commutator error $\|c - \bar{P}c\|/\|c\|$, magnitude of commutator error in the total filtered right hand side $\|c\|/\|\bar{P}\bar{F} + \bar{c}\|$, and resolved kinetic energy $\|\bar{u}\|^2_{\bar{\Omega}}/\|u\|^2_{\Omega}$. The norms are defined as $\|u\| = \sqrt{\sum_{\alpha=1}^{d} \|u^\alpha\|^2}$ for vector fields such as u and $\|u^\alpha\| = \sqrt{\sum_I |u^\alpha_I|^2}$ for scalar fields such as u^α. Additionally, the norm $\|\cdot\|_\Omega$ is weighted by the volume sizes.

It is clear that both $\bar{u}$ and $c(u)$ are divergence-free for the face-averaging filter, in both 2D and 3D. For the volume-averaging filter on the other hand, $\bar{u}$ and $c(u)$ are not divergence-free. At $\bar{N} = 32^2$, the orthogonal projected part $\bar{u} - \bar{P}\bar{u}$ comprises about 1.7% of $\bar{u}$. This is more visible for the commutator error, for which the orthogonal part is 11% of the total commutator error. When the grid is refined to $\bar{N} = 256^2$ however, the non-divergence-free parts of $\bar{u}$ shrink to 0.059% since the flow is more resolved. However, the non-divergence-free part of the commutator error is 12%, since the commutator error itself is smaller. For volume-averaging with $\bar{N} = 32^3$, the non-

N	Bits	Filter	$\bar{N}$	$\frac{\|\bar{D}\bar{u}\|}{\|\bar{u}\|}$	$\frac{\|\bar{u}-\bar{P}\bar{u}\|}{\|\bar{u}\|}$	$\frac{\|c-\bar{P}c\|}{\|c\|}$	$\frac{\|c\|}{\|\bar{P}\bar{F}+c\|}$	$\frac{\|\bar{u}\|_\Omega^2}{\|u\|_\Omega^2}$
4096^2	64	FA	32^2	1.5×10^{-14}	2.5×10^{-16}	2.3×10^{-13}	0.56	0.92
		FA	64^2	2.1×10^{-14}	1.9×10^{-16}	3.4×10^{-13}	0.35	0.98
		FA	128^2	3.4×10^{-14}	1.6×10^{-16}	6.1×10^{-13}	0.18	0.99
		FA	256^2	5.3×10^{-14}	1.3×10^{-16}	1.3×10^{-12}	0.077	1.0
		VA	32^2	1.1	0.017	0.11	0.54	0.89
		VA	64^2	0.67	0.0058	0.096	0.33	0.97
		VA	128^2	0.39	0.0019	0.088	0.18	0.99
		VA	256^2	0.19	0.00059	0.12	0.08	1.0
1024^3	32	FA	32^3	2.2×10^{-5}	2.8×10^{-7}	3.9×10^{-6}	0.85	0.63
		FA	64^3	2.2×10^{-5}	1.4×10^{-7}	3.5×10^{-6}	0.70	0.80
		FA	128^3	2.2×10^{-5}	7.6×10^{-8}	4.0×10^{-6}	0.51	0.91
		FA	256^3	2.4×10^{-5}	4.5×10^{-8}	7.6×10^{-6}	0.31	0.96
		VA	32^3	3.3	0.043	0.19	0.80	0.60
		VA	64^3	4.2	0.028	0.15	0.66	0.77
		VA	128^3	4.8	0.017	0.13	0.49	0.89
		VA	256^3	5.0	0.0094	0.13	0.30	0.95

Table 3.2: Magnitude of various quantities derived from two DNS trajectories $u(t)$ (one in 2D, one in 3D). All quantities are averaged over time. The machine precision for 64-bit numbers is $\epsilon \approx 2.22\times10^{-16}$, and for 32-bit numbers $\epsilon \approx 1.19\times10^{-7}$.

divergence-free parts of $\bar{u}$ and $c(u)$ are 4.3% and 19%, respectively, shrinking to 0.94% and 13% for $\bar{N} = 256^3$. Note that the volume-averaging filter width was chosen to be equal to the grid spacing, but these divergence errors would be larger if we increased the filter width.

For both 2D and 3D, and both filter types, the commutator error becomes smaller when the grid is refined, which is expected since more scales are resolved. The commutator error magnitude does not seem to depend much on whether we use face-averaging or volume-averaging, since they both have the same characteristic filter width. For face-averaging with $\bar{N} = 32^2$ and $\bar{N} = 128^3$, more than half of the total right-hand side is due to the commutator error, even though 92% and 91% of the kinetic energy is resolved by $\bar{u}$. For these resolutions, it is very important to have a good closure model. For $\bar{N} = 256^2$, the filtered DNS right hand side is much closer to the corresponding coarse unfiltered DNS right hand side, with $c(u)$ comprising 7.7% of the total right-hand side $\bar{P}\bar{F}(\bar{u}) + c(u)$. A closure model is still clearly needed, even though 100% of the energy is resolved by $\bar{u}$. Bae et al. [7] also found the discretization part of the commutator errors to be quite significant, in particular near walls (which we do not consider in this study).

In summary, the DNS and filtered DNS results confirm the theoretical analysis in section 3.3.3. The volume-averaging filter lacks divergence-consistency of the solution and the closure term. The magnitude of the closure term is similar for both filters. The benefits of divergence-consistency will be demonstrated in the a-posteriori analysis in the subsequent section.

3.5.2 *LES (2D)*

Next, we turn to the results for the key challenge set out in this chapter: testing our neural closure models in an LES setting, with divergence-consistent filters, aiming to approximate the trajectory $\bar{u}(t)$ given $\bar{u}(0)$. We now only consider the 2D problem to reduce the computational time, since the same network is trained repeatedly in multiple configurations. For 3D results, see section A.6.2. The DNS resolution is set to $N = (4096, 4096)$. The Reynolds number is $\mathrm{Re} = 6000$. The initial peak wavenumber is $\kappa_p = 20$. We use single precision floating point numbers for all computations, including DNS trajectory generation, to reduce memory usage and increase speed. Details about the datasets are found in section A.1.4.

For the closure model m, we consider three options:

1. No closure model, $m_0 = 0$. This is the baseline model.
2. Smagorinsky closure model, m_S.
3. Convolutional neural closure model, m_CNN. The architecture is shown in section A.1.3.

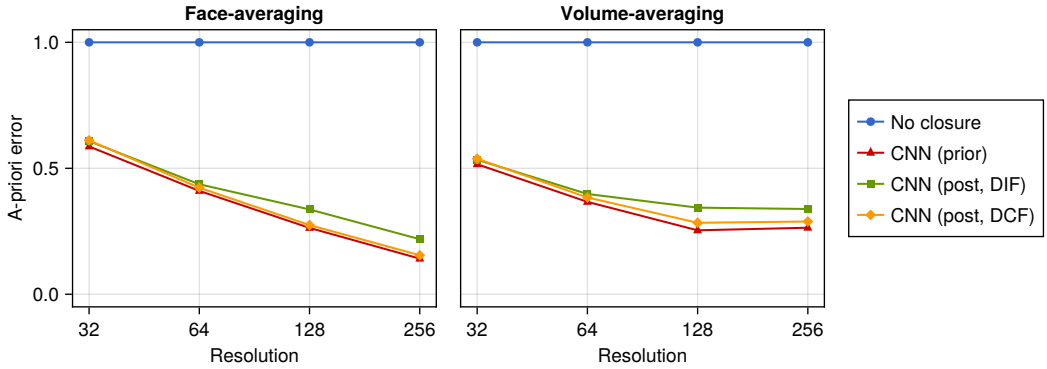


Figure 3.10: Relative a-priori errors $\frac{1}{n_u}\sum_u \frac{\|m-c\|}{\|c\|}$ for the testing dataset. **Left:** Face-averaging filter. **Right:** Volume-averaging filter.

The no-closure model can be thought of as "coarse DNS", and it is included to show the necessity of a closure model when compared to the filtered DNS reference data. The Smagorinsky model is a traditional closure model. While it is conventionally used in combination with volume-averaging filters, it can also be used with a face-averaging filter. The idea behind the Smagorinsky eddy viscosity $\nu_t = (\theta\bar{\Delta})^2\sqrt{2\,\mathrm{tr}(\bar{S}\bar{S})}$ is that it is equal to the large-scale strain weighted by the "average sub-filter eddy size" $\theta\bar{\Delta}$, which is parameterized by the fractional constant $\theta \in [0,1]$. The largest unresolved eddy then has the size $\bar{\Delta}$. This argument also holds for the face-averaging filter, where the largest unresolved eddy has a characteristic size $\bar{\Delta}$, and the averaged unresolved eddy has a size $\theta\bar{\Delta}$ for some coefficient θ (potentially with a different value than for volume-averaging). See section A.1.5 for details about the training.

3.5.2.1 *A-priori errors*

Figure 3.10 shows the average relative a-priori errors $\frac{1}{n_u}\sum_u \frac{\|m-c\|}{\|c\|}$ for the testing dataset and the three CNN parameters θ^{prior}, $\theta^{\text{post}}_{\text{DIF}}$, and $\theta^{\text{post}}_{\text{DCF}}$. For the no-closure model m_0, the a-priori error is always $\|0-c\|/\|c\| = 1$. For both FA and VA and for all grid sizes, the a-priori trained parameters θ^{prior} perform the best. This is expected, since the a-priori loss function used to obtain θ^{prior} is similar to the a-priori error.

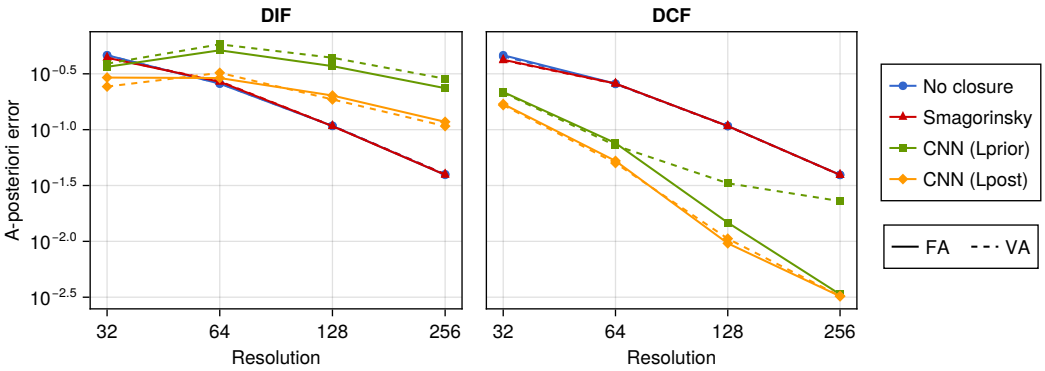


Figure 3.11: Relative a-posteriori errors $\frac{1}{n_t}\sum_{i=1}^{n_t} \|\bar{v}_i - \bar{u}_i\|/\|\bar{u}_i\|$ at time $t = 0.27$ for the testing dataset. The CNN is trained separately for each resolution and each filter type with L^{prior} (green squares) and L^{prior}-then-L^{post} (yellow diamonds). **Solid lines:** Face-averaging (Φ^{FA}). **Dashed lines:** Volume-averaging (Φ^{VA}). **Left:** General model $\mathcal{M}_{\text{DIF}}$. **Right:** Divergence-consistent model $\mathcal{M}_{\text{DCF}}$.

3.5.2.2 *A-posteriori errors*

The relative a-posteriori error $\frac{1}{n_t}\sum_{i=1}^{n_t} \|\bar{v}_i - \bar{u}_i\|/\|\bar{u}_i\|$ is computed for the trajectory in the testing dataset. A closure model parameter set θ is only used for testing on the same coarse grid and same filter type that it was trained for. Since turbulent flow is a chaotic system, it does not make sense to plot the point-wise error over long time periods. We therefore report errors at time $t = 0.27$; for later times, one needs to consider statistical quantities. Figure 3.11 shows the average error over time.

Main result: DCF with face-averaging achieves high accuracy with a-priori training alone — no a-posteriori fine-tuning needed.

For $\mathcal{M}_{\text{DCF}}$ (right), the no-closure and the Smagorinsky closure have similar errors. The CNN is clearly outperforming the other two closures. The a-priori trained CNN is performing significantly better for face-averaging than for volume-averaging. Since the commutator errors are fully consistent with $\mathcal{M}_{\text{DCF}}$ in the face-averaging case, high accuracy can be achieved with a-priori training alone. In the volume-averaging case, the commutator errors are inconsistent with $\mathcal{M}_{\text{DCF}}$, leading to higher errors. When switching to the a-posteriori loss function however, the volume-averaging CNN drastically improves and catches up with the face-averaging one, and the errors become indistinguishable for the two filters. In the face-averaging case, the a-priori trained CNN is already performing well, and training with L^{post} only leads to minor improvements. Note that the a-posteriori training is done starting from the best performing a-priori trained parameters.

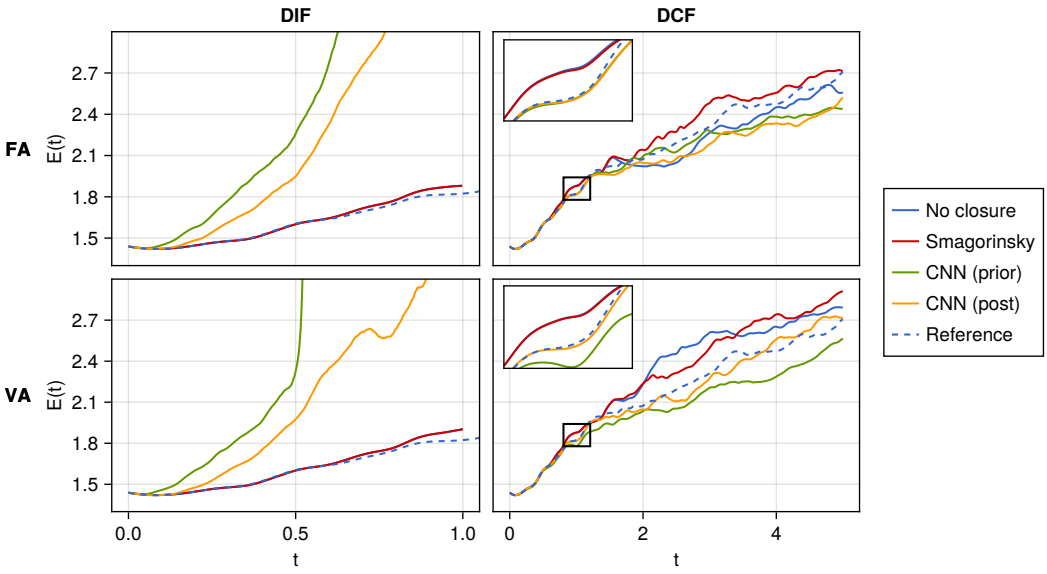


Figure 3.12: Total kinetic energy evolution for $\bar{n} = 256$. **Left:** Unprojected closure model $\mathcal{M}_{\text{DIF}}$. **Right:** Constrained model $\mathcal{M}_{\text{DCF}}$. **Top:** Face-averaging filter. **Bottom:** Volume-averaging filter.

For $\mathcal{M}_{\text{DIF}}$ (left), the no-closure and the Smagorinsky closure have similar profiles as for $\mathcal{M}_{\text{DCF}}$. For the no-closure, the two LES formulations are actually identical. For all resolutions except for $\bar{n} = 32$, $m_{\text{CNN}}(\cdot, \theta^{\text{prior}})$ is showing signs of instability and is performing worse than m_0. The a-posteriori training does manage to detect and reduce this error, but it is still unstable and higher than m_0.

3.5.2.3 *Stability*

To further investigate the stability, we compute the evolution of the total kinetic energy as a function of time. This is shown in fig. 3.12. The model DIF becomes unstable in the long term, so we only compute the energy until the time $t = 1$. The no-closure solution and Smagorinsky solution both stay close to the target energy during the first time unit, after which they become completely decorrelated from the reference solution due to the chaotic nature of turbulence. However, their energy profile does follow the same trend as the reference, staying slightly above. For $\mathcal{M}_{\text{DCF}}$, the a-posteriori trained CNN stays on the reference energy level for much longer than the two other models. As in fig. 3.11, $m_{\text{CNN}}(\cdot, \theta^{\text{prior}})$ and $m_{\text{CNN}}(\cdot, \theta^{\text{post}})$ are very similar in the face-averaging case (as can be seen in the zoom-in window), until they reach the point of decorrelation. In the volume-averaging

case, however, $m_{\text{CNN}}(\cdot, \theta^{\text{prior}})$ performs more poorly than $m_{\text{CNN}}(\cdot, \theta^{\text{post}})$. For $\mathcal{M}_{\text{DIF}}$, the high CNN errors from fig. 3.11 are confirmed by the rapid growth of the total kinetic energy. Similar growth in energy of unconstrained neural closure models after a period of seemingly good overlap with the reference energy has been observed by Beck and Kurz [12, 91], although in a different configuration. Training with L^{post} improves the stability, and the energy stays close to the reference for a longer period. This was not sufficient to stabilize $\mathcal{M}_{\text{DIF}}$, however, and the energy eventually begins increasing.

DIF instability explained: without divergence-free solutions, the skew-symmetric convective discretization loses its energy-conservation guarantee.

The growth in kinetic energy and resulting lack of stability for the divergence-inconsistent model can be explained by the energy-conservation properties of our spatial discretization. The convective terms are discretized with a second order central scheme (in so-called divergence form), which can be shown to be equivalent to a skew-symmetric, energy-conserving form *provided that the velocity field is divergence-free* [191]. If the velocity field is not divergence-free, there is no guarantee that the convective terms are energy-conserving, which can lead to growth in kinetic energy and loss of stability.

It is worth noting that during the first time unit for $\mathcal{M}_{\text{DCF}}$, the CNN models manage to stay close to the target energy level without being explicitly trained to do so. It is also possible to add an energy mismatch term in the loss function, thus encouraging the CNN to produce a correct energy level [107]. Such training could potentially be used to further improve the stability of $\mathcal{M}_{\text{DIF}}$.

3.5.2.4 *Energy spectra*

The energy spectra at different times are shown in fig. 3.13. The no-closure model spectrum is generally close to the reference spectrum, but contains too much energy in the high wavenumbers. This is likely due to the oscillations discussed in the previous sections. The Smagorinsky closure is correcting for this (as intended). For $\mathcal{M}_{\text{DCF}}$, the a-posteriori trained CNN spectrum is visually very close to the reference spectrum, also at the final time, long after the LES trajectory has diverged from the filtered DNS trajectory. This shows that the trained model can capture turbulence statistics correctly even though the system is chaotic. For $\mathcal{M}_{\text{DIF}}$, all the CNNs produce too much energy in the high wave numbers. Training with L^{post} seems to improve this somewhat.

3.5.2.5 *LES solution fields*

The curl of the LES solutions (LES vorticity) at the final time is shown in fig. 3.14. The reference solution $\bar{u}$ has a smooth vorticity field for all resolutions, since the low-pass filtering operation removes high wavenumber components. The no-closure model solution, on the other hand, shows sharp oscillations. Only the initial conditions are smooth, since $\bar{v}(0) = \bar{u}(0)$. As the

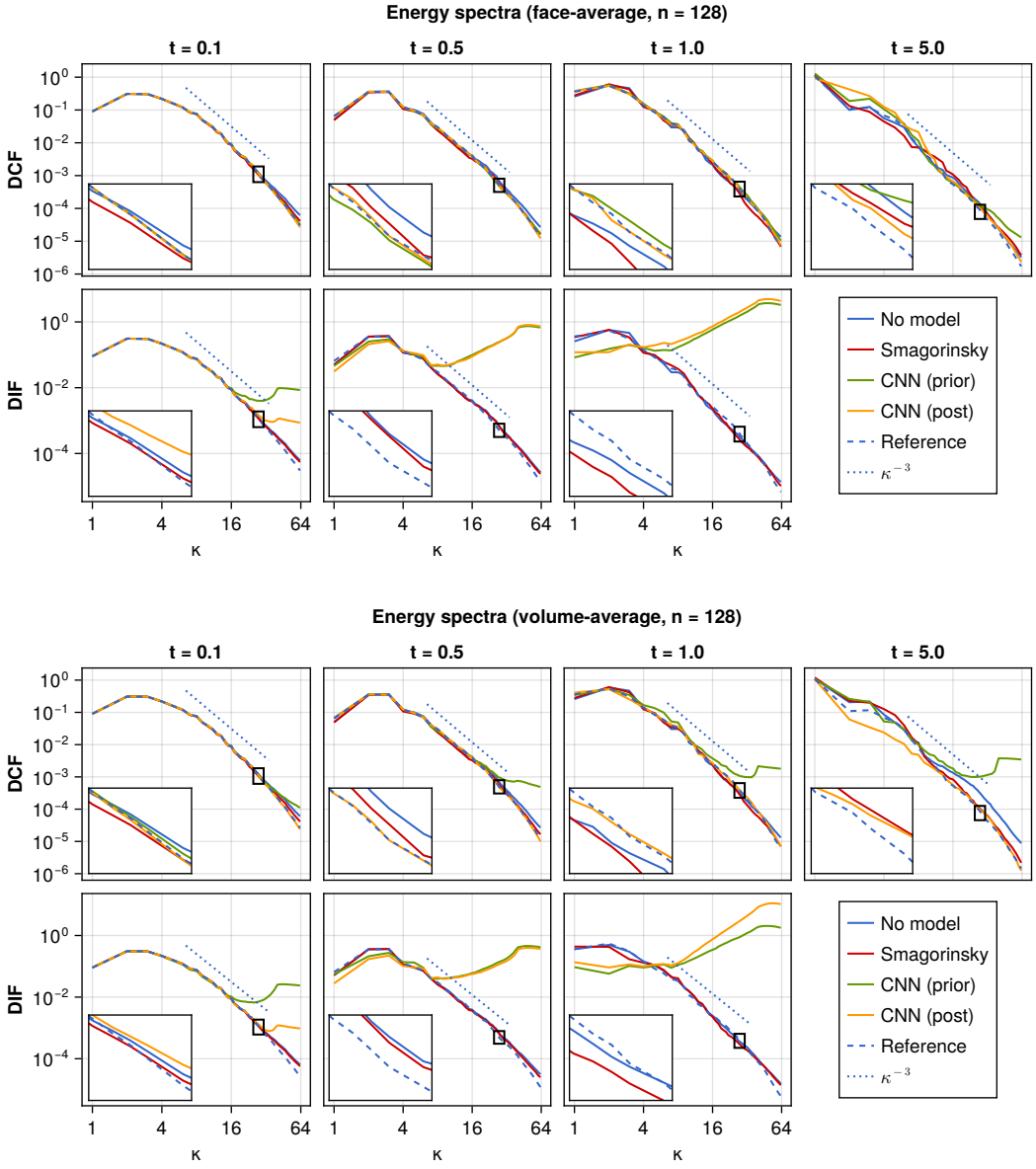


Figure 3.13: Energy spectra at final time for $\bar{n} = 128$. **Top:** Face-averaging filter. **Bottom:** Volume-averaging filter. Since the $\mathcal{M}_{\text{DIF}}$ simulations become unstable, we do not show their spectrum at time $t = 5$.

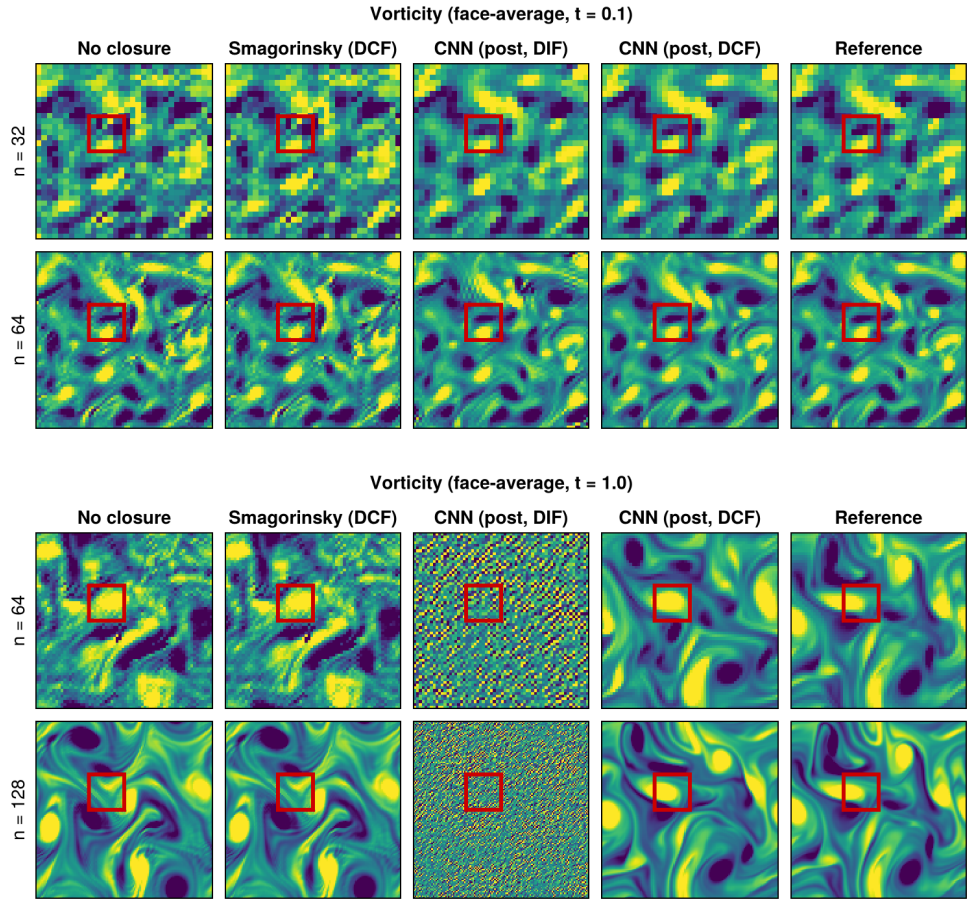


Figure 3.14: Vorticity of filtered DNS solution $\bar{u}$ (right) and LES solutions $\bar{v}$ computed using no closure (first column), Smagorinsky model (second column), $\mathcal{M}_{\text{DIF}}$ (third column), and $\mathcal{M}_{\text{DCF}}$ (fourth column) at different times.

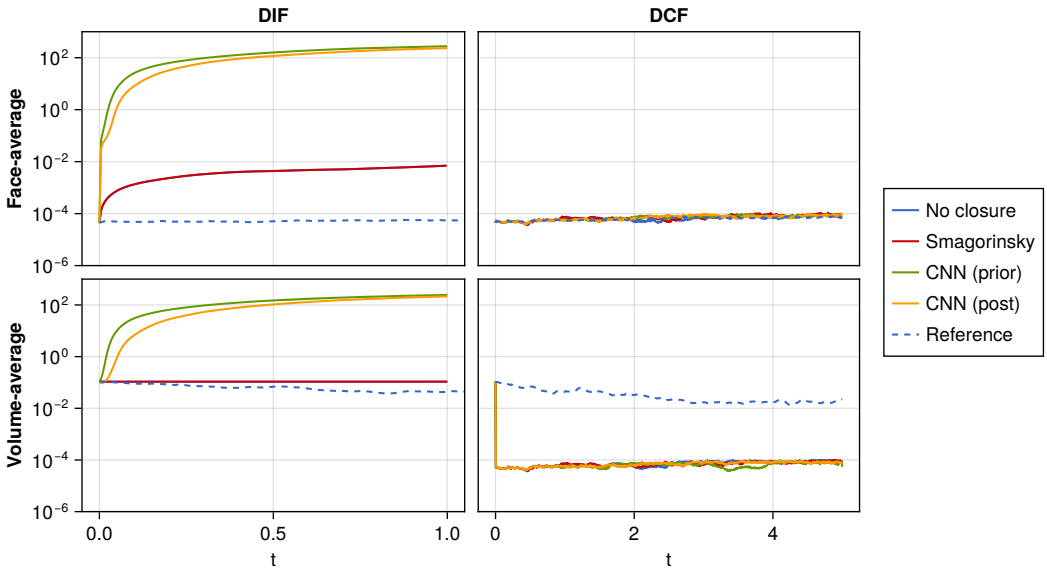


Figure 3.15: Divergence evolution for $\bar{n} = 256$. **Left:** Unprojected closure model $\mathcal{M}_{\text{DIF}}$. **Right:** Constrained model $\mathcal{M}_{\text{DCF}}$. **Top:** Face-averaging filter. **Bottom:** Volume-averaging filter.

solution evolves in time, the central difference scheme used in the right hand side $\bar{P}\bar{F}$ is too coarse for the given Reynolds numbers, and produces well-known oscillations [163]. At higher resolutions, these oscillations diminish, and $\bar{v}$ begins to visually resemble $\bar{u}$.

Adding an $\mathcal{M}_{\text{DCF}}$ constrained CNN closure term trained using $L^{\text{post}}_{\text{DCF}}$ seems to correct for the oscillations of the no-closure model, and we recover the smooth fields with recognizable features from the reference solution (compare $\bar{u}$ and $\bar{v}$ inside the red square). We are effectively doing *large eddy* simulation, as large eddies are visually found in the right positions after simulation. However, if we remove the projection and use the $\mathcal{M}_{\text{DIF}}$ CNN closure model, the solution becomes unstable, even after accounting for this by training with $L^{\text{post}}_{\text{DIF}}$. This confirms the observations from fig. 3.11.

We stress again that the CNN closure model is accounting for the total commutator error resulting from the discrete filtering procedure. This commutator error includes the coarse grid discretization error, and also the oscillations produced by the central difference scheme.

$\bar{u}$	Filter	$\bar{n}$	L^{prior}	$L^{\text{post}}_{\text{DIF}}$	$L^{\text{post}}_{\text{DCF}}$
1.5739×10^4	FA	32	160.6	4460.6	4449.3
	FA	64	244.0	4509.1	4584.7
	FA	128	497.4	4901.2	4853.9
	FA	256	1562.3	5822.8	5803.7
	VA	32	157.8	4490.8	4569.4
	VA	64	243.9	4498.5	4608.6
	VA	128	513.2	4914.2	4873.3
	VA	256	1583.6	5793.1	5849.2

Table 3.3: Computational time (in seconds) for DNS and filtering of training, validation, and testing data ($\bar{u}$), a-priori training for 10000 iterations (L^{prior}), and a-posteriori training for 1000 iterations (L^{post}).

3.5.2.6 *Divergence*

Figure 3.15 shows the evolution of the average divergence $\sqrt{\frac{1}{\bar{N}} \sum_I (\bar{D}\bar{v})_I^2}$ for the two LES models $\mathcal{M}_{\text{DIF}}$ and $\mathcal{M}_{\text{DCF}}$.

For the face-averaging filter, the filtered DNS divergence (reference) is of the order of 10^{-4} due to single-precision arithmetic. For $\mathcal{M}_{\text{DIF}}$, all closure models produce a divergence increasing in time. This is likely due to the instability observed in the previous figures. For $\mathcal{M}_{\text{DCF}}$, all closure models produce divergences at the same order of magnitude as the reference, since the solution is projected at every time step.

For the volume-averaging filter, the filtered DNS divergence (reference) is of the order of 10^{-1}, since Φ^{VA} does not preserve the divergence constraint. For $\mathcal{M}_{\text{DIF}}$, all closure models produce an increasing divergence, just like for Φ^{FA}. For $\mathcal{M}_{\text{DCF}}$, all closure models produce divergence free solutions, even though the reference solution is actually *not* divergence-free.

3.5.2.7 *Computational cost*

The computational time for generating filtered DNS data and training the neural networks is shown in table 3.3. One single DNS trajectory is used to compute filtered DNS trajectories for all filters and coarse grid sizes. In total, we compute 8 DNS trajectories. The two models $\mathcal{M}_{\text{DIF}}$ and $\mathcal{M}_{\text{DCF}}$ both produce similar timings, as exactly the same number of operations are performed (but in a different order). The similar times for the three lowest resolutions are likely because the same number of time steps are unrolled for all LES resolutions, resulting in an equal number of GPU kernel calls. Since 32^2 and 128^2 are both relatively small, the kernels for the differential opera-

tors (convection, Poisson solves, etc.) do not show any speed-up for the lower resolution. This does not seem to be the case for the L^{prior} training however, where most of the kernel calls are to the highly optimized convolutional operators in CUDNN [36].

The benefit of a-posteriori training is clear in situations where the a-priori trained model is unstable (in our case: $\mathcal{M}_{\text{DIF}}$), but for a-priori trained models that are stable and accurate (such as $\mathcal{M}_{\text{DCF}}$ for the face-averaging filter), one could ask whether the additional cost of a-posteriori training is worth it. We think the additional accuracy is useful, since the learned weights can be reused without having to retrain the model, as long as the same configuration is used (grid size, Reynolds number etc.).

3.6 CONCLUSION

The use of neural networks for LES closure models is a promising approach. Neural networks are discrete by nature and thus require consistent discrete training data. To achieve model-data consistency, we propose the paradigm "discretize, differentiate constraint, filter, and close" (in that particular order). This ensures full model-data consistency and circumvents pressure problems. No pressure filter needs to be defined; only a velocity filter is required. Our framework allows for training the neural closure model using a-priori loss functions, where the model is trained to predict the target commutator error directly, or using a-posteriori loss functions, where the model is trained to approximate the target filtered DNS-solution trajectory.

To ensure that the filtered DNS velocity stays divergence-free, we employed the divergence-consistent face-averaging filter used by Kochkov et al. [89]. This allows for using a divergence-constrained LES model similar to those commonly used in LES, but with the important difference that the training data obtained through discrete DNS is fully consistent with the LES environment. This resulted in our new divergence-consistent LES model, which was found to be stable with both a-priori and a-posteriori training. Using the same formulation with a divergence-inconsistent filter (VA), however, required a-posteriori training to achieve the same accuracy. A-posteriori training is more expensive than a-priori training but offers the advantage of increased accuracy.

The divergence-consistent filter stands apart from commonly used (volume-averaging) discrete filters, for which the filtered DNS solution is generally not divergence-free. We showed that the resulting LES model can produce instabilities. A-posteriori trained models improved the stability over a-priori trained models, but this was not sufficient to fully stabilize the model in our experiment. With a divergence-consistent formulation, such stability issues did not occur. Another common approach to achieve stability is to add noise to the training data. However, this could destroy the divergence-free

property of the data, and appropriate measures would need to be taken to avoid this. We tried adding different levels of noise to the training data, but the resulting validation error was consistently lower without noise.

Another important property of our discrete approach is that the (coarse grid) discretization error is included in the training data and learned by the neural network. Turbulence simulations with DNS and LES rely on non-dissipative discretization methods, such as the second order central difference discretization we used in this work, but they produce oscillations and instabilities on coarse grids. This could limit their use in LES, where using coarse grids is one of the main goals. Various smoothing methods, such as explicit LES or (overly) diffusive closure models, are commonly used to address this issue. We instead let the closure model learn to account for the oscillations. The fact that the coarse grid discretization effects (and thus the oscillations) are included in the training data allows the neural network to recognize and correct for these oscillations, even when training with a-priori loss functions.

We acknowledge that the choice of a divergence-consistent face-averaging filter imposes certain constraints. The weights have to be uniform (top-hat filter like) to ensure that all the sub-filter velocities cancel out, and the extension to other filters like Gaussians is an open problem. In addition, we took the filter width to be equal to the grid spacing. The face-averaging filter naturally extends to unstructured grids, as shown in fig. 3.2. However, this requires that the DNS grid perfectly overlaps with the faces of the LES grid, which can be achieved by designing the DNS and LES grids simultaneously. Lastly, the face-averaging filter is only divergence-consistent with respect to the second-order central difference divergence operator on a staggered grid. We intend to explore the use of filters that are divergence-consistent with respect to higher-order discretization methods (such as the discontinuous Galerkin element method) and non-staggered grids. The "discretize first" approach can also be applied on collocated grids. While it is possible to define a face-averaging filter on a collocated *Cartesian* grid, the divergence-consistent property of the filter would no longer hold. On an unstructured collocated grid, it might not make sense to define a face-averaging filter.

Future work will include the use of non-uniform grids with solid-wall boundary conditions. This will require a change in the neural network architecture, but not in the face-averaging filtering procedure itself. The CNN architecture assumes that the grid is uniform, but weighting the kernel stencil with non-uniform volume sizes could be a way to design a discretization-informed neural network. We intend to exploit the fact that discrete DNS boundary conditions naturally extend to the LES model for the face-average filter. We also intend to incorporate other constraints into the LES model, such as conservation of quantities of interest, in particular kinetic energy conservation, as studied in [187].

4

EXACT EXPRESSIONS FOR THE UNRESOLVED STRESS IN A FINITE-VOLUME BASED LARGE-EDDY SIMULATION

The previous chapter formulated the discrete closure problem using finite-volume filters. This chapter generalizes the framework to arbitrary convolutional LES filters (still uniform) and derives exact expressions for the discretization-induced unresolved stresses in conservative form. Unlike chapter 3, where the closure term is a raw residual in the discrete equations, here the total unresolved stress is written as a discrete divergence of a stress tensor, mirroring the divergence structure of the continuous LES equations and enabling physically consistent closure modeling.

We propose new discretization-informed expressions for the residual stress tensor (RST) in a finite-volume based large-eddy simulation (LES-FVM). In addition to the classical RST $\overline{uu}-\bar{u}\bar{u}$ resulting from the non-commutation between filtering and the nonlinear stress, our RST also contains contributions from the numerical flux, discrete divergence, and pressure terms. Unlike the classical RST, our proposed RST is non-symmetric and non-local. The proposed form of the RST is important for generating appropriate reference data for LES closure modeling.

Based on DNS results of the 1D Burgers and 3D incompressible Navier-Stokes equations, we show that the discretization-induced parts of the RST play an important role in the LES-FVM equation for common LES filter widths. When the discrete contribution is included, our RST expression gives zero a-posteriori error in LES, while existing RST expressions give errors that increase over time. For a Smagorinsky model, we show that the Smagorinsky coefficient is higher when fitted to our new RST than when fitted to the classical RST and gives improved results.

This chapter is based on the following article: Syver Døving Agdestein, Roel Verstappen, and Benjamin Sanderse. "Exact Expressions for the Unresolved Stress in a Finite-Volume Based Large-Eddy Simulation." In: Journal of Computational Physics *556 (July 2026), p. 114810.* DOI: *10.1016/j.jcp.2026.114810.*

CRediT author statement
***Syver Døving Agdestein**: Conceptualization, Formal analysis, Methodology, Software, Validation, Visualization, Writing – Original Draft*
***Roel Verstappen**: Supervision, Writing – Review & Editing*
***Benjamin Sanderse**: Funding acquisition, Project administration, Supervision, Writing – Review & Editing*

4.1 INTRODUCTION

The incompressible Navier-Stokes equations can be discretized using the finite volume method (FVM). Like LES, the FVM considers filtered velocities, with the filter being the average over a control volume. In LES, this filter can have other definitions, for example a convolution with an arbitrary kernel function. Unlike LES, the FVM equations are discrete by construction.

The continuous LES equations include a continuous divergence of a flux function. The discrete FVM equations contain an integral of a flux function over the control volume boundary. By defining this boundary integral as a *discrete* divergence operator applied to the given flux, the FVM equations

take the same form as the continuous LES equation. This allows for treating LES and the FVM as the same problem, as suggested by Verstappen [193].

Both the FVM and LES result in approximate equations for the filtered velocity field $\bar{u}$ that differ from the exact filtered conservation law. The mismatch between the approximate equations and the exact filtered conservation law can be written as a residual term, sometimes called a commutator error.

Most works on the closure problem focus on modeling the commutator between filtering and nonlinearities [17, 144, 152]. Assuming filtering commutes with spatial and temporal differentiation, the residual term takes the form of the divergence of a tensor, $\nabla \cdot \tau(u)$, which is a function of the resolved and unresolved velocity fields $\bar{u}$ and u through the residual stress tensor (RST) $\tau_{ij}(u) \coloneqq \overline{u_i u_j} - \bar{u}_i \bar{u}_j$. The exact LES equations are approximated by replacing $\tau(u)$ with a closure model $m(\bar{u})$. *Functional* models try to match the dissipation produced by $\tau(u)$, while *structural* models also try to model the structure of the RST $\tau(u)$ itself [109, 152].

Apart from the difficulty of correctly modeling the residual $\nabla \cdot \tau(u)$, another major challenge in LES is the appearance of additional residual terms when the equations are discretized [7, 14]. These terms are commonly referred to as *discretization errors*. For common discretization schemes, they are controlled by convergence bounds on the grid spacing h.

For typical LES scenarios, the filter width Δ is of the same order of magnitude as the grid spacing h [38, 65], sometimes with an exact equality $\Delta = h$ [48, 63, 132, 198]. For this reason the discretization errors can be of the same order of magnitude as $\nabla \cdot \tau(u)$ [41], which can strongly limit the usefulness of any closure modeling effort that only accounts for $\nabla \cdot \tau(u)$. This was illustrated by Bae and Lozano-Duran, who used DNS to assess the importance of the discretization error by computing the residual $\nabla \cdot \tau(u)$ explicitly [7]. Recognition of this issue has motivated the development of LES frameworks where the total residual from both nonlinearities and discretization is modeled [14, 46, 63]. However, the exact form of this total residual term has not yet been established.

We are interested in the exact discretization-informed expression for the total residual because the final goal is to use such an expression as target data for tuning the parameters of a closure model. In previous work, we showed that using a discretization-informed residual as target data for data-driven closure models gives stable models, while using incorrect target data leads to instabilities [2]. In this work, we introduce a new mathematical formalism to derive exact expressions for the residual appearing in the discretized LES equations. Furthermore, we write the residual in the form of a discrete divergence of an RST. We show how this RST differs between classical LES and the classical FVM, and propose a new discrete LES-FVM framework that merges the two.

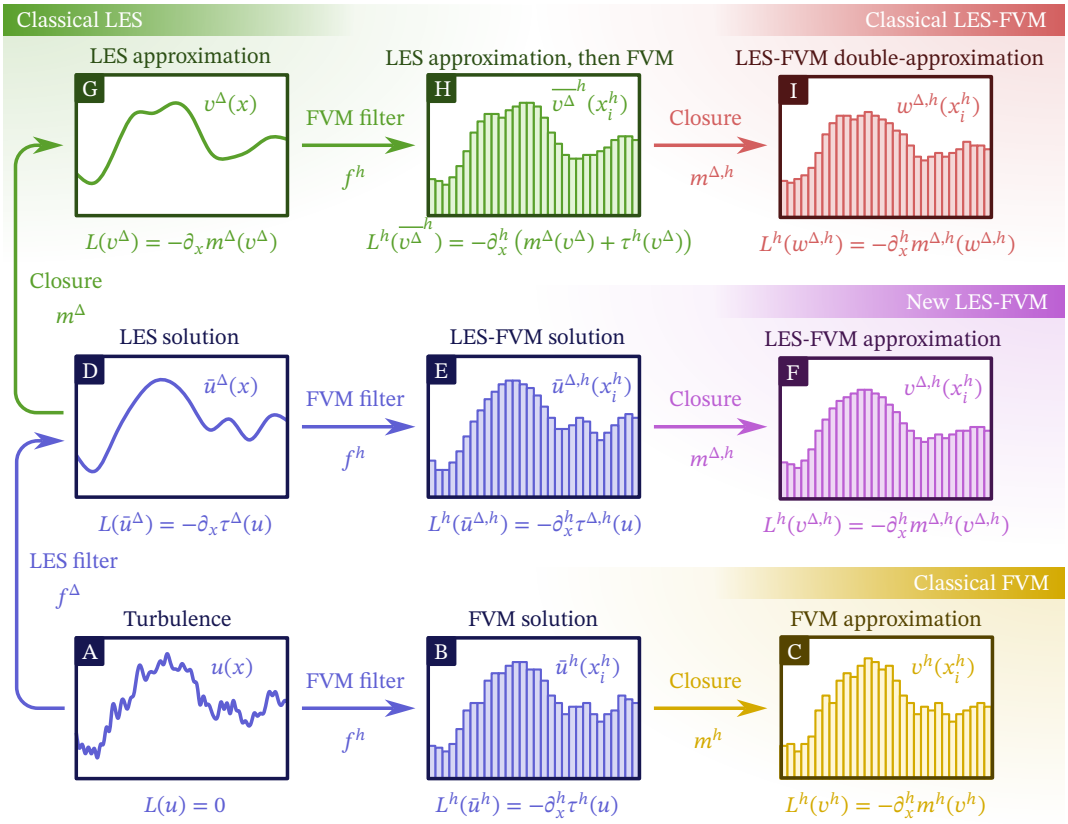


Figure 4.1: Three different routes to a closed system of discrete equations that simulate the large-scales of a turbulent flow u. A change of color indicates an approximation step, while a preservation of color indicates an exact operation. The grid points are $(x_i^h)_{i\in\mathbb{Z}}$. Classical LES-FVM relies on two separate approximation steps, whereas the FVM and our proposed LES-FVM framework only rely on one approximation step.

The main ideas in this article are visualized in fig. 4.1. The notation used in the figure is explained in detail in the following sections. All equations are derived from the original conservation law for the turbulent solution u in fig. 4.1-A. The approach we use is to first transform the equation for u by applying LES and FVM filters. The equations are then rewritten into the desired form, and the residual terms are gathered into an unresolved stress. The approximation steps (called "closures") are only performed after the exact residual stress has been defined. These approximation steps are indicated by a change of color in the figure.

In section 4.2, we first introduce our notation. For 1D conservation laws, we derive classical LES and the FVM in this notation, so that in section 4.3, we can merge classical LES and the FVM into a unified discrete LES-FVM framework. We derive new exact discretization-informed RST expressions for the given framework. In section 4.4, the importance of the RST definition is assessed for the 1D Burgers equation. We show that using our discretization-informed RST as a closure term gives a "perfect" closure model with zero a-posteriori error, while the classical RST gives errors that accumulate over time. This difference remains visible when the RSTs are used as target data in a Smagorinsky model. In section 4.5, we extend our LES-FVM framework to the 3D incompressible Navier-Stokes equations. Due to the incompressibility constraint, the RST is shown to be non-local, and the discrete divergence form makes it non-symmetric. In section 4.6, we repeat the Burgers experiments for a 3D decaying turbulence simulation, confirming that our expressions are correct for the full incompressible Navier-Stokes equations.

4.2 PRELIMINARIES: LES AND THE FVM

Let $\Omega = [0, \ell]$ be a 1D domain with length $\ell > 0$. Let $U \coloneqq \{u : \mathbb{R} \to \mathbb{R} \mid u(x) = u(x + \ell)\}$ be the space of periodic functions on Ω. Depending on the problem, U may need to be further restricted to more regular spaces such as $L^2(\Omega)$ or $H^1(\Omega)$ [17].

Consider the generic infinitesimal 1D conservation law

$$L(u) \coloneqq \partial_t u + \partial_x r(u) = 0 \tag{4.1}$$

(see fig. 4.1-A), where $u(x, t)$ is the solution at a given point x and time t, $L \coloneqq \partial_t + \partial_x r$ is the equation operator, $\partial_t \coloneqq \partial/\partial t$ and $\partial_x \coloneqq \partial/\partial x$ are partial derivatives, and the flux $r : U \to U$ is a non-linear spatial operator. We call the conservation law *infinitesimal* since the divergence ∂_x and potentially the flux r are infinitesimal operators (as opposed to discrete operators that can be computed on a grid). For simplicity, we omit a source term in (4.1). Including one is straightforward and does not affect the derivations in this article.

The conservation law (4.1) is a continuous equation defined by requiring that the field $L(u) \in U$ is zero everywhere. We can evaluate $L(u)$ in a point

$x \in \Omega$ and time $t \geq 0$ as $L(u)(x,t) \in \mathbb{R}$. In the following, we omit the time t and write $L(u)(x) \in \mathbb{R}$ and $u(x) \in \mathbb{R}$ etc.

An example of a non-linear conservation law is the viscous Burgers equation. The corresponding flux is defined as

$$r(u) \coloneqq \frac{1}{2}uu - \nu \partial_x u, \tag{4.2}$$

where $\nu > 0$ is a constant viscosity (diffusion coefficient).

The PDE (4.1) can be discretized and solved directly by using the FVM. Alternatively, the equation can first be filtered and approximated with an LES closure. To distinguish between these two approaches, we use the superscript $(\cdot)^\Delta$ for quantities related to LES, and $(\cdot)^h$ for quantities related to the FVM.

4.2.1 *Filtering and LES*

Equation (4.1) describes all the scales of motion for the given system. Filtering (4.1) with a convolutional LES filter

$$f^\Delta : U \to U,\ u \mapsto \bar{u}^\Delta \tag{4.3}$$

of filter-width $\Delta \geq 0$ gives the LES equation

$$\overline{L(u)}^\Delta = 0, \tag{4.4}$$

where $\bar{u}^\Delta \coloneqq f^\Delta u$ is the LES solution that we intend to solve for.

The convolution f^Δ is defined through a kernel $G^\Delta : \mathbb{R} \to \mathbb{R}$ by

$$\bar{u}^\Delta(x) \coloneqq \int_{\mathbb{R}} G^\Delta(x-y)u(y)\,\mathrm{d}y \tag{4.5}$$

for all $u \in U$ and $x \in \Omega$. Note that we integrate over $\mathbb{R}$ (not Ω), since the filter kernel G^Δ needs to be extended beyond the periodic boundary. Some kernels (such as the Gaussian kernel) have infinite support. In practice, such kernels are truncated, and one periodic extension is sufficient.

It is common to decompose the LES equation into a resolved and unresolved part as

$$\begin{aligned} L(\bar{u}^\Delta) &= -\left(\overline{L(u)}^\Delta - L(\bar{u}^\Delta)\right) \\ &= -\left(\overline{\partial_x r(u)}^\Delta - \partial_x r(\bar{u}^\Delta)\right), \end{aligned} \tag{4.6}$$

where the left-hand side only depends on the resolved scales $\bar{u}^\Delta$ and the right-hand side is a residual that still depends on the full solution u. This term is not yet the "divergence of a flux", meaning that (4.6) is not expressed

as a conservation law. However, since f^Δ is a convolution, it satisfies the *filter-swap* commutation property

$$f^\Delta \partial_x = \partial_x f^\Delta. \tag{4.7}$$

For proof, see theorem 1 in B.3. This can also be written as $\overline{\partial_x u}^\Delta = \partial_x \bar{u}^\Delta$ for all $u \in U$. This commutation property can be used to obtain a *conservative* form of the residual and rewrite eq. (4.6) as a conservation law

$$L(\bar{u}^\Delta) = -\partial_x \tau^\Delta(u) \tag{4.8}$$

(see fig. 4.1-D), where the commutator between L and f^Δ takes the form of the divergence of a residual flux

$$\boxed{\tau^\Delta(u) \coloneqq \overline{r(u)}^\Delta - r(\bar{u}^\Delta)}. \tag{4.9}$$

This flux is also known as the *sub-filter* flux. For the Burgers equation, we get the well-known expression $\tau^\Delta(u) = (\overline{uu}^\Delta - \bar{u}^\Delta \bar{u}^\Delta)/2$. As $\tau^\Delta(u)$ is a flux, it has dissipation properties that could potentially be replicated by a closure model. For example, a convective flux conserves energy, while a diffusive flux dissipates energy. Since $\partial_x \tau^\Delta(u)$ is of conservative form, the filtered momentum is conserved, which cannot be guaranteed otherwise. We highlight these properties and how they are obtained here since they do not automatically apply in the discrete case.

Equation (4.8) is exact, but unclosed. The next step in LES is to approximate the residual flux by a closure model $m^\Delta(\bar{u}^\Delta) \approx \tau^\Delta(u)$. Closure models are often chosen in the functional eddy-viscosity form:

We use the symbol "≈" solely to indicate the intent *of an approximation to guide the reader. The statement "a ≈ b" provides no guarantee that a is in any way similar or close to b. Instead, we take care to use different symbols for new quantities arising from an approximation step, for example $v^\Delta \neq \bar{u}^\Delta$.*

$$m^\Delta(\bar{u}^\Delta) \coloneqq -\nu^\Delta(\bar{u}^\Delta)\partial_x \bar{u}^\Delta, \tag{4.10}$$

where $\nu^\Delta : U \to U$ is an eddy viscosity chosen such that the dissipation produced by $m^\Delta(\bar{u}^\Delta)$ is similar to the one produced by $\tau^\Delta(u)$. For the basic Smagorinsky model [168], the viscosity is

$$\nu^\Delta(\bar{u}^\Delta) \coloneqq \theta^2 \Delta^2 |\partial_x \bar{u}^\Delta|, \tag{4.11}$$

where $\theta > 0$ is a model parameter.

The approximate LES equation is

$$L(v^\Delta) = -\partial_x m^\Delta(v^\Delta) \tag{4.12}$$

(see fig. 4.1-G), where $v^\Delta \approx \bar{u}^\Delta$ is the approximate LES solution. It is in general different from $\bar{u}^\Delta$ since the closure m^Δ cannot be exact when information is lost in the filtering process.

To summarize, we used the following steps to derive a closed form for the approximate equation in LES:

1. Apply the LES filter f^Δ.
2. Decompose the equation into a resolved part and a residual.
3. Rewrite the residual in conservative form using the filter-swap property.
4. Approximate the residual flux with a closure model.

These steps are visualized in the route A→D→G in fig. 4.1.

Equation (4.12) still needs to be discretized. The discretization we use is the FVM. As we will see, the equations for the FVM can be derived in a similar way as for LES.

4.2.2 *The FVM through filter-swap*

Let $h > 0$ denote the grid spacing of a uniform grid on Ω. The derivations in this section hold for all points $x \in \Omega$, and we do not restrict any quantities to the grid points until section 4.4. For now, h can be considered as a particular length scale that is independent from the LES filter width Δ from section 4.2.1.

4.2.2.1 *Numerical derivatives and fluxes*

For all $u \in U$ and $x \in \Omega$, define the staggered central finite difference and interpolation operators as

$$\partial_x^h u(x) \coloneqq \frac{1}{h}\left[u\left(x + \frac{h}{2}\right) - u\left(x - \frac{h}{2}\right)\right], \tag{4.13}$$

$$\eta_x^h u(x) \coloneqq \frac{1}{2}\left[u\left(x - \frac{h}{2}\right) + u\left(x + \frac{h}{2}\right)\right]. \tag{4.14}$$

These operators can be used to discretize the conservation law (4.1). If $r^h \approx r$ is a numerical flux, then $\partial_x^h r^h \approx \partial_x r$ is an approximation of the flux term, and $L^h \coloneqq \partial_t + \partial_x^h r^h$ is a (spatial) approximation of the conservation law. For the Burgers equation (defined by eq. (4.2)), we use the second-order accurate numerical flux

$$r^h(u) \coloneqq \frac{1}{2}\left(\eta_x^h u\right)\left(\eta_x^h u\right) - \nu \partial_x^h u. \tag{4.15}$$

The operators $\partial_x^h : U \to U$, $\eta_x^h : U \to U$, and $r^h : U \to U$ are both continuous and discrete at the same time. They are continuous in the sense that $\partial_x^h u(x)$ can be evaluated for all $x \in \Omega$. They are discrete in the sense that $\partial_x^h u(x)$ only depends on $u(x \pm h/2)$, and for the numerical Burgers flux (4.15), $\partial_x^h r^h(u)(x)$ only depends on $u(x-h)$, $u(x)$, and $u(x+h)$. We therefore say that these operators are *grid-compatible* or h-compatible. See B.1 for more details about infinitesimal, discrete, continuous, and grid-compatible operators.

4.2.2.2 *The FVM grid filter*

The finite difference ∂_x^h is closely related to the FVM grid filter

$$f^h : U \to U,\ u \mapsto \bar{u}^h \tag{4.16}$$

defined for all $x \in \Omega$ as

$$\bar{u}^h(x) \coloneqq \frac{1}{h}\int_{x-h/2}^{x+h/2} u(y)\,\mathrm{d}y. \tag{4.17}$$

This filter is sometimes called a "top-hat filter", "Schumann's filter" [158], or "volume-averaging filter", since it averages u over a control volume $[x \pm h/2]$. The FVM filter f^h constitutes a particular choice of convolutional LES filter f^Δ, where the filter width Δ is equal to the grid spacing h used in the finite difference ∂_x^h. The underlying top-hat kernel G^h is

$$G^h(x) \coloneqq \begin{cases} \frac{1}{h} & \text{if } |x| \leq \frac{h}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.18}$$

A filter width equal to the grid spacing is commonly the setting in implicit LES, but we emphasize that here the FVM filter definition is known explicitly.

The infinitesimal operator ∂_x is not grid-compatible. By applying the FVM filter f^h, the derivative ∂_x can be turned into the grid-compatible finite difference ∂_x^h through the discrete filter-swap property

$$\partial_x^h = f^h \partial_x, \tag{4.19}$$

which can also be written as $\partial_x^h u = \overline{\partial_x u}^h$ for all $u \in U$. The proof is given in theorem 3. This property entails that the finite difference ∂_x^h is equal to a filtered version of the exact derivative ∂_x, as noted by Schumann and others [111, 150, 158]. If we approximate an infinitesimal derivative ∂_x by ∂_x^h, the content of the derivative is implicitly filtered. The FVM filter f^h is therefore sometimes called a "discretization-induced filter" or an "implicit filter" induced by the choice of ∂_x^h. We prefer to make this statement explicit by turning it around and explicitly applying f^h to a derivative ∂_x. The finite difference ∂_x^h is therefore a "filter-induced discretization" of the infinitesimal derivative ∂_x.

The discrete filter-swap property $\partial_x^h = f^h \partial_x$ is a key building block of the LES-FVM framework.

We see the property (4.19) as an analogous version of the commutation property (4.7) for the finite difference operator ∂_x^h and grid filter f^h. The subtle, yet crucial, difference with (4.7) is that the left-hand side is no longer filtered and uses a discrete differentiation operator. An intuitive way to understand this property is through a telescoping sum. See fig. B.2 in B.3 for a visualization of how the inner terms in the integral defining f^h cancel out in the discrete filter-swap property. Unlike the property (4.7), we have $\partial_x^h f^h \neq f^h \partial_x$ (see theorem 2 for proof).

4.2.2.3 *FVM equations*

To obtain FVM equations, we use the same approach as we used for LES in section 4.2.1:

1. Apply the FVM filter f^h.
2. Decompose the equation into a resolved part and a residual.
3. Rewrite the residual in conservative form using the discrete filter-swap property.
4. Approximate the residual flux with a closure model.

These steps are visualized in the bottom route in fig. 4.1.

Applying the FVM filter f^h to eq. (4.1) yields the FVM equation

$$\overline{L(u)}^h = 0, \tag{4.20}$$

where $\bar{u}^h$ is the FVM solution that we intend to solve for. While the classical LES formulation could be obtained by using the filter-swap property $f^h\partial_x = \partial_x f^h$, our aim is to have an equation for $\bar{u}^h$ that involves the discrete divergence ∂_x^h. This is achieved by applying the discrete filter-swap property to eq. (4.20):

$$\partial_t \bar{u}^h + \partial_x^h r(u) = 0. \tag{4.21}$$

This equation is more commonly written as

$$h\partial_t \bar{u}^h(x) + r(u)\left(x + \frac{h}{2}\right) - r(u)\left(x - \frac{h}{2}\right) = 0, \tag{4.22}$$

emphasizing that we have two unknown fluxes at the left and right boundaries of the control volume $[x \pm h/2]$.

Equation (4.22) is often derived from the integral form of the conservation law (4.1):

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V u \,\mathrm{d}V + \int_{\partial V} r(u)\cdot n \,\mathrm{d}S = 0, \quad \forall V \subset \Omega. \tag{4.23}$$

In 1D, with $V = [x \pm h/2]$, $\partial V = \{x \pm h/2\}$, and $n = \pm 1$, this is exactly eq. (4.22).

Let $r^h : U \to U$ denote a grid-compatible numerical flux that approximates the original flux r. By adding the resolved term $\partial_x^h r^h(\bar{u}^h)$ to both sides of eq. (4.21) and rearranging the terms, we get

$$L^h(\bar{u}^h) = -\partial_x^h \tau^h(u) \tag{4.24}$$

(see fig. 4.1-B), where $L^h \coloneqq \partial_t + \partial_x^h r^h$ is a grid-compatible approximation of the original conservation law operator L and

$$\boxed{\tau^h(u) \coloneqq r(u) - r^h(\bar{u}^h)} \tag{4.25}$$

is the residual flux in the FVM equation. Unlike the LES equation (4.8), the FVM equation (4.24) is "∂_x^h-conservative", since the conservation law is written with respect to the discrete divergence ∂_x^h. The residual flux $\tau^h(u)$ is also known as the *sub-grid* flux. Calling it a sub-filter flux would be misleading, since it also depends on the choice of numerical divergence ∂_x^h and numerical flux r^h (in addition to the FVM filter f^h). We use the term *residual flux* for both $\tau^\Delta(u)$ and $\tau^h(u)$.

By approximating the residual flux with a grid-compatible dissipation model $m^h(\bar{u}^h) \approx \tau^h(u)$, we get the approximate FVM equation

$$L^h(v^h) = -\partial_x^h m^h(v^h) \tag{4.26}$$

(see fig. 4.1-C), where $v^h \approx \bar{u}^h$ is the approximate FVM solution. In a properly resolved DNS, where h is sufficiently small, the dissipation model m^h can be set to zero.

The operators L^h and m^h are still continuous and can be evaluated in every point $x \in \Omega$. "Fully discrete" equations for v^h can be obtained by simply evaluating eq. (4.26) at the grid points $x_i^h \coloneqq ih$, $i \in \{1, \dots, N\}$ on the condition that $h = \ell/N$ for some integer $N \in \mathbb{N}$. This restriction is visualized in fig. 4.1-B and fig. 4.1-C. The grid-compatible numerical fluxes r^h and m^h ensure that the restricted equations form a closed system of ordinary differential equations.

While eq. (4.24) can be seen as a discrete LES equation, the filter width is implied by the grid spacing h used in the discrete divergence ∂_x^h. In LES, we would like to be able to choose the filter width Δ independently from h. We therefore propose to combine the LES filter f^Δ with the FVM filter f^h.

4.3 COMBINING LES AND THE FVM: LES-FVM

Classical LES and the FVM both aim to correctly model the features of the flow that are larger than a certain length scale. In LES, the large scales are extracted using an LES filter f^Δ, which retains the scales that are larger than the filter width Δ. In the FVM, the size of the resolved scales is inherently linked to the grid size h through the FVM filter f^h, numerical flux r^h, and discrete divergence ∂_x^h. The fluxes in the LES equation (4.8) are defined with respect to the infinitesimal divergence ∂_x, but the fluxes in FVM equation (4.24) are defined with respect to the discrete divergence ∂_x^h.

In this section, we present a new discrete LES formalism—"LES-FVM"—that bridges the gap between infinitesimal LES and the discrete FVM. In LES-FVM, the LES filter is applied to selectively extract the scales of interest,

while the FVM filter is applied to make the divergence grid-compatible. We first show the classical approach in which one approximates before discretizing (top route in fig. 4.1), and then we propose our new approach in which we discretize first (middle route in fig. 4.1).

4.3.1 *The classical approach: LES first, then FVM*

After closure, the approximate LES equations are $L(v^\Delta) = -\partial_x m^\Delta(v^\Delta)$ (see fig. 4.1-G). Applying the FVM filter f^h, filter-swap property, adding the resolved term $\partial_x^h r^h(\overline{v^\Delta}^h)$, and rearranging the terms gives

$$L^h\left(\overline{v^\Delta}^h\right) = -\partial_x^h\left(m^\Delta(v^\Delta) + \tau^h(v^\Delta)\right) \tag{4.27}$$

(see fig. 4.1-H), where $\overline{v^\Delta}^h$ is the FVM average of the LES approximation v^Δ. This equation is unclosed in terms of $\overline{v^\Delta}^h$ since it contains the field v^Δ in the right-hand side flux $m^\Delta(v^\Delta) + \tau^h(v^\Delta)$. Note that $\tau^h(v^\Delta)$ is the residual FVM flux from eq. (4.25) applied to the LES approximation v^Δ instead of u.

By using a grid-compatible closure model $m^{\Delta,h}(\overline{v^\Delta}^h) \approx m^\Delta(v^\Delta) + \tau^h(v^\Delta)$, we get the approximate equation

$$L^h(w^{\Delta,h}) = -\partial_x^h m^{\Delta,h}(w^{\Delta,h}) \tag{4.28}$$

(see fig. 4.1-I), where $w^{\Delta,h} \approx \overline{v^\Delta}^h$ is the approximate solution. We call $m^{\Delta,h}$ a *closure* since it approximates an unresolved term and thus makes the equation closed. Typically, $m^{\Delta,h}$ is just a "discretization" of the infinitesimal LES closure m^Δ, but the expression $m^\Delta(v^\Delta) + \tau^h(v^\Delta)$ suggests that $m^{\Delta,h}$ should also account for the FVM discretization error $\tau^h(v^\Delta)$. This classical approach involves multiple steps: close first, then discretize, and then adjust the closure to account for the discretization (as visualized in the top route of fig. 4.1).

We now show what this classical approach would entail for the basic Smagorinsky closure model. The LES closure

$$m^\Delta(\bar{u}^\Delta) := -\theta^2\Delta^2\left|\partial_x\bar{u}^\Delta\right|\partial_x\bar{u}^\Delta \tag{4.29}$$

is used for the LES flux $\tau^\Delta(u)$, while its FVM variant

$$m^h(\bar{u}^h) := -\theta^2 h^2\left|\partial_x^h\bar{u}^h\right|\partial_x^h\bar{u}^h, \tag{4.30}$$

is used for the FVM flux $\tau^h(u)$. A natural choice for $m^{\Delta,h}$ is then

$$m^{\Delta,h}(\overline{v^\Delta}^h) := -\theta^2(\Delta^2 + h^2)\left|\partial_x^h\overline{v^\Delta}^h\right|\partial_x^h\overline{v^\Delta}^h. \tag{4.31}$$

The term $\theta^2\Delta^2|\partial_x^h \cdot |\partial_x^h \cdot$ is a discretization of m^Δ, while $\theta^2 h^2|\partial_x^h \cdot |\partial_x^h \cdot$ is a discrete Smagorinsky model for the residual FVM flux τ^h. The final model $m^{\Delta,h}$ takes the form of a discrete Smagorinsky model for a filter of width $\sqrt{\Delta^2 + h^2}$. The incorporation of h into the filter width can be seen as an adjustment to account for discretization error $\tau^h(v^\Delta)$.

Interestingly, if the LES filter f^Δ is a Gaussian or top-hat filter, then $\sqrt{\Delta^2 + h^2}$ is the width of the LES-FVM *double filter*

$$f^{\Delta,h} := f^h f^\Delta \tag{4.32}$$

Note that double filters in LES are also used for dynamically determining closure model coefficients by applying a test-filter on top of the LES filter [61]. Our double filter serves a different purpose: to filter out scales smaller than Δ while also coarse-graining the differential operators so that they become compatible with a grid of spacing h.

composed of the LES filter f^Δ and the FVM filter f^h. The field $w^{\Delta,h}$ can therefore be seen as a Smagorinsky approximation of the LES-FVM solution $\bar{u}^{\Delta,h} := f^{\Delta,h}u$.

In the next section, we propose to approximate $\bar{u}^{\Delta,h}$ directly, using one single approximation step $v^{\Delta,h} \approx \bar{u}^{\Delta,h}$, instead of performing successive approximations $v^\Delta \approx \bar{u}^\Delta$ and $w^{\Delta,h} \approx \overline{v^\Delta}^h$.

4.3.2 *Our new approach: LES-FVM*

We discretize the infinitesimal LES equation $\overline{L(u)}^\Delta = 0$ by applying the FVM filter f^h. This gives the double filtered conservation law

$$\overline{L(u)}^{\Delta,h} = 0. \tag{4.33}$$

By using the filter-swap property, adding the resolved term $\partial_x^h r^h(\bar{u}^{\Delta,h})$ to both sides, and rearranging the terms, we get an equation for $\bar{u}^{\Delta,h}$ in ∂_x^h-conservative form:

$$L^h(\bar{u}^{\Delta,h}) = -\partial_x^h \tau^{\Delta,h}(u) \tag{4.34}$$

(see fig. 4.1-E), where

$$\boxed{\tau^{\Delta,h}(u) := \overline{r(u)}^\Delta - r^h(\bar{u}^{\Delta,h})} \tag{4.35}$$

Novel residual flux expression: "discretize first, then approximate" instead of the classical "approximate first, then discretize."

is the residual flux in the LES-FVM equation. This is a novel residual flux expression that accounts for the errors in both LES and the FVM. The first term only uses the LES filter f^Δ and the infinitesimal flux r, while the second term uses the LES-FVM filter $f^{\Delta,h}$ and the numerical flux r^h.

Using a discretization-informed grid-compatible closure model $m^{\Delta,h}(\bar{u}^{\Delta,h}) \approx \tau^{\Delta,h}(u)$ gives the approximate LES-FVM equation

$$L^h\left(v^{\Delta,h}\right) = -\partial_x^h m^{\Delta,h}\left(v^{\Delta,h}\right) \tag{4.36}$$

(see fig. 4.1-F), where $v^{\Delta,h} \approx \bar{u}^{\Delta,h}$ is the approximate LES-FVM solution. Although the equation for $v^{\Delta,h}$, obtained by "discretizing first, then approximating", has the same form as the equation for $w^{\Delta,h}$ in (4.28), obtained

through multiple alternating filtering and approximation steps, it leads to a different solution because the closure is chosen differently. In eq. (4.28), $m^{\Delta,h}$ is chosen such that $w^{\Delta,h} \approx \overline{v^{\Delta}}^{h}$, which depends on a previous approximation $v^{\Delta} \approx \bar{u}^{\Delta}$. In eq. (4.36), $m^{\Delta,h}$ is chosen such that $v^{\Delta,h} \approx \bar{u}^{\Delta,h}$ directly.

Having defined our new LES-FVM framework, we now proceed to discuss its implications.

4.3.3 *Control over the LES and FVM filter widths*

For $f^{\Delta,h}$, we are free to choose Δ and h independently. Choosing $\Delta \gg h$ would result in a grid that resolves scales that are not present in the LES solution $\bar{u}^{\Delta}$, leading to unnecessary computational costs. This setting is sometimes referred to as *explicit LES*, where the effects of the discretization can be neglected. The advantage is that the classical LES framework can be used with small discretization errors. In this case, we have $\tau^{\Delta,h}(u) \approx \tau^{\Delta}(u)$. On the other hand, if $\Delta = 0$, we have $f^{\Delta,h} = f^{h}$, and we recover the classical FVM setting. This setting is sometimes referred to as *implicit LES*, where no explicit LES filter f^{Δ} is used. In literature, common choices for the ratio are $0 \leq \Delta/h \leq 4$ [63, 65]. In this range, the effect of the FVM filter is comparable to or somewhat smaller than that of the LES filter. It is generally acknowledged that the ratio Δ/h should ideally be higher to prevent interference from the discretization in the residual (for example $\Delta/h \geq 4$ for a second-order accurate discretization [38] or $\Delta/h \geq 2$ for a sixth-order accurate discretization [200]).

Unlike classical (infinitesimal) LES, the LES-FVM framework allows arbitrary Δ/h ratios.

Instead of circumventing the problem of the discretization in LES by choosing $\Delta \gg h$, we aim to allow for smaller ratios Δ/h by explicitly accounting for the discretization effects in our new residual flux $\tau^{\Delta,h}(u)$. In our new LES-FVM framework, all values of the ratio Δ/h are valid choices.

In fig. 4.2, we show the filter kernels G^{Δ}, G^{h}, and $G^{\Delta,h}$ corresponding to the filters f^{Δ}, f^{h}, and $f^{\Delta,h}$ for top-hat and Gaussian LES kernels

$$G^{\Delta}_{\text{top-hat}}(x) := \begin{cases} \frac{1}{\Delta} & \text{if } |x| \leq \frac{\Delta}{2}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.37}$$

$$G^{\Delta}_{\text{Gaussian}}(x) := \sqrt{\frac{6}{\pi\Delta^2}} \exp\left(-\frac{6x^2}{\Delta^2}\right). \tag{4.38}$$

The LES filter widths are $\Delta \in \{1h, 2h, 3h\}$. For top-hat G^{Δ} with $\Delta = h$, we have $G^{\Delta} = G^{h}$, and $G^{\Delta,h}$ becomes a triangular kernel. For top-hat G^{Δ} with $\Delta > h$, the double filter kernel $G^{\Delta,h}$ takes the shape of a trapezoid [200], with $G^{\Delta,h}(x) = G^{\Delta}(x)$ for $|x| \leq (\Delta - h)/2$. For Gaussian G^{Δ}, the double filter kernel $G^{\Delta,h}$ resembles a Gaussian kernel of width $\sqrt{\Delta^2 + h^2}$. However, $G^{\Delta,h}$ is not exactly Gaussian. As the ratio Δ/h grows, G^{Δ} and $G^{\Delta,h}$ become

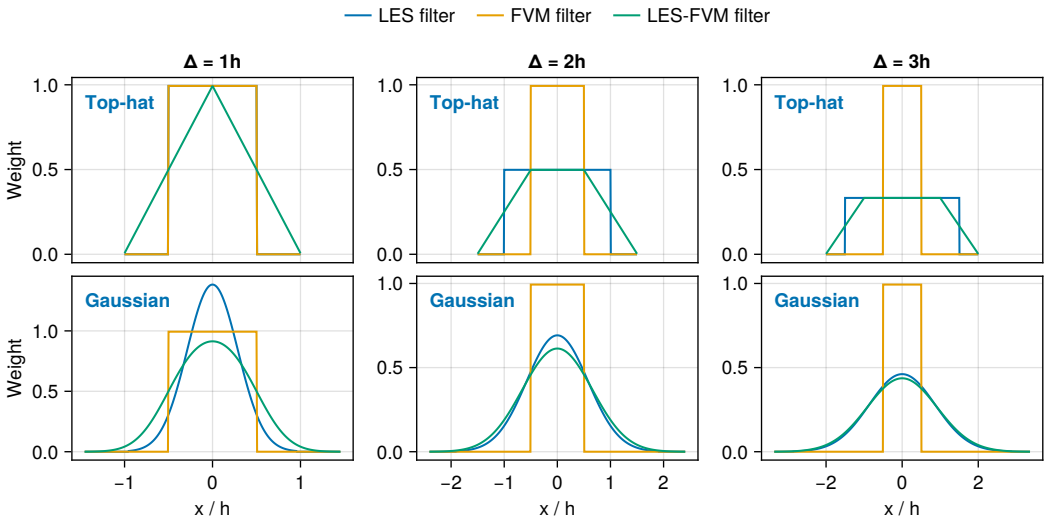


Figure 4.2: Spatial kernels of LES filter f^Δ (top-hat and Gaussian), FVM grid filter f^h (always top-hat), and LES-FVM double filter $f^{\Delta,h}$ for $\Delta \in \{1h, 2h, 3h\}$. In the top-left plot, we have $f^\Delta = f^h$.

more and more similar. For the ratios shown, G^Δ and $G^{\Delta,h}$ are still visually different.

For the top-hat and Gaussian LES filters, the spectral transfer functions are defined as [144]

$$\hat{G}^\Delta_{\text{top-hat}}(\kappa) := \frac{\sin(\kappa\Delta/2)}{\kappa\Delta/2}, \tag{4.39}$$

$$\hat{G}^\Delta_{\text{Gaussian}}(\kappa) := \exp\left(-\frac{\kappa^2\Delta^2}{24}\right), \tag{4.40}$$

where $\kappa := 2\pi k$ for $k \in \mathbb{Z}$. Interestingly, they both give the same Taylor series expansion up to second-order around $\kappa = 0$:

$$\hat{G}^\Delta_{\text{top-hat}}(\kappa) = 1 - \frac{\kappa^2\Delta^2}{24} + \mathcal{O}(\kappa^4), \tag{4.41}$$

$$\hat{G}^\Delta_{\text{Gaussian}}(\kappa) = 1 - \frac{\kappa^2\Delta^2}{24} + \mathcal{O}(\kappa^4). \tag{4.42}$$

For the double-filter $f^{\Delta,h}$, we therefore get the expansion

$$\hat{G}^{\Delta,h}(\kappa) = \hat{G}^\Delta(\kappa)\hat{G}^h(\kappa) = 1 - \frac{\kappa^2(\Delta^2 + h^2)}{24} + \mathcal{O}(\kappa^4) \tag{4.43}$$

for both the top-hat and Gaussian LES kernels. By comparing the second-order terms, we can make the argument that $\sqrt{\Delta^2 + h^2}$ is indeed the width of

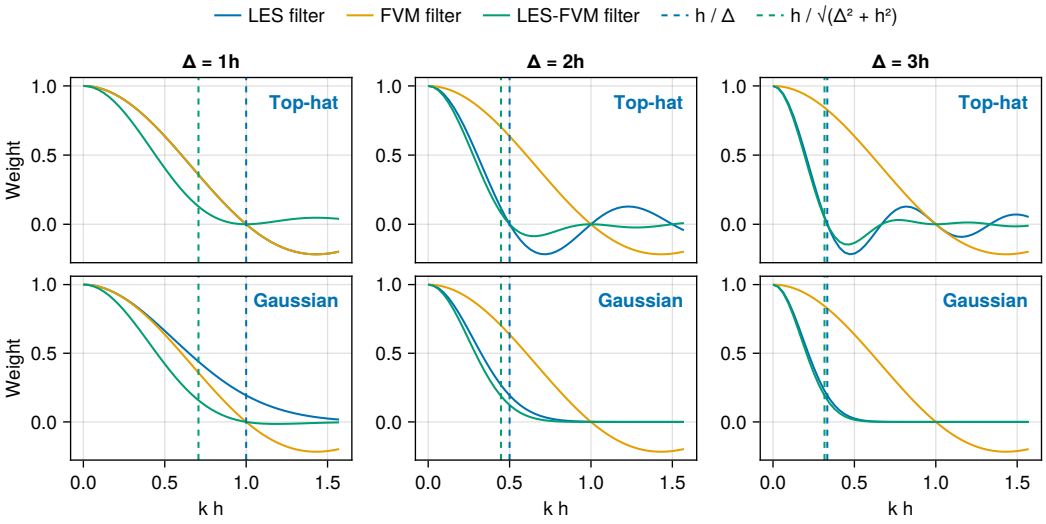


Figure 4.3: Spectral transfer functions of LES filter f^Δ (top-hat and Gaussian), FVM grid filter f^h (always top-hat), and LES-FVM double filter $f^{\Delta,h}$ for $\Delta \in \{1h, 2h, 3h\}$. In the top-left plot, we have $f^\Delta = f^h$.

the LES-FVM double filter. Vreman et al. showed that the same filter width can be obtained by minimizing the L^2-error between a single top-hat kernel and the trapezoidal kernel resulting from the composition of two top-hat filters [200].

The spectral transfer functions $\hat{G}^\Delta$, $\hat{G}^h$, and $\hat{G}^{\Delta,h}$ are shown in fig. 4.3 as a function of the wavenumber $k \in \mathbb{Z}$. As the ratio Δ/h grows, the transfer functions $\hat{G}^\Delta(\kappa)$ and $\hat{G}^{\Delta,h}(\kappa) = \hat{G}^\Delta(\kappa)\hat{G}^h(\kappa)$ become more and more similar. Their respective filter widths (represented by vertical dashed lines) also get closer. For $\Delta = h$, $\hat{G}^{\Delta,h}(\kappa)$ is quite different from $\hat{G}^\Delta(\kappa)$ and $\hat{G}^h(\kappa)$, and dampens at much lower wavenumbers.

4.3.4 *Interpretation in terms of existing residual flux expressions*

The residual LES-FVM flux $\tau^{\Delta,h}$ can be decomposed in two ways in order to relate it to the classical residual LES flux τ^Δ. The first decomposition is

$$\tau^{\Delta,h}(u) = \tau^\Delta(u) + \tau^h(\bar{u}^\Delta), \tag{4.44}$$

meaning that the FVM step induces an additional residual flux $\tau^h(\bar{u}^\Delta)$ in addition to the residual LES flux $\tau^\Delta(u)$. The second decomposition is obtained

by writing $\tau^{\Delta,h}$ in terms of the classical residual LES flux *for the double filter* $f^{\Delta,h}$:

$$\tau^{\Delta,h}(u) = \tau^{\Delta,h}_{\text{classic}}(u) + \tau^{\Delta,h}_{\text{flux}}(u) + \tau^{\Delta,h}_{\text{div}}(u), \tag{4.45}$$

where

$$\tau^{\Delta,h}_{\text{classic}}(u) \coloneqq \overline{\overline{r(u)}}^{\Delta,h} - r(\bar{u}^{\Delta,h}), \tag{4.46}$$

$$\tau^{\Delta,h}_{\text{flux}}(u) \coloneqq r(\bar{u}^{\Delta,h}) - r^h(\bar{u}^{\Delta,h}), \tag{4.47}$$

$$\tau^{\Delta,h}_{\text{div}}(u) \coloneqq \overline{\overline{r(u)}}^{\Delta} - \overline{\overline{r(u)}}^{\Delta,h} \tag{4.48}$$

are the classical residual LES flux for the double filter $f^{\Delta,h}$, a discretization error from the numerical flux $r^h \approx r$, and a discretization error from the discrete divergence $\partial^h_x \approx \partial_x$, respectively. Due to the filter-swap property, $\tau^{\Delta,h}_{\text{div}}(u)$ can be written as a *high-pass grid filter* $1 - f^h$ applied to the LES flux $\overline{\overline{r(u)}}^{\Delta}$. A similar observation was made by Geurts and van der Bos [63]. Our residual flux $\tau^{\Delta,h}(u)$ still has two important differences from the expression of Geurts and van der Bos. First, our residual flux contains the numerical flux r^h in addition to the exact flux r. Second, we define the residual flux with respect to the *discrete* divergence ∂^h_x (through the filter-swap property), whereas Geurts and van der Bos defined their residual flux with respect to the *infinitesimal* divergence ∂_x (through the *reverse* filter-swap property).

For the FVM, Winckelmans [205], Denaro [49, 50] and Verstappen [193] derived a similar expression for the residual flux as τ^h in eq. (4.25), where only one of the two terms involves the filter. In our notation, their residual flux resembles $r(u) - r(\bar{u}^h)$, with r instead of r^h in the resolved term. Our expression τ^h is different in the sense that it also accounts for the numerical flux r^h. For the Burgers flux (4.15), r^h includes the interpolation and finite difference operators η^h_x and ∂^h_x. Verstappen instead considered the interpolation η^h_x explicitly, as a second filter. He analyzed the filtered equations in terms of the "1h-filter" f^h and the "2h-filter" $\eta^h_x f^h$ (their filter widths are h and $2h$ respectively). This led to an orthogonal decomposition of the energy spectrum into three parts: a resolved part, a sub-2h-filter part, and a sub-1h-filter part.

4.3.5 *Can we discretize without the FVM filter?*

We can write discrete equations for $\bar{u}^\Delta$ with a resolved and unresolved part without applying the FVM filter f^h. For example, the LES equation $\overline{\overline{L(u)}}^{\Delta} = 0$ can be rewritten as

$$L^h(\bar{u}^\Delta) = -(\partial_x \overline{\overline{r(u)}}^{\Delta} - \partial^h_x r^h(\bar{u}^\Delta)). \tag{4.49}$$

The left-hand side is grid-compatible, and if the right-hand side is approximated by a grid-compatible closure model we do indeed get a discrete approximation of $\bar{u}^\Delta$ without applying the FVM filter. However, the residual in the right-hand side cannot be written as a discrete flux term (unless f^Δ is the FVM filter f^h), so this equation is not ∂_x^h-conservative. To obtain a discrete conservation law, the FVM filter must be applied.

4.3.6 *Closure modeling implications*

In LES, it is common to model the residual flux $\tau^\Delta(u)$ with artificial dissipation, as in eddy viscosity models such as the Smagorinsky model (4.10). For the FVM, it is common to incorporate artificial numerical dissipation directly into the numerical flux r^h [46], notably to prevent oscillations around sharp gradients and shocks [81]. When used to model subgrid scale effects, this approach is called *implicit* LES. Dissipation can also be explicitly added through a dissipative flux $m^h(\bar{u}^h) \approx \tau^h(u)$ on top of a DNS-like flux $r^h(\bar{u}^h)$ that has low numerical dissipation.

In our combined LES-FVM framework, the closure $m^{\Delta,h}$ should account for both LES and FVM effects. The advantage of including the numerical flux r^h in the definition of $\tau^{\Delta,h}$ is that the implicit dissipation produced by r^h is automatically accounted for in $\tau^{\Delta,h}$. This makes it possible to better determine how much additional dissipation is required from the explicit closure $m^{\Delta,h}(\bar{u}^{\Delta,h}) \approx \tau^{\Delta,h}(u)$ [57]. If $m^{\Delta,h}$ is a Smagorinsky model, then the Smagorinsky constant can be tuned so that $m^{\Delta,h}(\bar{u}^{\Delta,h})$ and $\tau^{\Delta,h}(u)$ have similar dissipation profiles for a few reference snapshots u obtained from DNS. If r^h is a low-dissipation flux, such as the central difference based Burgers flux (4.15), the resulting Smagorinsky constant would be higher. If r^h already contains implicit numerical dissipation (as is common for upwinding fluxes), then the resulting Smagorinsky constant would be lower, since less additional explicit dissipation would be required.

The LES-FVM RST bridges the gap between implicit and explicit LES.

In the classical LES-FVM approach "approximate first, then discretize", the LES closure m^Δ is in principle made without knowledge of the discretization effects. As a remedy, information about the discretization is loosely reinjected into the definition of m^Δ by relating Δ to the grid size h.

Next, we turn to the Burgers equation to illustrate the consequences of the new LES-FVM framework. In section 4.4.2, we investigate how the definition of the residual flux affects the LES-FVM equation. In section 4.4.3, we investigate how the choice of "approximate first" vs "discretize first" leads to different closure models in practice.

4.4 EXPERIMENT: LES-FVM FOR BURGERS' EQUATION

The code is available at https://zenodo.org/records/16267799. The results were obtained using a desktop computer.

We first consider the one-dimensional viscous Burgers' equation defined by the flux (4.2). This equation can be seen as a simplified version of the Navier-Stokes equations, without the pressure term. In the inviscid limit, the solution forms shocks, which can cause oscillations with a coarse grid discretization with a low-dissipation numerical flux such as the one in eq. (4.15) [81]. These issues also persist in the viscous case if the grid is too coarse, which is why discretization-aware closure models are needed.

To assess the correctness of the LES-FVM framework, we require reference solutions to Burgers' equation. We use DNS to approximate the reference solution u and all derived quantities (such as $\bar{u}^{\Delta,h}$ and $\tau^{\Delta,h}(u)$) on a DNS grid of sufficiently small grid spacing $h_{\mathrm{DNS}} \ll h$. In B.2, we show how the quantities in our LES-FVM framework can be computed from DNS data in a consistent manner (such that the filter-swap property stills holds). This step involves discretizing the two infinitesimal filters f^{Δ} and f^{h} on the DNS grid. It also adds the constraint that the refinement factor h/h_{DNS} must be odd. Since the notation required to keep track of quantities related to both the fine h_{DNS}-grid and coarse h-grid becomes quite verbose, we will keep the infinitesimal notation for the reference solution u in the following, even though it is approximated on the h_{DNS}-grid in the code. Note also that h here refers to the coarse LES-FVM grid spacing, while in B.2, h refers to the fine DNS grid spacing.

4.4.1 *Simulation setup*

We use the following unitless parameters. The domain size is $\ell := 2\pi$. The viscosity is $\nu := 5 \times 10^{-4}$. The numerical flux is the central difference based flux in eq. (4.15). The grid spacings are $h := \ell/N$ and $h_{\mathrm{DNS}} := \ell/N_{\mathrm{DNS}}$. We consider one DNS grid size $N_{\mathrm{DNS}} := 13500$ and multiple LES grid sizes $N \in \{300, 900, 2700\}$. The compression factors are $N_{\mathrm{DNS}}/N \in \{45, 15, 5\}$ respectively. These resolutions give odd refinement factors for multiple LES grid sizes at a fixed DNS grid size. The filter-swap compatibility between the DNS and LES grids requires N_{DNS}/N to be an odd integer (see B.2). The LES filter kernel is the Gaussian (4.38). Unless otherwise stated, the LES filter width is $\Delta = 2h$.

The initial conditions for u are prescribed through the Fourier coefficients

$$\hat{a}(k) := \left(\frac{k}{k_0}\right)^2 \exp\left(-\left(\frac{k}{k_0}\right)^2\right) \exp\left(2\pi \mathrm{i} \epsilon_k\right), \tag{4.50}$$

where $\widehat{(\cdot)}$ is the Fourier transform, $k \in \mathbb{Z}$ is the wavenumber, i is the imaginary unit, $k_0 = 10$ is the peak wavenumber, and $\epsilon_k \sim \mathcal{U}(0, 1)$ is a uniformly

sampled random number between 0 and 1 for $k \geq 0$ and $\epsilon_k = -\epsilon_{-k}$ otherwise. The velocity field is then normalized as

$$u \coloneqq \sqrt{2E_0}a/\|a\|_\Omega, \tag{4.51}$$

where $E_0 \coloneqq 2$ is the chosen initial total energy and $\|a\|_\Omega^2 \coloneqq \int_\Omega |a|^2 \,\mathrm{d}x$. This gives the initial energy spectrum profile $|\hat{u}(k)|^2 \propto k^4 \mathrm{e}^{-2(k/k_0)^2}$ which is commonly used for decaying turbulence problems [118, 153, 193]. In total, we generate 1000 different initial conditions $u \in \mathcal{U}_{\text{initial}}$.

4.4.2 *DNS-aided LES-FVM*

To evaluate the correctness of different residual fluxes, we employ the "DNS-aided LES(-FVM)"-framework of Bae and Lozano-Duran [6, 7]. It consists of running a DNS alongside an LES-FVM, where the DNS solution is used to compute the closure term used in the LES-FVM equation. Ideally, by using the DNS solution to compute the right-hand side, one would expect to be able to recover the filtered DNS solution with the DNS-aided LES-FVM. We here solve the coupled DNS/LES-FVM equations

$$L(u) = 0, \quad L^h(v^{\Delta,h}) = -\partial_x^h \tau^{\Delta,h}_{\text{model}}(u), \tag{4.52}$$

where $\tau^{\Delta,h}_{\text{model}}(u)$ is a "closure" model computed from the DNS solution u. This term does not depend on the LES-FVM approximation $v^{\Delta,h}$. The LES-FVM approximation is initialized with $v^{\Delta,h} = \bar{u}^{\Delta,h}$.

We test four different expressions for $\tau^{\Delta,h}_{\text{model}}$:

$$\tau^{\Delta,h}_{\text{no-model}}(u) \coloneqq 0, \tag{4.53}$$

$$\tau^{\Delta,h}_{\text{classic}}(u) \coloneqq \overline{r(u)}^{\Delta,h} - r(\bar{u}^{\Delta,h}), \tag{4.54}$$

$$\tau^{\Delta,h}_{\text{classic+flux}}(u) \coloneqq \overline{r(u)}^{\Delta,h} - r^h(\bar{u}^{\Delta,h}), \tag{4.55}$$

$$\tau^{\Delta,h}_{\text{classic+flux+div}}(u) \coloneqq \overline{r(u)}^{\Delta} - r^h(\bar{u}^{\Delta,h}). \tag{4.56}$$

From eq. (4.35) and the decomposition (4.45), we know that

$$\tau^{\Delta,h}_{\text{classic+flux+div}} = \tau^{\Delta,h} = \tau^{\Delta,h}_{\text{classic}} + \tau^{\Delta,h}_{\text{flux}} + \tau^{\Delta,h}_{\text{div}} \tag{4.57}$$

is the correct expression. The two other non-zero models are obtained by neglecting the discretization errors: $\tau^{\Delta,h}_{\text{classic}}$ neglects both the divergence error $\tau^{\Delta,h}_{\text{div}}$ and the numerical flux error $\tau^{\Delta,h}_{\text{flux}}$, while $\tau^{\Delta,h}_{\text{classic+flux}} = \tau^{\Delta,h}_{\text{classic}} + \tau^{\Delta,h}_{\text{flux}}$ only neglects the divergence error $\tau^{\Delta,h}_{\text{div}}$.

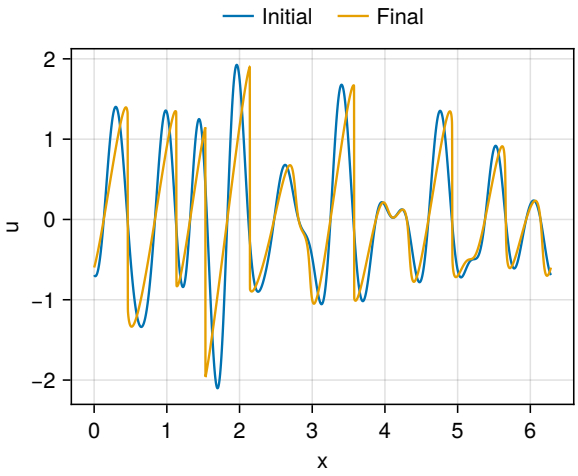


Figure 4.4: Initial and final solution to the Burgers equation.

Both u and $v^{\Delta,h}$ are advanced forward in time using the forward-Euler scheme

$$u_{k+1} \coloneqq u_k - \Delta t_k \partial_x r(u_k) \tag{4.58}$$

$$v^{\Delta,h}_{k+1} \coloneqq v^{\Delta,h}_k - \Delta t_k \partial^h_x \left(r^h(v^{\Delta,h}_k) + \tau^{\Delta,h}_{\text{model}}(u_k) \right) \tag{4.59}$$

where u_k and $v^{\Delta,h}_k$ denote the forward-Euler approximations to the DNS and LES-FVM solutions at time $t_k \coloneqq \sum_{i=0}^{k-1} \Delta t_i$. The time step

$$\Delta t_k \coloneqq C \times \min\left(\frac{h_{\text{DNS}}}{\max_{x\in\Omega} |u_k(x)|}, \frac{h^2_{\text{DNS}}}{\nu} \right) \tag{4.60}$$

is chosen based on the CFL condition for u_k with $C = 0.4$. Both equations use the same time step. While the time-stepping scheme is only first-order accurate, it has the advantage of requiring only one evaluation of the right-hand side per time step. This simplifies the injection procedure, where the residual flux from DNS is injected into the LES-FVM equation. Injecting the DNS solution into the LES-FVM equation would be more involved for higher-order schemes with multiple stages per time step.

In fig. 4.4, we show the DNS solution for one initial condition at initial and final time ($t = 0.1$). The solution is slightly damped due to dissipation, and some shock-like structures are forming. These sharp gradients cannot be properly resolved on the coarse h-grid and cause oscillations with the central difference ∂^h_x, which is why a discretization-informed closure model is needed.

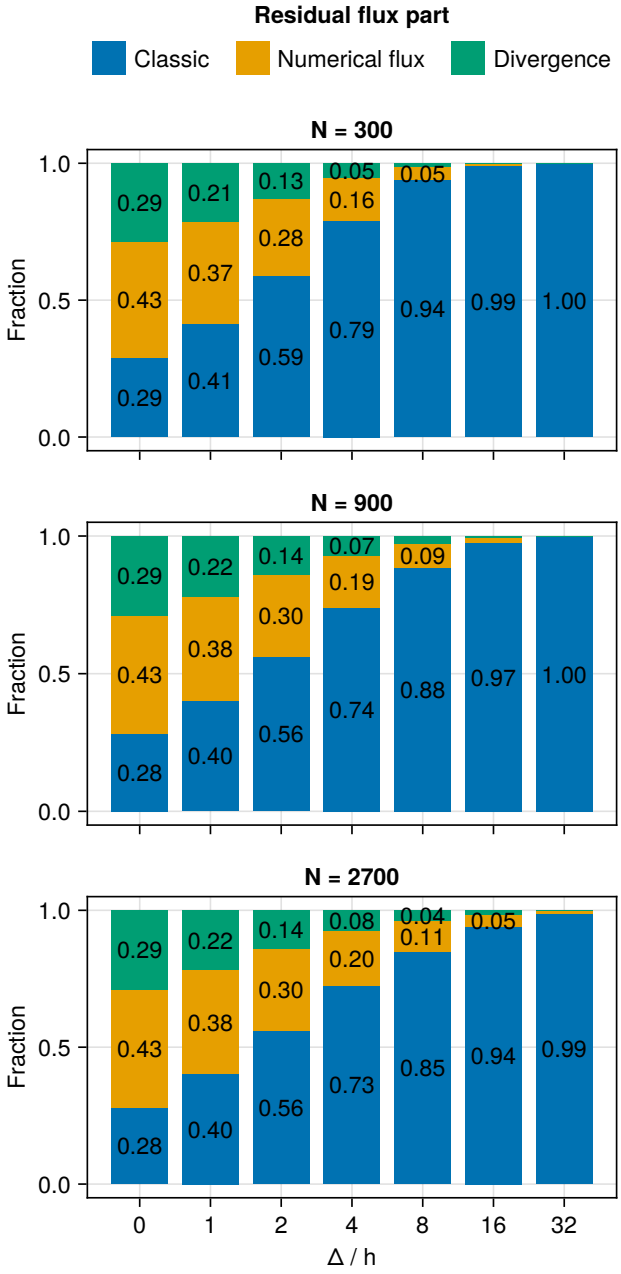


Figure 4.5: Relative contribution of the different flux parts in the decomposition (4.45).

In fig. 4.5, we show the average fractions

$$\frac{1}{1000}\sum_{u\in\mathcal{U}_{\text{final}}}\frac{\left|\tau^{\Delta,h}_{\text{label}}(u)\right|}{\left|\tau^{\Delta,h}_{\text{classic}}(u)\right|+\left|\tau^{\Delta,h}_{\text{flux}}(u)\right|+\left|\tau^{\Delta,h}_{\text{div}}(u)\right|} \tag{4.61}$$

for label $\in \{\text{classic}, \text{flux}, \text{div}\}$, where $\mathcal{U}_{\text{final}}$ contains the DNS solutions at the final time. For $\Delta = 0$, the classic flux only accounts for about 28% of the total residual flux. For $\Delta = 4h$, the classic flux accounts for about $73-79\%$ of the total. For $\Delta = 32h$, less than 1% of the residual is due to the discretization errors $\tau^{\Delta,h}_{\text{flux}}$ and $\tau^{\Delta,h}_{\text{div}}$. For explicit LES, at $\Delta \geq 32h$, we can therefore safely neglect the discretization errors in the residual flux. For $\Delta \leq 4h$, the discretization errors are significant and should not be neglected. This corresponds with the observation of Chow and Moin that $\Delta \geq 4h$ is necessary for the second order discretization errors to no longer dominate the residual flux [38]. For $4h < \Delta < 32h$, the discretization errors may or may not be neglected depending on what accuracy is required. For an actual closure model, which is going to have a significant modeling error anyway, the 21% contribution of the discretization to the residual flux at $(\Delta, N) = (4h, 300)$ may not be important. But if the residual flux is to be used as a target for tuning the closure model coefficients, including the 6% contribution of the discretization at $(\Delta, N) = (8h, 300)$ could still be useful.

Figure 4.5 only shows the importance of the discretization errors in the a-priori setting. In fig. 4.6, we show the average a-posteriori relative errors

$$e_{\text{model}} := \frac{1}{1000}\sum_{u_0\in\mathcal{U}_{\text{initial}}}\frac{\|v^{\Delta,h}_{\text{model}}-\bar{u}^{\Delta,h}\|_\Omega}{\|\bar{u}^{\Delta,h}\|_\Omega} \tag{4.62}$$

at the final time for different values of N and Δ. For all N and Δ, $e_{\text{classic+flux+div}}$ is at machine precision. For $(N, \Delta) = (300, 0h)$, we have $e_{\text{no-model}} \approx 60\%$, $e_{\text{classic}} \approx 16\%$, and $e_{\text{classic+flux}} \approx 8\%$. For $(N, \Delta) = (300, 4h)$, we have $e_{\text{no-model}} \approx 65\%$, $e_{\text{classic}} \approx 7\%$, and $e_{\text{classic+flux}} \approx 2\%$. For $(N, \Delta) = (300, 32h)$, we have $e_{\text{no-model}} \approx 35\%$, $e_{\text{classic}} \approx 0.1\%$, and $e_{\text{classic+flux}} \approx 0.03\%$. This confirms that the discretization errors are significant for $\Delta \leq 4h$, and insignificant for $\Delta \geq 32h$.

Further insight into the four models is obtained through the energy spectrum. We define the energy spectrum of a field u as $|\hat{u}(k)|^2/2$, where $\hat{u}(k)$ is the Fourier transform of u at wavenumber k. Figure 4.7 shows the energy spectra at the final time, averaged over the 1000 solutions. Individual DNS solutions have noisy spectra that fluctuate around the theoretical k^{-2} slope in the inertial range. The averaged DNS spectrum is smooth and follows the theoretical slope. The DNS grid spacing is small enough to resolve both the dissipation and inertial ranges.

The filtered DNS spectrum stays on top of the DNS spectrum for the low wavenumbers, and becomes damped for the highest wavenumbers.

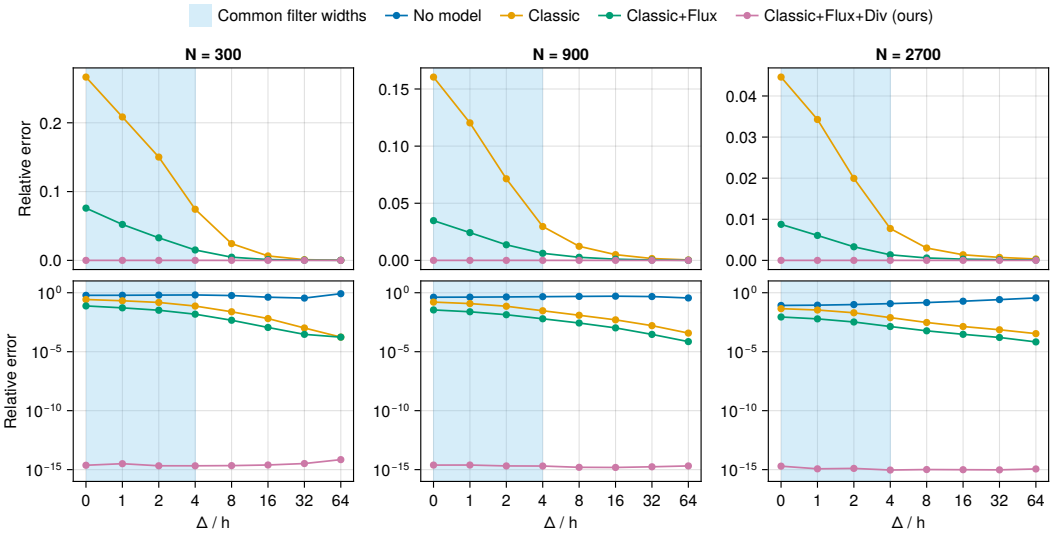


Figure 4.6: Relative errors at final time for DNS-aided LES-FVM of Burgers' equation. The shaded area indicates common ratios Δ/h in literature. Top: Linear scale (the no-model error is above the plotted region). Bottom: Logarithmic scale.

This is because the transfer function of the filter $f^{\Delta,h}$ is close to 1 for low wavenumbers, but decays for larger wavenumbers (see fig. 4.3). The filtered DNS spectra are shown beyond the cut-off wavenumbers $N/2$ of the coarse h-grids. These spectra are obtained by evaluating $\bar{u}^{\Delta,h}(x)$ *at all the DNS grid points* x, and then applying the Fourier transform on the DNS grid. Since $f^{\Delta,h}$ is not a spectral cut-off filter, the spectrum of $\bar{u}^{\Delta,h}(x_i^h)$ restricted to the coarse h-grid points x_i^h, computed with a coarse h-grid discrete Fourier transform, is slightly different from the spectrum of $\bar{u}^{\Delta,h}(x)$ computed on the DNS grid. In the more detailed zoom-in box, we therefore show the spectrum of $\bar{u}^{\Delta,h}(x_i^h)$, which is not defined beyond the cut-off wavenumber $N/2$ of the h-grid.

The LES-FVM spectra are all evaluated on the h-grid, and they are therefore not defined beyond $N/2$. For all three grid sizes, the no-closure solution has too much energy in the highest resolved wavenumbers. This is a well-known issue in LES and part of the motivation for using a dissipative closure model. The classic flux $\tau^{\Delta,h}_{\text{classic}}$ is not dissipative enough. For the first two resolutions, it even shows signs of instability in the highest wavenumbers. The improved flux $\tau^{\Delta,h}_{\text{classic+flux}}$ gives a spectrum closer to the filtered DNS than $\tau^{\Delta,h}_{\text{classic}}$, but at the highest resolved wavenumbers, it is generally *too dissipative*. It also shows a sign of instability at the highest wavenumbers.

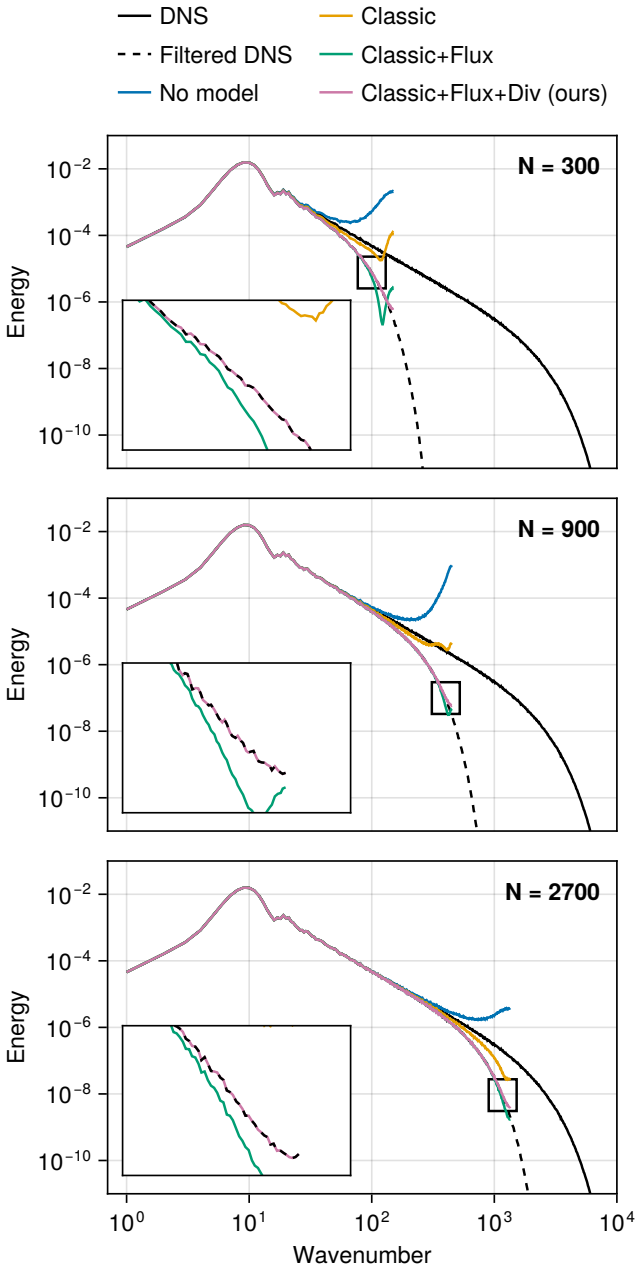


Figure 4.7: Energy spectra for the DNS-aided LES-FVM of Burgers equation. Our discretization-informed expression leads to an exact correspondence with the filtered DNS, while the classical expression is not dissipative enough.

This is likely because the DNS-aided closure term $\tau^{\Delta,h}_{\text{classic+flux}}(u)$ does not depend on the LES-FVM solution $v^{\Delta,h}_{\text{classic+flux}}$ at all. As $v^{\Delta,h}_{\text{classic+flux}}$ starts to decorrelate from $\bar{u}^{\Delta,h}$, the DNS-aided closure term becomes less accurate over time and even detrimental at the highest wavenumbers. The correct flux $\tau^{\Delta,h}_{\text{classic+flux+div}}$ gives a spectrum that perfectly overlaps with the h-grid variant of the filtered DNS spectrum (in the zoom-in box).

CONCLUSION Using the correct expression for the residual LES-FVM flux as a closure model gives *perfect* results, unlike the classical residual LES flux expression. The exact residual flux $\tau^{\Delta,h}_{\text{classic+flux+div}}(u) = \tau^{\Delta,h}(u)$ can therefore be seen as the *best case scenario* for an LES-FVM closure model $m^{\Delta,h}(\bar{u}^{\Delta,h})$. However, in general $\tau^{\Delta,h}(u)$ cannot be computed from $\bar{u}^{\Delta,h}$ alone, since information is lost in the filtering process.

Next, we turn to an actual LES-FVM closure model that does not require a concurrent DNS simulation for evaluation. The question is whether the discretization-informed expression for the residual flux can be used to produce better closure models.

4.4.3 *Smagorinsky closure*

Define the infinitesimal and discrete Smagorinsky fluxes as

$$s(u) \coloneqq \left|\partial_x u\right| \partial_x u, \quad s^h(u) \coloneqq \left|\partial_x^h u\right| \partial_x^h u. \tag{4.63}$$

For the double-filter $f^{\Delta,h}$ of width $\sqrt{\Delta^2 + h^2}$, the classical LES Smagorinsky model is defined at the infinitesimal level as

$$m^{\Delta,h}_{\text{classic}}(u;\theta) \coloneqq -\theta^2(\Delta^2 + h^2)s(u), \tag{4.64}$$

where θ is a model parameter. This classical LES model is then discretized as

$$m^{\Delta,h}_{\text{discrete}}(u;\theta) \coloneqq -\theta^2(\Delta^2 + h^2)s^h(u). \tag{4.65}$$

The coefficient θ can be estimated in various ways. For the infinitesimal basic Smagorinsky model (before discretizing the flux), a value can be derived analytically [101]. In the dynamic Smagorinsky model, the coefficient is estimated from the LES state itself using a hypothesis of scale-similarity combined with a test-filter [61]. We employ the basic Smagorinsky framework, but with a data-driven estimate of the coefficient using a least-squares fit to the residual flux.

In the classical LES framework, the infinitesimal model $m^{\Delta,h}_{\text{classic}}(\bar{u}^{\Delta,h};\theta)$ is used to approximate the residual LES flux $\tau^{\Delta,h}_{\text{classic}}(u) \coloneqq \overline{r(u)}^{\Delta,h} - r(\bar{u}^{\Delta,h})$. The discretization is then only accounted for in the filter definition. For a

set of reference solution snapshots $\mathcal{U}$ (obtained from DNS), we therefore estimate the coefficient for the classical Smagorinsky model as

$$\min_{\theta} \sum_{u \in \mathcal{U}} \left| -\theta^2(\Delta^2 + h^2) s(\bar{u}^{\Delta,h}) - \tau^{\Delta,h}_{\text{classic}}(u) \right|^2, \tag{4.66}$$

where the infinitesimal operators are approximated on the DNS grid. Similarly, in the new LES-FVM framework, a discretization-informed Smagorinsky model can be obtained by choosing θ such that $m^{\Delta,h}_{\text{discrete}}(\bar{u}^{\Delta,h}; \theta)$ matches the discretization-informed residual flux $\tau^{\Delta,h}(u) \coloneqq \overline{r(u)}^{\Delta} - r^h(\bar{u}^{\Delta,h})$ as

$$\min_{\theta} \sum_{u \in \mathcal{U}} \left| -\theta^2(\Delta^2 + h^2) s^h(\bar{u}^{\Delta,h}) - \tau^{\Delta,h}(u) \right|^2. \tag{4.67}$$

We thus get the two different least-squares estimates for θ:

$$-\theta^2_{\text{classic}}(\Delta^2 + h^2) = \frac{\sum_{u \in \mathcal{U}} \left\langle s(\bar{u}^{\Delta,h}), \tau^{\Delta,h}_{\text{classic}}(u) \right\rangle}{\sum_{u \in \mathcal{U}} \left\langle s(\bar{u}^{\Delta,h}), s(\bar{u}^{\Delta,h}) \right\rangle}, \tag{4.68}$$

$$-\theta^2_{\text{informed}}(\Delta^2 + h^2) = \frac{\sum_{u \in \mathcal{U}} \left\langle s^h(\bar{u}^{\Delta,h}), \tau^{\Delta,h}(u) \right\rangle}{\sum_{u \in \mathcal{U}} \left\langle s^h(\bar{u}^{\Delta,h}), s^h(\bar{u}^{\Delta,h}) \right\rangle}, \tag{4.69}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on Ω. Solving for θ is then straightforward. These expressions are similar to the ones in the dynamic Smagorinsky model (where the coefficient is also tuned using a least-squares estimate), but here we use DNS data u instead of LES data to compute the target residual flux.

What is the difference between θ_{classic} and θ_{informed}? In the classical LES setting, θ_{classic} is obtained *before* discretization (information about the discretization is only injected through the definition of the filter width). In our framework, θ_{informed} is obtained *after* discretization. It should therefore be slightly higher than θ_{classic} to account for the additional dissipation required by the discretization. Our hypothesis is that the classical discretized model $m^{\Delta,h}_{\text{discrete}}(\cdot; \theta_{\text{classic}})$ should perform worse than the discretization-informed model $m^{\Delta,h}_{\text{discrete}}(\cdot; \theta_{\text{informed}})$, since the choice of θ_{classic} assumes an infinitesimal setting, while θ_{informed} is obtained in the discrete setting, using the correct discrete residual flux as a target.

In fig. 4.8, we show the Smagorinsky coefficients estimated from DNS data for Gaussian f^{Δ} and $\Delta \in \{0h, 2h\}$. Indeed, θ_{discrete} is larger than θ_{classic} for all grid sizes. The two coefficients are more different for $\Delta = 0h$ than for $\Delta = 2h$, likely because $\tau^{\Delta,h}$ converges to $\tau^{\Delta,h}_{\text{classic}}$ when the ratio Δ/h increases (see fig. 4.5). For even larger Δ, for example $\Delta = 32h$, the two coefficients would likely be very similar.

Since all three grid sizes lie within the inertial range of the energy spectrum, we could also choose to fit one single Smagorinsky coefficient for all

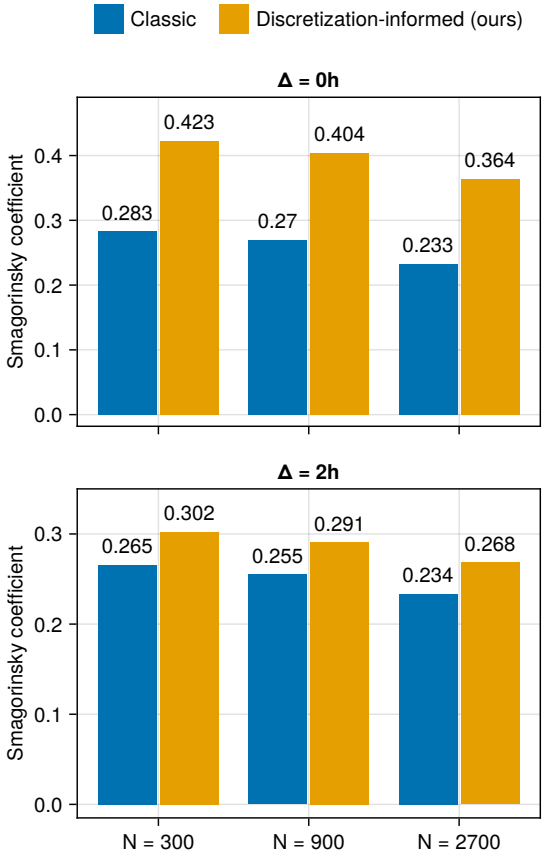


Figure 4.8: Fitted Smagorinsky coefficients for Burgers' equation.

grid sizes. Currently, we fit separate coefficients for each grid size N and filter ratio Δ/h, resulting in larger coefficients for the smaller grid sizes and filter ratios. This is likely because the sharp gradients are less resolved, and therefore more dissipation is needed.

In fig. 4.9, we show the relative errors at the final time between the Smagorinsky LES-FVM approximation and the exact LES-FVM solution. Our discretization-informed Smagorinsky model has lower errors than the classical Smagorinsky model. The difference is significant for $\Delta = 0h$, but smaller for $\Delta = 2h$. For even larger filter ratios, the difference would likely vanish.

Even a simple model like Smagorinsky benefits from the discretization-informed RST target.

The difference is further visible in the energy spectra shown in fig. 4.10. We see that the classical Smagorinsky model is not dissipative enough with a larger spectrum than the filtered DNS. The discretization-informed

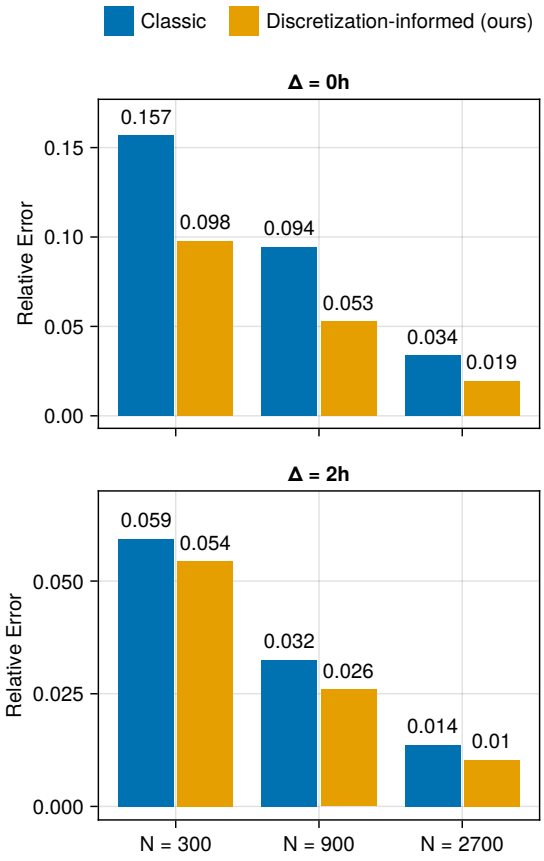


Figure 4.9: Relative errors at final time for LES-FVM of Burgers' equation with the Smagorinsky model.

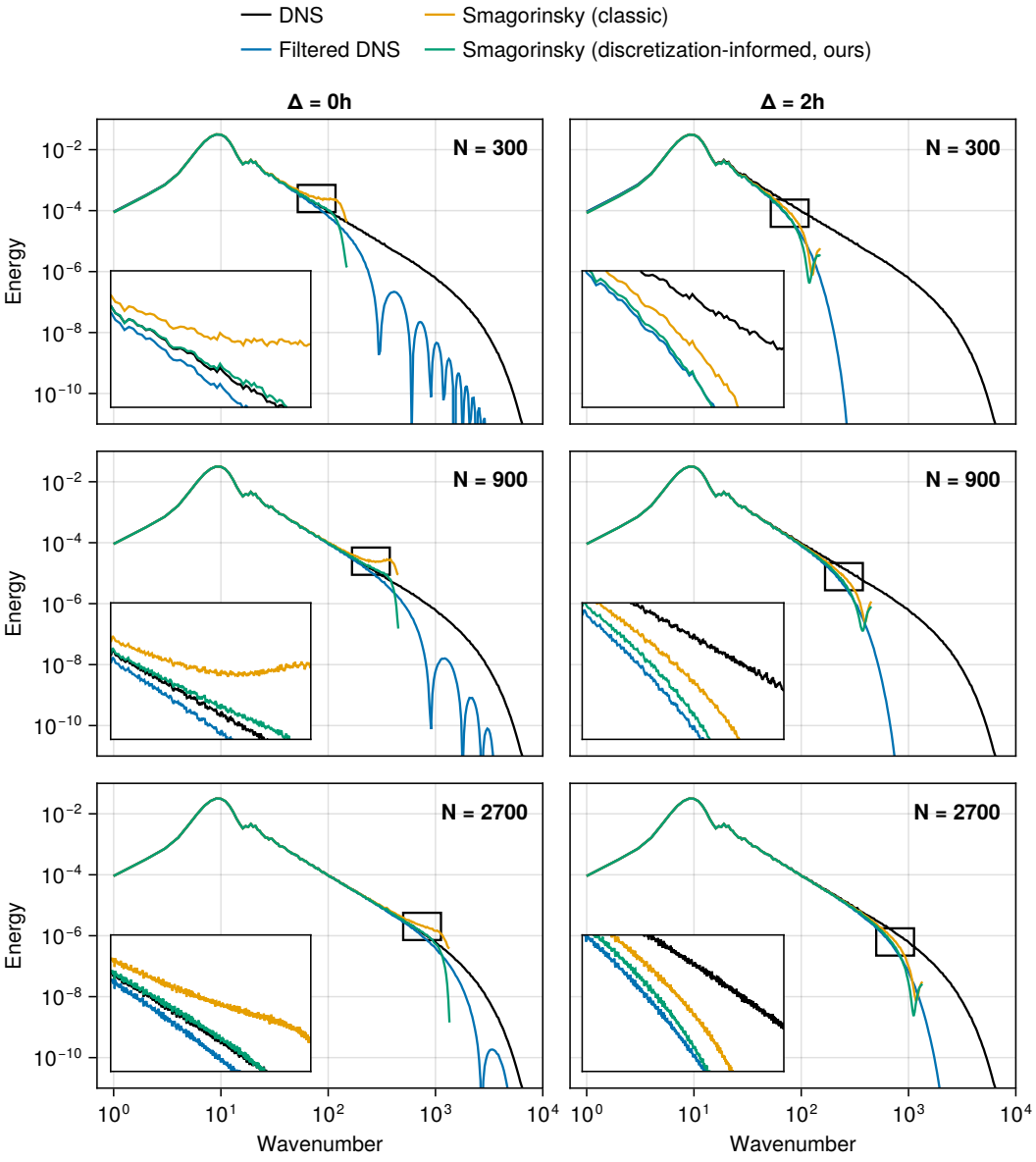


Figure 4.10: Energy spectra for the Burgers equation solved with the Smagorinsky model.

Smagorinsky model has a spectrum closer to the filtered DNS. For $\Delta = 2h$, both models have a peak at the highest wavenumbers. This is because the central difference scheme used in r^h gives rise to oscillations around the sharp gradients, and the dissipative Smagorinsky model is not able to fully remove them without causing over-dissipation across all wavenumbers. The Smagorinsky coefficient obtained from the least-squares fit is a compromise between removing these oscillations and preserving the rest of the spectrum.

We stress that the gain in accuracy obtained by tuning the Smagorinsky model with the correct residual flux instead of the classical one is limited by the eddy-viscosity hypothesis. We expect that with more expressive closure models, such as machine-learning-based ones, much lower errors can be achieved, and the benefits of our new residual flux expression will be more pronounced.

In conclusion, we have shown that for common filter ratios, the discretization-informed residual flux for the 1D Burgers equation both leads to zero error in a DNS-aided LES and improves the performance of an actual closure model when used as a target for fitting the model parameters. Next, we turn to the 3D incompressible Navier-Stokes equations. As we show, additional complexities must be addressed to obtain a discretization-informed expression for the residual.

4.5 LES-FVM FOR THE 3D INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

The incompressible Navier-Stokes equations in index-form are given by

$$\partial_j u_j = 0, \quad \partial_t u_i + \partial_j \left(\sigma_{ij}(u) + p\delta_{ij}\right) = 0, \tag{4.70}$$

where

$$\sigma_{ij}(u) \coloneqq u_i u_j - \nu \left(\partial_j u_i + \partial_i u_j\right) \tag{4.71}$$

is the convective-diffusive stress tensor, δ_{ij} is the identity tensor (Kronecker delta-symbol), $(i, j) \in \{1, 2, 3\}^2$ are indices, $\partial_t \coloneqq \partial/\partial t$ and $\partial_i \coloneqq \partial/\partial x_i$ are partial derivatives, $x \coloneqq (x_1, x_2, x_3)$ is the position, t is the time, $u_i(x, t)$ is the velocity in direction i, $p(x, t)$ is the pressure, and $\nu > 0$ is the kinematic viscosity. Repeated indices imply summation (Einstein notation). Since our derivations involve mainly spatial derivatives, we will write $u_i(x)$ instead of $u_i(x, t)$ to ease the notation. For simplicity, we assume the equations are defined in a periodic box $\Omega = [0, \ell]^3$ of side length $\ell > 0$.

The space of periodic scalar-valued fields on Ω is denoted U. Although the entries of the tensor $\sigma(u) \in U^{3\times 3}$ are scalar fields $\sigma_{ij}(u) \in U$, we still use the word "tensor" to describe both $\sigma(u)$ and $\sigma_{ij}(u)$ interchangeably (and similarly for other vector and tensor fields).

4.5.1 *Pressure-free Navier-Stokes equations*

The system (4.70) does not have the form of a conservation law for the state $(u, p) \in U^4$. To repeat the procedure from the previous sections (1D), we first need to rewrite the Navier-Stokes equations as a self-contained conservation-law. To achieve this, we will eliminate the pressure p from the momentum equations.

The pressure p is a Lagrange multiplier that enforces the continuity equation for u. A related viewpoint is that $p\delta_{ij}$ is a correction that makes the non-divergence-preserving stress tensor $\sigma_{ij}(u)$ divergence-preserving. We say that a stress tensor $\sigma \in U^{3\times 3}$ is divergence-preserving if $\partial_i\partial_j\sigma_{ij} = 0$, i.e. the force $\partial_j\sigma_{ij}$ is divergence-free.

In B.4, we introduce the two pressure projection operators

$$\pi_{ij} \coloneqq \delta_{ij} - \partial_i \left(\partial_k\partial_k\right)^\dagger \partial_j \tag{4.72}$$

and

$$\pi_{ij\alpha\beta} \coloneqq \delta_{i\alpha}\delta_{j\beta} - \delta_{ij} \left(\partial_k\partial_k\right)^\dagger \partial_\alpha\partial_\beta, \tag{4.73}$$

where $(\partial_k\partial_k)^\dagger$ is the inverse Laplacian operator subject to the constraint of an average pressure of zero. The vector-projector π_{ij} can be used to make vector fields divergence-free (since $\partial_i\pi_{ij} = 0$), while the tensor-projector $\pi_{ij\alpha\beta}$ can be used to make tensor fields divergence-*preserving* (since $\partial_i\partial_j\pi_{ij\alpha\beta} = 0$). The proofs are given in theorem 6 and theorem 7.

Since u is divergence-free, we can rewrite the momentum equation in a pressure-free way using either of the two projectors as $\partial_t u_i + \pi_{ij}\partial_k\sigma_{jk}(u) = 0$ or $\partial_t u_i + \partial_j\pi_{ij\alpha\beta}\sigma_{\alpha\beta}(u) = 0$. We will use the latter form, since it has the form of a conservation law (divergence of a tensor). The pressure-free momentum equations can thus be written as

$$L_i(u) \coloneqq \partial_t u_i + \partial_j r_{ij}(u) = 0, \tag{4.74}$$

where

$$r_{ij}(u) \coloneqq \pi_{ij\alpha\beta}\sigma_{\alpha\beta}(u) = \sigma_{ij}(u) + p\delta_{ij} \tag{4.75}$$

is the projected stress tensor and $L_i \coloneqq \partial_t + \partial_j r_{ij}$ is the pressure-free momentum equation operator. Given $\sigma(u)$, the projector π computes the unique pressure p (up to a constant) such that $\sigma(u) + p\delta$ is divergence-preserving. The pressure projection only modifies the diagonal, so $r_{ij} = \sigma_{ij}$ for $i \neq j$.

In eq. (4.70), there is a spatial constraint of divergence-freeness, which was not present in section 4.2. In the projected form (4.74), this constraint is hidden inside $r_{ij}(u)$. As a result, the 3D stress tensor $r_{ij}(u)$ is non-local in u, and requires solving a Poisson equation, whereas the 1D Burgers flux $r(u)$ was local. As long as the initial velocity field is divergence-free, the

Eliminating the pressure yields a self-contained conservation law, enabling the 1D LES-FVM procedure to be reused in 3D.

continuity equation can be ignored since eq. (4.74) evolves the velocity field in a divergence-preserving way.

Since we incorporated the divergence-free constraint, the 3D conservation law (4.74) has the same form as the 1D conservation law (4.1). We can therefore repeat the procedure from section 4.3 to obtain LES-FVM equations in conservative form. We use the same notation as in sections 4.2 to 4.4 to highlight the similarities and differences. One difference is the presence of direction indices i and j. The scalar flux $r(u)$ from section 4.2 is now a 3×3 stress tensor $r_{ij}(u)$.

4.5.2 *Classical LES*

Consider a convolutional homogeneous spatial filter $f^\Delta : u \mapsto \bar{u}^\Delta$ defined for all scalar fields $u \in U$ as

$$\bar{u}^\Delta(x) \coloneqq \int_{\mathbb{R}^3} G^\Delta(x-y)u(y)\,\mathrm{d}y \tag{4.76}$$

for some kernel G^Δ. We integrate over $\mathbb{R}^3$ instead of Ω to allow for periodic extension. As in 1D, this filter commutes with differentiation:

$$f^\Delta \partial_i = \partial_i f^\Delta. \tag{4.77}$$

The filtered Navier-Stokes equations therefore take the conservative form

$$\partial_j \bar{u}_j^\Delta = 0, \quad \partial_t \bar{u}_i^\Delta + \partial_j \left(\sigma_{ij}(\bar{u}^\Delta) + \xi_{ij}^\Delta(u) + \bar{p}^\Delta \delta_{ij}\right) = 0, \tag{4.78}$$

where $\bar{u}_i^\Delta$ and $\bar{p}^\Delta$ are filtered fields and

$$\xi_{ij}^\Delta(u) \coloneqq \overline{u_i u_j}^\Delta - \bar{u}_i^\Delta \bar{u}_j^\Delta \tag{4.79}$$

is the classical RST. We reserve the symbol τ^Δ for the RST $\tau^\Delta(u) \coloneqq \pi\xi^\Delta(u) = \overline{r(u)}^\Delta - r(\bar{u}^\Delta)$, which contains the pressure projector π. Since filtering commutes with differentiation, we also have $f^\Delta \pi = \pi f^\Delta$.

For classical structural LES models, the unprojected tensor $\xi_{ij}^\Delta(u)$ is replaced by a closure model $m_{ij}^\Delta(\bar{u}^\Delta)$ that only depends on $\bar{u}^\Delta$. Since ξ_{ij}^Δ is a symmetric tensor ($\xi_{ij}^\Delta = \xi_{ji}^\Delta$), the closure model is designed to be symmetric as well ($m_{ij}^\Delta = m_{ji}^\Delta$). If the filter f^Δ is local in space, then ξ^Δ is also local, and m^Δ can be chosen to be a local closure. To solve the LES equations, the closed equations are discretized. Since m^Δ is symmetric, its discretized variant is also symmetric.

4.5.3 *Classical FVM*

Using the staggered spatial discretization scheme of Harlow and Welch [71], we define the DNS equations as

$$\partial_j^h u_j^h = 0, \quad \partial_t u_i^h + \partial_j^h \left(\sigma_{ij}^h \left(u^h \right) + p^h \delta_{ij} \right) = 0. \tag{4.80}$$

Here, $u^h \in U^3$ and $p^h \in U$ are the DNS velocity and pressure fields,

$$\sigma_{ij}^h(u) \coloneqq \left(\eta_j^h u_i \right) \left(\eta_i^h u_j \right) - \nu \left(\partial_j^h u_i + \partial_i^h u_j \right) \tag{4.81}$$

is a grid-compatible numerical stress tensor analogous to the 1D numerical Burgers flux r^h from eq. (4.15), $\partial_i^h : U \to U$ and $\eta_i^h : U \to U$ are finite difference and interpolation operators defined for all $u \in U$ as

$$\partial_i^h u(x) \coloneqq \frac{1}{h} \left[u\left(x + \frac{h}{2} e_i \right) - u\left(x - \frac{h}{2} e_i \right) \right], \tag{4.82}$$

$$\eta_i^h u(x) \coloneqq \frac{1}{2} \left[u\left(x - \frac{h}{2} e_i \right) + u\left(x + \frac{h}{2} e_i \right) \right], \tag{4.83}$$

where h is a uniform grid spacing and $(e_i)_{i=1}^3$ are the unit vectors. These operators are second-order accurate in h, and, as a result, $u^h(x,t)$ is a second-order accurate approximation of $u(x,t)$ for all x and t if $u^h(x,0) = u(x,0)$.

We use the projected form of the DNS equations (4.80):

$$L_i^h(u^h) \coloneqq \partial_t u_i^h + \partial_j^h r_{ij}^h(u^h) = 0, \tag{4.84}$$

where L_i^h is the pressure-free finite volume momentum equation operator,

$$r_{ij}^h \coloneqq \pi_{ij\alpha\beta}^h \sigma_{\alpha\beta}^h \tag{4.85}$$

is the projected discrete stress tensor, and

$$\pi_{ij\alpha\beta}^h \coloneqq \delta_{i\alpha}\delta_{j\beta} - \delta_{ij} \left(\partial_k^h \partial_k^h \right)^\dagger \partial_\alpha^h \partial_\beta^h \tag{4.86}$$

is a discrete version of $\pi_{ij\alpha\beta}$ that makes stress tensors discretely divergence-preserving (i.e. $\partial_i^h \partial_j^h \pi_{ij\alpha\beta}^h = 0$). Given the stress $\sigma_{ij}^h(u^h)$, the projector $\pi_{ij\alpha\beta}^h$ computes the isotropic pressure correction $p^h \delta_{ij}$ that makes $r_{ij}^h(u^h)$ divergence-preserving, so we get the identity $r_{ij}^h(u^h) = \sigma_{ij}^h(u^h) + p^h \delta_{ij}$.

We divide the domain $\Omega = [0, \ell]^3$ into $N \coloneqq \ell / h$ reference volumes in each dimension (N^3 volumes in total). When restricting the FVM solution, we use a staggered representation as depicted in fig. 4.11. The pressure p^h is restricted to the volume centers (pressure points). The velocity components u_i^h are restricted to the centers of the volume faces orthogonal to e_i (velocity points). The positioning of the tensor components σ_{ij}^h follows naturally. They

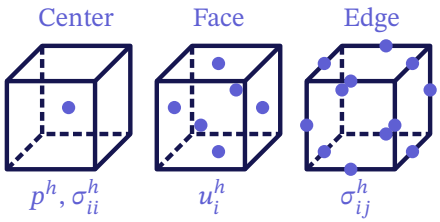


Figure 4.11: Staggered positions in a reference volume of a scalar p^h, vector u^h, and tensor σ^h. The diagonal tensor components are in the volume center, while the off-diagonal tensor components are on the volume edges.

are in the pressure points if $i = j$, and in the centers of the volume edges otherwise. The continuity equation is evaluated in the pressure points, and the momentum equations in the velocity points.

Note that while we still use the continuous notation $u^h(x,t)$, the degrees of freedom depicted in fig. 4.11 contain all the information needed to evaluate the DNS equations in the required points. This is because σ^h_{ij} is chosen such that the restricted DNS equations are closed in the discrete sense.

We use the divergence-form for the convective term, which is energy-conservative if $\partial^h_j u^h_j = 0$ [129]. The continuity equation $\partial^h_j u^h_j = 0$ is therefore enforced strictly by using semi-explicit time discretization schemes (applying a pressure projection *after* each momentum time step) [68, 154].

4.5.4 *Grid filters and the filter-swap property in 3D*

To obtain LES-FVM equations in conservative form, we need grid filters that satisfy a filter-swap property. As in the 1D case, the finite difference operator ∂^h_i is associated to a one-dimensional top-hat filter $g^h_i : U \to U$ that is only applied in the direction e_i. For all $u \in U$, we define it as

$$g^h_i u(x) := \frac{1}{h} \int_{-h/2}^{h/2} u(x + \alpha e_i)\, \mathrm{d}\alpha. \tag{4.87}$$

We can use the fundamental theorem of calculus to show the relation between ∂^h_i and g^h_i (with no summation over i):

$$\partial^h_i = g^h_i \partial_i, \tag{4.88}$$

meaning that the finite difference ∂^h_i can be induced by filtering the exact derivative ∂_i.

The Navier-Stokes momentum and continuity equations include derivatives in each of the cardinal directions x_1, x_2, and x_3. For example, the i-th momentum equation includes the term $\partial_j r_{ij} = \partial_1 r_{i1} + \partial_2 r_{i2} + \partial_3 r_{i3}$. If we

filter the i-th momentum equation with the 1D grid filter g_i^h in the direction i, the equation for $g_i^h u_i$ would include the term $g_i^h \partial_j r_{ij}$, and we could only use the filter-swap property on one of the three terms, where $j = i$. A similar remark was made by Lund, who argued that ideally, we would like to filter each of the three derivatives ∂_j with their associated filters g_j^h separately, but such equations cannot be obtained by applying one single filter to the Navier-Stokes momentum equations, since the same filter has to be applied to all of the terms [111]. We therefore resort to multi-dimensional grid filters.

From the 1D filters g_i^h, we define the multi-dimensional volume-averaging filter

$$f^h := g_1^h g_2^h g_3^h \tag{4.89}$$

and surface-averaging filters

$$f^{h,1} := g_2^h g_3^h, \quad f^{h,2} := g_1^h g_3^h, \quad f^{h,3} := g_1^h g_2^h. \tag{4.90}$$

For all $u \in U$, we employ the short-hand notation

$$\bar{u}^h := f^h u, \quad \bar{u}^{h,i} := f^{h,i} u. \tag{4.91}$$

A similar notation was used by Schumann [158].

By using the 1D property (4.88) for g_i^h, we obtain the following filter-swap commutation property for the multi-dimensional grid filters (with no sum over i):

$$f^h \partial_i = \partial_i^h f^{h,i}. \tag{4.92}$$

This is a discrete equivalent of the infinitesimal property $f^\Delta \partial_i = \partial_i f^\Delta$ which allows for switching between infinitesimal and discrete derivatives. The important observation is that the filter definition changes with the derivative definition. The 1D case in eq. (4.19) can be seen as a special case of eq. (4.92), where the 1D volume is $[x \pm h/2]$ and the 1D surface is a single point in $\{x \pm h/2\}$.

4.5.5 *LES-FVM*

The LES equation is $\overline{L(u)}^\Delta = 0$. Applying the FVM filter f^h gives the LES-FVM equation

$$\overline{L(u)}^{\Delta,h} = 0. \tag{4.93}$$

Using the filter-swap commutation property (4.92), we obtain the form

$$\partial_t \bar{u}_i^{\Delta,h} + \partial_j^h \overline{r_{ij}(u)}^{\Delta,h,j} = 0. \tag{4.94}$$

We can already note a couple of interesting observations. The stress tensor $\overline{r_{ij}(u)}^{\Delta,h,j}$ (no sum over j) is non-symmetric (because it is averaged over the surface orthogonal to j, but not i) and non-local in u (due to the projection present inside r). Furthermore, the LES-FVM solution $\bar{u}^{\Delta,h}$ is not discretely divergence-free, since $\partial_j^h \bar{u}_j^{\Delta,h} \neq \overline{\partial_j u_j}^{\Delta,h} = 0$ (see theorem 2). [1] Since the divergence-free constraint is important for stability in the energy-conserving convective term, we consider the divergence-free part of the volume-averaged field $\bar{u}^{\Delta,h}$. This is precisely the projected field

$$\bar{u}_i^{\Delta,h,\pi} := \pi_{ij}^h \bar{u}_j^{\Delta,h}. \tag{4.95}$$

Applying the projector π_{ij}^h to eq. (4.94) and using theorem 8 to swap projection and divergence gives

$$\partial_t \bar{u}_i^{\Delta,h,\pi} + \partial_j^h \pi_{ij\alpha\beta}^h \overline{r_{\alpha\beta}(u)}^{\Delta,h,\beta} = 0, \tag{4.96}$$

which is a discrete conservation law for a divergence-free velocity field.

Adding the discrete flux term $\partial_j^h r_{ij}^h(\bar{u}^{\Delta,h,\pi})$ to both sides gives the projected LES-FVM equation in the form

$$L_i^h(\bar{u}^{\Delta,h,\pi}) = -\partial_j^h \tau_{ij}^{\Delta,h,\pi}(u), \tag{4.97}$$

where the projected RST $\tau_{ij}^{\Delta,h,\pi} := \pi_{ij\alpha\beta}^h \xi_{\alpha\beta}^{\Delta,h,\pi}$ is defined by the RST (with no sum over j)

$$\boxed{\xi_{ij}^{\Delta,h,\pi}(u) := \overline{r_{ij}(u)}^{\Delta,h,j} - \sigma_{ij}^h(\bar{u}^{\Delta,h,\pi}).} \tag{4.98}$$

For comparison, the classical LES RST for the projected double-filter $\pi f^h f^\Delta$ would be $\tau_{\text{Classic},ij}^{\Delta,h,\pi} := \pi_{ij\alpha\beta} \xi_{\text{Classic},\alpha\beta}^{\Delta,h,\pi}$, where

$$\xi_{\text{Classic},ij}^{\Delta,h,\pi}(u) := \overline{\sigma_{ij}(u)}^{\Delta,h} - \sigma_{ij}(\bar{u}^{\Delta,h}). \tag{4.99}$$

Note that $\bar{u}^{\Delta,h}$ is already infinitesimally divergence-free, so $\pi_{ij}\bar{u}_j^{\Delta,h} = \bar{u}_i^{\Delta,h}$.

Like in the 1D case, the discretization-informed RST $\xi_{ij}^{\Delta,h,\pi}(u)$ accounts for both the filter-swap mechanism and the numerical stress σ_{ij}^h. Unlike the 1D case, it also accounts for the pressure, since the pressure that makes $\bar{u}^{\Delta,h,\pi}$ discretely divergence-free is not equal to a filtered variant of the pressure that makes u infinitesimally divergence-free.

In 3D, the LES-FVM RST is non-symmetric and non-local— fundamentally different from the classical RST.

1 An alternative is therefore to solve for the *surface-averaged* field $\bar{u}_i^{\Delta,h,i}$ (no sum over i), since it becomes ∂_j^h-divergence-free after filtering ($\partial_j^h \bar{u}_j^{\Delta,h,j} = 0$) [2, 89]. However, this field is not governed by a ∂_j^h-conservation law, since the surface-averaged momentum is not conserved discretely [2]. We will therefore not consider this option here.

The LES-FVM RST $\xi^{\Delta,h,\pi}$ has two important differences from the classical LES RST $\xi^{\Delta,h,\pi}_{\text{Classic}}$ (classical LES with $\pi f^h f^\Delta$ as the filter). First, it is non-symmetric ($\xi^{\Delta,h,\pi}_{ij} \neq \xi^{\Delta,h,\pi}_{ji}$), while the classical RST is symmetric ($\xi^{\Delta,h,\pi}_{\text{Classic},ij} = \xi^{\Delta,h,\pi}_{\text{Classic},ji}$). Second, it is non-local in u (due to the pressure projector), while the classical RST is local in u.

4.5.6 *LES-FVM closure*

Let $m^{\Delta,h,\pi}(\bar{u}^{\Delta,h,\pi}) \approx \text{dev}\left(\xi^{\Delta,h,\pi}(u)\right)$ be a grid-compatible LES-FVM closure model for the deviatoric part of the RST defined as $\text{dev}(\sigma)_{ij} \coloneqq \sigma_{ij} - \delta_{ij}\sigma_{kk}/3$ for all $\sigma \in U^{3\times 3}$. Since $\partial^h_j \pi^h_{ij\alpha\beta} p\delta_{\alpha\beta} = 0$ for all scalar fields $p \in U$, we only need a closure model for the deviatoric part of the RST (the isotropic part will be absorbed by the pressure projector). The approximate LES-FVM equations are

$$L^h_i(v^{\Delta,h,\pi}) = -\partial^h_j \pi^h_{ij\alpha\beta} m^{\Delta,h,\pi}_{\alpha\beta}(v^{\Delta,h,\pi}), \tag{4.100}$$

where $v^{\Delta,h,\pi} \approx \bar{u}^{\Delta,h,\pi}$ is the approximate LES-FVM solution.

We say that the closure $m^{\Delta,h,\pi}$ is structural if it is chosen to approximate $\xi^{\Delta,h,\pi}$ directly, and functional if it is chosen to produce the same dissipation as $\xi^{\Delta,h,\pi}$. In 1D, an RST and its dissipation coefficient are fields of the same dimension (scalar field). The distinction between structural and functional models is therefore less clear in 1D. In 3D, the RST has 9 components, while the dissipation coefficient has 1 component (scalar field). There is therefore more freedom in how to choose a functional model than a structural model in 3D.

In the exact LES-FVM equation (for $\bar{u}^{\Delta,h,\pi}$), the RST is non-symmetric and non-local in u. This observation would suggest that the closure model in the approximate LES-FVM equation (for $v^{\Delta,h,\pi}$) should also be non-symmetric and non-local in $v^{\Delta,h,\pi}$. In the exact LES equation (for $\bar{u}^\Delta$), the RST is both symmetric and local in u, which justifies using a symmetric and local closure model in classical LES (before discretization). A "perfect" LES-FVM closure model therefore needs to predict all 9 tensor components, whereas classical LES closures only need to predict 6 independent symmetric components.

We now illustrate the new RST expression with a numerical experiment.

4.6 EXPERIMENT: DNS-AIDED LES-FVM FOR 3D TURBULENCE

Like for the Burgers equation, we employ a "DNS-aided LES-FVM" approach to assess the importance of the RST definition in the LES-FVM equations. We consider a 3D decaying turbulence test case in a periodic box. The equations are defined by the following unitless parameters. The domain size is $\ell \coloneqq 1$. The viscosity is $\nu \coloneqq 2 \times 10^{-4}$. The number of finite volumes in each of the

The code is available at https://zenodo.org/records/16267799. The results were obtained using a single Nvidia H100 GPU on the Dutch National Supercomputer Snellius. To allow for reproducibility on accessible hardware, the code also includes a scaled down version of the 3D experiment (90^3 DNS grid points and $\{18^3, 30^3\}$ LES grid points) that can be run with multithreading on a modern laptop CPU.

3 dimensions for DNS and LES are $N_{\text{DNS}} := 500$ and $N := 100$ respectively, with a compression factor of 5. The grid spacings are $h_{\text{DNS}} := \ell / N_{\text{DNS}}$ and $h := \ell / N$ (note that these are called h and H in section B.2, respectively). The LES filter f^Δ is a Gaussian of width $\Delta := 2h$. In the DNS discretization of the Gaussian kernel G^Δ, we include points up to 2 standard deviations out on each side (see B.2 for details). The infinitesimal expressions from section 4.5 are all approximated on the DNS grid so that we can evaluate them based on the DNS solution, but we will still use the infinitesimal notation to avoid visual clutter.

4.6.1 *Initialization*

Let $u \in U^3$ be a velocity field and $\kappa \in \mathbb{N}$ be a scalar wavenumber. We define the energy spectrum of u at κ as

$$E(u, \kappa) := \frac{1}{2} \sum_{k \in K(\kappa)} \|\hat{u}(k)\|^2, \tag{4.101}$$

where $\hat{u}(k)$ is the Fourier transform of u at a wavenumber k and $K(\kappa) := \{k \in \mathbb{Z}^3 \mid \kappa \leq \|k\| < \kappa + 1\}$ is the shell of vector wavenumbers with magnitude between κ and $\kappa + 1$.

We initialize the solution u on the DNS grid through the following procedure, where $\leftarrow$ denotes the assignment operator.

1. Sample a random field $u_i(x) \sim \mathcal{N}(0, 1)$ from a normal distribution for each $i \in \{1, 2, 3\}$ and each DNS grid point x. We do not define u_i outside the grid points.
2. Project the velocity: $u \leftarrow \pi u$.
3. Compute the discrete Fourier transform $\hat{u} \leftarrow \text{FFT}(u)$.
4. For all wavenumbers $\kappa \in \{0, 1, \dots, \lfloor \sqrt{3} N_{\text{DNS}} / 2 \rfloor\}$, compute the current shell energy $E(u, \kappa)$, where $\lfloor \cdot \rfloor$ denotes the integer part. For $\kappa \geq N/2$, the shells are only partially filled, since the discrete Fourier transform gives a finite number of Fourier modes. Adjust the coefficients in the shell $K(\kappa)$ as

$$\hat{u}(k) \leftarrow \sqrt{\frac{P(\kappa)}{E(u, \kappa)}} \hat{u}(k), \quad \forall k \in K(\kappa), \tag{4.102}$$

   where $P(\kappa)$ is a prescribed energy profile defined as the Kolmogorov scaling in the inertial range as

$$P(\kappa) := \kappa^{-5/3}. \tag{4.103}$$

5. Apply inverse Fourier transform $u \leftarrow \text{IFFT}(\hat{u})$.

6. Reproject the velocity field (since the shell normalization may slightly perturb the DNS divergence of u): $u \leftarrow \pi u$.

7. Scale the velocity field such that the total energy adds up to 1: $u \leftarrow \frac{1}{\sqrt{1/2\|u\|^2}} u$.

The resulting velocity field u is represented as an array of size $N_{\text{DNS}}^3 \times 3$. It is divergence-free, the spectrum is proportional to the profile P (with some deviations due to the second projection), and the total energy is 1.

Since the initial spectrum is artificial, we first run the DNS simulation for 0.1 time units to obtain a more realistic distribution of velocity scales.

4.6.2 *DNS-aided LES-FVM*

The DNS-aided LES-FVM formulation is defined as

$$L(u) = 0, \quad L^h(v^{\text{model}}) = -\nabla^h \cdot \tau^{\text{model}}(u), \tag{4.104}$$

where u is the DNS solution, v^{model} is the modeled LES-FVM solution, $\nabla^h := (\partial_1^h, \partial_2^h, \partial_3^h)$, $\tau^{\text{model}} := \pi^h \xi^{\text{model}}$, and $\xi^{\text{model}} : U^3 \to U^{3\times 3}$ is an RST expression taking the DNS solution as input (to be defined in section 4.6.3). These equations are still continuous in time. A forward-Euler time discretization with adaptive time stepping is

$$\Delta t^n := C \times \min\left(\frac{h_{\text{DNS}}}{\max|u^n|}, \frac{h_{\text{DNS}}^2}{6\nu}\right), \tag{4.105}$$

$$u^{n+1} := u^n - \Delta t^n \nabla \cdot r(u^n), \tag{4.106}$$

$$v^{n+1} := v^n - \Delta t^n \nabla^h \cdot \left(r^h(v^n) + \tau^{\text{model}}(u^n)\right), \tag{4.107}$$

where Δt^n is chosen based on the DNS solution, u^n and v^n are the Forward-Euler approximations to the u and v^{model} at time $t^n := \sum_{k=0}^{n-1} \Delta t^k$, and $C := 0.4$ is a margin in the CFL expression. The initial conditions are given by the final DNS field from the warm-up simulation with $v^{\text{model}} = \bar{u}^{\Delta,h,\pi}$. The goal is for v^{model} to stay close to the projected filtered DNS velocity field $\bar{u}^{\Delta,h,\pi}$ over time.

Since both $u = \pi u$ and $v^h = \pi^h v^h$ are divergence-free on their respective grids, and theorem 8 can be used to commute projection and divergence, we can use the "semi-explicit" form of the scheme:

$$u^{n+1} = \pi\left[u^n - \Delta t^n \nabla \cdot \sigma(u)\right], \tag{4.108}$$

$$v^{n+1} = \pi^h\left[v^n - \Delta t^n \nabla^h \cdot \left(\sigma^h(v^n) + \xi^{\text{model}}(u^n)\right)\right], \tag{4.109}$$

which consists of doing an explicit forward-Euler step with the unprojected stresses before reprojecting the velocity field back onto the space of divergence-free fields.

Since turbulent flows are chaotic, we run the simulation for a short duration of 0.1 time units. If we run for longer, the LES and DNS solutions will decorrelate and the DNS-aided closure term will be of little use (unless ξ^{model} is the new correct RST expression).

4.6.3 *RST expressions*

We consider five DNS-aided "closure" models defined (with no sum over j) by:

$$\xi^{\mathrm{A}}_{ij}(u) \coloneqq 0, \tag{4.110}$$

$$\xi^{\mathrm{B}}_{ij}(u) \coloneqq \overline{\sigma_{ij}(u)}^{\Delta,h} - \sigma_{ij}\left(\overline{u}^{\Delta,h}\right), \tag{4.111}$$

$$\xi^{\mathrm{C}}_{ij}(u) \coloneqq \overline{\sigma_{ij}(u)}^{\Delta,h} - \sigma^{h}_{ij}\left(\overline{u}^{\Delta,h,\pi}\right), \tag{4.112}$$

$$\xi^{\mathrm{D}}_{ij}(u) \coloneqq \overline{\sigma_{ij}(u)}^{\Delta,h,j} - \sigma^{h}_{ij}\left(\overline{u}^{\Delta,h,\pi}\right), \tag{4.113}$$

$$\xi^{\mathrm{E}}_{ij}(u) \coloneqq \frac{1}{2}\left(\xi^{\mathrm{D}}_{ij}(u) + \xi^{\mathrm{D}}_{ji}(u)\right). \tag{4.114}$$

The respective LES-FVM solutions are denoted $\{v^{\mathrm{A}}, \dots, v^{\mathrm{E}}\}$. ξ^{A} is the no-model case (DNS on the coarse grid). ξ^{B} is the classical RST expression for the given double-filter, with all operators evaluated on the DNS grid. Since $\pi f^{\Delta,h} = f^{\Delta,h}\pi$ and $\pi u = u$, the second term in $\xi^{B}_{ij}(u)$ simplifies from $\sigma(\pi\bar{u}^{\Delta,h})$ to $\sigma(\bar{u}^{\Delta,h})$. ξ^{C} accounts for the coarse grid discretization through the h-grid numerical stress σ^h in the second term. Since the filtered field is not divergence-free on the h-grid ($\pi^h f^{\Delta,h} \neq f^{\Delta,H}\pi$), the second term in ξ^{C} includes the projector π^h. ξ^{D} further accounts for the filter-swap mechanism. The FVM filter is therefore surface-averaging in the first term and not volume-averaging. ξ^{E} is defined as the symmetric part of ξ^{D} (the surface-averaged term makes ξ^{D} non-symmetric).

From eq. (4.98), we know that ξ^{D} is the correct expression, since it accounts for the h-grid FVM discretization. In an "explicit LES", which can be thought of as letting $h \to 0$, ξ^{B} would be the correct expression. ξ^{C} and ξ^{E} are included to assess the importance of the filter-swap mechanism and the non-symmetric part of the correct RST ξ^{D}, respectively. As classical LES closure models are designed to be symmetric, they are *at best* able to represent ξ^{E}, but not the non-symmetric RST ξ^{D}.

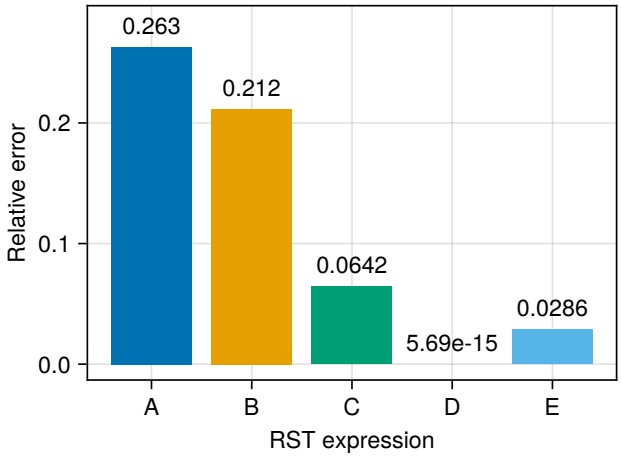


Figure 4.12: Errors from DNS-aided LES-FVM for the 3D decaying turbulence simulation.

4.6.4 *Results*

In fig. 4.12, we show the relative errors

$$\frac{\|v^{\text{model}} - \bar{u}^{\Delta,h,\pi}\|_\Omega}{\|\bar{u}^{\Delta,h,\pi}\|_\Omega} \tag{4.115}$$

at the final time $t = 0.1$ for the five RST expressions. Our proposed RST expression ξ^{D} gives an error at machine precision, confirming that it is the correct RST expression for the LES-FVM formulation. The error for ξ^{E} is at 2.86%, showing that the non-symmetric part of the RST is indeed non-zero. The error for ξ^{C} is at 6.42%, a bit more than twice as large as ξ^{E}. This shows that accounting for the filter-swap mechanism is also important. The error for the classical expression ξ^{B} is at 21.2%, a bit more than three times larger than for ξ^{C}. Finally, the no-model ξ^{A} gives the largest error at 26.3%. When comparing the errors for all the expressions, the largest absolute improvement is obtained by accounting for the numerical flux (from ξ^{B} to ξ^{C}). All errors are evaluated after a short simulation of 0.1 time units. They will therefore continue to grow for longer simulations.

The proposed RST gives zero error; the classical RST gives 21.2% error after only 0.1 time units.

In fig. 4.13, we show the energy spectra at the final time. The spectrum of ξ^{D} perfectly overlaps with the reference spectrum, while the one of ξ^{E} is lower, and the one of ξ^{C} is lower still. This means that not accounting for the non-symmetry and the filter-swap property leads to a slight over-estimation of the energy dissipation. On the other hand, the spectrum of the classical RST ξ^{B} is too high, meaning that not accounting for the discretization in the RST leads to an under-estimation of the energy dissipation. Finally, the

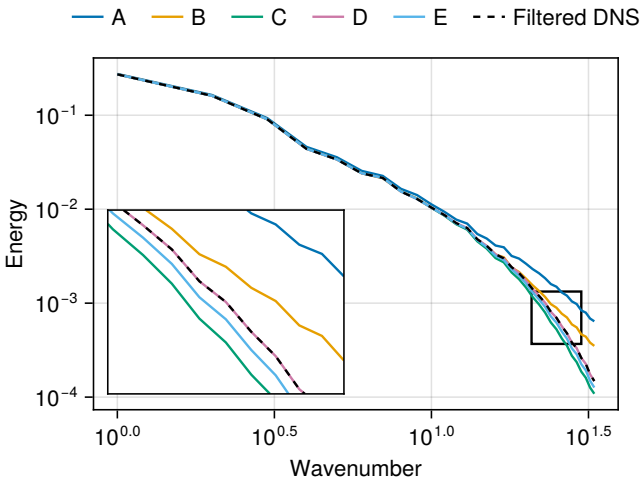


Figure 4.13: Energy spectra from DNS-aided LES-FVM for the 3D decaying turbulence test case.

no-model ξ^{A} has the highest spectrum, showing that a dissipative closure model is required to get the correct spectrum.

In the 1D Burgers' case (fig. 4.7), many of the residual flux expressions led to a sharp increase at the very end of the spectrum. This was likely due to the shock-like structures that cannot be resolved on the coarse grid. In fig. 4.13, no such sharp increase at the end of the spectrum is observed. Although the 3D resolution is smaller, which makes it difficult to compare with the 1D case, the 3D experiment is incompressible, and therefore cannot form the shock-like structures that were present in the Burgers' case.

4.7 CONCLUSION

In this work, we have proposed new exact expressions for the residual stress tensor (RST) appearing in discrete filtered conservation laws when using combined LES and FVM filters. Unlike the classical RST expression $\overline{uu} - \bar{u}\bar{u}$ which is commonly studied in LES, the RST in LES-FVM also includes contributions from the discrete divergence, numerical flux, and discrete incompressibility constraint. Writing the residual in discrete divergence form makes the RST non-symmetric, while accounting for the discrete incompressibility constraint makes the RST non-local. To make these two properties more intuitive, we show how they are derived using an alternative notation in B.5.

In a DNS-aided LES framework, where DNS data is used to compute the RST, our RST expression gives errors at machine precision, while the classical RST gives errors that grow over time, confirming the importance of accounting for the discretization errors in the RST.

In practice, DNS-aided LES is only useful for evaluating the correctness of the RST expression. Does the new RST provide any practical benefits for LES closure modeling? We believe so, and we have emphasized two main benefits:

1. The new framework informs us about the structure of the RST, which can guide the development of new or existing closure models. Notably, the LES-FVM RST expression suggests that a "perfect" structural closure model for incompressible flows should be non-symmetric and non-local in the velocity field. From the experiments in section 4.6, we have some idea of the importance of the non-symmetric part, but the benefit of including non-local values in a closure model for incompressible flows is still unclear and requires further study.

2. The new RST can be used as a target for tuning closure model parameters. By correctly accounting for the discretization in the RST, the RST automatically gives the correct desired dissipation profile that a functional closure model should reproduce. Notably, the correct RST accounts for the implicit dissipation produced by the numerical flux, so that the amount of explicitly modeled dissipation can be reduced accordingly. Our LES-FVM framework can thus be used as a tool to bridge the gap between implicit and explicit LES closure modeling.

For the 1D Burgers equation, we demonstrated the benefit of the second point by fitting the Smagorinsky model coefficient to the old and new RST expression. Indeed, the coefficient fitted to the new RST was slightly higher, adding some of the additional dissipation required by the FVM discretization and leading to better results. This is despite the fact that the Smagorinsky model only has one coefficient and is a rather inaccurate closure.

To compute the RST exactly from DNS reference data, we have introduced a new *two-grid formulation* that accounts for two different overlapping discretization levels (see B.2). This formulation is constructed such that the filter-swap commutation properties hold discretely. Making sure the DNS and LES grids are compatible in this way ensures that the exact RST (with respect to the DNS reference) is computable. Furthermore, as high-dimensional function approximators such as neural networks are receiving more attention for LES closure modeling [157], access to accurate discretization-informed target data becomes increasingly important [14].

For 1D conservation laws, the central insight that leads to these discrete formulations was the partial restoration of commutation between filtering and differentiation for finite differences and coarsening grid filters. We

showed that coarsening filters do not commute with differentiation, but the partial commutation property of finite differences was sufficient to obtain a discrete RST expression. This commutation property also holds for higher-order finite difference schemes, such as the ones studied by Geurts and van der Bos [63]. A similar formulation can be derived for finite volume schemes on unstructured grids, since a volume-averaged divergence over an unstructured grid cell can be written as a surface integral over the volume faces. Such an RST expression was proposed by Denaro [49, 50].

The key to making the LES-FVM framework valid for incompressible flows was to rewrite the incompressible Navier-Stokes equations as a self-contained conservation law for the velocity field. This was possible by incorporating the incompressibility constraint and pressure gradient correction into a pressure projection operator. This operator then became part of the stress tensor in the self-contained conservation law for the velocity field. In the presence of no-slip boundary conditions, this approach can still be used. However, it is not yet clear whether the pressure term can be written solely in terms of the velocity for certain types of outflow boundary conditions that prescribe a value for the pressure at the boundary. This topic requires further study.

The LES-FVM framework we propose relies on first choosing a discretization and a grid size, before choosing a closure model and tuning its parameters. One therefore cannot expect the closure model to transfer to a different discretization or grid size without retraining. The discrete closure model must be recalibrated to discretization-informed target data obtained from DNS when the LES grid size is changed.

We demonstrated the LES-FVM framework for staggered grids, which are composed of a primal and a dual grid (velocity and pressure points). The nodes on the primal and dual grids do not overlap but are separated by half a grid spacing. This led to the requirement that the primal and dual LES grids overlap with the primal and dual DNS grids, respectively, by using uniform Cartesian grids with odd compression factors. This is a limitation of our framework on staggered grids. With some care, *unstructured* staggered grids can also be designed such that the primal and dual LES grids overlap with the primal and dual DNS grids. This can be achieved by *first* choosing the LES grid and then refining it to obtain the DNS for generating training data. For flows around objects (such as airfoils), the LES grid would therefore need to be sufficiently fine to resolve the shape of the object considered. No further refinement of the object boundary would be possible without losing the exactness of the formulation. By allowing for some error at the boundary, this requirement could be relaxed, while retaining the exact formulation in the interior of the domain.

5

COMPARISON OF DATA-DRIVEN SYMMETRY-PRESERVING CLOSURE MODELS FOR LARGE-EDDY SIMULATION

This chapter is based on the following article: *Syver Døving Agdestein and Benjamin Sanderse.* Comparison of Data-Driven Symmetry-Preserving Closure Models for Large-Eddy Simulation. *Mar. 2026.* DOI: 10.48550/arXiv.2603.05325. *arXiv:* 2603.05325 [math].

CRediT author statement
Syver Døving Agdestein*: Conceptualization, Formal analysis, Methodology, Software, Validation, Visualization, Writing – Original Draft*
Benjamin Sanderse*: Funding acquisition, Project administration, Supervision, Writing – Review & Editing*

The previous chapters focused on discretization-consistency of LES closure models. This chapter shifts focus to another structural property of the Navier-Stokes equations: their symmetries, and how these can be built into neural network closure models.

Symmetries are fundamental to both turbulence and differential equations. The large-eddy simulation (LES) equations inherit these symmetries provided the LES closure respects them [133, 164]. Classical LES closures based on eddy viscosity or scale similarity preserve many of the original symmetries by design [148, 164].

Recently, data-driven neural network closures have been applied to LES to improve accuracy, but stability and generalizability remain challenges, as symmetries are not automatically enforced (see also chapters 3 and 4). In this work, we compare approaches for constructing symmetry-preserving data-driven LES closures, including tensor-basis neural networks (TBNNs) and group-convolutional neural networks, alongside unconstrained convolutional networks. All three data-driven closures outperform classical models in both the functional sense (producing the right amount of dissipation) and the structural sense (direct stress prediction). While unconstrained networks achieve comparable prediction accuracy, symmetry-preserving models produce more physically consistent velocity-gradient statistics, suggesting that enforcing symmetries improves the quality of the learned closure beyond what aggregate error metrics such as relative tensor prediction errors capture.

5.1 INTRODUCTION

Symmetries are fundamental properties of both turbulence [56] and differential equations in general [21, 134]. The incompressible Navier-Stokes equations exhibit symmetries including Galilean invariance and rotation invariance. The LES equations, obtained by filtering the Navier-Stokes equations, may inherit these symmetries depending on the filter, but the complete LES system preserves them only if the closure model does as well. Closure models that violate these symmetries can introduce spurious forces and unphysical behavior [133, 148, 164]. Classical functional closures such as eddy-viscosity models enforce Galilean invariance by construction, since they depend only on invariants of the velocity-gradient tensor. Structural

closures, including Bardina's scale-similarity model [10] and Clark's gradient model [41], also preserve most symmetries of the original equations [164].

Machine learning (ML) has been successfully applied to both functional [93] and structural [145, 146] LES closure modeling, achieving high a-priori accuracy. However, models that perform well on training data can still exhibit instabilities when deployed in actual simulations [91], a problem that affects both ML-based and classical closures [180]. Symmetry preservation addresses one source of such instabilities: closure models that respect the symmetries of the Navier-Stokes equations cannot introduce spurious frame-dependent forces or violate rotational isotropy, improving physical consistency and, in many cases, numerical stability. Ensuring symmetry preservation in ML closures is thus an important step toward their reliable deployment [120].

Several approaches have been developed to embed symmetries in ML-based closure models. One family of methods employs group equivariant neural networks [42, 95]. Group convolutional neural networks have been applied to LES [67], but their feature channels scale with the number of group elements, making them computationally intractable for large symmetry groups such as the continuous rotation group. Steerable CNNs address this scaling limitation by representing equivariance-preserving weights via a truncated Fourier series [202, 203]. Connolly et al. applied rotation-equivariant steerable CNNs to atmospheric LES [44], though only for rotations around the vertical axis. Jalaali and Okabayashi [80] achieved rotational invariance in a CNN-based SGS model by incorporating spatial transformer networks. Graph neural networks (GNNs) provide yet another equivariant framework [88]: Shankar et al. used GNNs to preserve symmetries in 2D LES [160], Kurz et al. applied them to learn eddy viscosity coefficients in 3D [93], and Lino et al. [104] and List et al. [105] developed rotation-equivariant GNN architectures for fluid dynamics.

Rather than enforcing equivariance through the network architecture, an alternative approach encodes symmetries through the choice of input features. This approach employs Pope's tensor basis [143], rooted in the theory of Spencer and Rivlin [171, 172]. Expressing the subgrid-scale stress tensor as a linear combination of basis tensors that are equivariant to the symmetry groups of the Navier-Stokes equations automatically enforces the required symmetries, provided the expansion coefficients depend only on invariants of the velocity-gradient tensor (VGT). Tensor basis closures for LES remain an active area of research [112, 164]. Ling et al. [103] pioneered the use of tensor basis neural networks (TBNNs), in which a neural network predicts the invariant coefficients, initially for Reynolds-averaged Navier-Stokes (RANS) closures, with subsequent developments in [16, 26, 40, 125, 126]. TBNNs have also been applied to LES [24, 161, 207]. Stallcup et al. [174, 175] proposed an alternative tensor basis derived from Smith's analysis [169], later employed in a TBNN by Wu and Lele [207]. A key advantage of TBNNs

is that they enforce symmetries regardless of the functional form of the invariant coefficients, imposing no architectural constraints on the neural network. This contrasts with group convolutional neural networks, steerable CNNs, and GNNs, where the architecture itself must be designed to preserve equivariance. Kaszuba et al. [84] proposed Euclidean neural networks as an alternative that implicitly encodes the tensor basis structure.

In this chapter, we compare TBNNs and group convolutional neural networks as symmetry-preserving closure models for LES, alongside unconstrained CNNs that do not enforce symmetries. Our goal is to determine whether symmetry-preserving models outperform their unconstrained counterparts, and to assess whether simple neural network architectures can accurately learn the subgrid-scale stress directly. The remainder of this chapter is organized as follows. Section 5.2 reviews the symmetries of the incompressible Navier-Stokes equations and their implications for LES. Section 5.3 presents our TBNN implementation for LES closure modeling. Section 5.4 describes the G-conv implementation. Section 5.5 compares the performance of the closure models.

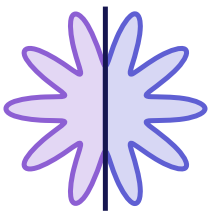
5.2 SYMMETRIES IN LARGE-EDDY SIMULATION

The Navier-Stokes equations admit a set of symmetries, but not all are preserved under discretization or closure modeling. In this section, we review the symmetry groups of the continuous Navier-Stokes equations and discuss their implications for LES closures.

5.2.1 *Transforms and symmetries*

Consider a scalar-, vector-, or tensor-valued field p, u, or σ. For a symmetry-group G with representation ρ in physical space $\mathbb{R}^3$, the group action for an element $g \in G$ is defined as

$$\begin{aligned} gp(x) &= p(\rho_g^{-1}x), \\ gu(x) &= \rho_g u(\rho_g^{-1}x), \\ g\sigma(x) &= \rho_g \sigma(\rho_g^{-1}x)\rho_g^{-1}. \end{aligned} \tag{5.1}$$



The shape above is invariant to reflectional (vertical line) and rotational symmetries. Only 10 *different rotations are possible, meaning that the symmetry group is discrete.*

For example, consider a rotation matrix

$$R \coloneqq \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.2}$$

representing a 90-degree rotation around the x_3-axis, i.e. $R = \rho_g$ for the corresponding group element g. Then

$$g\sigma = R\sigma R^T = \begin{pmatrix} \sigma_{22} & -\sigma_{21} & \sigma_{23} \\ -\sigma_{12} & \sigma_{11} & -\sigma_{13} \\ \sigma_{32} & -\sigma_{31} & \sigma_{33} \end{pmatrix}. \tag{5.3}$$

An operator $\mathcal{L}$ is *invariant* under G if, for all $g \in G$,

$$\mathcal{L}(g\varphi) = \mathcal{L}(\varphi), \tag{5.4}$$

where φ is a scalar-, vector-, or tensor-valued field. Similarly, $\mathcal{L}$ is *equivariant* if, for all $g \in G$,

$$\mathcal{L}(g\varphi) = g\mathcal{L}(\varphi). \tag{5.5}$$

Let $\mathcal{N}(u) = 0$ denote an equation for a field $u(x,t)$, and let $\tilde{x}(x,t)$, $\tilde{t}(x,t)$, and $\tilde{u}(\tilde{x},\tilde{t})$ denote transformed coordinates and fields. The transformed equation operator is denoted $\tilde{\mathcal{N}}(\tilde{u})$. A transform $\tilde{\cdot}$ is a *symmetry* of the equation $\mathcal{N} = 0$ if

$$\mathcal{N}(u) = 0 \implies \tilde{\mathcal{N}}(\tilde{u}) = 0, \tag{5.6}$$

i.e. solutions are mapped to solutions under the transform.

5.2.2 *Symmetries of the incompressible Navier-Stokes equations*

Let Ω be a spatial domain and $U = \{u : \Omega \times \mathbb{R} \to \mathbb{R},\ (x,t) \mapsto u(x,t)\}$ be the space of time-dependent scalar fields on Ω. To simplify notation, we omit the time variable t and write $u(x)$ instead of $u(x,t)$ when the meaning is clear from context.

The incompressible Navier-Stokes equations (see also chapters 3 and 4) are given by

$$\partial_j u_j = 0, \quad \partial_t u_i + \partial_j \left(\sigma_{ij}(u) + p\delta_{ij}\right) = f_i, \tag{5.7}$$

where $u \in U^3$ is the velocity field, $p \in U$ is the pressure field, $(i,j) \in \{1,2,3\}$ are spatial indices, δ_{ij} is the Kronecker symbol, $f \in U^3$ is a body force, $\sigma(u) \in U^{3\times 3}$ contains the nonlinear and viscous stresses defined as

$$\sigma_{ij}(u) \coloneqq u_i u_j - \nu\left(\partial_j u_i + \partial_i u_j\right), \tag{5.8}$$

and $\nu > 0$ is the viscosity. Unless otherwise stated, the Einstein summation convention applies for repeated indices.

We also define the velocity gradient tensor (VGT) $A_{ij} \coloneqq \partial_j u_i$, the strain-rate and rotation-rate tensors $S_{ij} \coloneqq (A_{ij} + A_{ji})/2$ and $W_{ij} \coloneqq (A_{ij} - A_{ji})/2$,

and the second and third invariants of the VGT, $q := -\frac{1}{2}A_{ij}A_{ji}$ and $r := -\frac{1}{3}A_{ij}A_{jk}A_{ki}$.

The complete symmetries of the incompressible Navier-Stokes equations (5.7) are listed below [133].

- Time invariance: $\tilde{t} = t + a$, $\tilde{x} = x$, $\tilde{u} = u$, $\tilde{p} = p$, where $a \in \mathbb{R}$ is a constant.
- Rotation invariance: $\tilde{x}_i = Q_{ij}x_j$, $\tilde{t} = t$, $\tilde{u}_i = Q_{ij}u_j$, $\tilde{p} = p$, where $Q \in \mathbb{R}^{3\times 3}$ is an orthogonal matrix.
- Reflection invariance in the direction x_i (with $j \neq i$): $\tilde{x}_i = -x_i, \tilde{x}_j = x_j$, $\tilde{t} = t, \tilde{u}_i = -u_i, \tilde{u}_j = u_j, \tilde{p} = p$.
- Generalized Galilean invariance: $\tilde{x}_i = x_i + X_i(t), \tilde{t} = t, \tilde{u}_i = u_i + \dot{X}_i(t)$, $\tilde{p} = p - x_j\ddot{X}_j(t)$, where $X : \mathbb{R} \to \mathbb{R}^3$ is a time-dependent twice differentiable frame translation.
- Scaling invariance: $\tilde{x}_i = ax_i, \tilde{t} = bt, \tilde{u}_i = \frac{a}{b}u_i, \tilde{p} = \left(\frac{a}{b}\right)^2 p, \tilde{\nu} = \frac{a^2}{b}\nu$, where $a \in \mathbb{R}$ and $b > 0$ are constants. Note that ν must be scaled to preserve the symmetry (unless $a^2 = b$).
- Pressure invariance: $\tilde{x} = x$, $\tilde{t} = t$, $\tilde{u} = u$, $\tilde{p} = p + a(t)$, where $a : \mathbb{R} \to \mathbb{R}$ is constant in space.
- Material frame indifference.

Not all of these symmetries are automatically preserved in the LES equations, as discussed next.

5.2.3 *Symmetries in LES*

Consider a convolutional filter $\overline{(\cdot)}$ defined by

$$\bar{u}(x) := \int_\Omega G(x-y)u(y)\,\mathrm{d}y \tag{5.9}$$

for some kernel G. The filter commutes with differentiation: $\partial_j\bar{u} = \overline{\partial_j u}$. The filtered Navier-Stokes equations are given by

$$\partial_j\bar{u}_j = 0, \quad \partial_t\bar{u}_i + \partial_j\left(\sigma_{ij}(\bar{u}) + \tau_{ij}(u) + \bar{p}\delta_{ij}\right) = \bar{f}_i, \tag{5.10}$$

where

$$\tau_{ij}(u) := \overline{u_iu_j} - \bar{u}_i\bar{u}_j \tag{5.11}$$

is the sub-filter stress tensor (SFS).

Introducing a closure model $m(\bar{u})$ for $\tau(u)$ yields the LES equations

$$\partial_j v_j = 0, \quad \partial_t v_i + \partial_j \left(\sigma_{ij}(v) + m_{ij}(v) + p_v \delta_{ij}\right) = \bar{f}_i, \tag{5.12}$$

where p_v is the pressure that enforces the divergence-free constraint on the LES solution v, which in general differs from the filtered pressure $\bar{p}$ (see chapters 3 and 4 for a detailed discussion).

Since replacing m_{ij} with $m_{ij} + n\delta_{ij}$ for any scalar function n does not change the dynamics—the isotropic part is absorbed by the pressure p_v—it is convenient to choose m to be deviatoric (trace-free) and to fit it to the deviatoric part $\tau_{ij}^{\text{dev}} \coloneqq \tau_{ij} - \tau_{kk}\delta_{ij}/3$.

Depending on the choice of closure m, the symmetry properties of the original equations may be violated. To enforce Galilean invariance, it is standard practice to express m as a function of $\bar{A}$ rather than $\bar{u}$ [112], since $\bar{u} + a(t)$ and $\bar{u}$ have identical velocity gradients for any constant $a(t)$. Eddy viscosity models and Clark's gradient model depend solely on $\bar{A}$, and thus preserve Galilean invariance. More precisely, since $\bar{A}$ is invariant under the generalized Galilean transform of section 5.2.2, any closure depending solely on $\bar{A}$ preserves generalized Galilean invariance as well. In the following sections, we present two neural network closure architectures that additionally preserve rotational and reflectional symmetries.

5.3 TENSOR-BASIS NEURAL NETWORK

Consider a closure model that depends on the VGT $\bar{A}$. For notational simplicity, we drop the overbar on A, S, W, etc. Following tensor representation theory [171, 172], Pope [143] proposed expressing such a model as

$$m^{\text{TB}} \coloneqq \Delta^2 \sum_k \alpha_k(\lambda) T^{(k)}, \tag{5.13}$$

where Δ is the filter width, $(T^{(k)})_k$ is a tensor basis (TB) of predetermined size, and the coefficients α_k are scalar functions of the invariants λ of the VGT. These invariants are

$$\begin{aligned} \lambda_1 &\coloneqq \operatorname{tr}(S^2), & \lambda_4 &\coloneqq \operatorname{tr}(SW^2), \\ \lambda_2 &\coloneqq \operatorname{tr}(W^2), & \lambda_5 &\coloneqq \operatorname{tr}(S^2W^2). \\ \lambda_3 &\coloneqq \operatorname{tr}(S^3), \end{aligned} \tag{5.14}$$

Pope's original basis consists of 10 tensors (or 11 in the compressible case) [143], and was used by Ling et al. [103] in the context of RANS closures.

Based on the analysis of Smith [169], Stallcup et al. [175] showed that a slightly different basis, consisting of 7 tensors (or 8 in the compressible case),

is both minimal and complete, whereas Pope's basis is neither minimal nor complete. We adopt this basis, defined as

$$\begin{aligned} T^{(0)} &:= I, & T^{(4)} &:= SW - WS, \\ T^{(1)} &:= S, & T^{(5)} &:= WSW, \\ T^{(2)} &:= S^2, & T^{(6)} &:= S^2W - WS^2, \\ T^{(3)} &:= W^2, & T^{(7)} &:= WSW^2 - W^2SW. \end{aligned} \tag{5.15}$$

The fifth tensor $T^{(5)} := WSW$ was not included in Pope's basis, which instead included fifth-order terms in S and W.

Smith's basis: 7 tensors that are both minimal and complete, unlike Pope's original 10-tensor basis.

The key property of this formulation is that the invariants λ and basis tensors T transform appropriately under the rotation and reflection symmetry groups (i.e. the orthogonal group $O(3)$) of the Navier-Stokes equations, so the full model m^{TB} is equivariant regardless of the functional form of the coefficients α_k. For example, let $R \in \mathbb{R}^{3\times3}$ be a rotation matrix. The invariants, being scalar fields, satisfy $\lambda_1(A) = \mathrm{tr}(S^2) = \mathrm{tr}(RS^2R^T) = \mathrm{tr}((RSR^T)^2) = \lambda_1(RAR^T)$, where we use the invariance of the trace under orthogonal transformations: $\mathrm{tr}(\sigma) = \mathrm{tr}(R\sigma R^T)$ for all orthogonal R. Similarly, the basis tensors, being tensor-valued fields, satisfy $RT^{(2)}(A)R^T = T^{(2)}(RAR^T)$ (and analogously for the other basis tensors).

Key advantage of TBNNs: equivariance is guaranteed by the basis, with no constraints on the neural network architecture.

This motivates the tensor basis neural network (TBNN) approach: the coefficients α_k are predicted by a neural network NN, i.e., $\alpha = (\alpha_1, \dots, \alpha_K) = \mathrm{NN}(\lambda_1, \dots, \lambda_5)$. Ling et al. [103] introduced this approach for RANS closures using Pope's basis, and TBNNs have subsequently been applied to LES closure modeling [24, 145].

In our experiments, we employ the TBNN with the reduced and complete basis of Stallcup et al. [175]. For incompressible flow, we retain only the deviatoric part of the basis. To improve training stability, we normalize the VGT by its Frobenius norm as $A^* := A/\|A\|_F$, following Prakash et al. [145]. The model then takes the form

$$m^{\mathrm{TBNN}}(\bar{u}) := \Delta^2\|A\|_F^2 \sum_{k=1}^{7} \alpha_k(\lambda(A^*))T^{(k),\mathrm{dev}}(A^*), \tag{5.16}$$

where the invariants and basis tensors are computed from the normalized VGT. The identity tensor $T^{(0)}$ is omitted since its deviatoric part vanishes. The coefficients $(\alpha_k)_k$ are predicted by a standard feedforward neural network with 5 inputs $\lambda_1(A^*), \dots, \lambda_5(A^*)$ and 7 outputs $\alpha_1, \dots, \alpha_7$. The model is applied locally in physical space, independently at each grid point, so $m^{\mathrm{TBNN}}(\bar{u})(x)$ depends only on $A(x)$. We implement this as a convolutional neural network with kernel size $1\times1\times1$, meaning the convolutions do not access neighboring points. The convolutional framework is used for consistency with the G-conv model described below, and because it allows efficient batched evaluation over the entire spatial grid using GPU-accelerated libraries.

5.4 STRUCTURAL CLOSURE: CNNS

Direct structural closure: Convolutional neural networks

As an alternative to the TBNN, we consider structural closure models that predict the subgrid-scale stress tensor using a neural network, bypassing the tensor basis decomposition. This approach offers greater flexibility, as the network is not constrained to the span of the tensor basis, but symmetry preservation must instead be enforced through the network architecture.

5.4.1 *Feed-forward neural network closure*

We first define a standard feed-forward neural network closure that is not equivariant to rotation or reflection, serving as an unconstrained baseline. A feed-forward layer $\ell : u \mapsto v$ is defined by

$$v_i = \varphi\left(w_{ij}u_j + b_i\right), \tag{5.17}$$

where w_{ij} are weights, b_i are biases, and φ is a non-linear activation function. The input and output shapes are given by the size of the weight matrix w.

We define a neural structural closure model as

$$m^{\text{conv}}(\bar{u}) \coloneqq \ell_n \circ \ell_{n-1} \circ \ldots \circ \ell_1(\bar{A}). \tag{5.18}$$

This is a structural closure: the network directly predicts the six independent components of the symmetric, deviatoric stress tensor m_{ij}. The model is composed of n layers. For simplicity, and to maintain consistency with the local TBNN formulation, we apply the closure at each grid point independently. We therefore implement each layer as a convolutional layer with a point kernel of physical size $1 \times 1 \times 1$. Galilean invariance is enforced by using the VGT $\bar{A}$ as input (as shown in eq. (5.18)), since the velocity gradient is invariant under Galilean transforms. However, this architecture is not equivariant to rotation or reflection. To enforce these additional symmetries, we next introduce group-equivariant layers.

5.4.2 *Group-convolutions*

Since Galilean and scaling invariance are already enforced through the input representation and output prefactor (see section 5.5), we focus on the remaining symmetries and require the model to be equivariant to rotations and reflections in 3D. The continuous symmetry group is the *orthogonal group* $O(3)$. However, on a discrete Cartesian grid, we only enforce equivariance to rotations by multiples of $\pi/2$ and reflections aligned with the grid axes. Infinitesimal rotations cannot be represented exactly on a Cartesian grid. Note that the TBNN from section 5.3 is equivariant to all of $O(3)$ in the continuous setting, but after discretization it too preserves only discrete

rotations by multiples of $\pi/2$. The resulting symmetry group consists of all rotations and reflections along the x_1, x_2, and x_3 axes, which is the octahedral group, denoted G. This group has $|G| = 48$ unique elements. In C.1, we describe our parametrization of this group.

To make a layer ℓ group-equivariant, we require that $g\ell(u) = \ell(gu)$ for all group elements $g \in G$. A simple way to achieve this is to use group-convolutional layers [42].

In physical space $\mathbb{R}^3$, a group element $g \in G$ is represented by a roto-reflection matrix $R_g \in \mathbb{R}^{3\times3}$. These matrices can have negative entries, causing the signs of transformed physical quantities to change depending on the group element. For sign-sensitive activation functions such as $\mathrm{ReLU}(x) \coloneqq \max(0, x)$, this representation is problematic. While Shang et al. proposed concatenated activation functions compatible with signed permutation matrices [159], adding biases to layers operating on physical 3×3 matrices also breaks rotational equivariance, since biases do not transform under group actions. We therefore employ the *regular representation* space $\mathbb{R}^{|G|} = \mathbb{R}^{48}$ for the hidden layers of the neural network. In this representation, group elements are represented by permutation matrices $P_g \in \mathbb{R}^{48\times48}$ with entries that are either 0 or 1 [42, 43]. Intuitively, each of the 48 channels corresponds to one group element, and the group action simply permutes these channels.

The regular representation maps each group element to a permutation matrix, enabling the use of standard activation functions and biases.

For a layer mapping n_{in} regular representation vectors $u_j \in \mathbb{R}^{48}$ to n_{out} regular representation vectors $v_i \in \mathbb{R}^{48}$ as $v_i \coloneqq \varphi\left(w_{ij}u_j + b_i\right)$, the requirement is to find $w_{ij} \in \mathbb{R}^{48\times48}$ and $b_i \in \mathbb{R}$ such that

$$P_g\varphi\left(w_{ij}u_j + b_i\right) = \varphi\left(w_{ij}P_g u_j + b_i\right) \tag{5.19}$$

for all $g \in G$. Since only one entry in each row of P_g is non-zero, the element-wise activation φ commutes with P_g [43]. Intuitively, since P_g only reorders the entries of a vector (without mixing or sign-flipping them), applying φ element-wise before or after the permutation yields the same result. This simplifies the requirement to

$$P_g w_{ij} = w_{ij}P_g. \tag{5.20}$$

While the weight projection approach follows Cohen and Welling [42], our contribution lies in applying it to the octahedral symmetry group for 3D Navier-Stokes closure modeling, including the lifting and projection layers between physical tensor space and the regular representation. Let $\tilde{w}_{ij} \in \mathbb{R}^{48\times48}$ be an arbitrary weight matrix (for each i and j). Define

$$w_{ij} \coloneqq \sum_{g\in G} P_g^{-1}\tilde{w}_{ij}P_g \tag{5.21}$$

as a projected weight tensor. We then verify that w_{ij} commutes with P_g:

$$\begin{aligned} P_g w_{ij} &= \sum_{h\in G} P_g P_h^{-1} \tilde{w}_{ij} P_h \\ &= \sum_{h\in G} P_g P_h^{-1} \tilde{w}_{ij} P_h P_g^{-1} P_g \\ &= \sum_{h\in G} P_{gh^{-1}} \tilde{w}_{ij} P_{hg^{-1}} P_g \\ &= \sum_{h\in G} P_{hg^{-1}}^{-1} \tilde{w}_{ij} P_{hg^{-1}} P_g \\ &= \sum_{h\in G} P_h^{-1} \tilde{w}_{ij} P_h P_g \\ &= w_{ij} P_g, \end{aligned} \tag{5.22}$$

where we substituted $h \leftarrow hg^{-1}$, which is valid since right multiplication by g^{-1} is a bijection on G. Hence w_{ij} commutes with P_g for all $g \in G$ and all choices of $\tilde{w}_{ij}$.

For the first neural network layer, we need a similar result. The entries of the gradient tensor $\bar{A}$ are the input. The gradient tensor $\bar{A}$ transforms as

$$g\bar{A} = R_g \bar{A} R_g^T. \tag{5.23}$$

If we "flatten" the gradient tensor $\bar{A}$ into a vector $a \in \mathbb{R}^9$, we can combine the left and right multiplications by R_g into a single representation matrix $Q_g \in \mathbb{R}^{9\times 9}$. Then a transforms as $ga = Q_g a$. Repeating the argument above, we find that

$$w_i = \sum_{g\in G} P_g^{-1} \tilde{w}_i Q_g \tag{5.24}$$

gives an equivariant initial layer $\ell : a \mapsto v$ defined by

$$v_i = \varphi\left(w_i a + b_i\right), \tag{5.25}$$

where $\tilde{w}_i \in \mathbb{R}^{48\times 9}$ is an arbitrary weight tensor for each output channel i (there is only one tensor-valued input channel). Finally, in the last layer we predict the sub-filter stress tensor. We represent the 9 components as a flattened tensor, and the equivariant weights are given by

$$w_j = \sum_{g\in G} Q_g^{-1} \tilde{w}_j P_g, \tag{5.26}$$

where $\tilde{w}_j \in \mathbb{R}^{9\times 48}$ is an arbitrary weight tensor for each input channel j (there is only one tensor-valued output channel). We use no activation function and no bias in the final layer, since Q_g^{-1} does not commute with elementwise activation functions φ (unlike P_g which only has one entry per

row). Since the output tensor should be symmetric ($m_{ij} = m_{ji}$), we only retain 6 out of 9 output components.

For the initial, inner, and final layers, the mapping $\mathcal{P} : \tilde{w} \mapsto w$ can be seen as a projection step, retaining only the equivariant part of the original weights $\tilde{w}$. The group-convolutional closure model $m^{\text{G-conv}}$ therefore has a similar architecture as m^{conv} in (5.18), except that each layer $\ell^{\mathcal{P}} : u \mapsto v$ includes the projection as

$$v_i := \varphi\left(\mathcal{P}(\tilde{w}_{ij})u_j + b_i\right). \tag{5.27}$$

The final model is

$$m^{\text{G-conv}}(\bar{u}) := \ell^{\mathcal{P}}_n \circ \ell^{\mathcal{P}}_{n-1} \circ \ldots \circ \ell^{\mathcal{P}}_1(\bar{A}). \tag{5.28}$$

During training, the projection $\mathcal{P}$ is part of the computational graph, and gradients are backpropagated through it. The matrix representing $\mathcal{P}$ is precomputed once and reused at each gradient-descent step. After training, the projected weights $w = \mathcal{P}(\tilde{w})$ are precomputed and stored, so that at inference time $m^{\text{G-conv}}$ operates as a standard neural network with fixed weights w.

5.4.3 *Weight sharing for octahedral group convolutions*

Novel contribution: eigendecomposition-based weight sharing reduces the number of learnable parameters from 48×48 to 48 per hidden layer.

The eigendecomposition-based weight sharing described below is, to the best of our knowledge, a novel contribution of this work.

In fig. 5.1, we show the weight projection mechanism for classical convolutional neural networks (CNNs) in 1D. In CNNs, the group is the 1D-translation group. The projection $\mathcal{P}^{\text{naive}}$ is the same as in eq. (5.21), except that instead of summing over the roto-reflections in the octahedral group, we sum over all the 1D-translations on a periodic grid.

Classical CNNs exploit two forms of sparsity: weight sharing (each row of the weight matrix is a shifted version of the previous row, so only one row needs to be learned) and locality (most weights are constrained to zero). Our octahedral group-convolutional neural network (G-conv) employs only weight sharing, not locality constraints. The weight matrices are dense. Unlike 1D translations, where locality is well-defined (some translations are closer to the identity than others), roto-reflections in the octahedral group lack a natural notion of locality, making it unclear which weights should be zero. Moreover, the octahedral group has only 48 elements, whereas translation groups scale linearly/quadratically/cubically with grid size in 1D/2D/3D. The computational benefit of sparsity would therefore be small. However, locality constraints might become useful if smaller rotation angles (finer than $\pi/2$) were included.

Weight sharing for the octahedral group is less intuitive than for the 1D translation group. We therefore use eigenvalue decomposition to compress

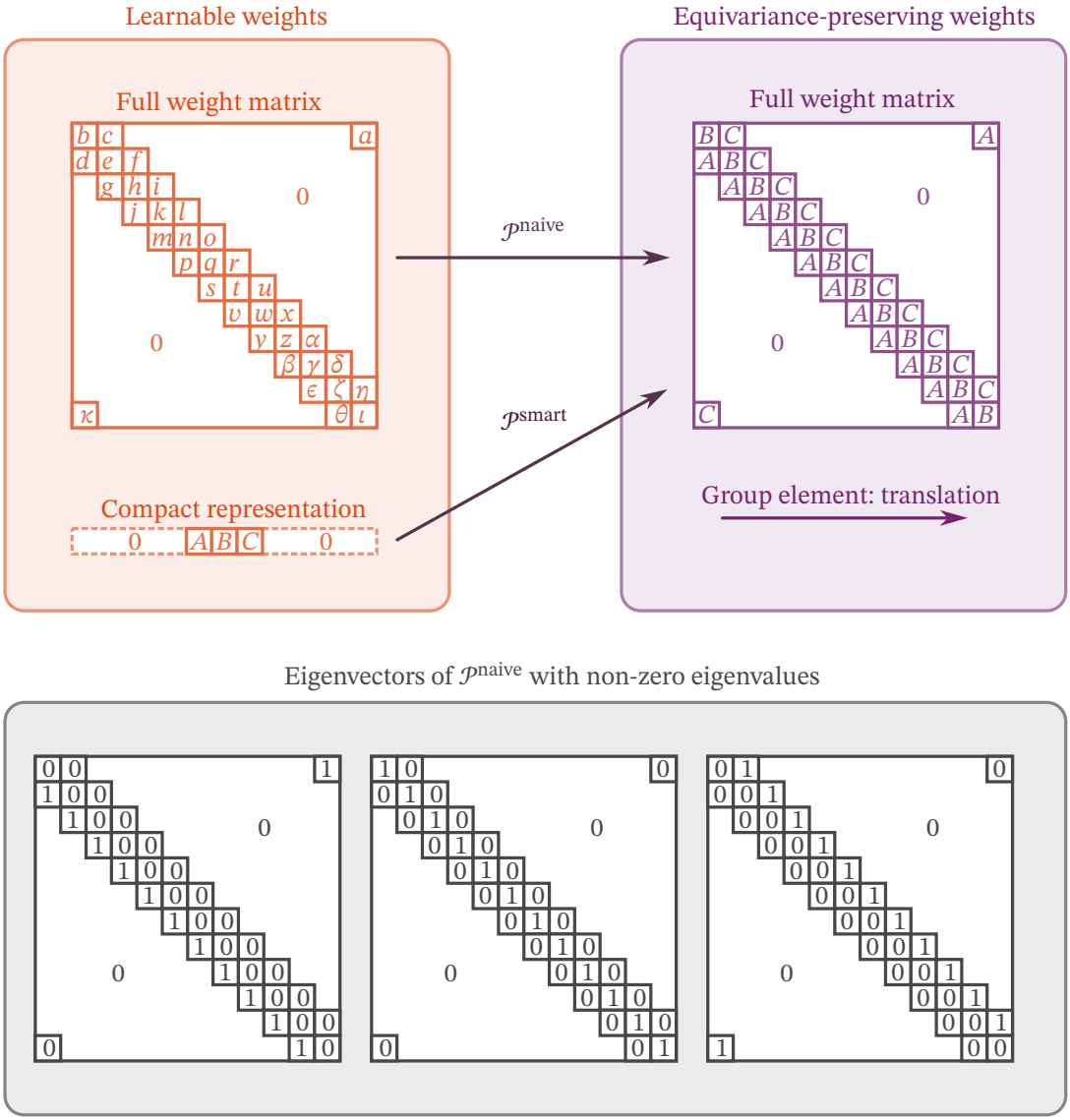


Figure 5.1: Example of weight projection for translation-equivariant group-convolutions in 1D (classical CNNs). For our octahedral group-convolutions, we employ a similar procedure, but without the constraint of locality (enforcing zeros).

the naive projection operator $\mathcal{P}$ (without weight sharing) as $\mathcal{P} = E\Lambda E^{-1}$, where Λ is a diagonal matrix of eigenvalues. We retain only the columns of E corresponding to non-zero eigenvalues, denoting the result by E^{trunc}. The initial and final layers have 9 non-zero eigenvalues (with weight matrix size 9×48), while the hidden layers have 48 non-zero eigenvalues (with weight matrix size 48×48). The simplified projector $\mathcal{P}^{\text{smart}} := E^{\text{trunc}} \in \mathbb{R}^{(9\times48)\times9}$ or $\mathcal{P}^{\text{smart}} := E^{\text{trunc}} \in \mathbb{R}^{(48\times48)\times48}$ therefore only takes 9 (initial and final layers) or 48 (inner layers) learnable weights as input (instead of 9×48 and 48×48 redundant full weights). For comparison, the naive projectors have sizes $\mathcal{P}^{\text{naive}} \in \mathbb{R}^{(9\times48)\times(9\times48)}$ and $\mathcal{P}^{\text{naive}} \in \mathbb{R}^{(48\times48)\times(48\times48)}$ (they map weight matrices to weight matrices). The eigenvectors with non-zero eigenvalues are visualized for the 1D-translation group in the bottom of fig. 5.1. In the figure, due to the locality constraint, only three eigenvectors (or "eigen-weight-matrices") have non-zero eigenvalues.

5.5 RESULTS

The code is available at https://github.com/agdestein/SymmetryCode.jl.

We consider forced homogeneous isotropic turbulence in a periodic box $\Omega = [0, 2\pi]^3$. A pseudo-spectral discretization is used for both DNS and LES (C.2), and details of the simulation setup and data generation are given in C.3.

We consider three non-data driven LES models:

- No-model: $m^{\text{no-model}}(\bar{u}) := 0$.
- Basic Smagorinsky:

$$m^{\text{smag}}(\bar{u}) := -2(C_s\Delta)^2|\bar{S}|\bar{S}, \tag{5.29}$$

  where $C_s = 0.17$ is the Smagorinsky constant and $|\bar{S}| = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$.
- Clark's gradient model:

$$m_{ij}^{\text{clar}}(\bar{u}) := \frac{\Delta^2}{12}\bar{A}_{ik}\bar{A}_{jk}. \tag{5.30}$$

These models illustrate two contrasting approaches to closure modeling. Smagorinsky is a purely dissipative functional model that does not account for backscatter of energy from small to large scales. Clark is a structural model derived by truncating the Taylor expansion of the filter kernel; it captures the tensor structure of the SFS but is insufficiently dissipative when used on its own [199].

We consider three data-driven LES models:

- Tensor basis Neural Network (TBNN):

$$m^{\text{TBNN}}(\bar{u}) := \Delta^2|\bar{A}|^2\sum_{k=1}^{7}\alpha_k T^{(k)}, \tag{5.31}$$

as described in section 5.3.

- Group-convolutional neural network (G-conv):

$$m^{\text{G-conv}}(\bar{u}) := \Delta^2|\bar{A}|^2 \ell_n^{\mathcal{P}} \circ \ell_{n-1}^{\mathcal{P}} \circ \ldots \circ \ell_1^{\mathcal{P}}(\bar{A}/|\bar{A}|), \tag{5.32}$$

  as described in section 5.4.

- Unconstrained convolutional neural network (Conv):

$$m^{\text{conv}}(\bar{u}) := \Delta^2|\bar{A}|^2 \ell_n \circ \ell_{n-1} \circ \ldots \circ \ell_1(\bar{A}/|\bar{A}|), \tag{5.33}$$

  with the same number of free parameters as $m^{\text{G-conv}}$.

All three data-driven models share the prefactor $\Delta^2|\bar{A}|^2$, which ensures dimensional consistency (the neural networks operate on dimensionless inputs $\bar{A}/|\bar{A}|$ and produce dimensionless outputs, while $\Delta^2|\bar{A}|^2$ provides the correct units of length^2/time^2) and enforces scaling invariance: under the scaling transform $\tilde{x} = ax$, $\tilde{u} = (a/b)u$ from section 5.2, both Δ and $|A|^{-1}$ scale as a, so $\Delta^2|A|^2$ is invariant. This choice matches the scaling of Clark's gradient model, which is second order in the velocity-gradient tensor. Since $\Delta^2|A|^2$ is the unique combination of Δ and $|A|$ with units of τ (length^2/time^2), incorporating higher-order contributions requires the viscosity ν. The dimensionless filter-scale Reynolds number $\text{Re}_\Delta := \Delta^2|\bar{A}|/\nu$ provides such a mechanism: a more general model takes the form $m = \Delta^2|\bar{A}|^2 f(\text{Re}_\Delta, \bar{A}/|\bar{A}|)$, where the current models correspond to f independent of Re_Δ. Buaria and Sreenivasan [29] adopted this approach for modeling velocity-gradient dynamics, learning the Reynolds number dependence explicitly, though using a non-dimensionalization based on the Taylor-scale Reynolds number Re_λ rather than a filter-scale quantity. We revisit this direction in section 5.6. All data-driven models are trained using an a-priori loss function, minimizing the error between predicted and reference stress tensors computed from discretization-consistent expressions that account for the numerical scheme (dealiasing, discrete Fourier transforms, etc.). For details about the training procedure, see C.4.

The closure models are evaluated in two settings:

- *A-priori*: The models predict the SFS using the exact filtered velocity field as input. We compute $m(\bar{u})$ for all snapshots $\bar{u}$ and models m.

- *A-posteriori*: The models serve as closures in LES simulations, initialized from the filtered DNS velocity field. The LES solution at time t is denoted $S_t(m, \bar{u}_0)$, where $\bar{u}_0$ is the initial condition and m is the closure model. The same adaptive time-stepping and forcing schemes as the DNS are used. We compute S_t at all snapshot times $t \in T$, at which the corresponding filtered DNS solutions are denoted $\bar{u}_t$.

Clark's gradient model is accurate at short times but becomes unstable without sufficient dissipation [199].

All models ran stably in the a-posteriori setting except for Clark, which diverged around $t = 2.1$.

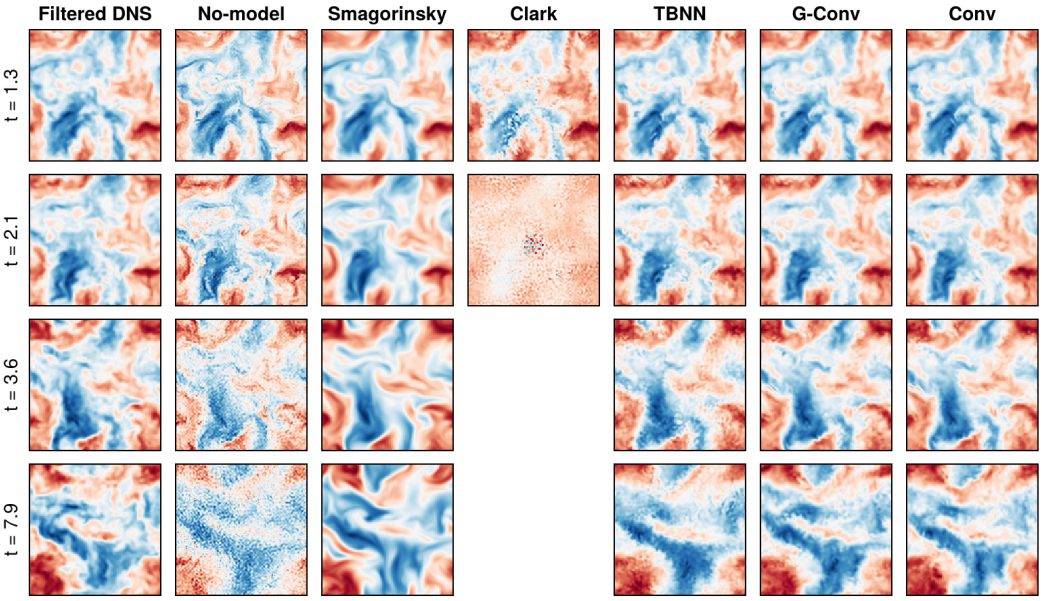


Figure 5.2: 2D section of predicted velocity component u_3 at $x_3 = 1$ at various times. At the initial time, all LES solutions are equal to the filtered DNS. For Clark, the solution became unstable around $t = 2.1$ and could not be finished.

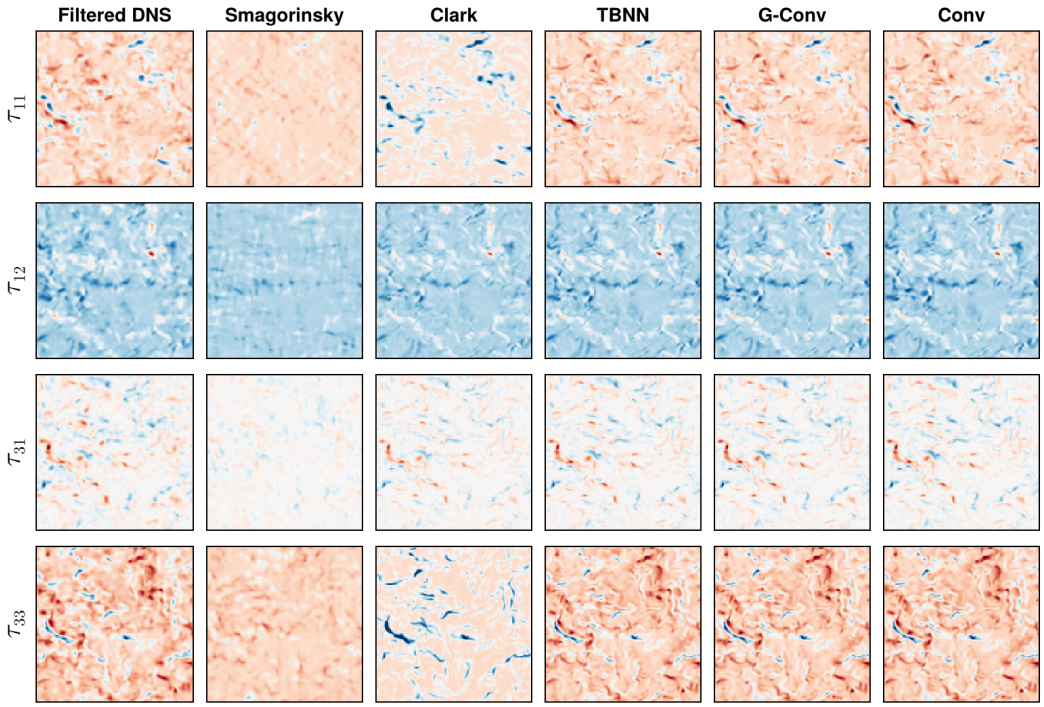


Figure 5.3: 2D section at $x_3 = 1$ of predicted SFS for a given snapshot $\bar{u}$. The reference SFS $\tau(u)$ is also shown.

VELOCITY FIELDS Figure 5.2 shows a 2D section of the z-component of the LES velocity fields at various times. All models produce similar large-scale features, except for No-model and Clark, which amplify small-scale features on the LES grid due to insufficient sub-filter dissipation. This is explained by fig. C.2: the Gaussian filter damps the spectrum *before* the LES-grid cutoff wavenumber, so a model without sufficient dissipation allows energy to accumulate at the highest resolved wavenumbers, making the solution resemble the DNS rather than the filtered DNS. Smagorinsky, being overly dissipative, appears excessively smooth compared to the filtered DNS.

TENSOR FIELDS Figure 5.3 shows a 2D section at $x_3 = 1$ of the SFS components τ_{11}, τ_{12}, τ_{31}, and τ_{33}, along with the corresponding a-priori model predictions. All four structural closures visually resemble the reference SFS, reproducing recognizable features. The functional Smagorinsky stress, by contrast, does not resemble the reference.

In table 5.1, we show four relative errors for the different models m:

Table 5.1: Various relative errors for different LES models. For Clark, the LES simulation was unstable and could not be finished.

Model	Tensor	Solution	Tensor-equi.	Solution-equi.
No-model	1.0000	0.3374	N.A.	1.317×10^{-15}
Smagorinsky	0.9398	0.2437	7.248×10^{-17}	1.074×10^{-15}
Clark	0.4544	N.A.	7.233×10^{-17}	5.563×10^{-15}
TBNN	0.4093	0.2154	1.699×10^{-16}	3.261×10^{-15}
G-conv	0.4196	0.2112	1.957×10^{-15}	2.066×10^{-15}
Conv	0.4225	0.2102	5.764×10^{-2}	1.736×10^{-2}

- The a-priori tensor error

$$\frac{1}{|U|} \sum_{u \in U} \frac{\|m(\bar{u}) - \tau(u)\|}{\|\tau(u)\|}, \tag{5.34}$$

  where U denotes the set of all DNS snapshots and $\|\cdot\|$ denotes the Frobenius norm summed over all grid points.

- The a-posteriori solution error

$$\frac{1}{|T|} \sum_{t \in T} \frac{\|S_t(m, \bar{u}_0) - \bar{u}_t\|}{\|\bar{u}_t\|}, \tag{5.35}$$

  where T contains all the snapshot times.

- The a-priori tensor-equivariance error

$$\frac{1}{|G|} \sum_{g \in G} \frac{\|gm(\bar{u}) - m(g\bar{u})\|}{\|m(g\bar{u})\|}, \tag{5.36}$$

  where gm and $g\bar{u}$ denote the group action of g on m and $\bar{u}$. This is computed for one snapshot $\bar{u}$ only.

- The a-posteriori solution-equivariance error

$$\frac{1}{|G|} \sum_{g \in G} \frac{\|gS_t(m, \bar{u}_0) - S_t(m, g\bar{u}_0)\|}{\|S_t(m, g\bar{u}_0)\|}. \tag{5.37}$$

  This is computed for one initial snapshot $\bar{u}_0$ and fixed time $t = 0.1$.

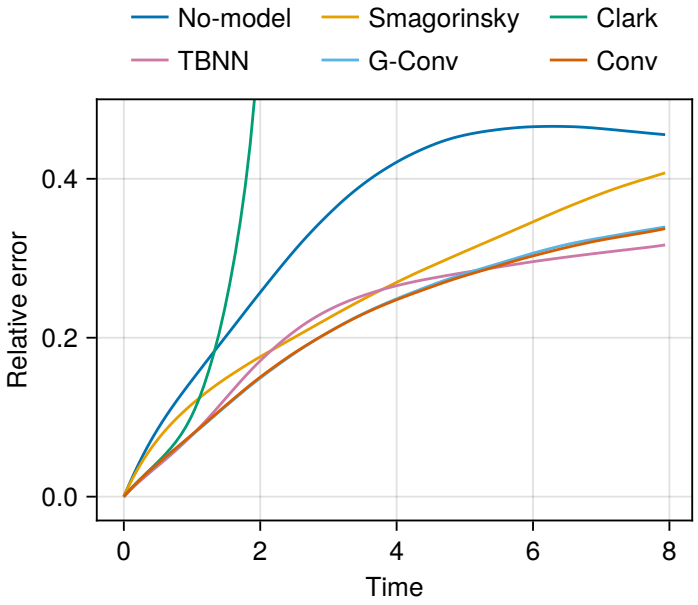


Figure 5.4: Relative LES solution errors as a function of time, see eq. (5.35).

PREDICTION ERRORS For tensor prediction, No-model performs worst ($\|0-\tau\|/\|\tau\| = 100\%$). Smagorinsky has an a-priori error of 94%, which is expected since it is a functional model designed to match energy dissipation rather than the stress tensor itself. Nevertheless, Smagorinsky achieves much lower a-posteriori errors than No-model because it provides subgrid-scale dissipation. The four structural models all outperform Smagorinsky and No-model in tensor prediction, ranked from least to most accurate: Clark, TBNN, Conv, G-conv. For a-posteriori solution error (Clark omitted due to instability), the ranking is: No-model, Smagorinsky, TBNN, G-conv, Conv, though the three data-driven models have nearly identical errors. The time evolution of the solution error is shown in fig. 5.4. Clark is as accurate as the data-driven models for short times but diverges for longer simulations.

Conv does not learn equivariance from isotropic data alone: 5.8% a-priori equivariance error despite symmetric training distribution.

TOTAL EQUIVARIANCE ERRORS All models except Conv exhibit equivariance errors at machine precision, confirming strict equivariance. Conv has a substantial a-priori equivariance error of 5.8%, indicating that it has not learned rotational and reflectional equivariance despite the training data being isotropic. The a-posteriori equivariance error for Conv is lower (1.7%), since the closure term is small relative to the resolved-scale dynamics, which are inherently equivariant. Despite these equivariance violations,

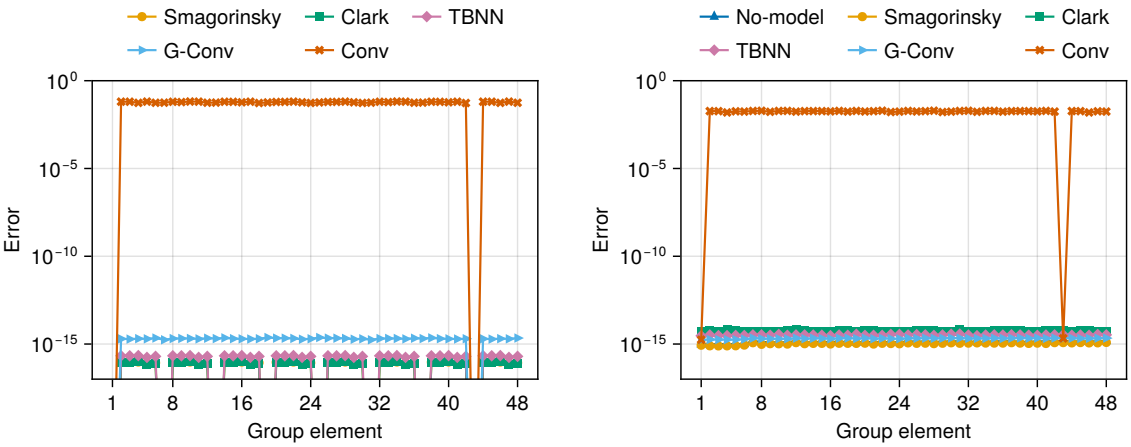


Figure 5.5: Equivariance errors. Left: a-priori errors. Right: a-posteriori errors.

Conv achieves prediction errors comparable to the equivariant models, suggesting that strict roto-reflectional equivariance may not be essential for accurate LES of isotropic turbulence. Note, however, that Conv still preserves Galilean and scaling invariance through its use of $\bar{A}$ (rather than $\bar{u}$) as input and the output scaling by $\Delta^2|\bar{A}|^2$.

INDIVIDUAL EQUIVARIANCE ERRORS In fig. 5.5, we show the individual contributions to the a-priori and a-posteriori equivariance errors (5.36)-(5.37) for each of the 48 group elements. All models except for Conv are at machine precision. For Conv, the a-priori error is around 6% for all the group elements except for g_1 and g_{43}, where the error is exactly 0. The corresponding roto-reflection matrices are

$$R_1 \coloneqq \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad R_{43} \coloneqq \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}. \tag{5.38}$$

Since g_1 is the identity element, there is no equivariance error. The element g_{43} is the only other element whose group action leaves 3×3 tensors unchanged ($R_{43}\sigma R_{43}^T = \sigma$ for all $\sigma \in \mathbb{R}^{3\times3}$). Since both the input and output of the closure model are 3×3 tensors, transforming the input or output with g_1 and g_{43} does not modify the result, leading to zero equivariance error.

ENERGY SPECTRA Figure 5.6 shows the time-averaged energy spectra for the different LES solutions. Time averaging reduces temporal fluctuations, providing a clearer comparison of the spectral energy distribution

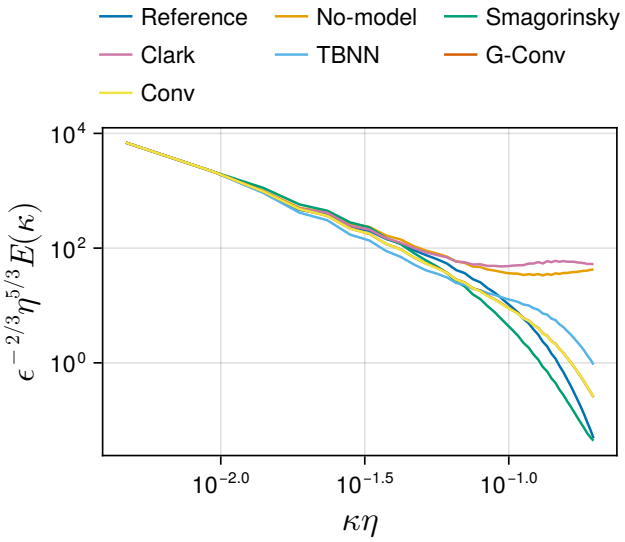


Figure 5.6: Time-averaged energy spectrum for different LES models.

across models. The Kolmogorov normalizations are applied such that the Kolmogorov spectrum is equal to one at the Kolmogorov length scale, informing us how far the different resolved scales are from the dissipation range (see eq. (C.41) for more details). For Clark, only spectra from before the instability at $t \approx 2.1$ are included. No-model, Clark, and TBNN are insufficiently dissipative, exhibiting a spurious energy pile-up at the highest wavenumbers. Smagorinsky is overly dissipative, with energy levels consistently below the reference. Conv and G-conv closely match the reference spectrum but remain slightly elevated at the highest wavenumbers.

TENSOR DISTRIBUTIONS To further distinguish the models, we examine their a-priori tensor distributions. Figure 5.7 shows the distributions of τ_{11}, τ_{12}, and the dissipation coefficient $\tau_{ij}\bar{S}_{ij}$, obtained via kernel density estimation over all snapshots. No-model is omitted (it would appear as a Dirac delta at zero). For τ_{11} and τ_{12}, Smagorinsky produces distributions that are too narrow. The four structural closures are accurate in high-density regions but differ in tail behavior: Clark fails to capture the tails, while TBNN, Conv, and G-conv accurately reproduce them. For the dissipation coefficient, Smagorinsky is slightly too dissipative and produces no backscatter (positive values), unlike the reference. Clark captures backscatter well but lacks sufficient forward transfer (negative values). TBNN accurately captures forward transfer but slightly underestimates backscatter. Conv and G-conv have backscatter profiles similar to TBNN but are overly dissipative.

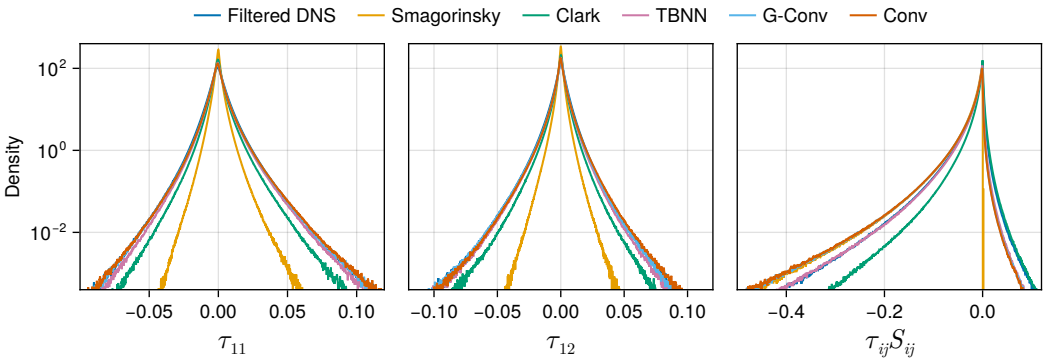


Figure 5.7: Distribution of SFS tensor and dissipation coefficients.

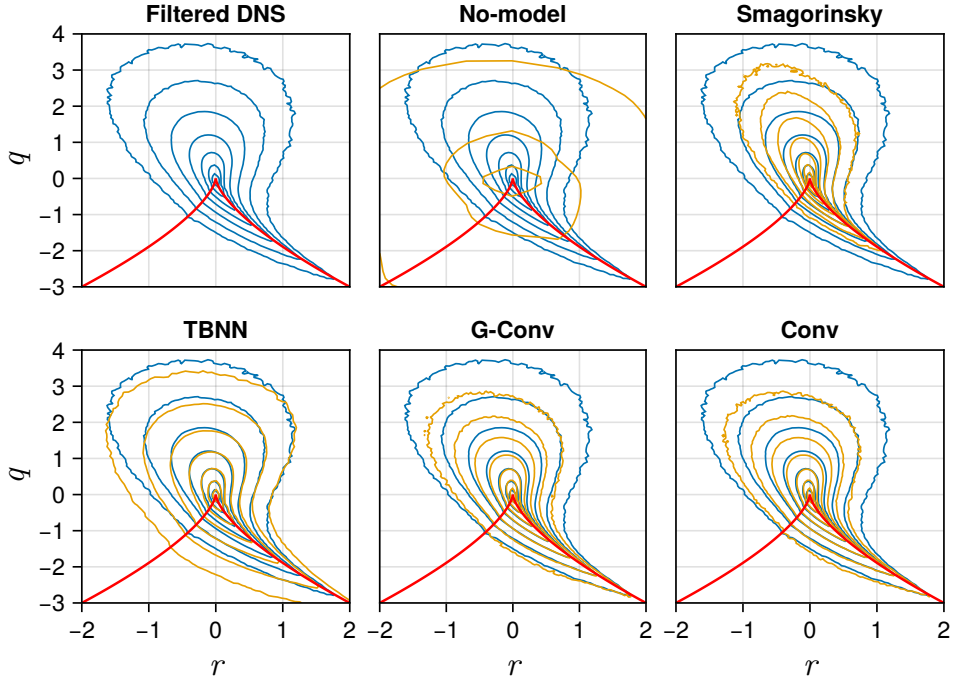


Figure 5.8: Isocontours of kernel density estimates of normalized (r, q)-pairs. The blue curves are for the filtered DNS. The yellow curves are for the LES solutions. The red line shows the Vieillefosse tail $(r/2)^2 + (q/3)^3 = 0$.

VGT INVARIANTS Figure 5.8 shows the joint distribution of the second and third velocity gradient invariants

$$q \coloneqq -\frac{1}{2}\operatorname{tr}(AA), \quad r \coloneqq -\frac{1}{3}\operatorname{tr}(AAA). \tag{5.39}$$

The (q, r)-distribution reveals differences in flow topology that aggregate error metrics miss.

The joint (q, r)-distribution characterizes the local flow topology [123]: regions with $q > 0$ are vorticity-dominated, while $q < 0$ indicates strain-dominated flow. The characteristic teardrop shape is a universal feature of turbulence, and its reproduction by the LES models provides a stringent test of their ability to capture the correct velocity-gradient statistics beyond what aggregate error metrics reveal. The distributions are computed in the a-posteriori setting using kernel density estimation over all LES snapshots. The filtered DNS density is shown in blue and the LES densities in yellow. The red line marks the Vieillefosse tail [123, 194], $(r/2)^2 + (q/3)^3 = 0$, along which an increased density is observed. To compare distributions, we normalize the invariants as $\tilde{q} \coloneqq q t_{\text{scale}}^2$ and $\tilde{r} \coloneqq r t_{\text{scale}}^3$, where $t_{\text{scale}} \coloneqq \langle \|\bar{A}\|_F^{-1} \rangle$ is the time-averaged inverse Frobenius of the filtered reference VGT. Isocontours are plotted at 7 logarithmically spaced levels between 10^{-4} and 10^{1}, with the innermost teardrop shape at the highest density (10^{1}). The (r, q)-pairs are computed in physical space.

Clark is omitted since long-term data is unavailable due to instability. No-model fails to produce the characteristic teardrop shape, lacking sufficient dissipation to prevent energy accumulation at high wavenumbers. Smagorinsky, being overly dissipative, produces a more compact distribution than the reference. TBNN matches the reference well for positive q (vorticity-dominated regions) but overestimates the tails for negative q (strain-dominated regions). Conv and G-conv accurately capture negative q but underestimate the tails for positive q. This is consistent with fig. 5.7, where Conv and G-conv were overly dissipative: excess dissipation suppresses vortical structures, reducing the extent of the distribution in the positive-q direction.

COMPUTATION TIME Table 5.2 reports computation times for training and inference (solving the LES equations for $t = 7.9$ time units). Only the three data-driven models require training; the Smagorinsky constant is prescribed. The DNS has significantly higher inference time than all LES models, due to both higher spatial resolution (more expensive FFTs) and smaller time steps needed to resolve turbulent time scales. No-model is slightly slower than Smagorinsky, likely because smaller adaptive time steps (see chapter C for details on the time integration scheme) are required to maintain stability without subgrid-scale dissipation. Conv is approximately 4 times slower than No-model. G-conv is approximately 7 times slower than Conv, because the 48-channel regular representation produces a much larger

Table 5.2: Computation time in seconds for training and inference.

Model	Parameters	Training [s]	Inference [s]
Reference (DNS)	0	0	8115.6
No-model	0	0	11.3
Smagorinsky	1	0	8.9
Clark	0	0	N.A.
TBNN	13,760	64.8	41.9
G-Conv	12,544	176.4	276.4
Conv	12,320	30.9	37.5

Table 5.3: Comparison of different closure models and their symmetry properties.

Model	Gal. inv.	Roto-refl. inv.	Scaling inv.	Reynolds inv.	Data-driven
No-model	Yes	Yes	Yes	Yes	No
Smagorinsky	Yes	Yes	Yes	Yes	No
Clark	Yes	Yes	Yes	Yes	No
TBNN	Yes	Yes	Yes	No	Yes
G-conv	Yes	Yes	Yes	No	Yes
Conv	Yes	No	Yes	No	Yes

network despite having the same number of trainable parameters. TBNN is slightly slower than Conv.

All three networks were designed with approximately the same number of trainable parameters. However, TBNN could potentially use a smaller network, since it predicts only 7 scalar coefficients for a basis that already encodes the tensor structure, rather than learning the full SFS directly. This could further reduce both training and inference costs.

5.6 CONCLUSION

Symmetries play a fundamental role in turbulence, and LES closures are expected to preserve them [170]. Neural networks do not automatically satisfy these symmetries. In this work, we compared three neural network closure architectures—a tensor basis neural network (TBNN), a group-convolutional neural network (G-conv), and an unconstrained convolutional neural network (Conv)—alongside classical closures (Smagorinsky and Clark's gradient model) for forced isotropic turbulence. All three data-driven models

enforce Galilean invariance and scaling symmetry by construction, through their use of the velocity-gradient tensor as input and the factor $\Delta^2|\bar{A}|^2$ in the output. TBNN and G-conv additionally preserve rotational and reflectional equivariance, while Conv does not. A comparison of the symmetry properties of all models is shown in table 5.3. The table also includes Reynolds number invariance, i.e. whether a model can be applied at Reynolds numbers different from the one it was designed for. The classical models are Reynolds-invariant because their functional form does not depend on a particular Reynolds number; they are defined in terms of local velocity gradients and a model constant. The data-driven models, by contrast, are trained on data generated at a specific Reynolds number and may not generalize to other regimes. We return to this point in the outlook below.

All three data-driven closures outperform the classical models in both a-priori tensor prediction and a-posteriori LES solution accuracy. Clark's gradient model, despite good short-time accuracy, became unstable around $t = 2.1$, illustrating the well-known difficulty of deploying purely structural closures without sufficient dissipation. Among the data-driven models, the direct structural approaches (Conv and G-conv) achieve lower a-priori tensor prediction errors than the TBNN. However, all three data-driven models produce nearly identical a-posteriori solution errors, indicating that the tensor prediction advantage of direct models does not translate into substantially better LES trajectories.

The energy spectra reveal further differences. TBNN exhibits a spurious energy pile-up at the highest wavenumbers, similar to the no-model and Clark baselines, indicating insufficient dissipation. Conv and G-conv closely match the reference spectrum but are slightly elevated at the highest wavenumbers. This is consistent with the dissipation distributions: Conv and G-conv are overly dissipative in the forward-transfer region, while TBNN better captures forward transfer but slightly underestimates backscatter. These complementary strengths suggest that combining the tensor basis formulation (which better captures forward transfer) with the flexibility of direct structural prediction (which better captures backscatter) could be a fruitful direction for future work.

Despite being trained on isotropic turbulence data, the unconstrained Conv network did not learn rotational and reflectional equivariance, exhibiting an a-priori equivariance error of 5.8%. At first glance, this suggests that strict roto-reflectional symmetry is not essential for accurate predictions, since the unconstrained Conv achieves comparable prediction errors. However, the velocity-gradient invariant distributions produced by Conv deviate more substantially from the reference than those of the symmetry-preserving models, indicating that symmetry constraints improve the physical consistency of the learned closure.

The computational cost of enforcing equivariance varies by approach. G-conv is approximately 7 times slower than Conv at inference due to the ex-

panded 48-channel regular representation required by the octahedral group. TBNN, which enforces equivariance through the tensor basis (i.e. through the input features and output reconstruction) rather than the network architecture, has inference cost comparable to Conv. This makes TBNNs an attractive option when symmetry preservation and computational efficiency are both required. Moreover, TBNNs predict only 7 scalar coefficients rather than the full stress tensor, so smaller networks may suffice, further reducing cost.

G-conv is 7× slower than Conv; TBNN achieves equivariance at comparable cost to Conv.

Several directions for future work remain. An important open question is generalization. As noted above, the data-driven models lack Reynolds number invariance (table 5.3): they are trained at a single Reynolds number and may not transfer to other regimes. Symmetry-preserving models may offer an advantage in this regard, as built-in physical constraints reduce the space of learnable functions and could improve out-of-distribution robustness. Buaria and Sreenivasan [29] explored Reynolds number generalization for velocity-gradient dynamics, but this has not yet been studied systematically for LES closures. First, this study considered only isotropic turbulence on a periodic domain. In anisotropic flows such as pipe or channel flows, rotational equivariance may not be physically appropriate: the SFS tensor itself may not be equivariant under arbitrary rotations due to macroscopic anisotropy persisting to the LES cutoff scales. To clarify: the Navier-Stokes equations themselves are equivariant, so rotating the domain and initial conditions and then time-stepping gives the same result as first time-stepping and then rotating. However, in anisotropic flows, the boundary conditions break rotational symmetry. Kolmogorov's hypothesis of local isotropy [56] suggests that small-scale turbulent fluctuations recover isotropy even when the large-scale flow is anisotropic, which would support enforcing rotational equivariance for sub-filter scale models. Yet this hypothesis may not hold near walls or at scales close to the energy-containing range, where anisotropy can persist down to the LES cutoff. In such cases, a closure model that enforces full rotational equivariance may be overly constrained. Wang et al. demonstrated relaxed group convolutions that accommodate partial symmetry breaking [201], providing a potential framework for extending our approach. Second, all neural networks in this study operate pointwise in space: the closure at each grid point depends only on the VGT in a single point. The full closure still depend on neighboring points through the VGT, which is computed from finite differences. Incorporating spatial context through larger convolutional stencils or graph-based message passing could improve accuracy, particularly for structural predictions. Third, we employed a-priori training throughout (as described in section 5.5). Combining symmetry-preserving architectures with a-posteriori training [107] could further improve long-term stability and accuracy, as demonstrated for non-symmetry-preserving models in chapters 3 and 4.

6

A DIFFERENTIABLE SOFTWARE SUITE FOR ACCELERATED SIMULATION OF TURBULENT FLOWS

This chapter is based on the following article: Syver Døving Agdestein and Benjamin Sanderse. A Differentiable Software Suite for Accelerated Simulation of Turbulent Flows. Apr. 2026. DOI: 10.48550/arXiv.2604.18536. arXiv: 2604.18536 [math].

The previous chapters developed discretization-consistent and symmetry-preserving closure models for LES. Training and deploying these models requires a numerical solver that supports both DNS data generation and neural-network-augmented LES. This chapter presents the software that was developed to this end.

6.1 INTRODUCTION

We present `IncompressibleNavierStokes.jl` [3] (hereafter referred to as `INS.jl`), an open-source software package written in the Julia programming language [20] for solving the incompressible Navier–Stokes equations using second-order staggered finite volume discretizations on Cartesian grids [71]. The package supports both direct numerical simulation (DNS) and large-eddy simulation (LES) in two and three dimensions, with various boundary conditions and non-uniform grids.

The code is archived at https://zenodo.org/records/19005988.

The package was initially based on the MATLAB codes INS2D/INS3D, where sparse matrices were precomputed and stored for all discrete operators [156]. While conceptually straightforward, storing sparse matrices for all the operators requires substantial memory, limiting the achievable resolution. `INS.jl` replaces these with matrix-free, hardware-agnostic kernels that are compiled for multi-threaded CPU or GPU from a single source implementation, drastically reducing memory requirements.

Original MATLAB code: https://github.com/bsanderse/INS2D

The solver is fully differentiable: hand-written adjoint kernels for all discrete operators are registered through the `ChainRules.jl` interface, enabling reverse-mode automatic differentiation through the entire solver using `Zygote.jl` [78]. This makes it possible to train neural network closure models *a posteriori*, while embedded in the LES solver.

The package is designed to be easy to modify and extend. Adding new physics—such as a temperature field for Rayleigh–Bénard convection [155], an Ornstein–Uhlenbeck process for stochastic forcing [74], an "evolve-filter-relax" scheme [79], or a neural network closure model for LES [2]—requires only implementing a new body force or right-hand side term with the appropriate interface.

CRediT author statement
***Syver Døving Agdestein**: Conceptualization, Formal analysis, Methodology, Software, Validation, Visualization, Writing – Original Draft*
***Benjamin Sanderse**: Funding acquisition, Project administration, Supervision, Writing – Review & Editing*

This package aims to provide the full computational pipeline—from DNS data generation through closure model training to LES evaluation—in a single environment and language. Julia was chosen for its ability to combine the ease of use of a high-level language with the performance of compiled

code [20], while its multiple-dispatch type system and composable package ecosystem make it straightforward to integrate GPU computing, automatic differentiation, and neural network libraries within a single codebase—without the inter-language interfacing required by frameworks such as Relexi [94] and SmartFlow [208], which couple Python-based machine learning libraries with external CFD solvers through middleware.

The key features that distinguish `INS.jl` are: (i) hand-written adjoint kernels for all discrete operators, enabling efficient reverse-mode differentiation through the entire solver; (ii) hardware-agnostic kernels compiled from a single source for multi-threaded CPU or GPU; (iii) seamless integration of neural network closure models from `Lux.jl` within the differentiable solver; (iv) support for both single and double precision arithmetic; and (v) memory optimizations that allow double-precision DNS at resolutions up to 840^3 on a single GPU.

This chapter is organized as follows. Section 6.2 presents the numerical methods, including the spatial discretization on a staggered Cartesian grid and the temporal discretization with pressure coupling. Section 6.3 discusses the software design, including hardware-agnostic kernels, differentiability, and memory optimization. Section 6.4 describes the integration of neural network closure models and the differentiable simulation pipeline. Section 6.5 describes the software development practices, including version control, documentation generation, testing, continuous integration, release management, and reproducibility. Section 6.6 compares numerical results for a turbulent channel flow simulation with reference data. Section 6.7 discusses hardware considerations and floating point arithmetic.

6.2 NUMERICAL METHODS

This section describes the spatial and temporal discretization of the incompressible Navier–Stokes equations used in `INS.jl`.

6.2.1 *Staggered Cartesian grid*

The d-dimensional domain $\Omega \coloneqq [a_1, b_1] \times \cdots \times [a_d, b_d]$ is partitioned into $N \coloneqq (N_1, \dots, N_d)$ finite volumes $\Omega_I \coloneqq \Delta^1_{I_1} \times \cdots \times \Delta^d_{I_d}$, where $I = (I_1, \dots, I_d)$ is a multi-index with half-integer entries $I_\alpha \in \{1/2,\ 2-1/2,\ \dots,\ N_\alpha - 1/2\}$. The volume is a product of 1D intervals $\Delta^\alpha_i = [x^\alpha_{i-1/2},\ x^\alpha_{i+1/2}]$ for $i \in \{1/2,\ 2-1/2,\ \dots\}$, with volume center coordinates $x^\alpha_i = (x^\alpha_{i-1/2} + x^\alpha_{i+1/2})/2$.

Following the staggered grid approach of Harlow and Welch [71], the pressure p is stored at volume centers x_I, while the velocity component u^α is stored at the faces $\Gamma^\alpha_I = \Omega_{I-h_\alpha} \cap \Omega_{I+h_\alpha}$, where $h_\alpha = e_\alpha/2$ is a half-index shift in the α-direction. This staggered placement naturally couples the pressure and velocity fields, avoiding the need for artificial stabilization.

To avoid index offset computations inside the computational kernels, we use the convention that each finite volume has exactly one pressure component and one of each velocity component (defined to the right of the volume center in their canonical direction). If a velocity component is required to the left of a volume near the boundaries, an additional ghost volume is added. Boundary values are included in the solution vectors, and before applying an operator kernel to a field, the boundary values are filled in (for example using a periodic extension). The memory footprint of including a ghost volume in a given direction is negligible.

6.2.2 *Discrete operators*

All discrete operators are built from the discrete derivative in the α-direction,

$$(\delta_\alpha \varphi)_I = \frac{\varphi_{I+h_\alpha} - \varphi_{I-h_\alpha}}{|\Delta^\alpha_{I_\alpha}|}, \tag{6.1}$$

which is a second-order approximation of $\partial\varphi/\partial x^\alpha$. Using this notation, the four operators appearing in the semi-discrete equations (D.5) are:

- **Divergence:** $(\delta_\alpha u^\alpha)_I = 0$ (mass conservation).
- **Diffusion:** $(\delta_\beta \delta_\beta u^\alpha)_I$ (applied twice for second-order derivatives).
- **Convection:** $(\delta_\beta(\eta^{\text{half}}_\beta u^\alpha \eta^{\text{lin}}_\alpha u^\beta))_I$, using a skew-symmetric form following Verstappen and Veldman [191] that ensures discrete conservation of kinetic energy. The product $u^\alpha u^\beta$ at off-diagonal positions requires interpolation with weights chosen to preserve skew-symmetry.
- **Pressure gradient:** $(\delta_\alpha p)_I$.

All operators are divided by the velocity volume sizes, giving them the same units as their continuous counterparts. The explicit stencil expressions for each operator, including the interpolation weights for the convective term and the adjoint kernels used for automatic differentiation, are given in chapter D.

6.2.3 *Semi-discrete equations*

Applying the discrete operators to the integral form of the Navier–Stokes equations yields a system of ordinary differential equations coupled with an algebraic constraint [154, 191]:

$$Du_h = 0, \tag{6.2}$$

$$\frac{\mathrm{d}u_h}{\mathrm{d}t} = F(u_h) - Gp_h, \tag{6.3}$$

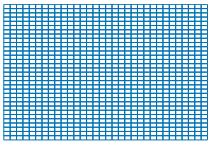
where D is the discrete divergence operator, G is the pressure gradient operator, u_h and p_h are the discrete velocity and pressure fields, and $F(u_h)$ collects the convective, diffusive, and forcing contributions. The momentum equation is only evaluated in the degrees of freedom of the velocity field. The velocity field also contains boundary values, for which the momentum equation is not defined.

The key property of the staggered discretization is that the divergence and pressure gradient operators are each other's transpose (up to a certain volume-wise scaling in the non-uniform case). This duality is a discrete analogue of the continuous integration-by-parts identity and is essential for discrete energy conservation [191].

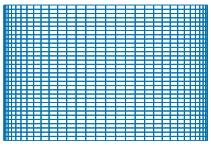
6.2.4 *Boundary conditions*

The software implements four types of boundary conditions, all through ghost volumes adjacent to the physical domain:

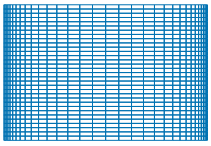
1. **Periodic.** The solution is periodic across the domain boundary. Ghost volumes duplicate values from the opposite side.
2. **Dirichlet.** The velocity is prescribed on the boundary, e.g., $u = 0$ for no-slip walls. Supports spatially and temporally varying boundary data $u = u_{\mathrm{BC}}(x, t)$.
3. **Symmetric.** The normal velocity vanishes at the boundary, while tangential components have zero normal gradient. Used to exploit planes of symmetry.
4. **Outlet.** The stress is prescribed on the boundary, with zero-gradient conditions on the velocity.



Uniform grid.



Cosine grid.



Tanh grid.

6.2.5 *Non-uniform grids*

Non-uniform grids are used to concentrate resolution near boundaries or regions of interest. Three grid stretching functions are provided. The *cosine grid* distributes $N + 1$ points on $[a, b]$ as

$$x_i = a + \frac{1 - \cos(\pi i/N)}{2}(b - a), \quad i = 0, \dots, N, \tag{6.4}$$

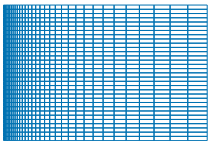
which clusters points near both ends. The *hyperbolic tangent grid* [182] distributes points as

$$x_i = a + \frac{b - a}{2}\left(1 + \frac{\tanh\big(\gamma(2\tilde{x}_i - 1)\big)}{\tanh(\gamma)}\right), \tag{6.5}$$

where $\tilde{x}_i = i/N$ is a uniform parameter and $\gamma > 0$ controls the degree of stretching. A geometric stretching function

$$x_i = a + (b-a)\frac{1-s^i}{1-s^N} \tag{6.6}$$

is also available for clustering near one end.



Stretched grid.

6.2.6 *Temporal discretization and pressure coupling*

Time integration is split into a momentum update and a pressure projection step to enforce the divergence-free constraint.

6.2.6.1 *Pressure Poisson equation*

The pressure field is determined at each time step (or sub-step) by enforcing the divergence-free constraint (6.2). Differentiating the mass equation in time and substituting the momentum equation (6.3) yields the discrete pressure Poisson equation

$$Lp_h = DF(u_h), \tag{6.7}$$

where L is a symmetric positive (semi-)definite discrete Laplacian. In the absence of pressure boundary conditions, the pressure is determined only up to a constant, which does not affect the velocity field.

Three pressure solvers are available:

1. **Spectral solver.** For fully periodic domains with uniform grids, the Laplace operator L is diagonal in Fourier space. The transfer function $\hat{L}$ is written out explicitly, and the pressure is computed as $p_h = \mathrm{DFT}^{-1}\,\hat{L}^{-1}\,\mathrm{DFT}\,r$, where r is the right-hand side and DFT denotes the d-dimensional discrete Fourier transform. To preserve the average pressure, $\hat{L}_{1,1}$ is set to one.

2. **Direct solver.** The Laplace operator is singular, so we directly incorporate the zero-mean constraint by solving the augmented system

$$\begin{pmatrix} L & e \\ e^{\mathsf{T}} & 0 \end{pmatrix}\begin{pmatrix} p_h \\ \lambda \end{pmatrix} = \begin{pmatrix} DF(u_h) \\ 0 \end{pmatrix}. \tag{6.8}$$

   The augmented matrix is invertible and symmetric, but it is no longer positive. We therefore employ the "square-root free" Cholesky decomposition LDL^T (the L here is not the Laplace matrix, but a lower-triangular factor). This requires assembling and storing the sparse Laplace matrix, but only a single factorization is needed since the Laplace matrix is constant in time.

3. **Iterative solver.** The conjugate gradient method is applied using the Laplace kernel directly (a chain of scaling, divergence, and pressure gradient kernels), without assembling the matrix. The AMGX solver is also available for GPU.

The Poisson solver is self-adjoint, since L is symmetric. Its pullback (for reverse-mode automatic differentiation) is therefore itself a Poisson solve.

6.2.6.2 *Time integration methods*

The semi-discrete system (6.3)–(6.2) forms a system of differential-algebraic equations. The software provides a library of time integration methods, including explicit Runge–Kutta methods (e.g., SSP33, DOPRI6, Wray3) and the one-leg β-method of Verstappen and Veldman [190]. At each Runge–Kutta sub-step, the pressure Poisson equation (6.7) is solved to project the velocity onto the divergence-free subspace. This ensures that the velocity is divergence-free at every sub-step, not only at the end of the time step, and that the correct order of accuracy of the Runge–Kutta method is obtained.

Adaptive time stepping is supported based on a CFL condition that considers both convective and diffusive stability limits.

6.3 SOFTWARE IMPLEMENTATION

This section describes the key implementation choices that enable correctness, performance, and differentiability in `INS.jl`.

6.3.1 *Design philosophy*

The discrete operators in the source code map directly to the equations presented in section 6.2. A reader familiar with the mathematical formulation should be able to read the source code without difficulty. For example, the divergence operator $\sum_{\alpha=1}^{d}(\delta_\alpha u^\alpha)_I$ is implemented as the kernel

```
divergence(u, D, I) = sum(1:length(I)) do a
    (u[a][I] - u[a][left(I, a, 1)]) / D[a][I[a]]
end
```

Here, `I` is always a Cartesian index of integers: a quantity `phi[I]` corresponds to φ_I if φ is defined at pressure points, and to φ_{I+h_α} if defined at the α-velocity points.

Wherever possible, the code follows a pure functional programming style: operators are pure functions that take a field as input and return a new field as output, without modifying any of their arguments. This makes it straightforward to compose operators, reason about correctness, and reuse

parts of the code in other solvers. Pure functions are also naturally compatible with reverse-mode automatic differentiation, as discussed in section 6.3.4.

However, pure functions allocate a new output array on every call, which conflicts with the memory constraints of high-resolution DNS (section 6.3.3). To address this, the package also provides *mutating* variants of all operators, which write their result into a preallocated output buffer. Following Julia convention, mutating functions are distinguished by an exclamation mark suffix (e.g., `divergence!` vs. `divergence`). The mutating variants are used in the time-stepping loop, where memory must be carefully managed, while the pure variants are used when differentiability is required. This separation keeps the mutating code isolated and clearly identified, preserving the benefits of functional style for the majority of the codebase.

The package is designed for extensibility. The solver acts on a "state"—a named tuple of fields that by default contains only the velocity field. Additional scalar fields (such as a temperature field for Rayleigh–Bénard convection) can be added to this state, and the right-hand side function (excluding the pressure projection) is user-defined. Adding new physics therefore requires only implementing a new right-hand side function acting on the augmented state, without modifying the solver itself. New closure models can similarly be added by implementing a function with the appropriate interface.

6.3.2 *Hardware-agnostic kernels*

All differential operators (divergence, pressure gradient, diffusion, convection) are implemented as hardware-agnostic kernels using `KernelAbstractions.jl` [39] as in `WaterLily.jl` [204]. The same kernel code is compiled for multi-threaded CPU or GPU, depending on the type of the input arrays: if called on CPU arrays, the kernel is compiled to a multi-threaded CPU loop; if called on GPU arrays, it is compiled to a GPU kernel. The kernels are written in a dimension-agnostic style, so that the same code handles both two- and three-dimensional problems.

The kernels operate in-place, overwriting preallocated temporary arrays rather than allocating new memory. These are the mutating operator variants (with `!` suffix) described in section 6.3; their use is confined to the time-stepping loop, where memory must be carefully managed. In-place operation is important for performance, particularly on GPUs where repeated memory allocation during time stepping would dominate the computational cost.

6.3.3 *Memory optimization for single-GPU DNS*

A three-dimensional DNS with N grid points per direction requires storing velocity fields with dN^d components and a pressure field with N^d compo-

The Dutch national supercomputer Snellius outside CWI. A single H100 GPU on Snellius can run a double-precision DNS with $N = 840$ in each dimension.

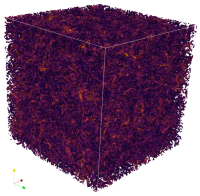
nents, along with temporary arrays for intermediate computations during time stepping. The goal is to fit all required arrays in the memory of a consumer GPU, such as an NVIDIA RTX 4090 with 24 GB of memory, or a datacenter GPU, such as an NVIDIA H100 with 94 GB of memory.

Three techniques are used to minimize memory consumption:

1. **Array reuse.** The mutating operator variants (section 6.3.2) overwrite preallocated temporary arrays rather than allocating new memory. This is where the departure from pure functional style pays off: the same buffer arrays are reused across operators within a single time step, avoiding the allocation of new arrays for every intermediate result.

2. **Low-storage Runge–Kutta methods.** Classical Runge–Kutta methods require storing all intermediate stages simultaneously, while low-storage variants (such as Wray3 [206]) require only two or three storage registers per degree of freedom, independent of the number of stages.

3. **Lazy computation of tensor components.** Rather than assembling the full $d \times d$ tensor $u^\alpha u^\beta$ at once, each component is computed and consumed on the fly. This removes the $d^2 N^d$ memory footprint of the convective term.



Isosurfaces of Q-criterion for a decaying turbulence simulation colored by velocity magnitude.

With these techniques, double-precision DNS with $N = 840$ was achieved on a single H100 in a triply periodic domain.

6.3.4 *Differentiability and adjoint operators*

When training neural network closure models, having access to the gradient of the loss function can greatly speed up the optimization process (see chapter 3). Reverse-mode automatic differentiation (AD) computes these gradients efficiently by propagating adjoints through the computational graph in reverse order.

Consider a program $f = f_N \circ \cdots \circ f_1$ composed of elemental building blocks f_n with intermediate states $x_{n+1} = f_{n+1}(x_n)$. If $x_N = f(x_1)$ is scalar (e.g., a loss function), the gradient of f with respect to the input x_1 can be computed using the pullback functions

$$\bar{f}_{n+1}(x_n) : \bar{x}_{n+1} \mapsto \bar{x}_n := \left(\frac{\mathrm{d}f_{n+1}}{\mathrm{d}x_n}(x_n)\right)^\mathsf{T} \bar{x}_{n+1}, \tag{6.9}$$

The notation $\overline{(\cdot)}$ here refers to the adjoint variable, and not to the filter in LES.

where $\bar{x}_n$ denotes the adjoint (cotangent) variable. Starting from the seed $\bar{x}_N = 1$, the back-propagation relation $\bar{x}_n := \bar{f}_{n+1}(x_n)(\bar{x}_{n+1})$ yields the gradient $\mathrm{d}f/\mathrm{d}x(x_1) = \bar{x}_1$. Note that the forward states x_n must be computed first before back-propagation can be performed.

We use `Zygote.jl` [78] to automatically generate the back-propagation code. Zygote expands any function f it encounters until it finds an elemental building block with a *pullback rule* (or "rrule") defined through the

`ChainRules.jl` framework. Standard operations such as matrix multiplication and neural network layers have pre-defined rules. However, since our discrete operators are implemented as matrix-free kernels rather than sparse matrix multiplications, their pullback rules are not available automatically. A key contribution of this work is the provision of hand-written adjoint kernels for all discrete operators, enabling efficient reverse-mode differentiation through the entire solver. Because `Zygote.jl` does not support mutation of arrays, the pure (non-mutating) operator variants are used during differentiation. The mutating variants used in the forward time-stepping loop (section 6.3.2) are bypassed by Zygote in favor of their pure counterparts, for which the hand-written pullback rules are defined.

Let $\kappa : u \mapsto \varphi$ be a kernel mapping a (possibly vector) field $u = (u_J)_J$ to a field $\varphi = (\varphi_I)_I$. The pullback kernel $\bar{\kappa}_u : \bar{\varphi} \mapsto \bar{u}$ at a given index J is

$$\bar{u}_J^\beta = \left\langle \bar{\varphi}, \frac{\mathrm{d}\varphi}{\mathrm{d}u_J^\beta}(u) \right\rangle = \sum_{\alpha=1}^{d} \sum_I \bar{\varphi}_I^\alpha \frac{\mathrm{d}\varphi_I^\alpha}{\mathrm{d}u_J^\beta}(u), \tag{6.10}$$

where $\bar{\varphi}$ is the incoming adjoint and $\bar{u}$ is the outgoing adjoint. If u or φ is scalar, the corresponding dimension index and sum are omitted. Since the kernels are local, most of the derivatives in the pullback are zero, and the sum over I reduces to a small neighborhood around J.

Except for convection, all operators are linear, and their pullbacks reduce to the transpose of the operator. The convection kernel is nonlinear but straightforward to differentiate. The explicit expressions for all kernels and their pullbacks are given in chapter D. When a user adds new physics—such as buoyancy, Coriolis terms, or reaction terms—they can implement the additional terms using standard vectorized Julia code, which `Zygote.jl` can differentiate automatically. Hand-written adjoint kernels are only needed if the user requires the performance of custom GPU kernels for the new terms; in that case, the corresponding pullback rules must also be provided.

All adjoint kernels are tested for correctness by comparing against finite difference gradients. For each operator κ, we verify that

$$\langle \bar{\varphi}, \kappa(u + \varepsilon\, \delta u) - \kappa(u) \rangle \approx \varepsilon \langle \bar{\kappa}_u(\bar{\varphi}), \delta u \rangle \tag{6.11}$$

for random perturbations δu and adjoint seeds $\bar{\varphi}$, at sufficiently small ε.

6.4 NEURAL NETWORK CLOSURE MODELS

A central motivation for developing `INS.jl` is the seamless integration of neural network closure models within the solver, providing the full computational pipeline—from DNS data generation through closure model training to LES evaluation—in a single differentiable framework.

Large-eddy simulation replaces the expensive resolution of all turbulent scales (DNS) with a coarse-grid simulation supplemented by a closure model

that accounts for the effect of unresolved sub-grid stresses. Classical closure models based on eddy viscosity, such as the Smagorinsky model [168] and the WALE model [130], are implemented in the software and can serve as baselines—for a full list, see section D.3.

As explored in chapters 3 and 5, an alternative to classical closure models is to use neural networks to learn the closure term from data. The software supports this through the following components:

- Neural network components from `Lux.jl`, a Julia framework for parameterized function approximation.
- Reverse-mode automatic differentiation through `Zygote.jl`, using the hand-written adjoint kernels described in section 6.3.4.
- A-posteriori training: the neural closure model is embedded in the LES solver and trained by differentiating through the entire time integration, unrolling the solver for a number of time steps and backpropagating through them.

This differentiable simulation capability distinguishes `INS.jl` from most existing CFD solvers, where neural network training typically requires an external coupling framework. Here, the hand-written adjoint kernels (section 6.3.4), the hardware-agnostic GPU kernels (section 6.3.2), and the neural network library all operate within the same Julia environment, avoiding the overhead and complexity of inter-language communication.

A challenge of reverse-mode differentiation through a time-stepping loop is that `Zygote.jl` requires the pure (non-mutating) operator variants, which allocate new arrays for every intermediate result. In the forward simulation, the mutating variants avoid this overhead (section 6.3.2), but during training the forward states at each sub-step must be stored for the backward pass, requiring continuous memory allocation. The resulting garbage collection overhead can affect performance, particularly for long training trajectories.

The neural closure modelling capabilities of `INS.jl` are used in the studies presented in chapter 3.

6.5 SOFTWARE DEVELOPMENT PRACTICES

While the preceding sections describe the mathematical and computational content of the software, the present section describes how the software is *developed*, tested, documented, and released. These practices—version control, automated testing, continuous integration, and documentation generation—are standard in professional software engineering but are less commonly adopted in scientific computing, where code is often developed by individual researchers without formal processes. We describe the practices used in `INS.jl` in some detail, as they are essential for the long-term maintainability

and correctness of the code. An overview of the different components and their interactions is shown in fig. 6.1.

6.5.1 *Version control*

The source code is managed with Git and hosted on GitHub. Every change to the code is recorded as a commit, providing a complete history of what was changed, when, and by whom. This makes it possible to trace the origin of any line of code, identify when a bug was introduced, and revert changes if needed.

Development follows a branch-based workflow. New features and bug fixes are developed on dedicated branches (e.g. `fix-CFL`) and merged into the main branch via pull requests. Pull requests allow changes to be reviewed and tested before they are accepted into the main codebase. Commit messages follow the conventional commits format (e.g., `feat: add default time step`, `docs: update examples to new syntax`), which makes the history easy to browse and filter by change type.

6.5.2 *Docstrings*

All public functions and types are documented with docstrings: inline documentation strings placed directly in the source code, adjacent to the function they describe. In Julia, docstrings are written in Markdown and placed immediately before the function definition. For example:

```
"""
Create a nonuniform grid of ‘N + 1‘ points from ‘a‘ to
‘b‘ using a cosine profile, i.e.

‘‘‘math
x_i = a + \\frac{1}{2} \\left(
    1 - \\cos \\left( \\pi \\frac{i}{n} \\right)
\\right) (b - a), \\quad i = 0, \\dots, N
‘‘‘

See also [‘stretched_grid‘](@ref).
"""
function cosine_grid(a, b, N)
    i = 0:N
    @. a + (b - a) * (1 - cospi(i / N)) / 2
end
```

Docstrings serve two purposes: they document the code for developers reading the source, and they are extracted automatically by the documentation

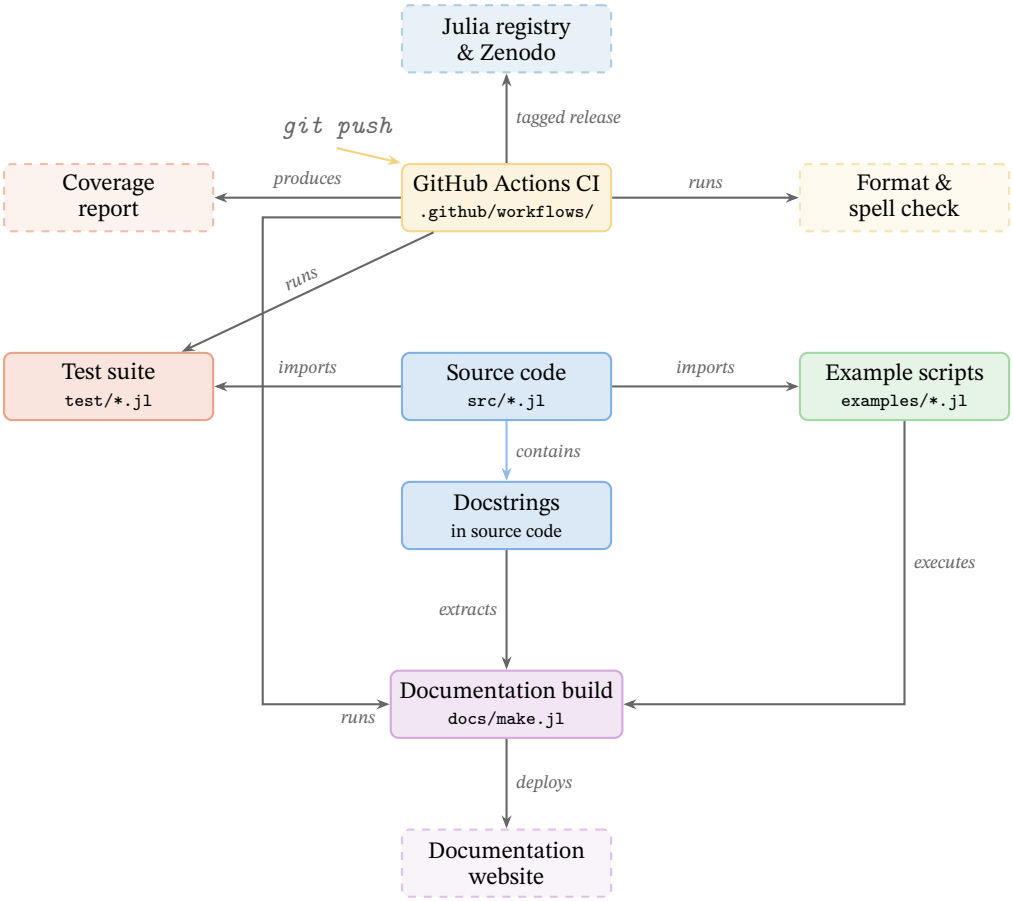


Figure 6.1: Overview of the software project components and their interactions. The source code (center) is imported by the test suite and example scripts, and contains docstrings that are extracted during the documentation build. Example scripts are executed by `Literate.jl` during the documentation build, producing the documentation website. A `git push` triggers GitHub Actions workflows that run the tests, build the documentation, and check code formatting and spelling. Tagged releases are published to the Julia General registry and archived on Zenodo. Solid boxes represent project components; dashed boxes represent generated outputs.

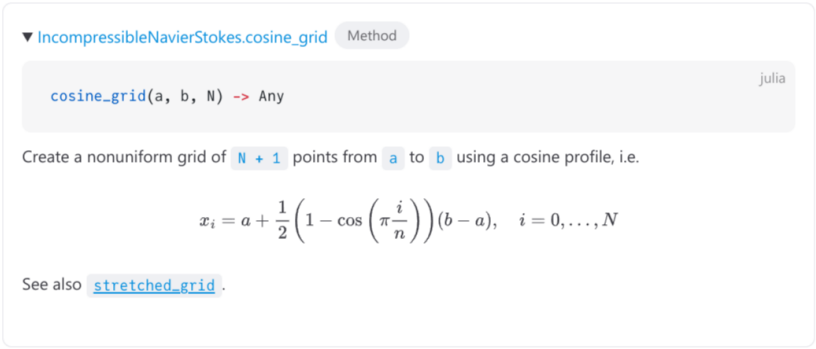


Figure 6.2: Rendered documentation from a docstring.

generator to produce a searchable API reference. This is shown in fig. 6.2. Cross-references between functions are created using the `@ref` macro.

6.5.3 *Documentation generation*

The package documentation is generated using `Documenter.jl`, which extracts docstrings from the source code, combines them with hand-written guide pages, and produces a complete documentation website (see fig. 6.3). The website is rendered using `VitePress` via the `DocumenterVitepress.jl` backend, and bibliographic references are handled by `DocumenterCitations.jl`.

The documentation build is defined in a script (`docs/make.jl`) that specifies the structure of the documentation, which modules to extract docstrings from, and how to process example scripts. The generated website includes an API reference (automatically collected from docstrings), tutorial examples (generated from literate scripts, described below), and guide pages.

6.5.4 *Literate programming for examples*

Tutorial examples are written as Julia scripts with embedded Markdown comments, using `Literate.jl`. Lines beginning with `#` are treated as Markdown text, while the remaining lines are Julia code. During the documentation build, `Literate.jl` converts each script into a Markdown page with executable code blocks. The code is then run by `Documenter.jl`, and any output—including plots and figures—is captured and included in the published documentation. Figures in the examples are produced using `Makie.jl` [47]. This ensures that all tutorial examples are always up to date with the current version of the code: if a breaking change is made to the API, the documentation build will fail, immediately signalling that the examples need to be updated.

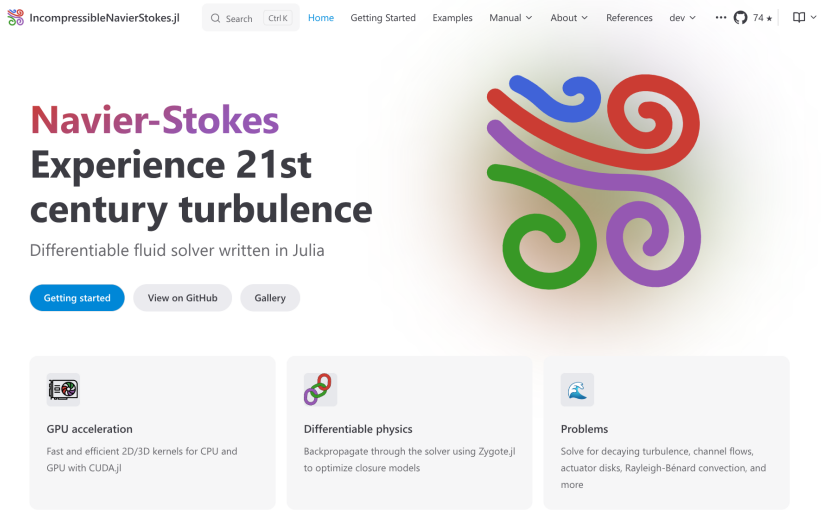


Figure 6.3: Generated documentation website.

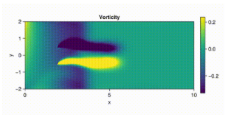
The package includes fifteen example scripts covering a range of configurations: periodic turbulence in two and three dimensions, lid-driven cavity flows, turbulent channel flow, backward-facing steps, actuator disk models, Rayleigh–Bénard convection, and Rayleigh–Taylor instability. They are shown in fig. 6.4. To manage computational cost, only a subset of the examples is executed during the documentation build; the remaining scripts are included as non-executed code listings.

6.5.5 *Continuous integration*

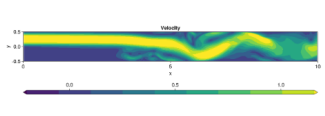
Every push to the main branch, every pull request, and every tagged release triggers a set of automated workflows on GitHub Actions. Continuous integration ensures that the code is always in a working state: any change that breaks the tests, the documentation, or the code formatting is detected immediately, before it can be merged.

The following workflows are configured:

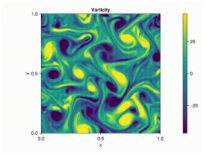
1. **Unit tests.** The full test suite is run on the latest stable Julia version on Ubuntu. The workflow has a timeout of 60 minutes to avoid runaway computations. Code coverage is measured and uploaded to Codecov, to see which lines of code covered by the tests. For pull requests, intermediate builds are cancelled when a new commit is pushed, avoiding unnecessary resource usage.



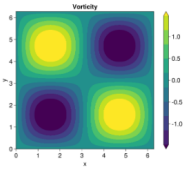
Actuator (2D)



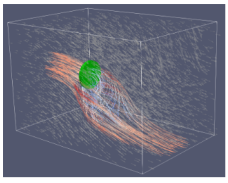
Backward facing step (2D)



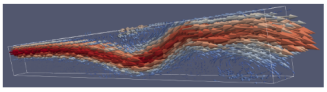
Decaying turbulence (2D)



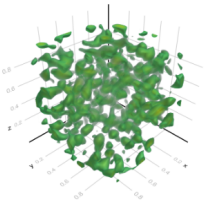
Taylor-Green vortex (2D)



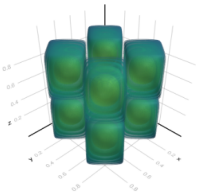
Actuator (3D)



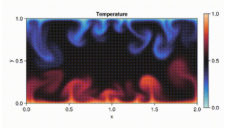
Backward facing step (3D)



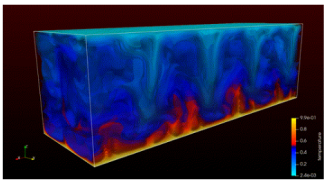
Decaying turbulence (3D)



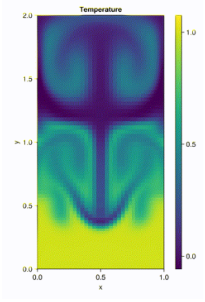
Taylor-Green vortex (3D)



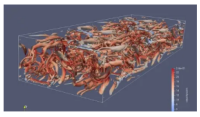
Rayleigh-Bénard (2D)



Rayleigh-Bénard (3D)



Rayleigh-Taylor (2D)



Turbulent channel flow (3D)

Figure 6.4: Different example scripts included in the documentation with `Literate.jl`.

2. **Documentation.** The documentation is built from source, executing the literate example scripts (section 6.5.4) and extracting docstrings. The result is deployed to GitHub Pages. Preview builds are generated for pull requests, allowing documentation changes to be reviewed before merging.

3. **Format check.** All source files are reformatted using `JuliaFormatter.jl`, and the workflow fails if any file was changed. This enforces a consistent code style across all contributors without requiring manual formatting.

4. **Spell check.** On pull requests, a spell checker (`typos`) scans the codebase for common typographical errors in code and documentation.

5. **Dependency compatibility.** `CompatHelper.jl` runs daily to check whether new versions of dependencies are available that may require updating the compatibility bounds in `Project.toml`. When updates are needed, it automatically opens a pull request. Similarly, Dependabot monitors the GitHub Actions workflow files for outdated action versions.

This automated pipeline means that a developer can push a commit and receive feedback within minutes on whether the change is correct, well formatted, and compatible with the existing documentation and dependencies.

6.5.6 *Release management*

The package follows semantic versioning: the version number (e.g., 3.0.0) encodes major, minor, and patch levels. Major version increments signal breaking API changes, minor increments add new features, and patch increments fix bugs. The version is specified in `Project.toml`, the central package manifest.

Releases are published through the Julia General registry. When a new version is ready, it is registered with the registry, after which TagBot—a GitHub Actions workflow—automatically creates a GitHub release and a corresponding Git tag. This makes every release permanently accessible and installable by any Julia user through the standard package manager.

Each release is also archived on Zenodo, which assigns a digital object identifier (DOI) to the archived snapshot. Unlike a GitHub repository, which can be deleted or made private, a Zenodo archive is intended to be permanent. This is important for reproducibility: a DOI in a publication points to the exact version of the code used to produce the results, regardless of future changes to the repository.

The package also includes a `CITATION.cff` file, a machine-readable citation metadata file that specifies the authors, title, license, and DOI. GitHub

recognizes this file and displays a "Cite this repository" button on the repository page, making it easy for users to generate a citation in various formats.

6.5.7 *Reproducibility*

All simulation results presented in scientific computing publications should be reproducible. The code used to generate the results should ideally be made publicly available. If this is not the case, the authors should be willing to help readers upon demand, or agree to share the code privately upon request with a non-disclosure agreement.

Simulation setups should be provided in full detail. While providing a Taylor-scale Reynolds number is descriptive for what kind of flow is being simulated, it is not sufficient for reproduction, since it is a *derived* quantity. The simulation setup should also include the exact domain size, kinematic viscosity, and initial conditions so that readers can reproduce the simulation setup and compare their results.

When using random numbers, the seed of the random number generators should always be fixed. This in itself is not a guarantee for perfect reproducibility, since different hardware architectures and compiler optimizations may lead to different results. Therefore, the exact hardware and software environment used to generate results should be explicitly stated.

Research code used to produce results for publications should be archived for long-term availability. Zenodo is such an archival service. A GitHub repository can disappear after some years, but a Zenodo archive is at least intended to be permanent. Zenodo provides convenient integration with GitHub for archiving repositories, and each version of the archived software gets a digital object identifier (DOI).

In addition to archiving code, one may also store the simulation results themselves. Full three-dimensional velocity fields from DNS or LES require substantial storage space, which makes long-term hosting nontrivial. Zenodo offers a solution for this as well, allowing researchers to upload large datasets with a persistent DOI [74]. The tradeoff is that raw field data is expensive to store and transfer, while derived statistics—such as mean velocity profiles, Reynolds stresses, and energy spectra—are compact and easily shared. Vreman and Kuerten [197] stored averaged DNS statistics for turbulent channel flow in small text files that are readily downloadable, providing a convenient reference dataset. We will use this reference data in section 6.6.

6.5.8 *Test suite*

The test suite of `INS.jl` comprises fourteen test files organized using `TestItemRunner.jl`. The entire suite is run with

```
julia --project=test test/runtests.jl
```

or equivalently through Julia's package manager with `Pkg.test()`. Individual tests can be filtered by name or file, which is useful during development when working on a specific component. The tests fall into three categories.

1. **Software correctness tests.** These verify that individual functions and complete simulation pipelines run without errors. A "master run" test steps through an entire simulation at low resolution to catch interface-level bugs. The numerical values produced are not checked; the purpose is to ensure that the code executes without crashing for representative use cases. Code quality is additionally checked by `Aqua.jl`, which detects common issues such as ambiguous method definitions, unbound type parameters, and stale dependencies.

2. **Numerical verification tests.** These check that the code produces correct numerical values. The 2D Taylor-Green vortex serves as a manufactured solution: the solver must converge at second order toward the known analytical velocity and pressure fields. The pressure Poisson solvers are verified by checking that the resulting velocity field is divergence-free up to a prescribed tolerance. All hand-written adjoint kernels (section 6.3.4) are compared against finite-difference gradients using `ChainRulesTestUtils.jl`.

3. **Physical verification tests.** These test that the discrete operators satisfy the physical invariants they are designed to preserve. The skew-symmetric convection operator must produce zero change in the total kinetic energy. The discrete divergence and pressure gradient operators must be each other's negative transpose.

Where possible, tests are run on both uniform and non-uniform grids and at small problem sizes, since edge cases—such as very non-uniform grid spacings or few degrees of freedom—are more likely to expose bugs that remain hidden on uniform grids at moderate resolution.

All tests are designed to complete within the resource limits of GitHub Actions (section 6.5.5), so that every commit receives automated feedback within minutes. This keeps the development cycle fast and ensures that new contributions do not break existing functionality.

Next, we compare the code against a turbulent channel flow to verify that the numerical solutions produce expected statistical properties. This is a visual confirmation, and is separate from the automated test suite.

6.6 TURBULENT CHANNEL FLOW

We simulate turbulent channel flow at friction Reynolds number $\mathrm{Re}_\tau = 180$ (fig. 6.5) and compare against the reference DNS data of Vreman and Kuerten [197].

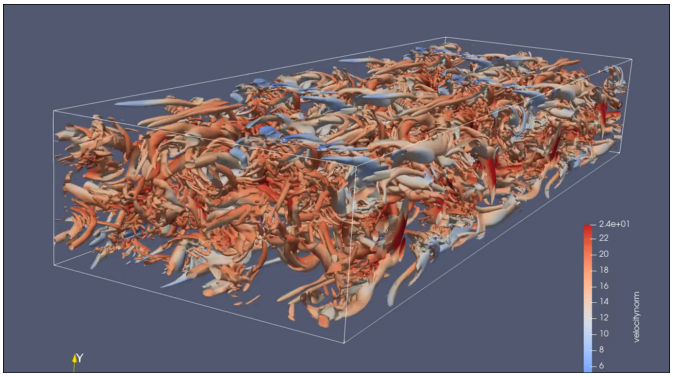


Figure 6.5: Snapshot of the turbulent channel flow during the transition from laminar to turbulent flow. The isosurfaces show the Q-criterion $Q = -\frac{1}{2}\operatorname{tr}(A^2)$, where A is the velocity gradient tensor. Positive values of Q identify regions where rotation dominates strain, highlighting vortical structures. The isosurfaces are colored by velocity magnitude.

6.6.1 *Setup*

The domain is a rectangular box $[0, L_x] \times [0, L_y] \times [0, L_z]$ with $L_x = 4\pi$, $L_y = 2$, and $L_z = 4\pi/3$. The velocity is denoted (u, v, w). The boundary conditions are periodic in the streamwise (x) and spanwise (z) directions, and no-slip at the walls $y = 0$ and $y = 2$. The flow is driven by a constant body force $f = (1, 0, 0)$ that acts as an artificial mean pressure gradient. The kinematic viscosity is $\nu = 1/180$.

The friction velocity is defined as $u_\tau \coloneqq \lim_{y\to 0} (\nu \mathrm{d}\bar{u}/\mathrm{d}y)^{1/2}$ at the wall, where $\bar{u}(y) \coloneqq \int u(x, y, z)\,\mathrm{d}x\,\mathrm{d}z/(L_x L_z)$ is the mean streamwise velocity. In the limit where $T \to \infty$, this setup gives $\langle u_\tau \rangle = 1$, where $T > 0$ is the averaging time and $\langle\cdot\rangle$ denotes time averaging. We therefore use the approximation $u_\tau \approx 1$ to define a velocity scale, corresponding to the "wall unit" length scale $L^+ \coloneqq \nu/u_\tau = 1/180$, time scale $T_\tau \coloneqq H/u_\tau = 1$, and Reynolds number $\mathrm{Re}_\tau \coloneqq u_\tau H/\nu = 180$, where $H = 1$ is the channel half-width. The initial conditions are a sinusoidally perturbed laminar flow. The flow is first integrated until $t = 15T_\tau = 15$ time units to reach a statistically stationary state, after which statistics are collected at 1000 equispaced times for $10T_\tau = 10$ time units.

The x-z averaging notation $\overline{(\cdot)}$ is not related to the LES filtering used in previous chapters.

6.6.2 *DNS on a uniform grid*

We first perform a DNS on a uniform grid with $N_x = 512$, $N_y = 1024$, $N_z = 256$ points. On this uniform grid, the pressure Poisson equation is solved spectrally using FFTs in the periodic directions and a discrete cosine

The DNS is performed on a single H100 GPU on the Dutch national supercomputer Snellius.

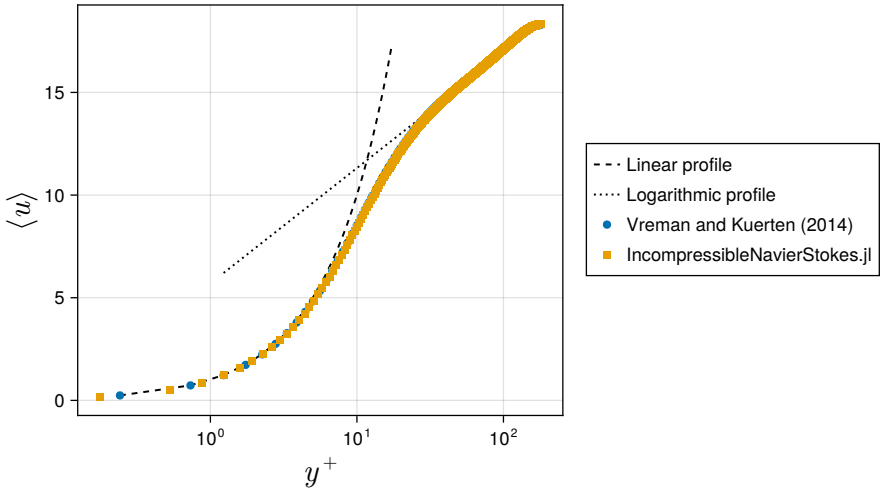


Figure 6.6: Mean streamwise velocity profile $\langle \bar{u} \rangle$ for DNS of turbulent channel flow with $512 \times 1024 \times 256$ grid points on a uniform grid, compared with reference data from Vreman and Kuerten [197].

transform (type-II) in the wall-normal direction, implemented via the `cuFFT` library on the GPU. In wall units, this gives grid spacings of $\Delta x^+ := \Delta x / L^+ = 4.418$, $\Delta y^+ = 0.3516$, and $\Delta z^+ = 2.945$. Vreman and Kuerten used $N_x = 512$, $N_y = 256$, $N_z = 256$ points with a tanh-stretching in the wall-normal direction, giving $\Delta y^+ = 0.49$ near the wall (with the same Δx^+ and Δz^+ as our uniform grid).

Figure 6.6 shows the mean streamwise velocity profile $\langle \bar{u} \rangle$ as a function of the wall-normal coordinate in viscous units, $y^+ = y / L^+ = 180\, y$. The reference data from Vreman and Kuerten [197] is also shown. The velocity profiles are in good agreement in both the linear viscous sublayer and the logarithmic region. The theoretical linear ($u/u_\tau = y^+$) and logarithmic ($u/u_\tau = \frac{1}{\kappa} \log y^+ + C$ with $\kappa = 0.41$, $C = 5.7$) profiles are also shown for reference.

Further insight can be gained by looking at the higher order moments of the velocity fluctuations

$$u' := u - \bar{u}. \tag{6.12}$$

These include the root-mean-square (RMS) velocity fluctuations $\sqrt{\langle \overline{u'u'} \rangle}$, which are shown in fig. 6.7. `INS.jl` is in good agreement with the reference data, but there is a tiny discrepancy at the peak of $\langle \overline{u'u'} \rangle$. This discrepancy is more visible in the third and fourth moments $\langle \overline{u'u'u'} \rangle$ and $\langle \overline{u'u'u'u'} \rangle$ shown in fig. 6.8 and fig. 6.9, respectively. The peaks are slightly underpre-

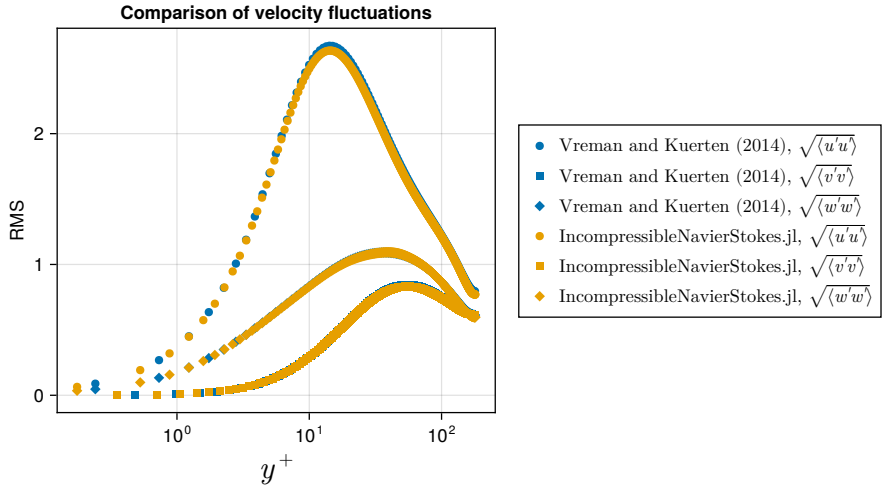


Figure 6.7: Average velocity fluctuations in the three directions for DNS of turbulent channel flow with $512 \times 1024 \times 256$ grid points on a uniform grid, compared with reference data from Vreman and Kuerten [197].

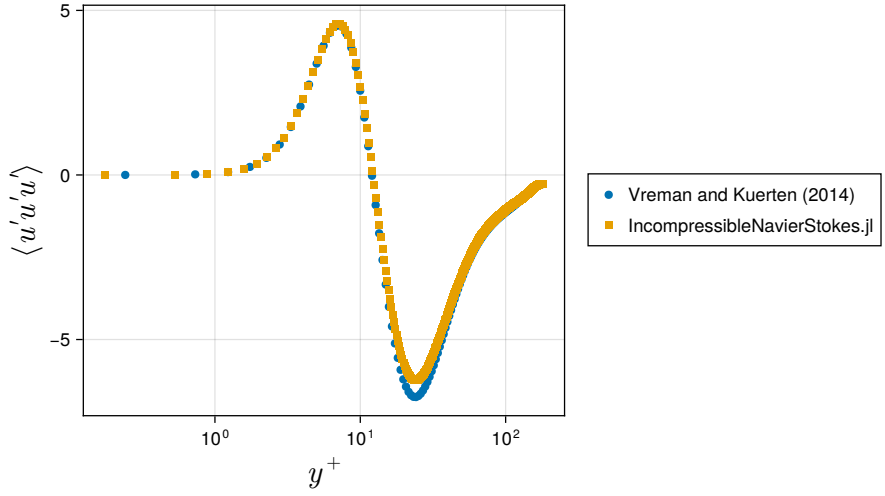


Figure 6.8: Third moment of x-velocity fluctuations $\langle \overline{u'u'u'} \rangle$ for DNS of turbulent channel flow with $512 \times 1024 \times 256$ grid points on a uniform grid, compared with reference data from Vreman and Kuerten [197].

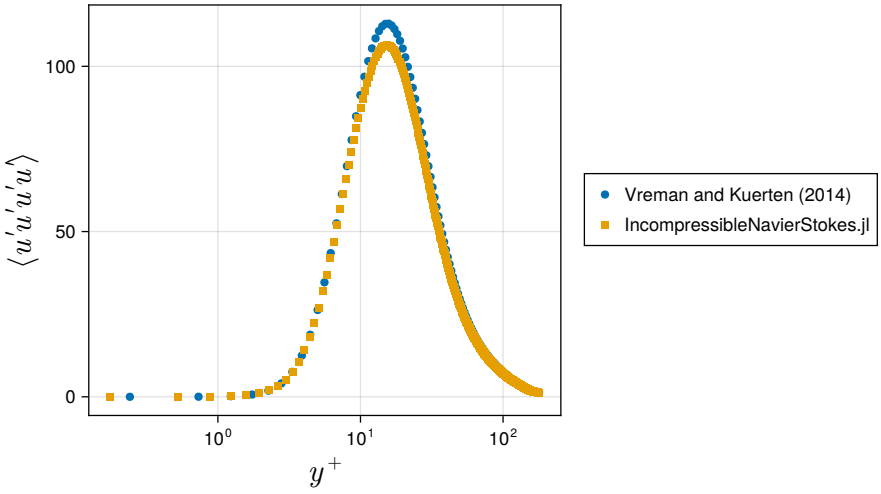


Figure 6.9: Fourth moment of x-velocity fluctuations $\left\langle \overline{u'u'u'u'} \right\rangle$ for DNS of turbulent channel flow with $512 \times 1024 \times 256$ grid points on a uniform grid, compared with reference data from Vreman and Kuerten [197].

dicted. We stress that we use a second-order accurate finite-volume scheme, while the reference data was produced using a fourth-order finite-difference scheme. Vreman and Kuerten also averaged over $200T_\tau = 200$ time units, while we only average over 10 time units, but we found little difference when running simulations for longer times.

Finally, figs. 6.10 and 6.11 show the cross-moments $\left\langle \overline{u'u'v'} \right\rangle$ and $\left\langle \overline{u'w'} \right\rangle$. Since the velocity components live on staggered face locations, they are first interpolated to volume centers before computing the cross-moments; the fluctuations $u' = u - \bar{u}$ are computed before interpolation. The cross-moment $\left\langle \overline{u'u'v'} \right\rangle$ is in good agreement with the reference data, apart from a small underprediction at the peak consistent with the moments discussed above. By contrast, $\left\langle \overline{u'w'} \right\rangle$ shows no agreement with the reference. This is expected: by the statistical symmetry of the channel in the spanwise direction, $\left\langle \overline{u'w'} \right\rangle$ should vanish identically. In practice, the finite averaging time leaves a residual of comparable magnitude to the reference but with random sign, depending on the initialization and floating-point perturbations. The streamwise and wall-normal directions do not share this sensitivity, as they are distinguished by the mean flow and the presence of the walls, respectively.

Having validated the solver against DNS reference data, we now turn to LES to demonstrate its ability to run standard eddy-viscosity closure models.

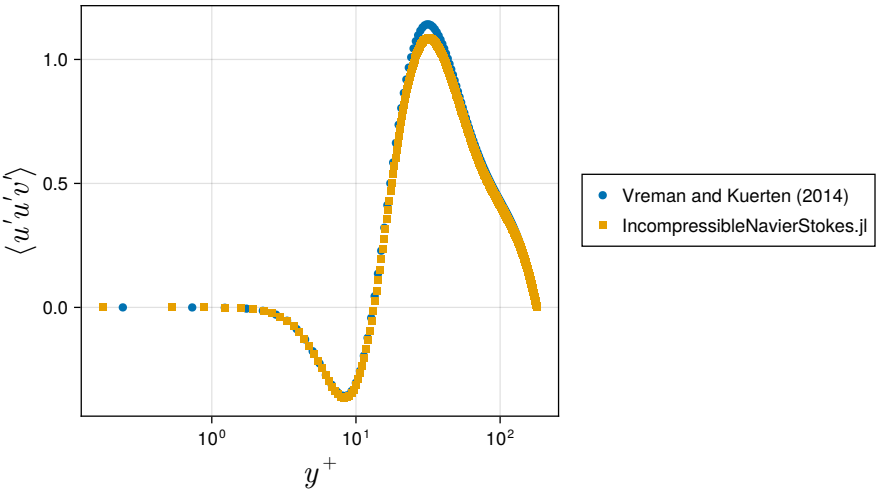


Figure 6.10: Third order cross-moment of the fluctuations $\langle \overline{u'u'v'} \rangle$ for DNS of turbulent channel flow with $512 \times 1024 \times 256$ grid points on a uniform grid, compared with reference data from Vreman and Kuerten [197].

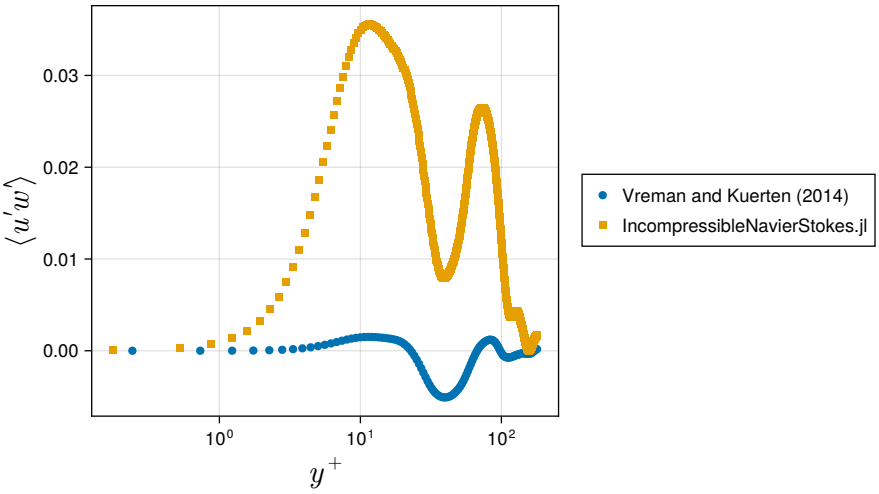


Figure 6.11: Second order cross-moment of the fluctuations $\langle \overline{u'w'} \rangle$ for DNS of turbulent channel flow with $512 \times 1024 \times 256$ grid points on a uniform grid, compared with reference data from Vreman and Kuerten [197].

6.6.3 *LES with eddy-viscosity models*

The LES is performed on a single RTX 4090 desktop GPU.

We then perform LES of the same channel flow on a coarser grid with $N_x = 128$, $N_y = 64$, $N_z = 64$ points. The grid in the wall-normal direction (y) is non-uniform, using the hyperbolic tangent stretching (6.5) to cluster points near the walls. Since the grid is non-uniform, the spectral pressure solver cannot be used; instead, the pressure Poisson equation is solved using a sparse direct solver with an LDL^T factorization, implemented via the `cuDSS` library on the GPU. This factorization requires substantial memory, which prevented us from running the DNS at full resolution on a non-uniform grid: neither the RTX 4090 (24 GB) nor the H100 (94 GB) had sufficient memory for the LDL^T decomposition at the $512 \times 256 \times 256$ resolution of Vreman and Kuerten. We employ five different eddy-viscosity models: no-model (zero eddy-viscosity), Smagorinsky [168], WALE [130], and QR [192], and Vreman's model [195]. The full model expressions are provided in section D.3.

We use the constants $C_S = 0.1$ for Smagorinsky, $C_W = 0.5$ for WALE, $C_Q = \sqrt{3/2}/\pi$ for QR, and $C_V = \sqrt{2.5C_S^2}$ for Vreman's model. The local filter width is defined based on the grid size with the common definition $(\Delta x \Delta y \Delta z)^{1/3}$. For Vreman's model, the individual grid spacings are incorporated separately. We note that the choice of constants depend on the choice of filter width definition. Verstappen argues that the filter width should also account for the interpolation scheme, potentially doubling the filter width (and thus halving the "optimal" values for the constants) [192]. For more discussions about the discretization-informed filter width definition, see chapter 4.

Figure 6.12 shows the mean eddy-viscosity $\langle \bar{\nu}^\Delta \rangle / \nu$ normalized by the kinematic viscosity ν. The DNS and no-model simulations do not produce any eddy-viscosity (we set $\nu^\Delta = 0$). The Smagorinsky viscosity does not decay to zero at the wall, which is a well-known issue with that model. All the other models decay to zero at the wall, with the WALE model showing the fastest decay. In the linear layer, Smagorinsky has the highest eddy-viscosity, followed by QR, Vreman, and WALE. In the logarithmic region, WALE has the highest eddy-viscosity, followed by QR, Smagorinsky, and Vreman. We emphasize that all the eddy-viscosities are scaled by the respective model constants. For different choices of constants, the magnitude of the eddy-viscosities can change, but not the shape of the profiles.

Figures 6.13 to 6.15 show the same statistics as in the DNS case, but for the LES. For the average streamwise velocity profile $\langle \bar{u} \rangle$ in fig. 6.13, Smagorinsky is too dissipative, resulting in an underprediction of the velocity. WALE and Vreman are closer to the reference than no-model. The QR model is slightly below in the viscous layer and higher than the reference in the logarithmic region. For the root mean square streamwise velocity fluctuations $\sqrt{\langle \overline{u'u'} \rangle}$

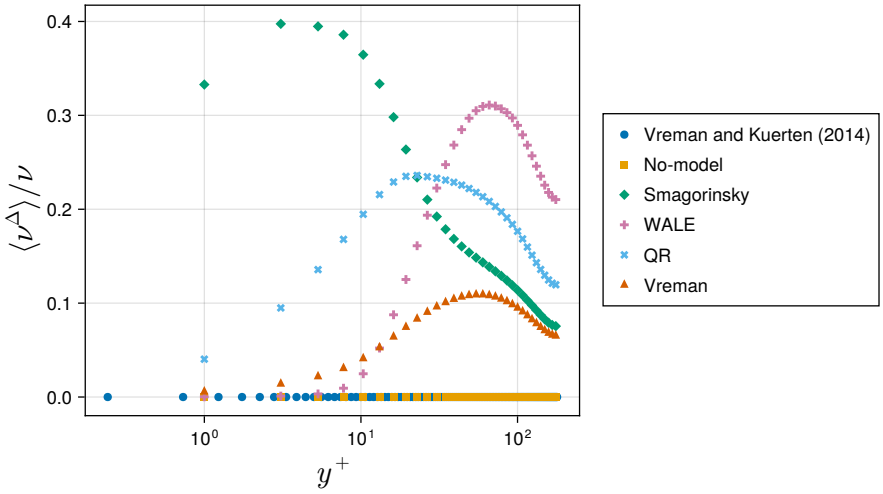


Figure 6.12: Mean eddy-viscosity $\langle \nu^{\Delta} \rangle$ normalized by the kinematic viscosity ν for LES of turbulent channel flow with $128 \times 64 \times 64$ grid points on a non-uniform grid, compared with reference data from Vreman and Kuerten [197].The DNS/no-model simulations do not produce any eddy-viscosity.

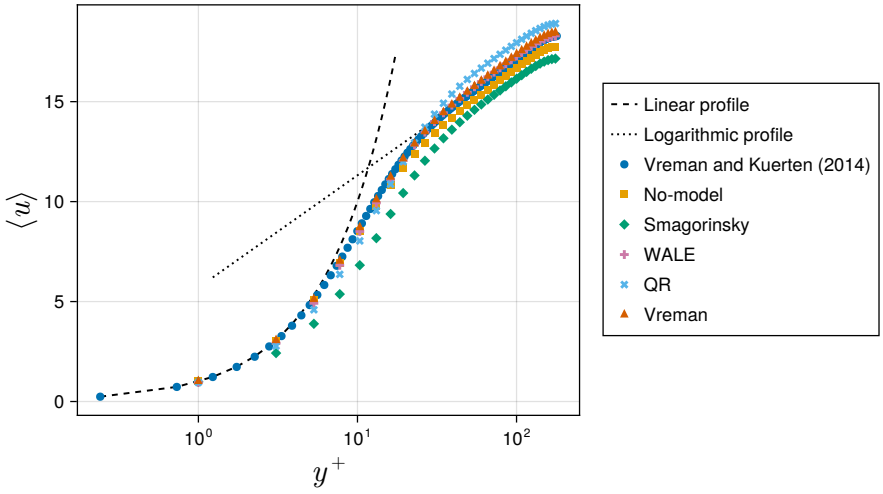


Figure 6.13: Mean streamwise velocity profile $\langle \bar{u} \rangle$ for LES of turbulent channel flow with $128 \times 64 \times 64$ grid points on a non-uniform grid, compared with reference data from Vreman and Kuerten [197].

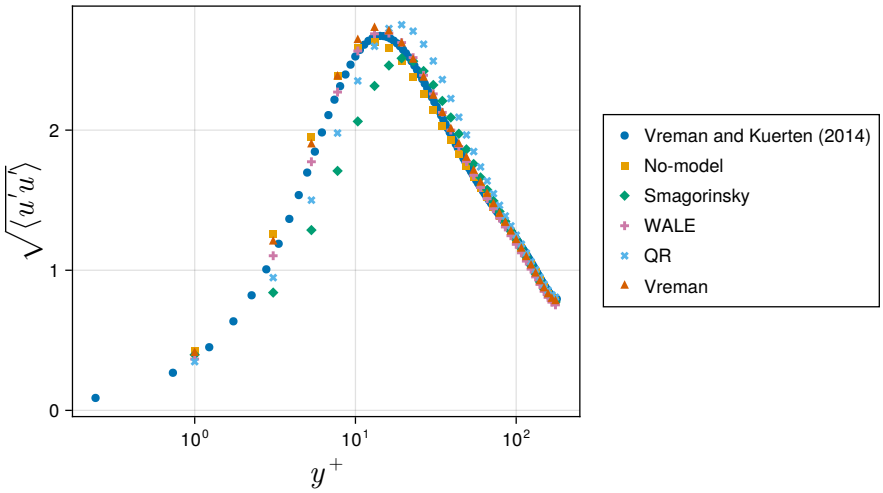


Figure 6.14: Average squared velocity fluctuations in the x-direction for LES of turbulent channel flow with $128 \times 64 \times 64$ grid points on a non-uniform grid, compared with reference data from Vreman and Kuerten [197].

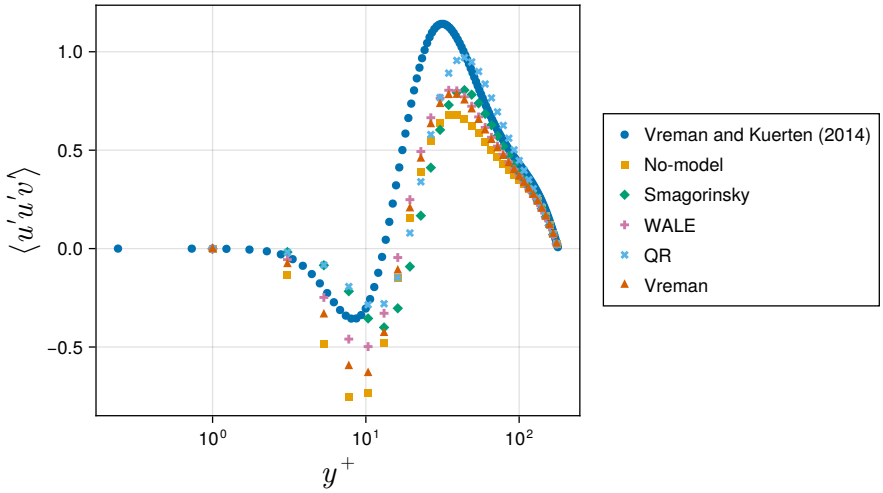


Figure 6.15: Third order cross-moment of the fluctuations $\langle \overline{u'u'v'} \rangle$ for LES of turbulent channel flow with $128 \times 64 \times 64$ grid points on a non-uniform grid, compared with reference data from Vreman and Kuerten [197].

in fig. 6.14, WALE and Vreman perform better than no-model near the wall and near the channel center, while no-model performs slightly better in the buffer layer. This is consistent with the well-known tendency of eddy-viscosity models to overdamp the turbulence production peak in the buffer layer. Smagorinsky is again too dissipative, resulting in an underprediction of the velocity fluctuations except for in the channel center. For the cross-moment $\langle \overline{u'u'v'} \rangle$ in fig. 6.15, the QR model performs best in the logarithmic region, while WALE is more accurate near the wall. All models perform better than no-model.

In conclusion, the solver correctly implements the eddy-viscosity models, and the observed behavior is consistent with the literature [130]: the Smagorinsky model is overly dissipative in wall-bounded flows due to its non-vanishing eddy viscosity at the wall, while WALE, QR, and Vreman's model all improve upon this by construction. We note that the model constants and filter width definition were not tuned for this case, and different choices can shift the results [181].

6.7 HARDWARE CONSIDERATIONS AND FLOATING POINT ARITHMETIC

The choice of floating point precision has direct consequences for both the accuracy and the computational cost of a simulation.

6.7.1 *Floating point formats and sources of error*

Real numbers are represented on a computer using the IEEE 754 floating point standard [77] as $(-1)^s \times 2^{e-b} \times (1+m)$, where s is a sign bit, e is the exponent, b is a format-dependent bias, and m is the significand (mantissa) with an implicit leading bit. The formats most relevant for scientific computing are listed in table 6.1. Each format trades precision (significand bits) and dynamic range (exponent bits) against storage size and computational throughput. The machine epsilon $\varepsilon = 2^{-(p-1)}$ bounds the relative error of a single arithmetic operation.

For DNS, the most relevant error mechanisms are *catastrophic cancellation*—the loss of significant digits when subtracting nearly equal numbers, which directly affects finite difference stencils—and *overflow*, where results exceed the representable range (Float16 overflows at approximately 6.5×10^4, which can be exceeded by common physical quantities). In long-running simulations, round-off errors accumulate as $O(\sqrt{N_t}\,\varepsilon)$ for stable schemes. For chaotic flows, even double-precision trajectories eventually diverge from the true solution, but the practical question is whether the statistical properties of the flow are affected by the precision.

Format	Bits	Sign	Exponent	Significand	ε	Range
Float64	64	1	11	52	$\approx 10^{-16}$	$\approx 10^{\pm 308}$
Float32	32	1	8	23	$\approx 10^{-7}$	$\approx 10^{\pm 38}$
Float16	16	1	5	10	$\approx 10^{-3}$	$\approx 10^{\pm 4.5}$
BFloat16	16	1	8	7	$\approx 10^{-2}$	$\approx 10^{\pm 38}$

Table 6.1: Properties of common floating point formats. The machine epsilon $\varepsilon = 2^{-(p-1)}$, where p is the number of significand bits plus the implicit leading bit, determines the smallest relative perturbation that is representable. The range indicates the approximate magnitude of the largest representable finite number.

6.7.2 *Precision trends in GPU hardware*

NVIDIA's A100 GPU provides 19.5 TFLOPS in Float64 but 156 TFLOPS in BFloat16—an eightfold difference that makes reduced precision difficult to ignore.

GPU manufacturers have progressively shifted transistor budgets toward lower-precision arithmetic units, driven by deep learning workloads [51]. In some architectures, Float64 throughput is only 1/32 or 1/64 of Float32 throughput. Halving the number of bits per value also halves the memory footprint and doubles the effective memory bandwidth, both of which are critical for memory-bound stencil-based PDE solvers. However, the reduced dynamic range of 16-bit formats can lead to overflow and loss of significance, so scientists must carefully evaluate numerical stability at reduced precision. Reduced-precision strategies have been explored in climate modelling [85, 179], where stochastic rounding can compensate for the loss of precision.

Mixed-precision strategies offer a middle ground: the bulk of the computation is performed in reduced precision, while critical accumulations (such as inner products in iterative solvers) use double precision [72]. `INS.jl` supports choosing the floating point format, but does not currently implement any mixed-precision strategies.

6.7.3 *Implications for DNS*

A finite difference approximation incurs both truncation error (decreasing with grid spacing h) and round-off error (increasing with smaller h due to catastrophic cancellation). Let

$$\partial_x^{h,1} f(x) := \frac{f(x+h) - f(x)}{h}, \tag{6.13}$$

$$\partial_x^{h,2} f(x) := \frac{f(x+h) - f(x-h)}{2h} \tag{6.14}$$

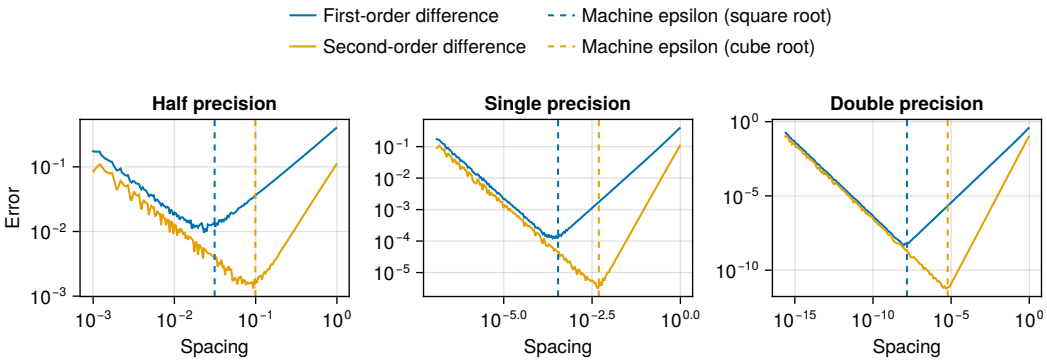


Figure 6.16: Finite difference errors for the derivative of $\sin(x)$, evaluated in half (Float16), single (Float32), and double (Float64) precision. As the grid spacing h decreases, the truncation error decreases until round-off error dominates, producing a characteristic V-shaped curve. The minimum error is reached at a grid spacing that depends on both the order of the finite difference and the machine epsilon of the floating point format. For second-order differences in single precision, the optimal spacing is comparable to typical DNS grid spacings, indicating that single precision is marginal at high resolutions.

denote the first- and second-order finite difference approximations. The error is minimized at $h \approx \varepsilon^{1/2}$ for the first-order and $h \approx \varepsilon^{1/3}$ for the second-order difference, as illustrated in fig. 6.16.

For a DNS with $N = 1024$ points per direction in a domain of size 2π, the grid spacing is $h \approx 6 \cdot 10^{-3}$. In single precision, the optimal spacing for second-order differences is $h \approx 5 \cdot 10^{-3}$, which is comparable to the DNS spacing. This suggests that single precision is marginal but feasible for second-order methods at this resolution, consistent with the findings of Karp et al. [83], who observed no significant discrepancies between single- and double-precision results across multiple turbulence solvers and test cases. At higher resolutions or for higher-order methods, double precision becomes necessary. Half precision is not viable for spatial discretization at practical DNS resolutions, but may find application in components where high accuracy is not required, such as neural network inference within a closure model. In the closure modelling work of chapters 3 and 5, neural networks are trained and evaluated in single and double precision, respectively, but the tolerance of neural networks to quantization suggests that half precision could be used for inference without significant loss of closure model quality, while benefiting from the higher throughput of 16-bit arithmetic units.

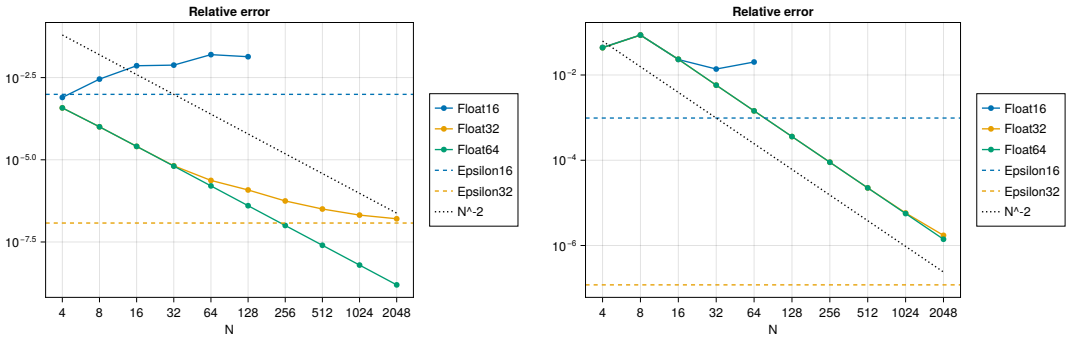


Figure 6.17: Convergence of the numerical solution to the analytical solution for the velocity field of the 2D Taylor-Green vortex at $t = 1$. The error is measured in the L^2 norm and plotted against the number of grid points N for different floating-point formats. The dotted line indicates second-order convergence. The dashed lines indicate the machine epsilon for single and half precision. Left: Uniform grid. Right: Non-uniform grid with tanh-stretching.

6.7.4 *Convergence study: the 2D Taylor-Green vortex*

The 2D Taylor-Green vortex is an exact solution to the incompressible Navier-Stokes equations on the periodic domain $\Omega = [0, 2\pi]^2$ (see section A.5 for the analytical expressions). Since the velocity and pressure fields are known in closed form, this test case allows us to measure the spatial discretization error directly.

We discretize the domain with $N \times N$ grid points for $N \in \{4, 8, 16, \dots, 2048\}$ and compute the numerical solution at $t = 1$ using a sufficiently small time step so that the temporal discretization error is negligible compared to the spatial error. The L^2 error between the numerical and analytical solutions is then evaluated for three floating-point formats: Float64 (double precision), Float32 (single precision), and Float16 (half precision). We perform this convergence study on both a uniform grid and a non-uniform grid with tanh-stretching.

UNIFORM GRID. On the uniform grid, the pressure Poisson equation is solved spectrally using FFTs. Figure 6.17 (left) shows that Float64 and Float32 both exhibit clean second-order convergence, consistent with the second-order finite-volume scheme. For Float32, the convergence rate begins to deteriorate at fine resolutions and eventually plateaus as the error reaches single-precision machine epsilon. Float16 does not display a meaningful convergence trend and produces large errors across all resolutions.

NON-UNIFORM GRID. On the non-uniform grid, the pressure is solved with a sparse direct (LDL^T) solver, which is only available for Float64 and Float32. Since no direct solver implementation is available for Float16, the Float16 case falls back to the conjugate gradient method in that case. Figure 6.17 (right) shows that Float64 and Float32 converge at the same rate, since the spatially varying grid spacing leads to larger errors at a given N compared to the uniform case, so that the single-precision plateau is not yet reached for the resolutions tested. Float16 tracks the other two formats at coarse resolutions but becomes unstable as N increases.

In summary, Float64 delivers reliable second-order convergence across all tested resolutions. Float32 also converges at second order for coarse and moderate resolutions, but the payoff of grid refinement diminishes at finer grids as rounding errors accumulate and the effective convergence rate drops below two. Float16, in its current form, is not viable for this test case. One possible path toward making reduced-precision arithmetic more effective is to scale the physical quantities so that they remain well within the representable range of the floating-point format, thereby reducing overflow, underflow, and loss of significance.

6.8 CONCLUSION AND OUTLOOK

We summarize the contributions of this work and outline directions for future development.

6.8.1 *Summary*

This chapter presented `INS.jl`, an open-source Julia package for DNS and LES of incompressible flows on staggered Cartesian grids. The solver uses a second-order finite-volume spatial discretization with explicit time integration and a pressure Poisson equation for divergence-free projection, supporting both uniform and non-uniform grids in two and three dimensions.

The software is designed around matrix-free, hardware-agnostic kernels that are compiled for multi-threaded CPU or GPU from a single source implementation. Each discrete operator is implemented in both a pure (non-mutating) variant for automatic differentiation and a mutating variant for memory-efficient forward simulation. Hand-written adjoint kernels registered through `ChainRules.jl` enable reverse-mode differentiation through the entire solver, making it possible to train neural network closure models *a posteriori* by backpropagating through the time integration. Neural network closure models from `Lux.jl` are supported directly within the solver, using the differentiable operator infrastructure for a-posteriori training.

Three memory-optimization techniques—array reuse, low-storage Runge–Kutta methods, and lazy computation of tensor components—allow double-precision DNS at resolutions up to 840^3 on a single datacenter GPU.

The code was validated against an analytical solution (the 2D Taylor–Green vortex, confirming second-order convergence) and against reference DNS data from Vreman and Kuerten [197] for turbulent channel flow at $\mathrm{Re}_\tau = 180$. The mean velocity profile and RMS velocity fluctuations are in good agreement with the reference. Higher-order moments (third and fourth) show small discrepancies at the peaks, which we attribute to the second-order spatial discretization. LES of the same channel flow on a coarser non-uniform grid with four classical eddy-viscosity models (Smagorinsky, WALE, QR, and Vreman) showed behavior consistent with the literature: Smagorinsky is overly dissipative due to its non-vanishing wall viscosity, while WALE, QR, and Vreman's model improve upon this by construction, producing better agreement with the DNS reference for both the mean velocity and velocity fluctuations.

We also investigated the effect of floating-point precision on accuracy: Float64 delivers reliable second-order convergence, Float32 is effective at moderate resolutions but offers diminishing returns at finer grids, and Float16 is not yet viable in its current form. Understanding these precision limits is important because modern GPU hardware provides substantially higher throughput at reduced precision, and neural network components within closure models may tolerate lower precision for inference.

Software development practices—including version control, automated testing, continuous integration, documentation generation, literate programming for examples, release management with archival on Zenodo, and reproducibility—ensure that the code remains maintainable and that published results can be reproduced. The code has been used by other researchers [74, 79], demonstrating its accessibility and extensibility.

6.8.2 *Outlook*

The most immediate limitation of the current software is that it runs on a single compute node. While the memory optimizations (section 6.3.3) allow DNS at resolutions up to 840^3 on a single datacenter GPU, higher resolutions require distributing the computation across multiple GPUs and multiple nodes. This requires domain decomposition with halo exchange—communication of ghost-layer values at subdomain boundaries—between neighboring subdomains, using CUDA-aware MPI for GPU-to-GPU communication. The spectral pressure solver would need to be replaced with a distributed variant based on pencil decomposition for the Fourier transforms. The iterative conjugate gradient solver is more straightforward to distribute, since it requires only local stencil operations and global reductions.

A second direction is to increase the spatial accuracy of the discretization. The current second-order staggered scheme is well suited for DNS at moderate resolutions, but higher-order stencils would reduce the number of grid points needed to achieve a given accuracy. Verstappen and Veldman [191] describe a fourth-order scheme that preserves the symmetry properties of the second-order discretization by combining the standard discretization with a coarser grid. Implementing this scheme would require extending the kernel infrastructure to support wider stencils and multiple grid levels, while preserving the energy-conservation properties that the current discretization guarantees.

On the automatic differentiation side, the current approach uses `Zygote.jl` for reverse-mode differentiation, which requires the pure (non-mutating) operator variants and stores all intermediate states for the backward pass. This limits the number of time steps that can be unrolled during training before memory is exhausted. Checkpointing strategies, where only a subset of intermediate states is stored and the rest are recomputed during the backward pass, would allow training over longer time horizons without proportionally increasing memory requirements. An alternative direction is to use `Enzyme.jl`, which operates at a lower level than `Zygote.jl` and can differentiate through mutating code directly, potentially avoiding the need for separate pure and mutating operator variants altogether. Preliminary support for `Enzyme.jl` is already included in the package as an extension module.

The analysis in section 6.7 showed that single precision is marginal for second-order methods at typical DNS resolutions. Systematic exploration of mixed-precision strategies—using reduced precision for memory-bound kernels while retaining double precision for accumulations—could improve performance on modern GPU hardware without compromising accuracy.

Finally, the functional programming design of the solver makes it straightforward to extract and reuse individual components in other contexts. The discrete operators, grid generation routines, and initial condition generators have already been reused in a pseudospectral solver developed separately by the author. Maintaining this composability as the package grows in scope will require careful attention to the software architecture, ensuring that new features are added as modular components rather than tightly coupled extensions.

7 CONCLUSIONS AND RECOMMENDATIONS

7.1 CONCLUSIONS

In this thesis, we have developed new data-driven closure models for large-eddy simulation (LES). The central insight is that the numerical discretization should be treated as a fundamental part of the LES formulation, rather than as a source of errors that vanish in the limit of infinite resolution. In LES, this limit is never reached—reaching it would effectively turn LES into DNS—and closure models must therefore be designed with the discretization in mind.

We revisit the three research sub-goals stated in chapter 1 and summarize the main conclusions.

1. **Discretization-consistent closure formulations.** In chapter 2, we studied the discretize-first approach in a simplified setting: the linear convection equation with a non-uniform filter. We compared intrusive (explicit reconstruction) and non-intrusive (derivative fitting, trajectory fitting) methods for inferring the discretely filtered operator. Derivative fitting proved to be the most practical approach, achieving good accuracy and generalization without requiring access to the full-order model. Trajectory fitting (embedded learning), despite its greater computational overhead, did not outperform simpler methods.

   In chapter 3, we extended the discretize-first approach to the nonlinear Navier-Stokes equations. We proposed a divergence-consistent face-averaging filter that ensures full model-data consistency and avoids pressure-related stability issues. We showed that if the discretization is accounted for correctly, a-priori training is sufficient for achieving accurate and stable neural-network based LES. If the discretization is not properly accounted for, a-posteriori training (which requires a differentiable solver) becomes necessary.

   In chapter 4, we derived exact expressions for the unresolved stress tensor in discrete filtered conservation laws, accounting for the discrete divergence, numerical flux, and incompressibility constraint. The commutation properties between filtering and finite-difference differentiation—the filter-swap property—enable exact computation of the stress from DNS data. Whereas chapter 3 used a surface-averaging (face-averaging) filter tied directly to the FVM staggered grid, chapter 4 uses a volume-averaging FVM filter in combination with general

convolutional LES filters, making the framework applicable beyond the specific FVM filter.

The overarching conclusion from these three chapters is that before any modeling can take place, the goal and target of an LES should be stated explicitly. The filter, the grid, the numerical fluxes, the boundary conditions, and the overall discretization scheme should be clearly defined. Only then can the target solution and the unresolved terms be identified. In particular, an explicit closure model should only act on top of any implicit dissipation already incorporated into the discretization scheme, so that the sub-grid dissipation is not accounted for twice.

2. **Incorporating physical structure.** In chapter 5, we investigated how known symmetry properties of the Navier-Stokes equations can be built into neural network closure models. We compared unconstrained direct networks, tensor basis neural networks (TBNNs), and symmetry-preserving group-convolution networks. Direct networks achieved the highest a-priori accuracy, while symmetry-preserving networks better respected the physical constraints of the equations. The results suggest that incorporating symmetries can reduce the space of functions the network explores, though the computational overhead of group-convolution networks is significant. For non-isotropic flows with privileged directions, relaxed symmetry constraints may be more appropriate than full rotational equivariance.

   Furthermore, a key finding from chapter 4 is that the unresolved stress tensor in the finite-volume setting is non-symmetric (due to the discrete divergence form) and non-local (due to the incompressibility constraint), unlike the classical continuous unresolved stress tensor. A discrete closure model should therefore aim to account for these properties of the tensor.

3. **Software for data-driven closure modeling.** In chapter 6, we presented IncompressibleNavierStokes.jl, an open-source Julia package that enables LES with machine-learning-based closure models embedded directly in the solver. The package supports DNS and LES on staggered Cartesian grids. The solver uses energy-conserving finite volume discretizations with hardware-agnostic kernels that run on both multi-threaded CPUs and GPUs from a single source implementation. Hand-written adjoint kernels for all discrete operators make the solver fully differentiable, enabling a-posteriori training of neural network closure models embedded in the LES solver. A pure functional programming style ensures composability and compatibility with automatic differentiation, while isolated mutating variants (marked with Julia's ! convention) enable the memory reuse

required for high-resolution DNS. With these optimizations, double-precision DNS with up to 840^3 degrees of freedom was achieved on a single consumer GPU. The implications of floating point precision for DNS were examined: single precision is marginal but feasible at typical resolutions for second-order methods, while the trend toward reduced-precision GPU hardware requires careful evaluation for scientific computing workloads. The chapter also described the software development practices underlying the project—version control, automated testing, continuous integration, documentation generation from source code and literate example scripts, and release management with archival on Zenodo—as a reference for scientific software projects. The software has been used by other researchers, demonstrating its accessibility beyond the original development team.

7.2 TECHNICAL RECOMMENDATIONS

Several aspects of the discretize-first framework were not part of this thesis and merit further study.

1. **Boundary conditions.** One of the motivations for the paradigm "discretize first" is that the LES boundary conditions are automatically inherited from the DNS formulation. Instead of postulating new LES boundary conditions, the boundary conditions from DNS are transferred to the LES equations through the definition of the filter and coarse-graining procedure. The surface-averaging procedure from chapter 3 is well-suited for boundary conditions, and no additional modeling is required. However, the volume-averaging procedure further studied in chapter 4 requires modifications at no-slip walls. A recommendation for future work is to investigate how to define the volume-averaging procedure close to no-slip walls such that the conservation properties are maintained.

2. **Time integration.** In chapters 3 to 5, the equations were discretized in space, then modeled with a closure. Only afterwards was a time-integration method applied, usually a low-storage explicit Runge-Kutta scheme. This means that the time integration introduces additional errors that are not accounted for by the spatial-discretization-aware LES formulation. Including the temporal discretization in the framework would have two advantages: first, the time integration errors would be absorbed into the closure model, removing a source of model-data inconsistency; second, the implicit temporal filtering introduced by finite time steps would be accounted for explicitly. For example, for a simple Forward Euler scheme, the procedure from chapter 4 could be generalized, with a volume-averaging filter that also averages forward

in time, thus converting a continuous time-derivative $\partial u/\partial t$ into a discrete time-difference $\partial^{\Delta t} u/\partial t^{\Delta t} := (u(t+\Delta t) - u(t))/\Delta t$. A space-time grid filter can be defined, and a continuous conservation law defined by a space-time divergence can be converted into a discrete space-time conservation law with a discrete space-time divergence.

3. **Higher-order schemes.** The procedure from chapter 4 was developed for second-order finite-volume schemes. A natural extension is to investigate whether it can be generalized to higher-order schemes, such as the fourth-order finite-volume scheme from [129, 191]. Such schemes can be obtained by superposing the second-order formulation for different coarse-grained versions of a base grid, such that higher-order terms cancel out. By using a similar superposition of volume-averaging filters, it may be possible to define a higher-order version of the filter-swap properties, as in [63].

4. **Revisiting existing closure models.** The exact discretization-aware unresolved stress tensor expressions from chapter 4 open the door to revisiting and adapting existing closure models. For example, the Clark model is defined through a Taylor series expansion of the non-linear unresolved stress tensor $\overline{u \otimes u} - \bar{u} \otimes \bar{u}$. In the discrete setting, this term includes numerical fluxes and interpolations, and the filter gets reduced by the filter-swap property. By applying the same Taylor series expansion to this new unresolved stress tensor, a discretization-aware Clark model could potentially be obtained, with a non-symmetric expression for the tensor. Could such a model be more stable than the classical Clark model when applied in a discrete LES setting?

   The dynamic Smagorinsky model is another candidate that could potentially be rederived using the new unresolved stress tensor expression. The classical dynamic Smagorinsky model relies on the Germano identity to write the difference between the classical unresolved stress tensors for two different filter widths [61]. With the new expression, the estimate for this term could potentially be modified to account for the discretization.

   New closure models could also be developed for the discretization-aware unresolved stress. The symmetry-preserving neural network models from chapter 5 could be modified for use in the FVM setting instead of the pseudo-spectral setting. The predicted stress tensor would then be non-symmetric, since the target tensor is known to be non-symmetric in that case. Non-symmetric eddy-viscosity models could also be developed, for example models of the form $m := -\nu^{\text{symm}} \bar{S} - \nu^{\text{skew}} \bar{W}$, where $\bar{S}$ and $\bar{W}$ are the symmetric and skew-symmetric parts of the resolved velocity gradient tensor, ν^{symm} is an eddy-viscosity in the traditional sense, and ν^{skew} is a new coefficient

weighing the non-symmetric part of the model. Closure model development for discrete LES would then be focused on finding an appropriate expression for ν^{skew} (without affecting energy stability).

5. **Prepare for changes in hardware.** Accelerator hardware is becoming increasingly important for scale resolving simulations (DNS and LES). However, the push towards reduced-precision floating point formats by hardware vendors potentially requires changes in numerical methods and their software implementations. A recommendation is therefore to already include support for mixed-precision and reduced-precision formats in software for turbulence simulations, and test where these formats fail to provide sufficient accuracy and stability. This may also require implementing new non-dimensionalization scaling techniques to ensure that numbers lie within the representable range of the reduced-precision formats.

7.3 BROADER OUTLOOK

The work in this thesis addresses the question of how to construct data-driven LES closure models that are consistent with the discretization and respect physical structure. Several broader challenges remain for the field of data-driven turbulence modeling.

A fundamental challenge of the discretize-first approach is the coupling between the closure model and the specific discretization for which it was trained. Because the unresolved stress tensor depends on the numerical fluxes, grid spacing, and filter definition, a model trained for one particular scheme may not transfer directly to a different grid resolution, a different numerical method, or a different test case. Generalizing discretization-aware closure models to unstructured meshes or higher-order schemes is therefore non-trivial and requires rethinking the filter-swap framework for those settings. One promising direction is to learn the dependence of the closure on the grid spacing explicitly, for example by incorporating it as a parameter, so that a single model can cover a range of resolutions. Ultimately, the discrete nature of the framework is both its strength—it provides exact, self-consistent closure targets—and its limitation in terms of portability across different computational setups.

A further fundamental limitation is the availability of training data. The DNS simulations used to generate training data in this thesis were performed at moderate Reynolds numbers. At higher Reynolds numbers, DNS becomes prohibitively expensive, and the question of how to obtain sufficient high-quality training data remains open. Transfer learning, where models trained at low Reynolds numbers are adapted to higher Reynolds numbers, is one promising direction. Experimental data could also complement DNS, although it is typically incomplete and noisy.

The computational cost of training and inference is another practical concern. Training a neural network closure model requires many evaluations of the LES solver (for a-posteriori training) or the DNS data pipeline (for a-priori training). At inference time, the neural network evaluation must be fast enough to preserve the cost savings of LES over DNS. Advances in hardware acceleration and efficient network architectures will be important for making data-driven LES practical at scale.

In chapter 1, we discussed the statistical nature of turbulence: since individual realizations are unpredictable, the goal of LES is to reproduce the correct statistical properties of the flow. This perspective suggests that *generative models*—machine learning models that produce samples from a learned probability distribution—may be a natural fit for turbulence modeling. Rather than predicting a single deterministic closure term, a generative closure model could sample from the distribution of possible sub-filter stresses, potentially capturing the stochastic nature of back-scatter and other intermittent phenomena.

Finally, a significant gap remains between the idealized test cases studied in this thesis (homogeneous isotropic turbulence, channel flows) and the complex geometries and multi-physics encountered in engineering applications. Bridging this gap will require extending the discretize-first framework to unstructured meshes, developing closure models that generalize across flow configurations, and validating data-driven models against experimental data in realistic settings.

A

SUPPLEMENTARY MATERIAL FOR CHAPTER 3

A.1 NUMERICAL EXPERIMENT DETAILS

A.1.1 *Energy spectra*

For our discretization, we define the energy at a wavenumber k as $\hat{E}_k = \frac{1}{2}\|\hat{u}_k\|^2$, where $\hat{u}^\alpha = \mathrm{DFT}(u^\alpha)$ is the discrete Fourier transform of u. Since $k \in \mathbb{Z}^d$, a discrete equivalent of the scalar energy spectrum as a function of $\|k\|$ is not straightforward. We proceed as follows. The energy at a scalar level $\kappa > 0$ is defined as the sum over all energy components in the dyadic bin $\mathcal{K}_\kappa = \{k \mid \kappa/a \leq \|k\| \leq \kappa a\}$ as

$$\hat{E}_\kappa = \sum_{k \in \mathcal{K}_\kappa} \hat{E}_k. \tag{A.1}$$

The parameter $a > 1$ determines the width of the interval. Lumley recommends using the golden ratio $a = (1 + \sqrt{5})/2 \approx 1.6$ [59]. Note that there is no normalization factor in front of the sum (A.1), even though the number of wavenumbers in the set increases with κ.

For homogeneous decaying isotropic turbulence, the spectrum should behave as follows. In the inertial region, for large Reynolds numbers, the theoretical decay of $\hat{E}$ should be $\hat{E}_\kappa = \mathcal{O}(\kappa^{-3})$ in 2D and $\mathcal{O}(\kappa^{-5/3})$ in 3D [144]. For the lowest wavenumbers, we should have $\hat{E}_\kappa = \mathcal{O}(\kappa^4)$ or $\hat{E}_\kappa = \mathcal{O}(\kappa^2)$.

It is also common to use a linear bin such as $\mathcal{K}_\kappa = \{k \mid \kappa - \frac{1}{2} \leq \|k\| < \kappa + \frac{1}{2}\}$ [117, 135, 153]. However, this leads to a different power law scaling in the inertial range than the well known κ^{-3} and $\kappa^{-5/3}$.

A.1.2 *Initial conditions*

To generate initial conditions in a periodic box, we prescribe an energy spectrum $\hat{E}_k$. We seek an initial velocity field u with the following properties:

- The Fourier transform of u, noted $\hat{u}$, should be such that $\frac{1}{2}\|\hat{u}_k\|^2 = \hat{E}_k$ for all k.
- u should be divergence-free with respect to our discretization: $Du = 0$.
- u should be controlled by random parameters, enabling the generation of a wide variety of initial conditions.

These properties are achieved by sampling a velocity field in spectral space, projecting it to make it divergence-free, transforming to physical space, and projecting again. In detail, let $a_k = \sqrt{2\hat{E}_k}\mathrm{e}^{2\pi\mathrm{i}\tau_k}$, where $\tau_k = \sum_{\alpha=1}^{d} \xi_k^\alpha$ is phase shift, $\xi_k^\alpha \sim \mathcal{U}[0,1]$ is a random uniform number if $k_\beta \geq 0$ for all β. If $k_\beta < 0$ for any β, we add the symmetry constraint $\xi_k^\alpha = \mathrm{sign}(k_\alpha)\xi_{|k|}^\alpha$ where $|k| = (|k_\beta|)_{\beta=1}^d$. Then $\|\hat{u}_k\| = |a_k|$, and a_k has a random phase shift. We then multiply the scalar a_k with a random unit vector e_k projected onto the divergence-free spectral grid as follows: $\hat{u}_k^\alpha = a_k \hat{P}_k e_k^\alpha / \|\hat{P}_k e_k^\alpha\|$, where $\hat{P}_k = I - \frac{kk^\mathsf{T}}{k^\mathsf{T}k} \in \mathbb{C}^{d\times d}$ is a projector for each k ensuring that $2\pi\mathrm{i}k^\mathsf{T}\hat{u}_k = 0$ for all k (which is the equivalent of $\nabla \cdot u = 0$ in spectral space) [144]. The normalization with respect to $\|\hat{P}_k e_k^\alpha\|$ ensures that no energy is lost in the projection step. In 2D, we choose a random vector on the unit circle $e_k = (\cos(\theta_k), \sin(\theta_k))$ with $\theta_k \sim \mathcal{U}[0, 2\pi]$. In 3D, we choose a random vector on the unit sphere $e_k = (\sin(\theta_k)\cos(\phi_k), \sin(\theta_k)\sin(\phi_k), \cos(\theta_k))$ with $\theta_k \sim \mathcal{U}[0,\pi]$ and $\phi_k \sim \mathcal{U}[0, 2\pi]$. Finally, we obtain the velocity field u by taking the inverse discrete Fourier transform and projecting again, since divergence-freeness on the spectral grid and on the staggered grid differ slightly. This yields the random initial field

$$u = P\,\mathrm{DFT}^{-1}(\hat{u}). \tag{A.2}$$

Note that the second projection may result in a slight loss of energy, but since u is already divergence-free on the spectral grid, we have found this loss to be negligible.

A.1.3 *CNN architecture*

The CNN architecture is shown in table A.1. We use periodic padding. For the last convolutional layer, we use no activation and no bias to avoid limiting the expressiveness of m. For the inner layers, we use bias and the tanh activation function.

The choice of channel sizes is based solely on having sufficient expressive capacity in the closure model. The kernel radius, on the other hand, is chosen to be small ($r = 2$, diameter 5) to ensure that the closure model uses local information. Furthermore, the resulting small stencils that are learned can possibly be interpreted as discrete differential operators of finite difference type, separated by simple non-linearities. In this way, the CNN can be viewed as a generalized Taylor series expansion of the commutator error in terms of the filtered velocity field, similar to certain continuous filter expansions [152]. We do not investigate this further in this study, but it could be a direction for future research.

The same CNN architecture m is used for all grids and filters. We choose a simple architecture since the goal of this study is not to develop the most

Layer	Radius	Channels	Activation	Bias	Parameters
$\text{Interpolate}_{u\to p}$					
Conv	2	$2 \to 24$	tanh	Yes	1224
Conv	2	$24 \to 24$	tanh	Yes	14424
Conv	2	$24 \to 24$	tanh	Yes	14424
Conv	2	$24 \to 24$	tanh	Yes	14424
Conv	2	$24 \to 2$	$x \mapsto x$	No	1200
$\text{Interpolate}_{p\to u}$					
					45696

Table A.1: CNN architecture, with $\epsilon = 1/100$. The total radius is 8, which means that the component $m(\bar{u}, \theta)_I$ depends on the components $\bar{u}_{I+J}$ for $J \in [-8, 8]^2$. In comparison, the diffusion operator has a radius of 1.

accurate closure model, but rather to compare different filters, LES formulations, and loss functions using the same closure architecture. For the LES formulation, we consider only the two models $\mathcal{M}_{\text{DIF}}$ and $\mathcal{M}_{\text{DCF}}$.

A.1.4 *Data generation*

To create data, we run 8 DNS simulations with $N = 4096^2$. We use adaptive time-stepping. For each random initial flow field $u(0)$, we let the DNS run for a burn-in time $t_{\text{burn}} = 0.5$ to initialize the flow beyond the artificial initial spectrum (3.30). We then start saving $\bar{u}$ and c at every time step until $t_{\text{end}} = 5$. Every 50 time steps we compute $\bar{u}$ and $c(u)$ on four coarse grids of size $\bar{n} \in \{32, 64, 128, 256\}$. The first 6 trajectories are used for training, and the remaining 2 for validation and testing.

A.1.5 *Training*

Both the Smagorinsky model and the CNN are parameterized and require training. Since the Smagorinsky parameter is a scalar, we perform a grid search to find the optimal parameter for each of the grid sizes, filter types and projection orders. We evaluate the relative a-posteriori error for the training set. We choose the value of $\theta \in \{0, 1/1000, 2/1000, \dots, 300/1000\}$ that yields the lowest training error. The resulting Smagorinsky constants are shown in table A.2. Note that the Smagorinsky constants are theoretically grid-independent, since information about the grid is incorporated through the filter width that enters the expression separately. We still optimize the coefficient for each grid size to ensure a fair comparison with the CNN.

$\bar{n}$	$\theta^{\text{FA}}_{\text{DIF}}$	$\theta^{\text{VA}}_{\text{DIF}}$	$\theta^{\text{FA}}_{\text{DCF}}$	$\theta^{\text{VA}}_{\text{DCF}}$
32	0.129	0.131	0.144	0.142
64	0.114	0.116	0.143	0.141
128	0.061	0.064	0.096	0.093
256	0.000	0.000	0.023	0.014

Table A.2: Optimized Smagorinsky coefficients for each coarse resolution, filter type, and LES model.

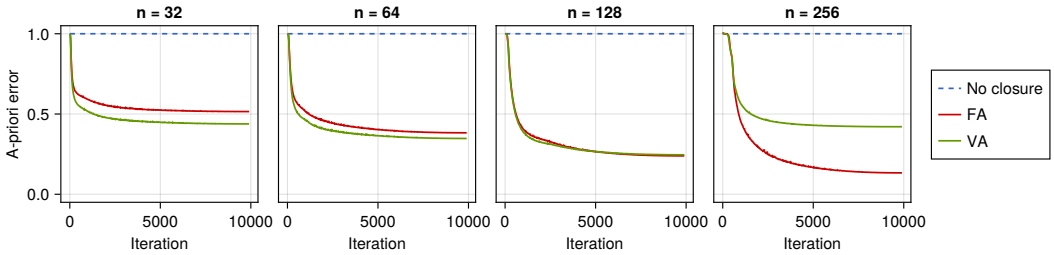


Figure A.1: Relative a-priori error on validation set during a-priori training for 10^4 iterations. From left to right: $\bar{n} = 32, 64, 128, 256$.

It is important to stress that the Smagorinsky closure term (divergence of weighted strain tensor in equation (3.26)) is *not* divergence-free, just like the other right hand side terms like convection and diffusion. As a result, the optimal Smagorinsky coefficients for $\mathcal{M}_{\text{DIF}}$ at the higher LES resolutions (where the CNN is unstable) go to zero, thus creating a divergence-free right hand side.

For the CNN, the initial model parameters θ_0 are sampled from a uniform distribution. They are improved by minimizing the stochastic loss function using the ADAM optimizer [87]. The gradients are obtained using reverse mode automatic differentiation.

We start by training using the a-priori loss function (3.24). We learn one set of parameters θ^{prior} for each of the training grids and filter types. Each time, the model is trained for 50 epochs using the Adam optimizer [87] with default weight decay and momentum parameters. Each epoch consists of iterating through the 198 training batches, each containing 64 $(\bar{u}, c)$ snapshot pairs, resulting in roughly 10^4 iterations of stochastic gradient descent. The learning rate is set at the start of each epoch using a cosine annealing scheduler [108] with initial learning rate 10^{-3} and final learning rate 10^{-6}. Every 20 iterations we evaluate the a-priori error on the validation dataset. The validation error is shown in figure A.1. Note that the commutator error

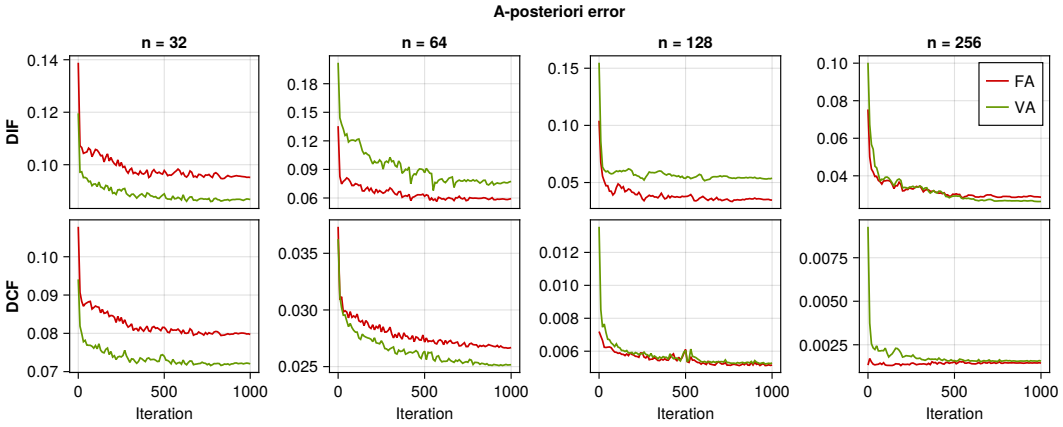


Figure A.2: Relative a-posteriori error on validation set during a-posteriori training for 1000 iterations. From left to right: $\bar{n} = 32, 64, 128, 256$.

differs for each LES resolution and filter type, so the relative a-priori error is not directly comparable between setups. The parameters giving the lowest validation error are retained after training.

Since the a-priori loss L^{prior} does not take into account the effect of the LES model we use, we fine-tune the a-priori trained CNN parameters by training using the a-posteriori loss function $L^{\text{post}}_{\mathcal{M}}$. This time, the LES model $\mathcal{M} \in \{\mathcal{M}_{\text{DIF}}, \mathcal{M}_{\text{DCF}}\}$ is part of the loss function definition. We use $n_{\text{unroll}} = 50$, and train for 1000 iterations, as shown in figure A.2. Since the weights are already converged from the a-priori training, we use a smaller learning rate with the scheduler, starting at 10^{-4} and ending at 10^{-6}. The best parameters from the a-priori training (θ^{prior}) are used as initial parameters for the a-posteriori training. The Adam optimizer is reinitialized without the history terms from the a-priori training session. The parameters $\theta^{\text{post}}_{\mathcal{M}}$ giving the lowest a-posteriori error on the validation dataset during training are retained after training.

For FA with $\bar{n} = 256$, the parameters found with a-priori training are already close to the optimum, and subsequent a-posteriori training yields only minor improvements. For VA with $\bar{n} = 256$, the same a-priori training validation error is much higher than for FA. A-posteriori training therefore improves the validation error significantly, and eventually reaches the same error as for FA. This is likely because for VA, the commutator error used in a-priori training is inconsistent with $\mathcal{M}_{\text{DCF}}$. Training with $L^{\text{post}}_{\text{DCF}}$ corrects for this inconsistency. For FA, the computed commutator errors are fully consistent with $\mathcal{M}_{\text{DCF}}$, so a-posteriori training is not needed.

A.2 DIVERGENCE-FREE FILTER

Formal proof that the face-averaging filter preserves the discrete divergence-free constraint.

In this appendix we prove that the face-averaging filter is divergence-free. The proof follows directly from the continuous divergence theorem.

Consider a coarse grid index J. The fine grid volumes Ω_I contained within $\bar{\Omega}_J$ are indexed by $I \in \mathcal{K}_J = \{I \mid \Omega_I \subset \bar{\Omega}_J\}$. The face-averaging filter is defined by

$$\bar{u}_J^\alpha = \sum_{I \in \mathcal{F}_J^\alpha} \rho_{J,I}^\alpha u_I^\alpha, \tag{A.3}$$

where $\mathcal{F}_J^\alpha = \{I \mid \Gamma_I^\alpha \in \bar{\Gamma}_J^\alpha\}$ contains the face indices and $\rho_{J,I}^\alpha$ are weights to be determined. We assume that $\rho_{J,I}^\alpha$ is independent of I_α and J_α. The fine grid divergence is given by

$$(Du)_I = \sum_{\alpha=1}^{d} (\delta_\alpha u^\alpha)_I = \sum_{\alpha=1}^{d} \frac{u_{I+h_\alpha}^\alpha - u_{I-h_\alpha}^\alpha}{|\Delta_{I_\alpha}^\alpha|} = 0. \tag{A.4}$$

The coarse grid divergence is given by

$$\begin{aligned}
(\bar{D}\bar{u})_J &= \sum_{\alpha=1}^{d} (\bar{\delta}_\alpha \bar{u}^\alpha)_J \\
&= \sum_{\alpha=1}^{d} \frac{\bar{u}_{J+h_\alpha}^\alpha - \bar{u}_{J-h_\alpha}^\alpha}{|\bar{\Delta}_{J_\alpha}^\alpha|} \quad \text{by definition of finite difference operator} \\
&= \sum_{\alpha=1}^{d} \frac{1}{|\bar{\Delta}_{J_\alpha}^\alpha|} \left(\sum_{I \in \mathcal{F}_{J+h_\alpha}^\alpha} \rho_{J,I}^\alpha u_I^\alpha - \sum_{I \in \mathcal{F}_{J-h_\alpha}^\alpha} \rho_{J,I}^\alpha u_I^\alpha \right) \quad \text{by definition of } \bar{u}_J^\alpha \\
&= \sum_{\alpha=1}^{d} \frac{1}{|\bar{\Delta}_{J_\alpha}^\alpha|} \sum_{I \in \mathcal{K}_J} \left(\rho_{J,I+h_\alpha}^\alpha u_{J,I+h_\alpha}^\alpha - \rho_{J,I-h_\alpha}^\alpha u_{J,I-h_\alpha}^\alpha \right) \quad \text{telescoping sum over } I_\alpha \\
&= \sum_{\alpha=1}^{d} \frac{1}{|\bar{\Delta}_{J_\alpha}^\alpha|} \sum_{I \in \mathcal{K}_J} \rho_{J,I}^\alpha \left(u_{I+h_\alpha}^\alpha - u_{I-h_\alpha}^\alpha \right) \quad \text{since } \rho_{J,I}^\alpha \text{ is independent of } I_\alpha \\
&= \sum_{I \in \mathcal{K}_J} \frac{|\Omega_I|}{|\bar{\Omega}_J|} \sum_{\alpha=1}^{d} \frac{|\bar{\Gamma}_J^\alpha|}{|\Gamma_I^\alpha|} \rho_{J,I}^\alpha \frac{u_{I+h_\alpha}^\alpha - u_{I-h_\alpha}^\alpha}{|\Delta_{I_\alpha}^\alpha|} \quad \text{rewrite terms with } |\Omega_I| = |\Gamma_I^\alpha||\Delta_I^\alpha| \ \forall \alpha \\
&= \sum_{I \in \mathcal{K}_J} \frac{|\Omega_I|}{|\bar{\Omega}_J|} (Du)_I \quad \text{if we choose } \rho_{J,I}^\alpha = |\Gamma_I^\alpha|/|\bar{\Gamma}_J^\alpha| \\
&= 0 \quad \text{since } (Du)_I = 0.
\end{aligned} \tag{A.5}$$

The chosen $\rho_{J,I}^\alpha$ is indeed independent of I_α and J_α. We also obtain the property $\sum_{I \in \mathcal{F}_J^\alpha} \rho_{J,I}^\alpha = 1$, so constant fine grid velocities are preserved upon

filtering. In other words, choosing $\bar{u}_J^\alpha$ as a weighted average of the DNS velocities passing through the coarse volume face $\bar{\Gamma}_J^\alpha$ yields a divergence-free $\bar{u}$. Note that the divergence constraint holds only for the face-size weights chosen above. Using arbitrary weights such as Gaussian weights would not preserve this property.

We also observe that for the general case, we can write $\bar{D}\Phi = \Psi D$ for a certain pressure filter Ψ. It is defined by

$$(\Psi p)_J = \sum_{I \in \mathcal{K}_J} \frac{|\Omega_I|}{|\bar{\Omega}_J|} p_I \tag{A.6}$$

for all fields p defined in the pressure points. If we consider a 3×3 compression on a uniform grid, with $J = (a, b)$, we effectively get $\mathcal{K}_{a,b} = \{i-1, i, i+1\} \times \{j-1, j, j+1\}$ and $|\Omega_I|/|\bar{\Omega}_J| = 1/9$ for all $I \in \mathcal{K}_J$.

A.3 DISCRETIZE-THEN-FILTER (WITHOUT DIFFERENTIATING THE CONSTRAINT)

In this appendix we show a problem that arises when discretely filtering the differential-algebraic system (3.3)-(3.4), rather than filtering the pressure-free equation (3.7). Since the discrete DNS system (3.3)-(3.4) includes divergence and pressure terms, we need to define a pressure filter Ψ (in addition to the velocity filter Φ), such that $\bar{p} = \Psi p$. This yields the following set of equations:

$$\Psi D u = 0, \tag{A.7}$$

$$\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} = \Phi F(u) - \Phi G p. \tag{A.8}$$

These can be rewritten as follows:

$$\bar{D}\bar{u} = c_D(u, \bar{u}), \tag{A.9}$$

$$\frac{\mathrm{d}\bar{u}}{\mathrm{d}t} = \bar{F}(\bar{u}) + \tilde{c}(u, \bar{u}) - \bar{G}\bar{p} + c_P(p, \bar{p}). \tag{A.10}$$

Filtering the DAE system directly introduces an extra pressure commutator error—avoided by the discretize-differentiate-filter approach.

Here $c_D(u, \bar{u}) := (\bar{D}\Phi - \Psi D)u$ represents the commutator error between the discrete divergence operator and filtering, $\tilde{c}(u, \bar{u}) := (\Phi F(u) - \bar{F}(\bar{u}))$ represents the commutator error arising from filtering $F(u)$ (note that it differs from the one used in equation (3.12)), and the pressure commutator error is $c_P(p, \bar{p}) = (\bar{G}\Psi - \Phi G)p$. In the case of a face-averaging filter, we have $c_D = 0$, which is the constraint that must be enforced by the filtered pressure when moving to the LES equations. However, in the above formulation an additional commutator error for the pressure appears, which is undesirable and it is unclear how it should be modelled. The discretize-differentiate-filter approach avoids this issue with the pressure.

A.4 CONTINUOUS FILTERS AND TRANSFER FUNCTIONS

The discrete volume-averaging filter Φ^{VA} is an approximation to the continuous top-hat volume-averaging filter g defined by

$$\begin{aligned}(g * \varphi)(x) &= \int_{\mathbb{R}^d} g(y)\varphi(x-y)\,\mathrm{d}\Omega(y) \\ &= \frac{1}{\bar{\Delta}^d}\int_{\left[-\frac{\bar{\Delta}}{2},\frac{\bar{\Delta}}{2}\right]^d} \varphi(x-y)\,\mathrm{d}\Omega(y),\end{aligned} \tag{A.11}$$

where φ is a scalar field,

$$g(x) = \frac{1}{\bar{\Delta}^d}\prod_{\alpha=1}^{d} I\left(x^\alpha \in \left[-\frac{\bar{\Delta}}{2},\frac{\bar{\Delta}}{2}\right]\right)$$

is the convolutional filter kernel, and $I(a \in A)$ is an indicator function equal to 1 if $a \in A$ and 0 otherwise. The continuous filter g can also be expressed through its transfer function G, defined by

$$G_k = \prod_{\alpha=1}^{d} \frac{\sin(\pi k_\alpha \bar{\Delta}/2)}{\pi k_\alpha \bar{\Delta}/2} \tag{A.12}$$

for a given wavenumber vector $k = (k_1, \ldots, k_d)$. The Fourier transform of $g * \varphi$ is then given by $\widehat{(g * \varphi)}_k = G_k \hat{\varphi}_k$.

Similarly, we can define the continuous top-hat face-averaging filter (with face normal in the direction α) as

$$(g^\alpha * \varphi)(x) = \int_{\mathbb{R}^d} g^\alpha(y)\varphi(x-y)\,\mathrm{d}\Omega(y), \tag{A.13}$$

where the kernel

$$g^\alpha(x) = \frac{1}{\bar{\Delta}^{d-1}}\delta(x^\alpha)\prod_{\beta\neq\alpha} I\left(x^\beta \in \left[-\frac{\bar{\Delta}}{2},\frac{\bar{\Delta}}{2}\right]\right) \tag{A.14}$$

is the same as for the volume-averaging filter, but with one interval indicator replaced by a Dirac delta function δ. The transfer function of the face-averaging filter is

$$G^\alpha_k = \prod_{\beta\neq\alpha} \frac{\sin(\pi k_\beta \bar{\Delta}/2)}{\pi k_\beta \bar{\Delta}/2}. \tag{A.15}$$

In other words, the α-face-averaging filter performs top-hat averaging in every direction except for direction α, where it leaves signals intact.

The transfer functions G, G^1, and G^2 are shown in figure A.3. The filter width is set to $\bar{\Delta} = 1/128$, corresponding to a cutoff wavenumber level at

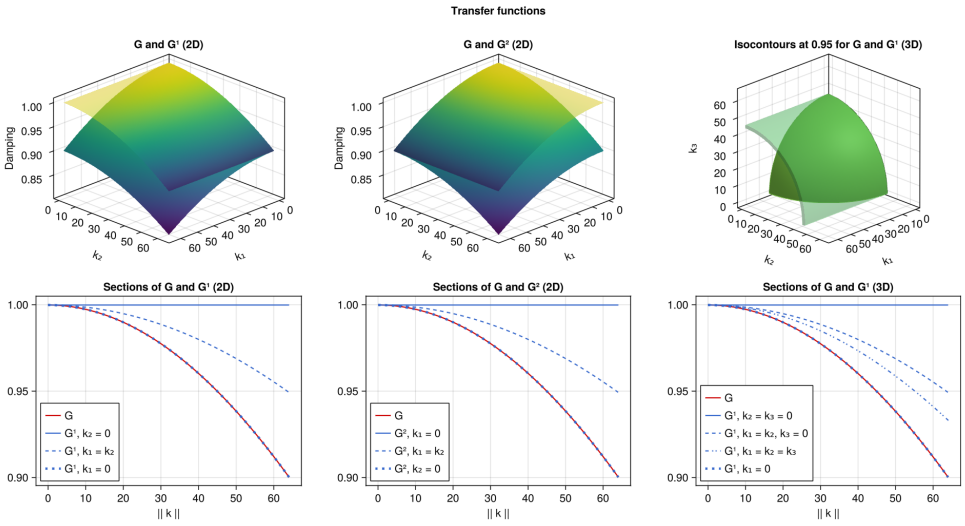


Figure A.3: Transfer functions in 2D and 3D. The two top left plots show the 2D transfer functions of the volume-averaging and face-averaging filters. The upper surface is G^{α}, the lower one is G. The top-right plot shows isocontours of the 3D transfer functions G (behind) and G^1 (transparent, in front) at a given damping level 0.95. The bottom row shows sections of the transfer functions in different wavenumber directions $k/\|k\|$ as a function of the wavenumber magnitude $\|k\|$. G does not depend on the direction.

$\|k\|_\infty = 64$. We also show the transfer functions G and G^1 for the 3D versions of the filters. In 3D, the isocontour of G at a given damping level takes the shape of a sphere, since G_k depends only on $\|k\|$. The isocontour of G^α takes the shape of a cylinder with axis along the α-direction, because the value of G^α_k does not depend on k_α.

We can clearly see that for *all* wavenumbers k, the volume-averaging filter dampens more than the face-averaging filter ($\forall k, G_k \leq G^\alpha_k$). For $k_\alpha = 0$, we have $G_k = G^\alpha_k$ (VA and FA coincide), and for $k_\alpha > 0$, we have the strict inequality $G_k < G^\alpha_k$. For all k such that $k_\beta = 0$ for $\beta \neq \alpha$, we obtain $G^\alpha_k = 1$ (FA does not filter in the normal direction).

A.5 DISCRETE AND CONTINUOUS COMMUTATOR ERRORS FOR THE 2D TAYLOR-GREEN VORTEX

In this section, we compare the discrete and continuous convective commutator errors. We use an analytical solution to the Navier-Stokes equations on a periodic domain $\Omega = [0, 2\pi]^2$, namely the Taylor-Green vortex [177]:

$$\begin{aligned} u(x,y,t) &= -\sin(x)\cos(y)\mathrm{e}^{-2\nu t}, \\ v(x,y,t) &= \cos(x)\sin(y)\mathrm{e}^{-2\nu t}, \\ p(x,y,t) &= \frac{1}{4}\left(\cos(2x)+\cos(2y)\right)\mathrm{e}^{-4\nu t}. \end{aligned} \tag{A.16}$$

In the remainder of this appendix, u, v, and p refer to this solution only, and the results that follow are specific to this solution. We also drop the time dependence and consider $u(x,y) = u(x,y,0)$ and similarly for v and p.

A.5.1 *Top-hat filter*

First, we note that

$$\int_{x-\Delta/2}^{x+\Delta/2} \sin(x')\,\mathrm{d}x' = 2\sin(x)\sin(\Delta/2), \tag{A.17}$$

$$\int_{x-\Delta/2}^{x+\Delta/2} \cos(x')\,\mathrm{d}x' = 2\cos(x)\sin(\Delta/2). \tag{A.18}$$

Applying a volume-averaging top-hat filter $\varphi \mapsto \bar{\varphi}$ of width Δ yields the following filtered solution:

$$\begin{aligned} \bar{u}(x,y) &= -\frac{1}{\Delta^2}\int_{x-\Delta/2}^{x+\Delta/2}\sin(x')\,\mathrm{d}x' \int_{y-\Delta/2}^{y+\Delta/2}\cos(y')\,\mathrm{d}y' \\ &= \mathrm{sinc}^2(\Delta/2)u(x,y), \end{aligned} \tag{A.19}$$

where $\mathrm{sinc}(x) = \sin(x)/x$ is the transfer function of the 1D top-hat filter. A similar derivation for $\bar{v}$ and $\bar{p}$ gives

$$\bar{u} = \mathrm{sinc}^2(\Delta/2)u, \quad \bar{v} = \mathrm{sinc}^2(\Delta/2)v, \quad \bar{p} = \mathrm{sinc}(\Delta)p. \tag{A.20}$$

A.5.2 *Continuous convective commutator error*

For the continuous commutator error in the Navier-Stokes equations, we consider the nonlinearities $\bar{u}\bar{u}$, $\bar{u}\bar{v}$, $\bar{v}\bar{u}$, and $\bar{v}\bar{v}$. They are given by

$$\begin{aligned}
\bar{u}\bar{u} &= \frac{1}{4}\,\mathrm{sinc}^4(\Delta/2)(1 - \cos(2x) + \cos(2y) - \cos(2x)\cos(2y)),\\
\bar{v}\bar{v} &= \frac{1}{4}\,\mathrm{sinc}^4(\Delta/2)(1 + \cos(2x) - \cos(2y) - \cos(2x)\cos(2y)),\\
\bar{u}\bar{v} &= -\frac{1}{4}\,\mathrm{sinc}^4(\Delta/2)\sin(2x)\sin(2y).
\end{aligned} \tag{A.21}$$

Next, we are interested in the quantities $\overline{uu}$, $\overline{uv}$, $\overline{vu}$, and $\overline{vv}$. Integrating uu, vv, and uv over x and y separately gives

$$\begin{aligned}
\overline{uu} &= \frac{1}{4}(1 - \mathrm{sinc}(\Delta)(\cos(2x) - \cos(2y)) - \mathrm{sinc}^2(\Delta)\cos(2x)\cos(2y)),\\
\overline{vv} &= \frac{1}{4}(1 + \mathrm{sinc}(\Delta)(\cos(2x) - \cos(2y)) - \mathrm{sinc}^2(\Delta)\cos(2x)\cos(2y)),\\
\overline{uv} &= -\frac{1}{4}\,\mathrm{sinc}^2(\Delta)\sin(2x)\sin(2y).
\end{aligned} \tag{A.22}$$

Finally, we can assemble the sub-filter stress tensor τ:

$$\begin{aligned}
\tau_{xx} &= \overline{uu} - \bar{u}\bar{u}\\
&= \frac{1}{4}\Big(1 - \mathrm{sinc}^4(\Delta/2) + (\mathrm{sinc}^4(\Delta/2) - \mathrm{sinc}(\Delta))(\cos(2x) - \cos(2y))\\
&\quad + (\mathrm{sinc}^4(\Delta/2) - \mathrm{sinc}^2(\Delta))\cos(2x)\cos(2y)\Big),\\
\tau_{xy} &= \overline{uv} - \bar{u}\bar{v}\\
&= \frac{1}{4}(\mathrm{sinc}^4(\Delta/2) - \mathrm{sinc}^2(\Delta))\sin(2x)\sin(2y),
\end{aligned} \tag{A.23}$$

and similarly for τ_{yy}. Finally, the sub-filter force terms $c_x = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y}$ and $c_y = \frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y}$ are given by

$$\begin{aligned}
c_x &= -\frac{1}{2}(\mathrm{sinc}^4(\Delta/2) - \mathrm{sinc}(\Delta))\sin(2x),\\
c_y &= -\frac{1}{2}(\mathrm{sinc}^4(\Delta/2) - \mathrm{sinc}(\Delta))\sin(2y).
\end{aligned} \tag{A.24}$$

Since the filter width is small compared to the domain size (2π), we can assume that higher powers of Δ may be negligible. Using the Taylor series expansions

$$\begin{aligned} \operatorname{sinc}(x) &= 1 - \frac{1}{6}x^2 + \frac{1}{120}x^4 + \mathcal{O}(x^6), \\ \operatorname{sinc}^4(x) &= 1 - \frac{2}{3}x^2 + \frac{1}{5}x^4 + \mathcal{O}(x^6), \end{aligned} \tag{A.25}$$

we get the coefficient

$$\operatorname{sinc}^4(\Delta/2) - \operatorname{sinc}(\Delta) = \frac{1}{240}\Delta^4 + \mathcal{O}(\Delta^6). \tag{A.26}$$

If we drop the terms of order $\mathcal{O}(\Delta^6)$ and higher, we obtain the following simplified expressions for the continuous commutator error:

$$\begin{aligned} c_x &= -\frac{\Delta^4}{480}\sin(2x) + \mathcal{O}(\Delta^6), \\ c_y &= -\frac{\Delta^4}{480}\sin(2y) + \mathcal{O}(\Delta^6). \end{aligned} \tag{A.27}$$

A.5.3 *Discrete convective commutator error*

Next, we consider the discrete commutator error computed from a discrete DNS. Assume for simplicity that the DNS field is the exact solution sampled at the DNS grid points. Consider a grid point (x, y). Let d denote the DNS grid spacing.

A.5.3.1 *Discrete filter*

Assume that $\Delta/2 = nd$ for some integer n. A straightforward discretization of the top-hat volume-averaging filter at a coarse grid point (x, y) is given by

$$\begin{aligned} (\Phi u)(x, y) &= \frac{1}{(2n+1)^2} \sum_{i=-n}^{n} \sum_{j=-n}^{n} u(x + id, y + jd) \\ &= -\frac{1}{(2n+1)^2} \sum_{i=-n}^{n} \sin(x + id) \sum_{j=-n}^{n} \cos(y + jd), \end{aligned} \tag{A.28}$$

and similarly for Φv and Φp. For the x-part, we get

$$\begin{aligned} \sum_{i=-n}^{n} \sin(x + id) &= \sum_{i=-n}^{n} (\sin(x)\cos(id) + \cos(x)\sin(id)) \\ &= \left(\sum_{i=-n}^{n} \cos(id) \right) \sin(x). \end{aligned} \tag{A.29}$$

A similar derivation for the y-part, v and p give

$$\Phi u = G_{n,d}^2 u, \quad \Phi v = G_{n,d}^2 v, \quad \Phi p = G_{n,2d} p, \tag{A.30}$$

where $G_{n,d} = \frac{1}{2n+1} \sum_{i=-n}^{n} \cos(id)$ is the transfer function of the discrete top-hat filter in 1D.

A.5.3.2 *Convective term*

We now compute the discrete convective term $\mathrm{conv}_{x,d}(u, v)$ which contributes to the equation for u. This term is required at a u-velocity grid point (x, y), so we need the quantity uu at $(x + d/2, y)$ and $(x - d/2, y)$, and the quantity uv at $(x, y + d/2)$ and $(x, y - d/2)$. Knowing that the native points of v are $(x \pm d/2, y \pm d/2)$ when (x, y) is the u-point, we proceed through interpolation.

$A_x(u)$ is obtained by interpolating u left and right:

$$\begin{aligned} A_x(u)(x + d/2, y) &= \frac{1}{2}(u(x, y) + u(x + d, y)) \\ &= -\frac{1}{2}\sin(x)\cos(y)(1 + \cos(d)) \\ &\quad + \frac{1}{2}\cos(x)\cos(y)\sin(d). \end{aligned} \tag{A.31}$$

$A_x(v)$ is obtained by interpolating v left and right:

$$\begin{aligned} A_x(v)(x, y + d/2) &= \frac{1}{2}(v(x - d/2, y + d/2) + v(x + d/2, y + d/2)) \\ &= \frac{1}{2}\cos(x)(\sin(y)(1 + \cos(d)) + \cos(y)\sin(d)), \end{aligned} \tag{A.32}$$

$A_y(u)$ is obtained by interpolating u up and down:

$$\begin{aligned} A_y(u)(x, y + d/2) &= \frac{1}{2}(u(x, y) + u(x, y + d)) \\ &= -\frac{1}{2}\sin(x)(\cos(y)(1 + \cos(d)) - \sin(y)\sin(d)), \end{aligned} \tag{A.33}$$

Now we can assemble the quadratic terms.

$$\begin{aligned} (A_x(u)A_x(u))(x + d/2, y) = \frac{1}{16}&((1 - \cos(2x))(1 + \cos(2y))(1 + \cos(d))^2 \\ &+ 2\sin(2x)(1 + \cos(2y))\sin(d)(1 + \cos(d)) \\ &+ (1 + \cos(2x))(1 + \cos(2y))\sin(d)^2), \end{aligned} \tag{A.34}$$

$$\begin{aligned} (A_y(u)A_x(v))(x, y + d/2) = &-\frac{1}{16}\sin(2x)\sin(2y)(1 + 2\cos(d) + \cos(2d)) \\ &- \frac{1}{8}\sin(2x)\cos(2y)\sin(d)(1 + \cos(d)), \end{aligned}$$

(A.35)

The discrete convective term at a point (x, y) is defined as

$$\begin{aligned} C_{xx,d} &= \frac{1}{d}\left((A(u)A(u))(x + d/2, y) - (A(u)A(u))(x - d/2, y)\right) \\ &= \frac{1}{4d}\sin(2x)(1 + \cos(2y))\sin(d)(1 + \cos(d)), \end{aligned} \tag{A.36}$$

$$\begin{aligned} C_{xy,d} &= \frac{1}{d}\left((A_y(u)A_x(v))(x, y + d/2) - (A_y(u)A_x(v))(x, y - d/2)\right) \\ &= -\frac{1}{4d}\sin(2x)\cos(2y)\sin(d)(1 + \cos(d)), \end{aligned} \tag{A.37}$$

$$\mathrm{conv}_{x,d}(u, v) = C_{xx,d} + C_{xy,d} = \frac{1}{4}\sin(2x)(\mathrm{sinc}(d) + \mathrm{sinc}(2d)), \tag{A.38}$$

A similar derivation for the y-component gives

$$\mathrm{conv}_{y,d}(u, v) = C_{yx,d} + C_{yy,d} = \frac{1}{4}\sin(2y)(\mathrm{sinc}(d) + \mathrm{sinc}(2d)). \tag{A.39}$$

Using the Taylor series expansion (A.25) for the expression

$$\mathrm{sinc}(d) + \mathrm{sinc}(2d) = 2 - \frac{5}{6}d^2 + \frac{17}{120}d^4 + \mathcal{O}(d^6), \tag{A.40}$$

and recalling that the continuous convective x-term is $\frac{1}{2}\sin(2x)$, we obtain

$$\mathrm{conv}_{x,d}(u, v) = \frac{\partial}{\partial x}(uu) + \frac{\partial}{\partial y}(uv) + \mathcal{O}(d^2), \tag{A.41}$$

and similarly for the y-component. This confirms that the computation of the derivatives is second order accurate.

Next, we compute the filtered convective force $\Phi\,\mathrm{conv}_x(u, v)$. We obtain

$$\begin{aligned} \Phi\,\mathrm{conv}_x(u, v) &= \frac{1}{(2n + 1)^2}\sum_{i=-n}^{n}\sum_{j=-n}^{n}\frac{1}{4}\sin(2(x + id))(\mathrm{sinc}(d) + \mathrm{sinc}(2d)) \\ &= G_{n,2d}\,\mathrm{conv}_{x,d}(u, v), \end{aligned} \tag{A.42}$$

where the transfer function is now applied on $2d$ instead of d. Noting that $\Phi u = G_{n,d}^2 u$ and $\Phi v = G_{n,d}^2 v$ everywhere on the domain, we get

$$\begin{aligned} A_{x,\Delta}(\Phi u)A_{x,\Delta}(\Phi u) &= G_{n,d}^4 A_{x,\Delta}(u)A_{x,\Delta}(u), \\ A_{y,\Delta}(\Phi u)A_{x,\Delta}(\Phi v) &= G_{n,d}^4 A_{x,\Delta}(u)A_{y,\Delta}(v), \\ \mathrm{conv}_{x,\Delta}(\Phi u, \Phi v) &= G_{n,d}^4\,\mathrm{conv}_{x,\Delta}(u, v) \\ &= \frac{1}{4}G_{n,d}^4(\mathrm{sinc}(\Delta) + \mathrm{sinc}(2\Delta))\sin(2x), \end{aligned} \tag{A.43}$$

Using a similar derivation for the y-component, we can now compute the *discrete* convective commutator error

$$\begin{aligned}\Phi\,\mathrm{conv}_{x,d}(u,v) - \mathrm{conv}_{x,\Delta}(\Phi u, \Phi v) &= -E\frac{1}{4}\sin(2x),\\ \Phi\,\mathrm{conv}_{y,d}(u,v) - \mathrm{conv}_{y,\Delta}(\Phi u, \Phi v) &= -E\frac{1}{4}\sin(2y),\end{aligned} \tag{A.44}$$

where

$$E = G_{n,d}^4(\mathrm{sinc}(\Delta) + \mathrm{sinc}(2\Delta)) - G_{n,2d}(\mathrm{sinc}(d) + \mathrm{sinc}(2d)) \tag{A.45}$$

is the commutator coefficient.

It is important to note that at any given point (x, y), the values of the discrete and continuous convective commutator errors (A.44) and (A.24) differ. The dependence on (x, y) is the same, but the coefficient in front differs. If a closure model is imposed using the traditional approach of filtering first, but trained using commutator error target data computed from DNS (discretizing first), there is an inconsistency between the model and the learning environment.

A.6 ADDITIONAL NUMERICAL EXPERIMENTS

We include additional numerical experiments to confirm that our methods work in a variety of settings.

A.6.1 *LES of decaying turbulence (2D)*

We repeat the experiment from 3.5.2, but without the body force. This results in a decaying turbulence setup, where energy dissipates over time.

Figure A.4 shows the total kinetic energy evolution for the decaying turbulence test case. Since our discretization is energy-conserving, the DNS energy cannot increase in the absence of a body force, and it can only decrease due to dissipation. If D_2 denotes the discrete diffusion operator, it can be shown that the kinetic energy $E = \frac{1}{2}\|u\|_\Omega^2 = \frac{1}{2}\langle u, u\rangle_\Omega$ satisfies

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \langle u, D_2 u\rangle_\Omega. \tag{A.46}$$

For $\mathcal{M}_{\mathrm{DCF}}$, all models produce decaying energy profiles, but the CNN energy is the most accurate. The no-closure model is slightly too dissipative, and the Smagorinsky model is even more dissipative. For $\mathcal{M}_{\mathrm{DIF}}$, the no-closure and Smagorinsky models have similar profiles as for $\mathcal{M}_{\mathrm{DCF}}$, but the CNN models become unstable. Training with the a-posteriori loss function corrects for this instability, leading to correct energy levels for a longer time, but the instability is still present.

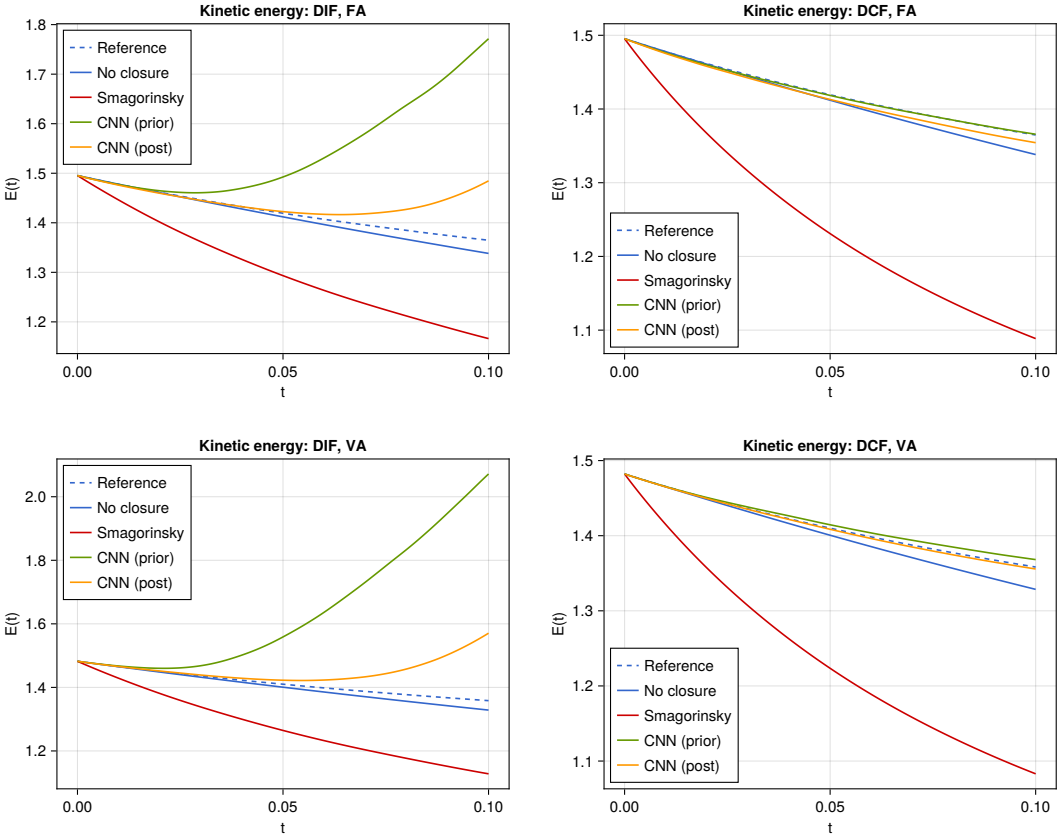


Figure A.4: Total kinetic energy evolution for the 2D decaying turbulence case at $\bar{n} = 128$. **Left:** Unprojected closure model $\mathcal{M}_{\mathrm{DIF}}$. **Right:** Constrained model $\mathcal{M}_{\mathrm{DCF}}$. **Top:** Face-averaging filter. **Bottom:** Volume-averaging filter.

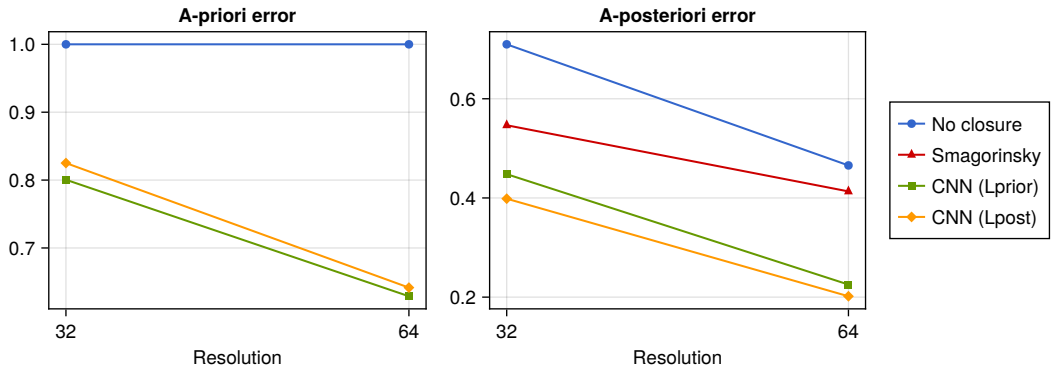


Figure A.5: A-priori and a-posteriori errors at time $t = 0.29$ for the 3D case.

A.6.2 *LES of forced turbulence (3D)*

To demonstrate that the formulation also works in 3D, we consider a decaying turbulence test case in a periodic box $\Omega = [0, 1]^3$ with $N = 1024^3$ finite volumes. We use single precision floating point numbers, Reynolds number $\mathrm{Re} = 2000$, simulation time $t_{\mathrm{end}} = 3$, burn-in time $t_{\mathrm{burn}} = 0.5$, and 10 random initial conditions. Unlike for the extensive analysis in section 3.5.2, we only consider 2 LES resolutions $\bar{n} \in \{32, 64\}$, one filter (face-averaging), and one LES model ($\mathcal{M}_{\mathrm{DCF}}$). We use the same body force as in the 2D case, namely f with $f^{\alpha}(x) = 5\delta_{\alpha=1} \sin(8\pi x^2)$. We use the same CNN architecture as in the 2D case, but with a kernel size $5 \times 5 \times 5$ instead of 5×5, resulting in 234 096 learnable parameters instead of 45 696. For the Smagorinsky model, we use the common choice of $\theta = 0.17$.

Figure A.5 shows the a-priori and a-posteriori errors for the 3D case at time $t = 0.29$. As in the 2D case (Chapters/Pap2/figures 3.10 and 3.11), the a-priori trained CNN produces lower a-priori errors, while the a-posteriori trained CNN produces lower a-posteriori errors. For both training methods, the CNN performs better than the no-closure model and Smagorinsky.

Figure A.6 shows the energy evolution in the 3D case. As in the 2D case (figure 3.12), the CNN energy stays close to the target energy much longer than the no-closure and Smagorinsky energies. Additionally, the Smagorinsky energy seems to be more dissipative, while for the 2D case this was not visible due to the energy injection from the body force.

Figure A.7 shows the energy spectra in the 3D case. A clear peak is visible at the body force injection wavenumber at $\kappa = 4$. The Smagorinsky model is clearly too dissipative in the high wavenumbers. The CNN is outperforming both of the two other models, and the spectrum stays close long after the LES solution has become decorrelated from the reference solution, confirming

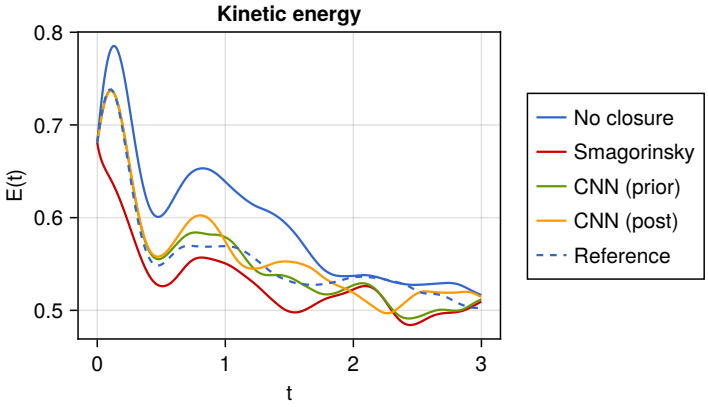


Figure A.6: Energy evolution for the 3D case with $\bar{n} = 64$.

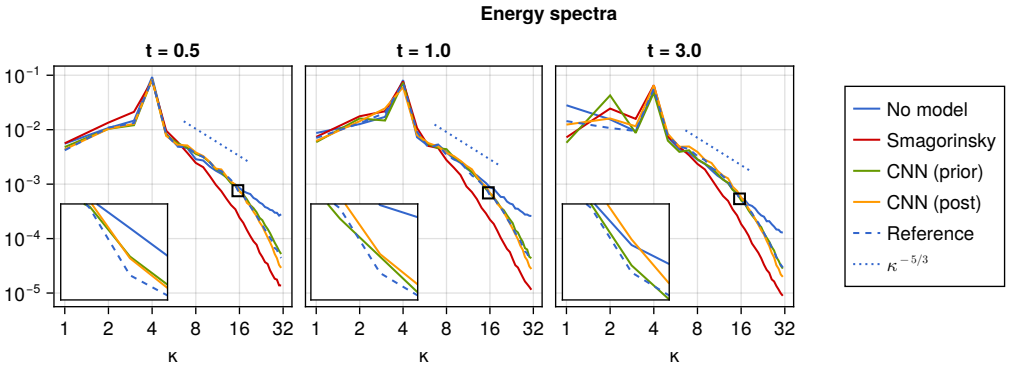


Figure A.7: Energy spectra for the 3D case with $\bar{n} = 64$.

that while the closure model is not able to predict the exact chaotic solution for long times, it is able to track the turbulent statistics.

B

SUPPLEMENTARY MATERIAL FOR CHAPTER 4

B.1 GRID-COMPATIBLE OPERATORS AND RESTRICTION

We say that an operator $o : U \to U$ is h-collocated if, for all $x \in \Omega$, $o(u)(x)$ only depends on the values $\{u(x + ih) \mid i \in \mathbb{Z}\}$. Similarly, o is h-staggered if $o(u)(x)$ only depends on $\{u(x + ih + h/2) \mid i \in \mathbb{Z}\}$.

The finite difference ∂_x^h, interpolator η_x^h, and numerical Burgers flux r^h from eqs. (4.13) to (4.15) are staggered since $\partial_x^h u(x)$, $\eta_x^h u(x)$, and $r^h(u)(x)$ depend on $u(x \pm h/2)$, and not on $u(x \pm h)$. When chained together, the staggered operators become collocated. For example, the expressions

$$\partial_x^h \partial_x^h u(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}, \tag{B.1}$$

$$\partial_x^h \eta_x^h u(x) = \frac{u(x+h) - u(x-h)}{2h}, \tag{B.2}$$

only depend on $u(x - h)$, $u(x)$, and $u(x + h)$. These expressions do not depend on u at the half-points $x \pm h/2$.

We say that an operator (such as the discrete Laplacian $\partial_x^h \partial_x^h$) is *compatible* with a grid of spacing h if the operator is collocated. This means that equations involving the operator form a closed system of equations once restricted to the grid points. In particular, a flux term of the form $\partial_x^h r^h$ is grid-compatible if $r^h : U \to U$ is a staggered numerical flux. We therefore say that a flux is grid-compatible if it is staggered.

Grid compatibility ensures that discrete operators form a closed system on the grid points.

At a point x, the numerical Burgers flux $r^h(u) \coloneqq (\eta_x^h u)(\eta_x^h u)/2 - \nu \partial_x^h u$ takes the form

$$\begin{aligned} r^h(u)(x) = \frac{1}{8}\left[u\left(x - \frac{h}{2}\right) + u\left(x + \frac{h}{2}\right)\right]^2 \\ - \frac{\nu}{h}\left[u\left(x + \frac{h}{2}\right) - u\left(x - \frac{h}{2}\right)\right], \end{aligned} \tag{B.3}$$

and the discrete flux divergence is

$$\begin{aligned} \partial_x^h r^h(u)(x) &= \frac{1}{h}\left[r^h(u)\left(x + \frac{h}{2}\right) - r^h(u)\left(x - \frac{h}{2}\right)\right] \\ &= \frac{1}{8h}\left[u(x+h)^2 - u(x-h)^2\right] \\ &+ \frac{1}{8h} u(x)\left[u(x+h) - u(x-h)\right] \\ &- \frac{\nu}{h^2}\left[u(x+h) - 2u(x) + u(x-h)\right]. \end{aligned} \tag{B.4}$$

This expression can be evaluated at any point $x \in \Omega$. If we evaluate the continuous field $\partial_x^h r^h(u)$ at the grid points $x_i^h := ih$, $i \in \mathbb{Z}$, and define the restriction $u_i^h := u(x_i^h)$, we can use the more common (but less general) notation style

$$\begin{aligned}\partial_x^h r^h(u)(x_i^h) = \frac{1}{8h}\left[(u_{i+1}^h)^2 - (u_{i-1}^h)^2 + u_i^h\left(u_{i+1}^h - u_{i-1}^h\right)\right] \\ - \frac{\nu}{h^2}\left[u_{i+1}^h - 2u_i^h + u_{i-1}^h\right].\end{aligned} \tag{B.5}$$

This restricted form, where the expression is written only in terms of $(u_i^h)_{i\in\mathbb{Z}}$, is only possible to write because $\partial_x^h r^h$ is a grid-compatible operator. We still use the continuous notation like in eqs. (B.3), (B.4) and (4.15) for analysis and only use the restricted form like (B.5) for computer evaluation.

B.2 GRID-COMPATIBLE COARSE-GRAINING

A practical limitation of the LES-FVM framework is that we cannot compute target solutions $\bar{u}^{\Delta,h}$ and target residual fluxes $\tau^{\Delta,h}(u)$ unless we have access to a continuous solution u at all points $x \in \Omega$. In practice, reference data is obtained by solving the DNS equation

$$L^h(u^h) = 0 \tag{B.6}$$

with a sufficiently small DNS grid spacing h. This gives the DNS solution u^h at the h-grid points $x_i^h := ih$, $i \in \mathbb{Z}$. The LES-FVM problem is then formulated on a coarser grid with spacing $H > h$. The goal is to approximate the fields $\bar{u}^{\Delta,H}(x_i^H)$ and $\tau^{\Delta,H}(u)(x_i^H)$ evaluated at the H-grid points $(x_i^H)_{i\in\mathbb{Z}}$ using only the DNS solution $u^h(x_i^h)$ evaluated at the h-grid points $(x_i^h)_{i\in\mathbb{Z}}$. This framework is illustrated in fig. B.1. This two-grid formulation should approximate the one-grid formulation and converge to it as h goes to 0.

The two-grid formulation makes the LES-FVM RST computable from DNS data.

B.2.1 *1D filters*

For the 1D case, we propose the following DNS approximations of the filters that fit our criteria. Choose $H := (2n+1)h$ for some $n \in \mathbb{N}$. The FVM filter f^H is replaced by the h-compatible filter $f_h^H : U \to U$ using the quadrature rule for all $u \in U$ and $x \in \Omega$ as

$$f_h^H u(x) := \frac{1}{2n+1}\sum_{i=-n}^{n} u(x + ih). \tag{B.7}$$

This filter is designed to satisfy a discrete equivalent of the filter-swap commutation property (4.19):

$$\partial_x^H = f_h^H \partial_x^h. \tag{B.8}$$

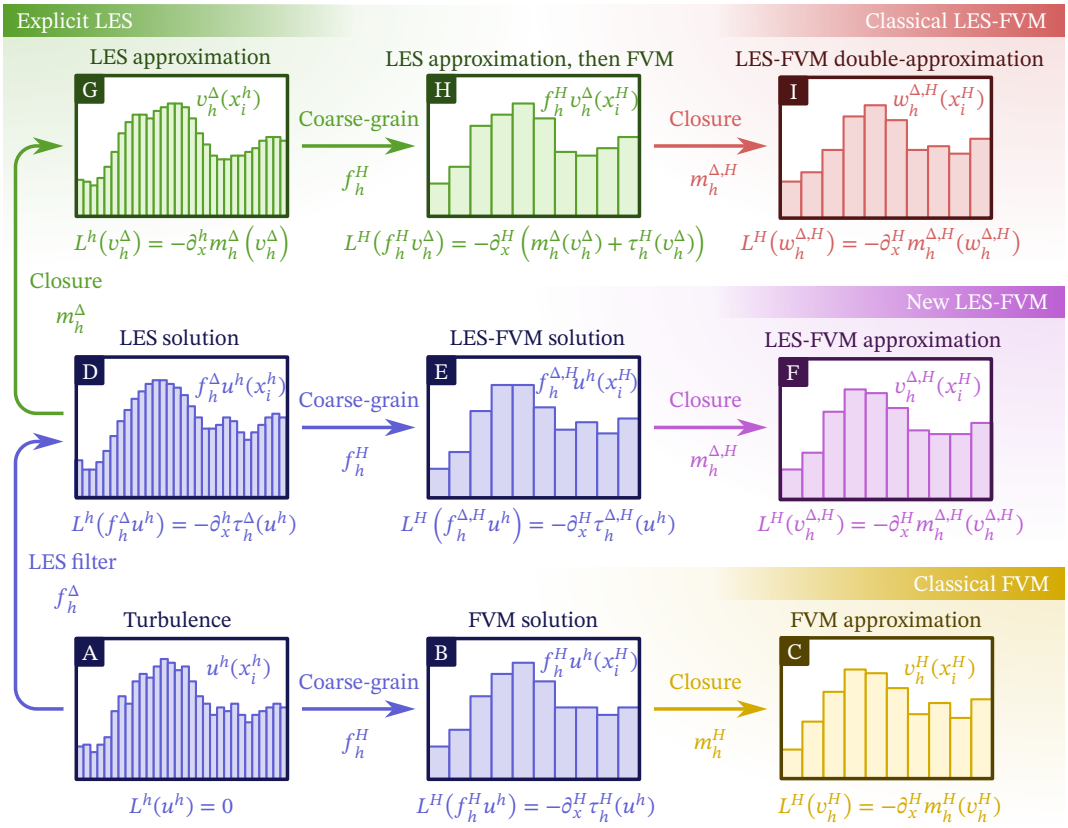


Figure B.1: Same as fig. 4.1, but starting from the DNS solution u^h instead of the continuous solution u. In the limit when h goes to zero, this figure becomes equivalent to fig. 4.1. The fine grid points are $(x_i^h)_{i\in\mathbb{Z}}$, and the coarse grid points are $(x_i^H)_{i\in\mathbb{Z}}$.

The discrete filter-swap property: coarse-graining and finite differencing commute exactly.

For proof, see theorem 5.

Assume that the LES kernel G^Δ of the LES filter f^Δ is compactly supported (e.g. top-hat) or decays sufficiently fast at infinity (e.g. Gaussian). We then replace f^Δ with the h-compatible filter $f_h^\Delta : U \to U$ defined as

$$f_h^\Delta u(x) = \frac{\sum_{r=-R}^{R} G^\Delta(rh) u(x - rh)}{\sum_{r=-R}^{R} G^\Delta(rh)} \tag{B.9}$$

for some sufficiently large $R \in \mathbb{N}$. The denominator ensures that the discrete kernel is normalized. For the Gaussian filter (4.38), the standard deviation is $\sigma = \Delta/\sqrt{12}$. We then choose R such that $Rh \geq 3\sigma$, i.e. $R := \lceil 3\Delta/(\sqrt{12}h) \rceil$, where $\lceil \cdot \rceil$ is the ceiling function. The filter f_h^Δ is h-compatible since $f_h^\Delta u(x)$ only depends on u at x modulo h.

Applying the coarse-graining h-compatible LES-FVM filter $f_h^{\Delta,H} := f_h^H f_h^\Delta$ to the DNS equation (B.6) gives the LES-FVM equation

$$f_h^{\Delta,H} L^h(u^h) = 0. \tag{B.10}$$

By adding the resolved term $\partial_x^H r^H(f_h^{\Delta,H} u^h)$ to both sides and using the discrete filter-swap property (B.8), we get the LES-FVM equation in H-grid conservative form:

$$L^H(f_h^{\Delta,H} u^h) = -\partial_x^H \tau_h^{\Delta,H}(u^h) \tag{B.11}$$

(see fig. B.1-E), where $f_h^{\Delta,H} u^h$ is the coarse-grained LES-FVM solution we intend to solve for and

$$\boxed{\tau_h^{\Delta,H}(u^h) := f_h^\Delta r^h(u^h) - r^H\left(f_h^{\Delta,H} u^h\right)} \tag{B.12}$$

is the residual flux in the LES-FVM equation. Equation (B.11) is analogous to the LES-FVM equation (4.34), but it is written using H-grid divergences ∂_x^H.

Note that the FVM filters f^h, f^H, and f_h^H are related through the property

$$f^H = f_h^H f^h. \tag{B.13}$$

For proof, see theorem 4. At fixed H, we can show that $f_h^H \to f^H$ and $f_h^\Delta \to f^\Delta$ as $h \to 0$. This means that if the DNS is fully resolved, we recover the continuous setting.

Odd compression factors are required for staggered grid consistency.

A limitation of our FVM filter f_h^H is that the compression factor is required to be odd, i.e. $H = (2n+1)h$ for some n, and not $H = 2nh$. For an odd compression factor, we can compute both $f_h^{\Delta,H} u^h$ and $\tau_h^{\Delta,H}(u^h)$ in the required staggered grid points exactly, without performing any interpolations. This would not be possible for an even compression factor, due to the way the staggered coarse and fine grids overlap.

B.2.2 *Coarse-graining filters in 3D*

The 1D h-compatible LES and FVM filters extend naturally to 3D by applying them successively in each coordinate direction. The 1D FVM filter $g_{h,i}^H : U \to U$ can be defined as

$$g_{h,i}^H u(x) \coloneqq \frac{1}{2n+1} \sum_{r=-n}^{n} u(x + rhe_i), \tag{B.14}$$

where e_i is the i-th unit vector. The volume-averaging and surface-averaging FVM filters then follow as $f_h^H \coloneqq g_{h,1}^H g_{h,2}^H g_{h,3}^H$, $f_h^{H,1} \coloneqq g_{h,2}^H g_{h,3}^H$, $f_h^{H,2} \coloneqq g_{h,1}^H g_{h,3}^H$, and $f_h^{H,3} \coloneqq g_{h,1}^H g_{h,2}^H$.

Like their one-grid counterparts, our proposed two-grid filters have the commutation properties (with no sum over i)

$$g_{h,i}^H \partial_i^h = \partial_i^H, \tag{B.15}$$

$$f_h^H \partial_i^h = \partial_i^H f_h^{H,i}. \tag{B.16}$$

Given a DNS solution u^h restricted to the h-grid, target data pairs $f_h^{\Delta,H,\pi} u^h$ and $\tau_h^{\Delta,H,\pi}(u^h)$ can be computed and then evaluated in the required H-grid points to assess the performance of LES-FVM closures.

B.3 PROOFS OF COMMUTATION PROPERTIES

Here we provide proofs for various properties used in this article. We recall that Ω is a periodic 1D domain, U is the space of periodic 1D fields on Ω, f^Δ is a spatial convolutional LES filter (see eq. (4.5)), f^h is an FVM filter (see eq. (4.17)), f_h^Δ is a h-grid-compatible LES filter (see eq. (B.7)), f_h^H is a h-grid-compatible FVM filter (see eq. (B.7)), ∂_x^h is a finite difference operator (see eq. (4.13)), h is a grid spacing, and $H = (2n+1)h$ is a coarse grid spacing for some $n \in \mathbb{N}$.

Note that for non-uniform filters or bounded domains, some of the commutation properties may no longer hold. Here, we only consider periodic domains.

Theorem 1. *Spatial convolutional filters f^Δ commute with differentiation [17]:*

$$f^\Delta \partial_x = \partial_x f^\Delta. \tag{B.17}$$

Proof. Let $u \in U$ and $x \in \Omega$. Then

$$\begin{aligned}
\partial_x \bar{u}^\Delta(x) &= \partial_x \left[\int_{\mathbb{R}} G^\Delta(x-y) u(y)\,\mathrm{d}y \right] \\
&= \int_{\mathbb{R}} \partial_x [G^\Delta(x-y)] u(y)\,\mathrm{d}y \\
&= \int_{\mathbb{R}} (\partial_x G^\Delta)(x-y) u(y)\,\mathrm{d}y \\
&= -\int_{\mathbb{R}} \partial_y [G^\Delta(x-y)] u(y)\,\mathrm{d}y \\
&= \int_{\mathbb{R}} G^\Delta(x-y) \partial_y u(y)\,\mathrm{d}y \\
&= \overline{\partial_x u}^\Delta(x).
\end{aligned} \tag{B.18}$$

where we first used Leibniz's rule to interchange differentiation and integration, then the chain rule, and then integration by parts (assuming that the kernel goes to zero at infinity). Since this holds for all u and x, we have $f\partial_x = \partial_x f$. □

Theorem 2. *Coarse-graining and differentiation do not commute:*

$$\partial_x^h f^h \neq f^h \partial_x. \tag{B.19}$$

Non-commutation is the source of discretization error in classical LES.

Proof. Let $u \in U$. The Taylor series expansion of u around a point x is

$$\begin{aligned}
u\left(x + \frac{h}{2}\right) = u(x) + \frac{h}{2}\partial_x u(x) + \frac{h^2}{8}\partial_{xx} u(x) \\
+ \frac{h^3}{48}\partial_{xxx} u(x) + \frac{h^4}{384}\partial_{xxxx} u(x) + \mathcal{O}(h^5),
\end{aligned} \tag{B.20}$$

where $\partial_{xx} \coloneqq \partial_x \partial_x$ etc. Subtracting a similar expansion of $u(x - h/2)$ makes the even terms cancel out. The expansion of the finite difference operator ∂_x^h is therefore reduced to

$$\partial_x^h = \partial_x - \frac{h^2}{24}\partial_{xxx} + \mathcal{O}(h^4). \tag{B.21}$$

This gives

$$\begin{aligned}
\partial_x^h f^h &= \partial_x f^h - \frac{h^2}{24}\partial_{xxx} f^h + \mathcal{O}(h^4) \\
&= f^h \partial_x - \frac{h^2}{24} f^h \partial_{xxx} + \mathcal{O}(h^4) \\
&\neq f^h \partial_x,
\end{aligned} \tag{B.22}$$

since the operator $f^h \partial_{xxx}$ is non-zero. Here we used theorem 1 to swap f^h and ∂_x. Note that for a few special cases, such as velocity fields with $\partial_{xxx} u = 0$ (and similarly for higher order derivatives), we do get $\partial_x^h \bar{u}^h = \overline{\partial_x u}^h$. □

Theorem 3. *The finite difference ∂_x^h can be written as a composition between the FVM filter and an exact derivative:*

$$\partial_x^h = f^h \partial_x. \tag{B.23}$$

Proof. The fundamental theorem of calculus states that

Proof by the fundamental theorem of calculus—elegant and exact.

$$\int_a^b \partial_x u \, \mathrm{d}x = u(b) - u(a) \tag{B.24}$$

for all $(a, b) \in \mathbb{R}^2$. For $u \in U$ and $x \in \Omega$, this gives

$$\begin{aligned} \overline{\partial_x u}^h(x) &= \frac{1}{h} \int_{x-h/2}^{x+h/2} \partial_y u(y) \, \mathrm{d}y \\ &= \frac{1}{h} \left[u\left(x + \frac{h}{2}\right) - u\left(x - \frac{h}{2}\right) \right] \\ &= \partial_x^h u(x). \end{aligned} \tag{B.25}$$

Since this holds for all u and x, we have $\partial_x^h = f^h \partial_x$. □

We now show the properties of the coarse-graining filter f_h^H.

Theorem 4. *The average over $H := (2n+1)h$ can be written as a composition between a coarse-graining filter and the average over h:*

$$f^H = f_h^H f^h, \tag{B.26}$$

where $n \in \mathbb{N}$.

Proof. Let $u \in U$ and $x \in \Omega$. Then

$$\begin{aligned} f_h^H \bar{u}^h(x) &= \frac{1}{2n+1} \sum_{i=-n}^{n} \frac{1}{h} \int_{x+ih-h/2}^{x+ih+h/2} u(y) \, \mathrm{d}y \\ &= \frac{1}{(2n+1)h} \int_{x-(2n+1)h/2}^{x+(2n+1)h/2} u(y) \, \mathrm{d}y \\ &= \frac{1}{H} \int_{x-H/2}^{x+H/2} u(y) \, \mathrm{d}y \\ &= \bar{u}^H(x), \end{aligned} \tag{B.27}$$

where we used the property $\int_a^b u(x)\mathrm{d}x + \int_b^c u(x)\mathrm{d}x = \int_a^c u(x)\mathrm{d}x$ for all $(a, b, c) \in \mathbb{R}^3$ to combine the integrals in the sum. Since this holds for all u and x, we have $f^H = f^{h \to H} f^h$. □

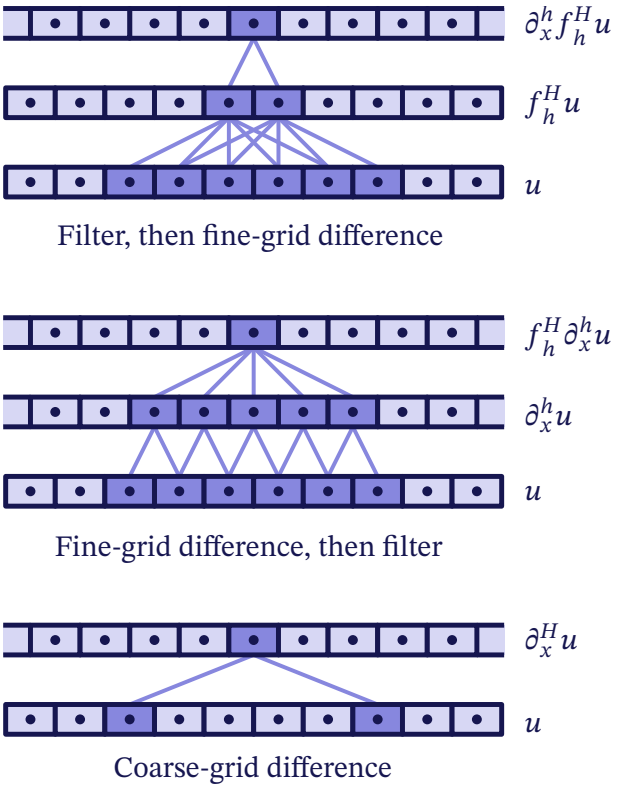


Figure B.2: For all $u \in U$, the three terms $\partial_x^H u$, $f_h^H \partial_x^h u$, and $\partial_x^h f_h^H u$ are equal. Here we show the three quantities on the h-grid with $H \coloneqq 5h$.

Theorem 5. *A finite difference over $H \coloneqq (2n+1)h$ can be written as a composition between a coarse-graining filter and a finite difference over h:*

$$\partial_x^H = f_h^H \partial_x^h, \tag{B.28}$$

where $n \in \mathbb{N}$.

Proof. Using theorems 3 and 4, we have

$$\partial_x^H = f^H \partial_x = f_h^H f^h \partial_x = f_h^H \partial_x^h. \tag{B.29}$$

□

Note that theorem 5 can be extended to include three terms: $\partial_x^H = \partial_x^h f_h^H = f_h^H \partial_x^h$, since filtering and finite differencing do commute *if we do not coarse-grain the derivative*. These three terms are shown in fig. B.2 for $H = 5h$. The three terms are equal since all the intermediate terms cancel out in the telescoping sum in f_h^H. We restrict the fields to the h-grid to visualize how the inner terms cancel out.

B.4 PRESSURE PROJECTION FOR VECTORS AND STRESS TENSORS

The pressure projection enables writing incompressible NS as a self-contained conservation law.

The system (4.70) consists of an evolution equation subject to a spatial constraint. By defining a pressure projection operator, these equations can be combined into one self-contained evolution equation.

In this appendix, we introduce two projection operators, one that makes vector fields in U^3 divergence-free and one that makes stress tensor fields in $U^{3\times3}$ divergence-preserving. We provide proofs for the continuous case. For the discrete case, the proofs are identical, the only difference being that ∂_i is replaced by ∂_i^h.

B.4.1 *Pressure projector for vector fields*

The pressure field p enforces the continuity equation $\partial_j u_j = 0$. In eq. (4.70), combining the continuity equation with the momentum equation gives the Poisson equation for the pressure field:

$$-\partial_k \partial_k p = \partial_i \partial_j \sigma_{ij}(u). \tag{B.30}$$

By solving this equation explicitly for the pressure, we can write a "pressure-free" momentum equation as

$$\partial_t u_i + \pi_{ij} \partial_k \sigma_{jk}(u) = 0, \tag{B.31}$$

where

$$\pi_{ij} \coloneqq \delta_{ij} - \partial_i (\partial_k \partial_k)^\dagger \partial_j \tag{B.32}$$

is a pressure projection operator (see theorem 6 for proof), δ_{ij} is the Kronecker symbol, and the inverse Laplacian $(\partial_k \partial_k)^\dagger : \varphi \mapsto p$ maps scalar fields φ to the unique solution to the Poisson equation $\partial_k \partial_k p = \varphi$ subject to the additional constraint of an average pressure of zero, i.e. $\int_\Omega p \, \mathrm{d}V = 0$. For our periodic domain, the pressure field is determined up to a constant. We are free to choose the constant this way since it subsequently disappears in the pressure gradient $\partial_i p$.

Theorem 6. *The operator $\pi_{ij} \coloneqq \delta_{ij} - \partial_i (\partial_k \partial_k)^\dagger \partial_j$ is a projector onto the space of divergence-free vector fields, i.e. $\partial_i \pi_{ij} = 0$ (the output of π is divergence-free) and $\pi\pi = \pi$ [37, 178, 196].*

Proof. We have $\partial_i \partial_i (\partial_j \partial_j)^\dagger = 1$, since $(\partial_j \partial_j)^\dagger$ gives solutions to the Poisson equation. This can be used to show that π_{ij} makes vector fields divergence-free. The divergence ∂_i composed with π_{ij} is

$$\partial_i \pi_{ij} = \underbrace{\partial_i \delta_{ij}}_{\partial_j} - \underbrace{\partial_i \partial_i (\partial_k \partial_k)^\dagger}_{1} \partial_j = \partial_j - \partial_j = 0. \tag{B.33}$$

This means that for all $u \in U^3$, $\partial_i \pi_{ij} u_j = 0$, so πu is divergence-free (even if u is not). Additionally, we get

$$\begin{aligned}\pi_{ij}\pi_{jk} &= \delta_{ij}\delta_{jk} \\ &\quad - \delta_{ij}\partial_j(\partial_\alpha\partial_\alpha)^\dagger\partial_k - \delta_{jk}\partial_i(\partial_\alpha\partial_\alpha)^\dagger\partial_j \\ &\quad + \partial_i(\partial_\alpha\partial_\alpha)^\dagger \underbrace{\partial_j\partial_j(\partial_\beta\partial_\beta)^\dagger}_{1}\partial_k \\ &= \delta_{ik} - 2\partial_i(\partial_\alpha\partial_\alpha)^\dagger\partial_k + \partial_i(\partial_\alpha\partial_\alpha)^\dagger\partial_k \\ &= \pi_{ik}.\end{aligned} \tag{B.34}$$

Since $\pi\pi = \pi$, we can conclude that π is idempotent. □

The projected momentum equation (B.31) automatically enforces the continuity equation at all times *by construction* (as long as $\partial_j u_j = 0$ at the initial time). The pressure term and continuity equation can be ignored, at the cost of making the momentum equations non-local (the inverse Laplacian is non-local).

B.4.2 *Pressure projector for tensor fields*

We say that a stress tensor $\sigma \in U^{3\times3}$ is divergence-preserving if $\partial_j\sigma_{ij}$ is divergence-free, i.e. $\partial_i\partial_j\sigma_{ij} = 0$. Define the tensor projector $\pi : U^{3\times3} \to U^{3\times3}$ as

$$\pi_{ij\alpha\beta} \coloneqq \delta_{i\alpha}\delta_{j\beta} - \delta_{ij}\left(\partial_k\partial_k\right)^\dagger \partial_\alpha\partial_\beta. \tag{B.35}$$

This operator maps stress tensors to stress tensors.

Theorem 7. *The operator $\pi_{ij\alpha\beta}$ is a projector onto the space of divergence-preserving stress tensors, i.e. $\partial_i\partial_j\pi_{ij\alpha\beta} = 0$ (the output of π is divergence-preserving) and $\pi\pi = \pi$.*

Proof. The double-divergence $\partial_i\partial_j$ composed with the operator $\pi_{ij\alpha\beta}$ is

$$\begin{aligned}\partial_i\partial_j\pi_{ij\alpha\beta} &= \delta_{i\alpha}\delta_{j\beta}\partial_i\partial_j - \delta_{ij}\partial_i\partial_j\left(\partial_k\partial_k\right)^\dagger\partial_\alpha\partial_\beta \\ &= \partial_\alpha\partial_\beta - \underbrace{\partial_i\partial_i\left(\partial_k\partial_k\right)^\dagger}_{1}\partial_\alpha\partial_\beta \\ &= \partial_\alpha\partial_\beta - \partial_\alpha\partial_\beta \\ &= 0.\end{aligned} \tag{B.36}$$

This means that for all $\sigma \in U^{3\times3}$, we have $\partial_i\partial_j(\pi_{ij\alpha\beta}\sigma_{\alpha\beta}) = 0$, so $\pi\sigma$ is a divergence-preserving tensor.

Furthermore, applying the operator twice gives

$$\begin{aligned}
\pi_{ij\alpha\beta}\pi_{\alpha\beta mn} =& \left(\delta_{i\alpha}\delta_{j\beta} - \delta_{ij}(\partial_k\partial_k)^\dagger\partial_\alpha\partial_\beta\right) \\
& \left(\delta_{\alpha m}\delta_{\beta n} - \delta_{\alpha\beta}(\partial_l\partial_l)^\dagger\partial_m\partial_n\right) \\
=&(\delta_{i\alpha}\delta_{j\beta})(\delta_{\alpha m}\delta_{\beta n}) \\
&-(\delta_{i\alpha}\delta_{j\beta})\delta_{\alpha\beta}(\partial_l\partial_l)^\dagger\partial_m\partial_n \\
&-(\delta_{\alpha m}\delta_{\beta n})\delta_{ij}(\partial_k\partial_k)^\dagger\partial_\alpha\partial_\beta \\
&+\delta_{ij}\delta_{\alpha\beta}(\partial_k\partial_k)^\dagger\partial_\alpha\partial_\beta(\partial_l\partial_l)^\dagger\partial_m\partial_n \\
=&\delta_{im}\delta_{jn} \\
&-\delta_{ij}(\partial_l\partial_l)^\dagger\partial_m\partial_n \\
&-\delta_{ij}(\partial_k\partial_k)^\dagger\partial_m\partial_n \\
&+\delta_{ij}(\partial_k\partial_k)^\dagger \underbrace{\partial_\alpha\partial_\alpha(\partial_l\partial_l)^\dagger}_{1}\partial_m\partial_n \\
=&\delta_{im}\delta_{jn} + (-2+1)\delta_{ij}(\partial_k\partial_k)^\dagger\partial_m\partial_n \\
=&\pi_{ijmn}.
\end{aligned} \tag{B.37}$$

Since $\pi\pi = \pi$, we can conclude that π is idempotent. □

The vector-projector π_{ij} and tensor-projector $\pi_{ij\alpha\beta}$ satisfy the following commutation property for the tensor-divergence.

Theorem 8. *Projection and tensor-divergence commute, i.e. for all stress tensors $\sigma \in U^{3\times3}$, we have*

$$\pi_{ij}\partial_k\sigma_{jk} = \partial_j\pi_{ij\alpha\beta}\sigma_{\alpha\beta}. \tag{B.38}$$

Proof. Let $\sigma \in U^{3\times3}$ be a stress tensor. The tensor-divergence ∂_j of the projected stress tensor $\pi_{ij\alpha\beta}\sigma_{\alpha\beta}$ is

$$\begin{aligned}
\partial_j\pi_{ij\alpha\beta}\sigma_{\alpha\beta} &= \partial_j\left(\delta_{i\alpha}\delta_{j\beta} - \delta_{ij}(\partial_k\partial_k)^\dagger\partial_\alpha\partial_\beta\right)\sigma_{\alpha\beta} \\
&= \left(\delta_{i\alpha}\partial_\beta - \partial_i(\partial_k\partial_k)^\dagger\partial_\alpha\partial_\beta\right)\sigma_{\alpha\beta} \\
&= \left(\delta_{i\alpha} - \partial_i(\partial_k\partial_k)^\dagger\partial_\alpha\right)\partial_\beta\sigma_{\alpha\beta} \\
&= \pi_{i\alpha}\partial_\beta\sigma_{\alpha\beta},
\end{aligned} \tag{B.39}$$

which is the projected tensor-divergence of σ. □

Note that we use the same symbol π for both the vector and stress tensor projectors. It should be clear from the context which version is used.

B.5 FVM FOR THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS IN ALTERNATIVE NOTATION

We present the incompressible Navier-Stokes equations in an alternative notation to highlight how the RST in the FVM equation becomes non-local

and non-symmetric. In Cartesian notation, the incompressible Navier-Stokes equations read

$$\partial_x u_x + \partial_y u_y + \partial_z u_z = 0, \tag{B.40}$$

$$\partial_t u_x + \partial_x (\sigma_{xx} + p) + \partial_y \sigma_{xy} + \partial_z \sigma_{xz} = 0, \tag{B.41}$$

$$\partial_t u_y + \partial_x \sigma_{yx} + \partial_y (\sigma_{yy} + p) + \partial_z \sigma_{yz} = 0, \tag{B.42}$$

$$\partial_t u_z + \partial_x \sigma_{zx} + \partial_y \sigma_{zy} + \partial_z (\sigma_{zz} + p) = 0, \tag{B.43}$$

where

$$\sigma \coloneqq u \otimes u - \nu \left(\nabla u + \nabla u^T \right), \tag{B.44}$$

$$u \otimes u \coloneqq \begin{pmatrix} u_x u_x & u_x u_y & u_x u_z \\ u_y u_x & u_y u_y & u_y u_z \\ u_z u_x & u_z u_y & u_z u_z \end{pmatrix}, \tag{B.45}$$

$$\nabla u \coloneqq \begin{pmatrix} \partial_x u_x & \partial_y u_x & \partial_z u_x \\ \partial_x u_y & \partial_y u_y & \partial_z u_y \\ \partial_x u_z & \partial_y u_z & \partial_z u_z \end{pmatrix}. \tag{B.46}$$

In vector notation, we can write the equations as

$$\nabla \cdot u = 0, \quad \partial_t u + \nabla \cdot (\sigma + p\delta) = 0. \tag{B.47}$$

If we eliminate the pressure, this system can be written in projection-form as three equations

$$\partial_t u_x + \partial_x r_{xx} + \partial_y r_{xy} + \partial_z r_{xz} = 0, \tag{B.48}$$

$$\partial_t u_y + \partial_x r_{yx} + \partial_y r_{yy} + \partial_z r_{yz} = 0, \tag{B.49}$$

$$\partial_t u_z + \partial_x r_{zx} + \partial_y r_{zy} + \partial_z r_{zz} = 0, \tag{B.50}$$

or, in vector notation,

$$\partial_t u + \nabla \cdot r = 0, \tag{B.51}$$

where

$$r \coloneqq \pi\sigma \coloneqq \sigma + p\delta \tag{B.52}$$

is the projected stress tensor,

$$p \coloneqq - \triangle^{\dagger} \nabla \cdot \nabla \cdot \sigma, \tag{B.53}$$

is a pressure field that depends non-locally on σ,

$$\triangle \coloneqq \partial_{xx} + \partial_{yy} + \partial_{zz}, \tag{B.54}$$

is the Laplacian, $A^{\dagger} : b \mapsto x$ gives the unique solution (up to a constant) to the equation $Ax = b$ (we can for example design $A^{\dagger}$ to return the solution with zero mean), and

$$\begin{aligned} \nabla \cdot \nabla \cdot \sigma \coloneqq{}& \partial_{xx}\sigma_{xx} + \partial_{yy}\sigma_{yy} + \partial_{zz}\sigma_{zz} \\ &+ 2 \left(\partial_{xy}\sigma_{xy} + \partial_{xz}\sigma_{xz} + \partial_{yz}\sigma_{yz} \right). \end{aligned} \tag{B.55}$$

B.5.1 *The stress tensor in the FVM equation*

Here we write the FVM equation in "unresolved" form, without introducing the resolved part $\nabla^h \cdot r^h(\bar{u}^h)$ or the numerical stress tensor r^h, in order to understand the properties of the stress tensor.

The FVM equation is

$$\partial_t \bar{u}^h + \overline{\nabla \cdot \pi\sigma}^h = 0. \tag{B.56}$$

For the tensor divergence, the discrete filter-swap property (4.92) can be written as $\overline{\nabla \cdot r}^h = \nabla^h \cdot \bar{r}^{h,*}$ for all tensors r, where $\nabla^h := (\partial_x^h, \partial_y^h, \partial_z^h)$ and

$$\bar{r}^{h,*} := \begin{pmatrix} \bar{r}_{xx}^{h,x} & \bar{r}_{xy}^{h,y} & \bar{r}_{xz}^{h,z} \\ \bar{r}_{yx}^{h,x} & \bar{r}_{yy}^{h,y} & \bar{r}_{yz}^{h,z} \\ \bar{r}_{zx}^{h,x} & \bar{r}_{zy}^{h,y} & \bar{r}_{zz}^{h,z} \end{pmatrix} \tag{B.57}$$

contains the surface-averaging filters $f^{h,x} := g_y^h g_z^h$, $f^{h,y} := g_x^h g_z^h$, and $f^{h,z} := g_x^h g_y^h$ for 1D top-hat filters g_i^h in direction i. This gives the ∇^h-conservation law for $\bar{u}^h$:

$$\partial_t \bar{u}^h + \nabla^h \cdot \overline{\pi\sigma}^{h,*} = 0. \tag{B.58}$$

The stress tensor appearing in the ∇^h-conservation law for $\bar{u}^h$ is therefore

$$\begin{aligned} \overline{\pi\sigma}^{h,*} &= \overline{\sigma + p\delta}^{h,*} \\ &= \begin{pmatrix} \bar{\sigma}_{xx}^{h,x} & \bar{\sigma}_{xy}^{h,y} & \bar{\sigma}_{xz}^{h,z} \\ \bar{\sigma}_{yx}^{h,x} & \bar{\sigma}_{yy}^{h,y} & \bar{\sigma}_{yz}^{h,z} \\ \bar{\sigma}_{zx}^{h,x} & \bar{\sigma}_{zy}^{h,y} & \bar{\sigma}_{zz}^{h,z} \end{pmatrix} + \begin{pmatrix} \bar{p}^{h,x} & 0 & 0 \\ 0 & \bar{p}^{h,y} & 0 \\ 0 & 0 & \bar{p}^{h,z} \end{pmatrix}. \end{aligned} \tag{B.59}$$

This tensor is clearly non-symmetric, since

$$f^{h,x} \neq f^{h,y} \neq f^{h,z}. \tag{B.60}$$

The FVM stress tensor is inherently non-symmetric due to different surface-averaging filters per direction.

Furthermore, the surface-averaged pressure tensor

$$\overline{p\delta}^{h,*} = \begin{pmatrix} \bar{p}^{h,x} & 0 & 0 \\ 0 & \bar{p}^{h,y} & 0 \\ 0 & 0 & \bar{p}^{h,z} \end{pmatrix} \tag{B.61}$$

is not an isotropic tensor (it cannot be written as $q^h\delta$ for some scalar field q^h). This is one of the reasons why the volume-averaged field $\bar{u}^h$ is not ∇^h-divergence-free, i.e. $\nabla^h \cdot \bar{u}^h \neq 0$.

Define the ∇^h-divergence-free part of $\bar{u}^h$ as

$$\bar{u}^{h,\pi} \coloneqq \pi^h \bar{u}^h \coloneqq \bar{u}^h - \nabla^h \left(\triangle^h\right)^\dagger \nabla^h \cdot \bar{u}^h, \tag{B.62}$$

where π^h is a vector-version of a discrete pressure projector and $\triangle^h \coloneqq \partial^h_{xx} + \partial^h_{yy} + \partial^h_{zz}$. The equation for $\bar{u}^{h,\pi}$ is

$$\partial_t \bar{u}^{h,\pi} + \nabla^h \cdot \pi^h \overline{\overline{\pi\sigma}}^{h,*} = 0, \tag{B.63}$$

where $\pi^h : \sigma \mapsto \sigma - (\triangle^h)^\dagger \nabla^h \cdot \nabla^h \cdot \sigma$ here denotes the tensor version of the discrete pressure projector (this should be clear from context). We used theorem 8 to swap ∇^h and π^h. In the tensor $\pi^h \overline{\overline{\pi\sigma}}^{h,*}$, all the isotropic parts of $\overline{\overline{\pi\sigma}}^{h,*}$ are absorbed by π^h. But since the tensor $\overline{\overline{p\delta}}^{h,*}$ is non-isotropic, it does not get absorbed, and cannot be removed from the expression $\pi^h \overline{\overline{\pi\sigma}}^{h,*}$. This is also why the pressure projector π cannot be taken outside the surface-averaging filter. The stress tensor in the ∇^h-conservation law for $\bar{u}^{h,\pi}$ is therefore non-local in the external unresolved field u, which p depends on. This also remains the case even if we *reintroduce* the incompressibility constraint for $\bar{u}^{h,\pi}$ as

The non-isotropic filtered pressure is the mechanism that makes the RST non-local.

$$\nabla^h \cdot \bar{u}^{h,\pi} = 0, \quad \partial_t \bar{u}^{h,\pi} + \nabla^h \cdot \left(\overline{\overline{\pi\sigma}}^{h,*} + q^h \delta \right) = 0, \tag{B.64}$$

where q^h is the unique pressure (up to a constant) such that $\overline{\overline{\pi\sigma}}^{h,*} + q^h\delta = \pi^h \overline{\overline{\pi\sigma}}^{h,*}$ and thus $\bar{u}^{\pi,h}$ remains ∇^h-divergence-free. Importantly, $q^h \neq \bar{p}^h$ is not the filtered pressure field, since ∇^h-incompressibility and ∇-incompressibility are different types of constraints. The original pressure field p is therefore still present in the stress tensor, even though the equations are in incompressibility-constraint-form.

The FVM analysis shows that closure models should be both non-symmetric and non-local—motivating CNN-based approaches.

The fact that the unresolved stress tensor in the equation for $\bar{u}^{h,\pi}$ is non-local in u also suggests that a closure model for this unresolved stress tensor should be non-local in $\bar{u}^{h,\pi}$. If $m(\bar{u}^{h,\pi}) \approx \overline{\overline{\pi\sigma(u)}}^{h,*}$ is a closure model for the total unresolved stress tensor, then it should ideally be

1. non-symmetric ($m^T \neq m$),
2. non-local ($m(\bar{u}^{h,\pi})(x)$ should depend on $\bar{u}^{h,\pi}(x+d)$ for potentially large displacements $d \in \mathbb{R}^3$).

C

SUPPLEMENTARY MATERIAL FOR CHAPTER 5

C.1 PARAMETRIZATION OF THE 3D ORTHOGONAL GROUP ON CARTESIAN GRIDS

For vectors x and tensors σ, we employ two indexing notations:

$$x_i = x[i], \quad \sigma_{ij} = \sigma[i, j]. \tag{C.1}$$

Square bracket notation is used when indices involve expressions more complex than a single symbol. This convention also applies to the Kronecker delta $\delta_{ij} = \delta[i, j]$, which equals 1 if $i = j$ and 0 otherwise.

Definition 1 (Orientation of space)**.** *Let $(i, j) \in \{1, 2, 3\}^2$ be two directions. We say that i and j are oriented* positively*, and write $i \prec j$, if $(i, j) \in \{(1, 2), (2, 3), (3, 1)\}$. If $j \prec i$ we say that i and j are oriented* negatively *and write $i \succ j$.*

If i, j, k are three distinct directions such that $i \prec j \prec k$, then e_i, e_j, e_k are oriented like the thumb, index and middle finger of the right hand.

The orthogonal group $O(3)$ comprises all rotations and reflections in 3D space. On uniform Cartesian grids, we restrict attention to a grid-compatible subgroup of $O(3)$, where rotations are limited to multiples of $\pi/2$. This restriction ensures that transformed grids align with the original grid. We denote this subgroup as $G \subset O(3)$. This is the *octahedral group* (or "cube" group), representing the symmetries of an octahedron (or equivalently, a cube).

The octahedral group G has 48 elements: 24 rotations and 24 roto-reflections, capturing all symmetries of a Cartesian grid.

G contains $|G| = 48$ elements: 24 are orientation-preserving rotations (elements of the special orthogonal group $SO(3)$), while the remaining 24 include a reflection.

We employ multiple *representations* of G. One representation uses 3D rotation matrices. Let

$$R_1(\theta) \coloneqq \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}, \tag{C.2}$$

$$R_2(\theta) \coloneqq \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}, \tag{C.3}$$

$$R_3(\theta) \coloneqq \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{C.4}$$

$$S_3(s) \coloneqq \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix}. \tag{C.5}$$

These are the operators that rotate 3D vectors around the e_1, e_2, and e_3 axes respectively, while $S_3(s)$ is a reflection across the plane orthogonal to e_3. In terms of Euler angles, any element in $O(3)$ can be represented as

$$R(s, \theta_1, \theta_2, \theta_3) \coloneqq S_3(s)R_3(\theta_3)R_2(\theta_2)R_1(\theta_1). \tag{C.6}$$

The angles θ_3, θ_2 and θ_1 are called *yaw*, *pitch*, and *roll*, respectively. We are free to choose S_3 or any similar reflection operator S_1 or S_2.

Euler's rotation theorem states that a rigid body displacement where one point remains fixed is a rotation around an axis. A way to represent a rotation (without the reflection) is through the *Euler-Rodrigues formula*. Denoting θ and u as the angle and axis of rotation, respectively, with $\|u\| = 1$, the formula reads

$$\begin{aligned} R(u, \theta) &\coloneqq \cos(\theta)I + \sin(\theta)U + (1 - \cos(\theta))u \otimes u \\ &= I + \sin(\theta)U + (1 - \cos(\theta))UU, \end{aligned} \tag{C.7}$$

where

$$U \coloneqq \begin{pmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{pmatrix} \tag{C.8}$$

is the *cross product matrix* of the vector u. For a vector $x \in \mathbb{R}^3$, we have $Ux = u \times x$, where $\times$ denotes the cross product. In index notation, the formula reads

$$R_{ij}(u, \theta) \coloneqq \delta_{ij} \cos(\theta) + u_i u_j (1 - \cos(\theta)) - \epsilon_{ijk} u_k \sin(\theta), \tag{C.9}$$

where ϵ is the Levi-Civita symbol defined as

$$\epsilon_{ijk} \coloneqq \begin{cases} +1 & \text{if } i \prec j \prec k, \\ -1 & \text{if } i \succ j \succ k, \\ 0 & \text{otherwise.} \end{cases} \tag{C.10}$$

The active 2D part of the rotation matrices R_1, R_2, and R_3, namely

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}, \tag{C.11}$$

only takes four distinct values for elements of the octahedral group G, where θ is a multiple of $\pi/2$:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \tag{C.12}$$

Hence, any roto-reflection matrix for elements in G only contains the values $+1$, and -1. Each row and column only has one non-zero entry. This means that for each element $g \in G$, the corresponding rotation matrix R can be written as a product between a diagonal sign-flipping matrix S and a permutation matrix P as $R = SP$.

There are 6 permutation matrices $P \in \mathbb{R}^{3\times 3}$ with entries

$$P_{ij} \coloneqq \delta[p_i, j], \tag{C.13}$$

where the 6 permutation vectors p are given by

$$p \in \left\{ \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} \right\}. \tag{C.14}$$

There are 8 sign-flipping matrices $S \coloneqq \operatorname{diag}(s) \in \mathbb{R}^{3\times 3}$ with diagonals given by

$$s \in \left\{ \begin{pmatrix} + \\ + \\ + \end{pmatrix}, \begin{pmatrix} - \\ + \\ + \end{pmatrix}, \begin{pmatrix} + \\ - \\ + \end{pmatrix}, \begin{pmatrix} + \\ + \\ - \end{pmatrix}, \begin{pmatrix} - \\ - \\ + \end{pmatrix}, \begin{pmatrix} + \\ - \\ - \end{pmatrix}, \begin{pmatrix} - \\ + \\ - \end{pmatrix}, \begin{pmatrix} - \\ - \\ - \end{pmatrix} \right\}, \tag{C.15}$$

where $+$ and $-$ mean $+1$ and -1, respectively.

For a vector $x = (x_1, x_2, x_3) \in \mathbb{R}^3$, we get the transformed vector (with no sum over i)

$$(Rx)_i = (SPx)_i = s_i x[p_i]. \tag{C.16}$$

For a tensor $\sigma \in \mathbb{R}^{3\times3}$, we get the transformed tensor (with no sum over i and j)

$$\left(R\sigma R^T\right)_{ij} = s_i s_j \sigma[p_i, p_j]. \tag{C.17}$$

We encode the elements of G as g_{ij}, where $i \in \{1, \dots, 6\}$ denotes the permutation number in (C.14) and $j \in \{1, \dots, 8\}$ denotes the sign-flipping number in (C.15). With a "column-major" ordering, the group elements are uniquely identified by a linear index $k \coloneqq i + 6(j-1)$, and we define the "flat ordering" $g_k \coloneqq g_{ij}$. Thus $G = \{g_k \mid k \in \{1, \dots, 48\}\}$.

Definition 2 (Regular representation)**.** *The regular representation of G is a mapping $\rho : G \to \{0,1\}^{|G|\times|G|}$. The entries for an element $g \in G$ are given by*

$$\rho_{ij}(g) = \delta[g_i, gg_j], \tag{C.18}$$

where $(g_i)_{i=1}^{|G|}$ are the group elements of G ordered with the flat ordering and gg_j is a composition of the elements g and g_j.

In other words, $\rho(g) \in \mathbb{R}^{48\times48}$ is a permutation matrix that maps all the group elements to the elements they become after applying g. For an element $g_j \in G$, the composed roto-reflection gg_j is equal to the unique roto-reflection g_i such that $\rho_{ij}(g) = 1$.

Definition 3 (Cayley table)**.** *The Cayley table of G is a matrix $C \in \mathbb{N}^{|G|\times|G|}$ with entries*

$$C_{ij} = k\delta[g_k, g_i g_j]. \tag{C.19}$$

If $g_k = g_i g_j$, then $C_{ij} = k$.

C.2 PSEUDO-SPECTRAL DISCRETIZATION OF THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

In a periodic box $\Omega \coloneqq [0, L]^3$ of side length $L > 0$, the incompressible Navier-Stokes equations can be expressed in spectral form as

$$\xi_j u_j = 0, \quad \partial_t u_i + \xi_j \left(\sigma_{ij}(u) + p\delta_{ij}\right) = f_i, \tag{C.20}$$

where

$$\sigma_{ij}(u) \coloneqq \widehat{\check{u}_i \check{u}_j} - \nu(\xi_j u_i + \xi_i u_j) \tag{C.21}$$

is the spectral stress tensor, $\xi \coloneqq 2\pi \mathrm{i} k/L$ is the spectral derivative operator, i is the imaginary unit, $k \in \mathbb{Z}^3$ is the wavenumber vector, $u(k,t) \in \mathbb{C}^3$ is the spectral velocity field, $p(k,t) \in \mathbb{C}$ is the spectral pressure, $f(k,t) \in \mathbb{C}^3$ is the spectral body force, $\nu > 0$ is the kinematic viscosity, and $\widehat{(\cdot)}$ and $\widetilde{(\cdot)}$ denote

the forward and inverse Fourier transforms, respectively. These equations result from applying the Fourier transform to the Navier-Stokes equations and replacing partial derivatives ∂_j with spectral derivatives ξ_j.

Define the spectral pressure projectors

$$\pi_{ij} \coloneqq \delta_{ij} - \frac{k_i k_j}{k_\gamma k_\gamma}, \tag{C.22}$$

$$\pi_{ij\alpha\beta} \coloneqq \delta_{i\alpha}\delta_{j\beta} - \delta_{ij}\frac{k_\alpha k_\beta}{k_\gamma k_\gamma}. \tag{C.23}$$

When $k = (0,0,0)$, the above expressions are ill-defined, and we set $\pi \coloneqq 1$. We can write the pressure-free momentum-equations as

$$\partial_t u_i + \xi_j \pi_{ij\alpha\beta}\sigma_{\alpha\beta}(u) = \pi_{ij} f_j. \tag{C.24}$$

C.2.1 *Discretization*

We represent the velocity field $\check{u}(x,t)$ on a Cartesian grid with N^3 collocated points (with N even). The forward and inverse Fourier transforms $\widehat{(\cdot)}$ and $\widecheck{(\cdot)}$ are approximated by the discrete Fourier transforms (DFTs) $\widehat{(\cdot)}^N$ and $\widecheck{(\cdot)}^N$. They are defined as follows, for a spectral field $u(k)$ and a physical field $v(x)$:

$$\hat{v}(k) \coloneqq \int_\Omega v(x)\exp(-\xi_j x_j)\,\mathrm{d}x \tag{C.25}$$

$$\check{u}(x) \coloneqq \sum_{k\in\mathbb{Z}^3} u(k)\exp(\xi_j x_j) \tag{C.26}$$

$$\tag{C.27}$$

This gives the spectral velocity field $u(k,t)$ at N^3 wavenumbers. For example, we have $k_1 \in \{-N/2, \dots, N/2-1\}$, and similarly for k_2 and k_3.

To avoid aliasing errors when computing the non-linear term, we employ the 2/3-rule [139]. The exact stress $\sigma(u)$ is approximated by the numerical stress

$$\sigma^N_{ij}(u) \coloneqq \widehat{\check{\tilde{u}}^N_i \check{\tilde{u}}^N_j}^N - \nu(\xi_j u_i + \xi_i u_j), \tag{C.28}$$

where

$$\tilde{u}(k) \coloneqq \begin{cases} u(k), & \text{if } \|k\|_\infty < N/3, \\ 0, & \text{otherwise,} \end{cases} \tag{C.29}$$

is obtained using a spectral cut-off filter at the frequency $N/3$.

The discrete equations (DNS equations) are then

$$\xi_j v_j = 0, \quad \partial_t v_i + \xi_j\left(\sigma^N_{ij}(v) + q\delta_{ij}\right) = f_i, \tag{C.30}$$

where q is the DNS pressure and v is the DNS velocity field.

C.2.2 *Large-eddy equations*

In spectral space, a convolutional filter reduces to wavenumber-wise scaling. Let $G(k)$ denote the spectral filter kernel. The filtered field is then $\bar{u}(k) \coloneqq G(k)u(k)$. The filtered DNS equations become

$$\xi_j \bar{v}_j = 0, \quad \partial_t \bar{v}_i + \xi_j \left(\overline{\sigma_{ij}^N(v)} + \bar{q}\delta_{ij} \right) = \bar{f}_i, \tag{C.31}$$

where $\bar{v}$ is the filtered DNS solution that we would like to solve for.

In LES, we solve the equations on a coarser grid than DNS. We introduce a coarse grid of size M^3 with $M < N$. The resolved numerical stress is computed on the coarse grid as $\sigma^M(\bar{v})$, where σ^M is defined analogously to σ^N using grid size M for both DFTs and the anti-aliasing procedure. To account for discretization effects in the commutator, we subtract the discretely resolved part from the filtered DNS equations [5], yielding

$$\xi_j \bar{v}_j = 0, \quad \partial_t \bar{v}_i + \xi_j \left(\sigma_{ij}^M(\bar{v}) + \tau^{N\to M}(v) + \bar{q}\delta_{ij} \right) = \bar{f}_i, \tag{C.32}$$

where the *discrete SFS* is defined as

$$\tau_{ij}^{N\to M}(v) \coloneqq \overline{\sigma_{ij}^N(v)} - \sigma_{ij}^M(\bar{v}). \tag{C.33}$$

Since the spectral derivative commutes with filtering, only the nonlinear part contributes to the SFS; the viscous commutator vanishes. However, $\tau^{N\to M}$ depends on the discretization sizes N and M through the DFTs and anti-aliasing procedures on both grids. The discrete SFS differs from the commonly used continuous SFS expression $\tau(u)(k)$ defined in physical space as $\check{\tau}(u) \coloneqq \overline{\check{u}\check{u}} - \check{\bar{u}}\check{\bar{u}}$. We numerically verified this discrete SFS expression using a DNS-aided LES approach [7].

The discrete SFS accounts for both the filter and the discretization differences between DNS and LES grids.

C.3 DATA GENERATION

We perform both DNS and LES on a single H100 GPU using double precision (64-bit) floating point arithmetic. The kinematic viscosity is $\nu \coloneqq 2 \times 10^{-4}$. This configuration, combined with our forcing scheme, produces forced homogeneous isotropic turbulence suitable for testing closure models. The DNS resolution is $N_{\text{DNS}} \coloneqq 810$ and the LES resolution is $N_{\text{LES}} \coloneqq 128$. Since only 2/3 of the wavenumbers are retained to avoid aliasing errors, the effective resolutions are 540 and 84 for DNS and LES, respectively. The DNS resolution was chosen to maximize grid size while fitting arrays in GPU memory. While powers of 2 are more efficient for FFTs, $N_{\text{DNS}} \coloneqq 1024$ exceeds available memory. Double precision is required for accurate estimation of equivariance errors.

The DNS velocity field is initialized using the following procedure:

1. Assign random numbers $u(k) \sim \mathcal{N}(0,1)$.
2. Project: $u(k) \leftarrow \pi(k)u(k)$.
3. Normalize shell energy by setting $u(k) \leftarrow \sqrt{P(\kappa)/E(u,\kappa)}u(k)$ for all $k \in K(\kappa)$, where

$$K(\kappa) \coloneqq \{k \in \mathbb{Z}^3 \mid \kappa \leq \|k\| < \kappa + 1\} \tag{C.34}$$

is the shell of wavenumbers at level $\kappa \in \mathbb{N}$,

$$E(u,\kappa) \coloneqq \sum_{k \in K(\kappa)} \|u(k)\|^2/2 \tag{C.35}$$

is the energy spectrum at κ, and $P(\kappa) \coloneqq \kappa^{-5/3}$ is a prescribed energy profile at level κ with a constant logarithmic slope of $-5/3$.

4. Rescale the total velocity field to have total kinetic energy $E = 0.2$.

The time integration is performed using Wray's third order low-storage Runge-Kutta scheme [206]. The time step is chosen as

$$\Delta t \coloneqq C \times \min\left(\frac{h}{\max_{i\in\{1,2,3\},x\in\Omega}|u_i(x)|}, \frac{h^2}{\nu}\right). \tag{C.36}$$

with $C \coloneqq 0.35$.

To prevent decay and to maintain the turbulence over time, we apply a body force. The force is chosen such that the energy contained in each of the first two shells $K(1)$ and $K(2)$ (see eq. (C.34)) is constant over time. After each unforced Runge-Kutta time step, we adjust the velocity coefficients in $K(1)$ and $K(2)$ as

$$u_i(k) \leftarrow u_i(k)\sqrt{\frac{E_f(\kappa)}{\frac{1}{2}\sum_{k' \in K(\kappa)}\|u(k')\|^2}}, \quad \forall\, k \in K(\kappa) \tag{C.37}$$

for $i \in \{1,2,3\}$ and $\kappa \in \{1,2\}$, where $E_f(\kappa)$ is the initial energy in shell $K(\kappa)$. Lundgren applied a similar linear forcing procedure to *all* wavenumbers to *exactly* maintain the total kinetic energy [113]. We force only the low-wavenumber shells to avoid interfering with scales requiring closure. This approach for injecting energy into decaying turbulence is widely used, including in the Johns Hopkins turbulence database [32, 100].

The modified Gaussian filter ensures that forcing and filtering commute, avoiding spurious commutator errors in the SFS.

For data generation, we employ a Gaussian filter. To ensure the closure model accounts only for the SFS rather than commutation errors between forcing and filtering, the body force should commute with the filter. The forcing procedure in eq. (C.37) does not generally commute with filtering. However, since the first two shells contain only large-scale features, filtering

has minimal effect on them. We therefore use a modified Gaussian filter that leaves the first two shells unchanged. The spectral filter kernel is given by

$$G(k) := \begin{cases} 1, & \text{if } \|k\| < 3, \\ \exp\left(-\frac{\|k\|^2\Delta^2}{24}\right), & \text{if } \|k\| \geq 3, \end{cases} \tag{C.38}$$

where $\Delta := 4h$ is the filter width and $h := L/N_{\text{LES}}$ is the LES grid spacing. This filter does commute with the forcing procedure in eq. (C.37).

We first run a warm-up DNS simulation for 5.0 time units to initialize the flow. After warm-up, we continue the DNS for 4950 additional time steps, corresponding to 7.94 time units. Every 50 time steps, we compute data pairs $(\bar{u}, \tau(u))$ and turbulence statistics. Here τ denotes the discrete SFS from eq. (C.33); for brevity, we omit the superscript $N_{\text{DNS}} \to N_{\text{LES}}$. This yields 100 snapshots of each quantity, including the initial snapshot after warm-up. The 100 time instances

$$T = \{t_0, \ldots, t_{99}\} \tag{C.39}$$

are not evenly spaced due to adaptive DNS time stepping. The average large-eddy turnover time is $t_{\text{int}} = 7.63$, so the simulation spans approximately one turnover time scale. The average Taylor-scale Reynolds number is $Re_{\text{tay}} = 268$. The first 50 snapshots are used for training, the remaining 50 for testing.

In fig. C.1, we show the evolution of the total kinetic energy $E(u) := \sum_k \|u(k)\|^2/2$ and the viscous dissipation rate $\epsilon := \nu \sum_k \|k\|^2\|u(k)\|^2$ (normalized by their maximum values). In fig. C.2, we show the time-averaged energy spectra $\langle E(u,\kappa)\rangle$ and $\langle E(\bar{u},\kappa)\rangle$, where $\langle\cdot\rangle$ denotes time-averaging and

$$E(u,\kappa) := \frac{1}{2}\sum_{k\in K(\kappa)} \|u(k)\|^2 \tag{C.40}$$

is the energy contained in the shell $K(\kappa)$. We also show the theoretical Kolmogorov spectrum for the inertial range given by

$$E_{\text{Kol}}(\kappa) := C\langle\epsilon\rangle^{2/3}\kappa^{-5/3}, \tag{C.41}$$

where $C := 1.6$ is the Kolmogorov constant [173]. We use the Kolmogorov normalizations $\tilde{E} := \epsilon^{-2/3}\eta^{5/3}E$ and $\tilde{\kappa} := \kappa\eta$ for plotting, where $\eta := (\nu^3/\epsilon)^{1/4}$ is the Kolmogorov length scale. This normalization is such that the Kolmogorov spectrum is 1 when $\tilde{\kappa} = 1$. The forced wavenumber shells are indicated with a banded area.

In fig. C.1, the dissipation rate peaks at the beginning of the warm-up period. This occurs because the simplified initial spectrum assumes an inertial range with constant logarithmic slope across all wavenumbers, including near the Kolmogorov scale where viscous effects dominate. The

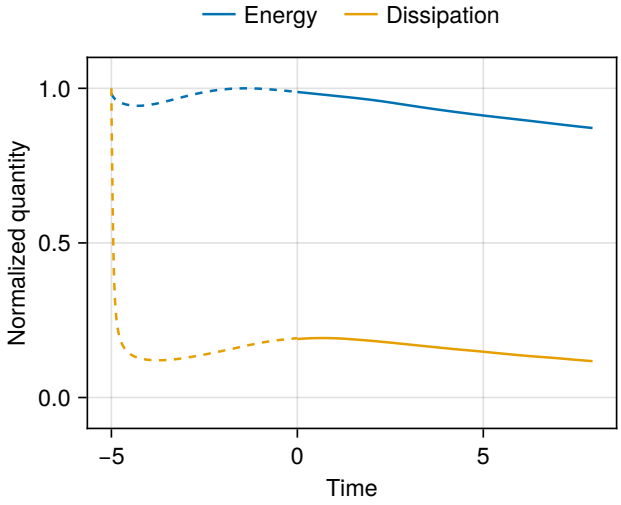


Figure C.1: DNS time series of the total kinetic energy E and viscous dissipation rate ϵ. Negative times (dashed lines) indicate the warm-up period. All time series are normalized by their respective maximum values.

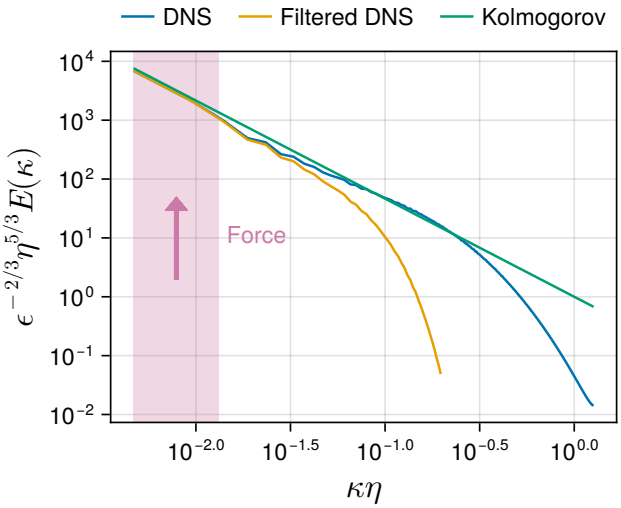


Figure C.2: Time-averaged energy spectrum after warm-up simulation. The banded area shows the range of wavenumbers with an active forcing.

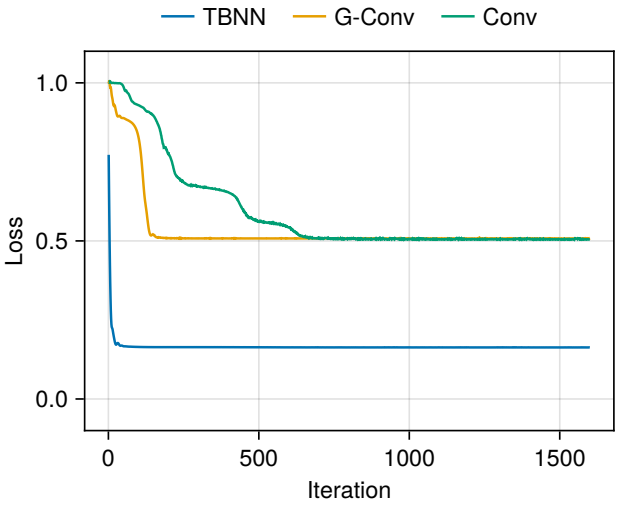


Figure C.3: Loss during training.

initial drop in dissipation reflects depletion of this excess high-wavenumber energy. In fig. C.2, the Kolmogorov scales are resolved by DNS but not by LES. The forced shells span scales approximately 100 times larger than the Kolmogorov scale, while the filtered DNS spectrum decays at scales roughly 10 times larger than the Kolmogorov scale.

C.4 TRAINING

For each closure model $m \in \{m^{\text{TBNN}}, m^{\text{G-conv}}, m^{\text{Conv}}\}$, we define the loss function

$$L_\theta \coloneqq \frac{1}{|B|} \sum_{u \in B} \frac{\|m_\theta(\bar{u}) - \tau(u)\|^2}{\|\tau(u)\|^2}, \tag{C.42}$$

where θ denotes the model parameters and B is a random mini-batch of samples. We minimize the loss over 5 epochs using the Adam optimizer. Each epoch consists of iterating through all mini-batches, each containing 10 snapshots.

The loss evolution is shown in fig. C.3. TBNN converges after only a few iterations, likely because it learns coefficients for a structured basis rather than the full SFS structure. The other two networks require more iterations to converge. G-conv converges faster than Conv, presumably because Conv must learn the underlying symmetries while G-conv has them built in by construction.

D

SUPPLEMENTARY MATERIAL FOR CHAPTER 6

D.1 FINITE VOLUME DISCRETIZATION

The integral form of the Navier-Stokes equations serves as the starting point for developing a spatial discretization:

$$\int_{\partial\mathcal{O}} u \cdot n \, \mathrm{d}\Gamma = 0, \tag{D.1}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathcal{O}} u \, \mathrm{d}\Omega = \int_{\partial\mathcal{O}} (-u \otimes u - pI + \nu\nabla u) \cdot n \, \mathrm{d}\Gamma + \int_{\mathcal{O}} f \, \mathrm{d}\Omega, \tag{D.2}$$

where $(u^1, \dots, u^d)$ is the velocity field, p is the pressure field, ν is the viscosity, f is the external forcing, I is the identity matrix, and $\mathcal{O} \subset \Omega$ is an arbitrary control volume with boundary $\partial\mathcal{O}$, normal n, surface element $\mathrm{d}\Gamma$, and volume size $|\mathcal{O}|$. We divide by the control volume sizes in the integral form so that the system (D.1)-(D.2) has the same dimensions as the continuous Navier-Stokes equations.

D.1.1 *Staggered grid configuration*

In this section, we describe a finite volume discretization of equations (D.1)-(D.2). Before doing so, we introduce our notation, which is such that the mathematical description of the discretization closely matches the software implementation.

The d spatial dimensions are indexed by $\alpha \in \{1, \dots, d\}$. The α-th unit vector is denoted $2h_\alpha = (2h_{\alpha\beta})_{\beta=1}^d$, where the (half) Kronecker symbol $h_{\alpha\beta}$ is $1/2$ if $\alpha = \beta$ and 0 otherwise. The Cartesian index $I = (I_1, \dots, I_d)$ is used to avoid repeating terms and equations d times, where I_α is a scalar index (typically one of i, j, and k in common notation). This notation is dimension-agnostic, since we can write u_I instead of u_{ij} in 2D or u_{ijk} in 3D. In our Julia implementation of the solver we use the same Cartesian notation (`u[I]` instead of `u[i, j]` or `u[i, j, k]`).

For the discretization, we use a staggered Cartesian grid as proposed by Harlow and Welch [71]. Staggered grids have excellent conservation properties [102, 141], and in particular their exact divergence-freeness is essential. Consider a rectangular domain $\Omega = \prod_{\alpha=1}^d [a_\alpha, b_\alpha]$, where $a_\alpha < b_\alpha$ are the domain boundaries and $\prod$ is a Cartesian product. Let $\Omega = \bigcup_{I\in\mathcal{I}} \Omega_I$ be a partitioning of Ω, where $\mathcal{I} = \prod_{\alpha=1}^d \{\frac{1}{2}, 2 - \frac{1}{2}, \dots, N_\alpha - \frac{1}{2}\}$ are volume center

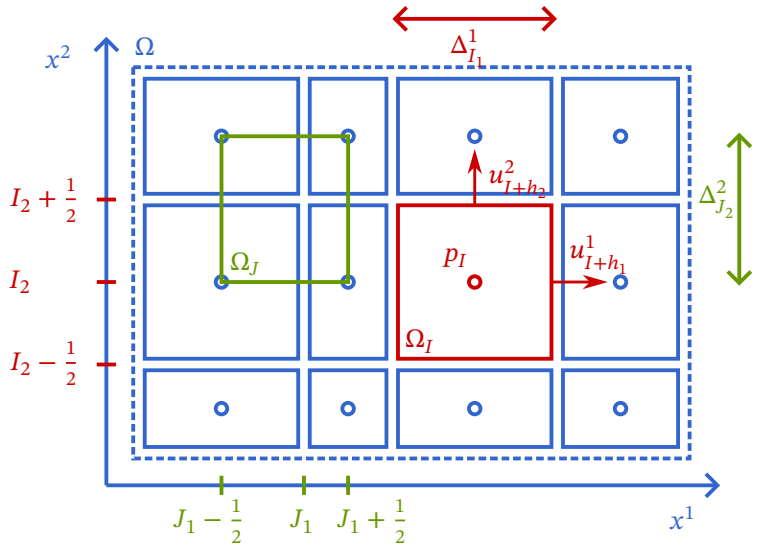


Figure D.1: Finite volume discretization on a staggered grid. Note that the grid can be non-uniform, as long as each volume in a given column has the same width and each volume in a given row has the same height. Here, I and J are two arbitrary Cartesian indices, with $I \in \mathcal{I}$ in a volume center and $J \in \mathcal{I} + h_1 + h_2$ in a volume corner for illustrative purposes.

indices, $N = (N_1, \dots, N_d) \in \mathbb{N}^d$ are the number of volumes in each dimension, $\Omega_I = \prod_{\alpha=1}^{d} \Delta^\alpha_{I_\alpha}$ is a finite volume, $\Gamma^\alpha_I = \Omega_{I-h_\alpha} \cap \Omega_{I+h_\alpha} = \prod_{\beta\neq\alpha} \Delta^\beta_{I_\beta}$ is a volume face, $\Delta^\alpha_i = \left[x^\alpha_{i-\frac{1}{2}}, x^\alpha_{i+\frac{1}{2}} \right]$ is a volume edge, $x^\alpha_0, \dots, x^\alpha_{N_\alpha}$ are volume boundary coordinates, and $x^\alpha_i = \frac{1}{2}\left(x^\alpha_{i-\frac{1}{2}} + x^\alpha_{i+\frac{1}{2}} \right)$ for $i \in \{1/2, \dots, N_\alpha - 1/2\}$ are volume center coordinates. We also define the operator δ_α which maps a discrete scalar field $\varphi = (\varphi_I)_I$ to

$$(\delta_\alpha \varphi)_I = \frac{\varphi_{I+h_\alpha} - \varphi_{I-h_\alpha}}{|\Delta^\alpha_{I_\alpha}|}. \tag{D.3}$$

It can be interpreted as a discrete equivalent of the continuous operator $\frac{\partial}{\partial x^\alpha}$. All the above definitions are extended to be valid in volume centers $I \in \mathcal{I}$, volume faces $I \in \mathcal{I} + h_\alpha$, or volume corners $I \in \mathcal{I} + \sum_{\alpha=1}^{d} h_\alpha$. The discretization is illustrated in figure D.1.

D.1.2 *Equations for unknowns*

We now define the unknown degrees of freedom. The average pressure in Ω_I, $I \in \mathcal{I}$ is approximated by $p_I(t)$. The average α-velocity on the face Γ_I^α, $I \in \mathcal{I} + h_\alpha$ is approximated by $u_I^\alpha(t)$. Note that the pressure p and the d velocity fields u^α are each defined at their own canonical positions x_I and x_{I+h_α} for $I \in \mathcal{I}$. This is illustrated for a volume I in figure D.1. We now derive the governing equations for these unknowns.

Using the pressure control volume $\mathcal{O} = \Omega_I$ with $I \in \mathcal{I}$ in the integral constraint (D.1) and approximating the face integrals with the mid-point quadrature rule $\int_{\Gamma_I} u \, \mathrm{d}\Gamma \approx |\Gamma_I| u_I$ results in the discrete divergence-free constraint

$$\sum_{\alpha=1}^{d} (\delta_\alpha u^\alpha)_I = 0. \tag{D.4}$$

Note that dividing by the volume size yields a discrete equation resembling the continuous one (since $|\Omega_I| = |\Gamma_I^\alpha||\Delta_{I_\alpha}^\alpha|$).

Similarly, choosing an α-velocity control volume $\mathcal{O} = \Omega_I$ with $I \in \mathcal{I} + h_\alpha$ in equation (D.2), approximating the volume and face integrals using the midpoint quadrature rule, and replacing the remaining spatial derivatives in the diffusive term with a finite difference approximation yields the discrete momentum equations

$$\frac{\mathrm{d}}{\mathrm{d}t} u_I^\alpha = -\sum_{\beta=1}^{d} \left[\delta_\beta \left(\eta_\beta^{\text{half}} u^\alpha \eta_\alpha^{\text{lin}} u^\beta \right) \right]_I + \nu \sum_{\beta=1}^{d} (\delta_\beta \delta_\beta u^\alpha)_I + f^\alpha(x_I) - (\delta_\alpha p)_I. \tag{D.5}$$

where we have assumed that f is constant in time for simplicity. The outer discrete derivative in $(\delta_\beta \delta_\beta u^\alpha)_I$ is required at position I, which means that the inner derivative is evaluated at $(\delta_\beta u^\alpha)_{I+h_\beta}$ and $(\delta_\beta u^\alpha)_{I-h_\beta}$, thus requiring $u^\alpha_{I-2h_\beta}$, u^α_I, and $u^\alpha_{I+2h_\beta}$, which are all at their canonical positions. The two velocity components $\eta_\beta^{\text{half}} u^\alpha$ and $\eta_\alpha^{\text{lin}} u^\beta$ in the convective term are required at positions $I - h_\beta$ and $I + h_\beta$, which are outside the canonical positions. Their values at the required positions are obtained using averaging with weights $1/2$ for the α-component and linear interpolation for the β-component. This preserves the skew-symmetry of the convection operator, ensuring energy conservation in the convective term [187].

D.2 DISCRETE OPERATOR STENCILS

Each operator is paired with its explicit pullback kernel, enabling reverse-mode AD for solver-in-the-loop training.

This appendix lists the explicit stencil expressions for the discrete operators introduced in section 6.2, along with their adjoint (pullback) kernels used for reverse-mode automatic differentiation (section 6.3.4). The adjoint kernels are tested against numerical finite-difference gradients to verify their correctness.

All operators are expressed in terms of the discrete derivative δ_α defined in (D.3). Below, I denotes a multi-index at a grid point, $h_\alpha = e_\alpha/2$ is a half-index shift, $|\Delta_i^\alpha|$ is the width of volume i in the α-direction, and $\bar{\varphi}$, $\bar{u}$, $\bar{p}$ denote adjoint (cotangent) variables. All pullbacks are derived from the general definition (6.10). Except for convection, all operators are linear, and their pullbacks reduce to the transpose of the operator.

DIVERGENCE. The divergence maps the velocity field u to a scalar field φ defined at the pressure points. The kernel and its pullback are

$$\varphi_I = \sum_{\alpha=1}^{d} \frac{u^\alpha_{I+h_\alpha} - u^\alpha_{I-h_\alpha}}{|\Delta^\alpha_{I_\alpha}|}, \qquad \bar{u}^\alpha_I = \frac{\bar{\varphi}_{I-h_\alpha}}{|\Delta^\alpha_{I_\alpha-1/2}|} - \frac{\bar{\varphi}_{I+h_\alpha}}{|\Delta^\alpha_{I_\alpha+1/2}|}. \tag{D.6}$$

PRESSURE GRADIENT. The pressure gradient maps a scalar field p to a vector field φ defined at the velocity points. The kernel and its pullback are

$$\varphi^\alpha_I = \frac{p_{I+h_\alpha} - p_{I-h_\alpha}}{|\Delta^\alpha_{I_\alpha}|}, \qquad \bar{p}_I = \sum_{\alpha=1}^{d} \left(\frac{\bar{\varphi}^\alpha_{I-h_\alpha}}{|\Delta^\alpha_{I_\alpha-1/2}|} - \frac{\bar{\varphi}^\alpha_{I+h_\alpha}}{|\Delta^\alpha_{I_\alpha+1/2}|} \right). \tag{D.7}$$

DIFFUSION. The diffusion maps a vector field u to a vector field φ defined at the velocity points. The kernel and its pullback are

$$\varphi^\alpha_I = \nu \sum_{\beta=1}^{d} \frac{1}{|\Delta^\beta_{I_\beta}|} \left(\frac{u^\alpha_{I+2h_\beta} - u^\alpha_I}{|\Delta^\beta_{I_\beta+1/2}|} - \frac{u^\alpha_I - u^\alpha_{I-2h_\beta}}{|\Delta^\beta_{I_\beta-1/2}|} \right) \tag{D.8}$$

and

$$\bar{u}^\alpha_I = \nu \sum_{\beta=1}^{d} \left(\frac{\bar{\varphi}^\alpha_{I-2h_\beta}}{|\Delta^\beta_{I_\beta-1}|\,|\Delta^\beta_{I_\beta-1/2}|} - \frac{1}{|\Delta^\beta_{I_\beta}|} \left(\frac{\bar{\varphi}^\alpha_I}{|\Delta^\beta_{I_\beta+1/2}|} + \frac{\bar{\varphi}^\alpha_I}{|\Delta^\beta_{I_\beta-1/2}|} \right) + \frac{\bar{\varphi}^\alpha_{I+2h_\beta}}{|\Delta^\beta_{I_\beta+1}|\,|\Delta^\beta_{I_\beta+1/2}|} \right). \tag{D.9}$$

CONVECTION. The convective term maps a vector field u to a vector field φ defined at the velocity points. The product $u^\alpha u^\beta$ at off-diagonal positions requires interpolation of velocity components to the appropriate face locations. The interpolation weights

$$A_i^{\alpha\beta+} = \frac{|\Delta^\beta_{i-\delta_{\alpha\beta}+1}|}{|\Delta^\beta_{i-\delta_{\alpha\beta}}| + |\Delta^\beta_{i-\delta_{\alpha\beta}+1}|}, \qquad A_i^{\alpha\beta-} = \frac{|\Delta^\beta_{i+\delta_{\alpha\beta}-1}|}{|\Delta^\beta_{i+\delta_{\alpha\beta}-1}| + |\Delta^\beta_{i+\delta_{\alpha\beta}}|} \tag{D.10}$$

map to the right (+) and left (−) in the β-direction, respectively, and are chosen to preserve skew-symmetry [191]. The convective kernel is

$$\begin{aligned}\varphi_I^\alpha = -\sum_{\beta=1}^{d} \frac{1}{|\Delta_{I_\beta}^\beta|}\Big(&\Big(A_{I_\beta}^{\alpha\beta+}u_I^\alpha + A_{I_\beta+1}^{\alpha\beta-}u_{I+2h_\beta}^\alpha\Big)\Big(A_{I_\alpha}^{\beta\alpha+}u_I^\beta + A_{I_\alpha+1}^{\beta\alpha-}u_{I+2h_\alpha}^\beta\Big)\\ &-\Big(A_{I_\beta-1}^{\alpha\beta+}u_{I-2h_\beta}^\alpha + A_{I_\beta}^{\alpha\beta-}u_I^\alpha\Big)\\ &\Big(A_{I_\alpha-\delta_{\alpha\beta}}^{\beta\alpha+}u_{I-2h_\beta}^\beta + A_{I_\alpha-\delta_{\alpha\beta}+1}^{\beta\alpha-}u_{I-2h_\beta+2h_\alpha}^\beta\Big)\Big).\end{aligned} \tag{D.11}$$

Since the convective kernel is nonlinear, the pullback contains components from the primal input u. The input u appears eight times in (D.11), and their contributions to the pullback are

$$\begin{aligned}\bar{u}_I^\alpha = &-\bar{\varphi}_I^\alpha \sum_{\beta=1}^{d} \frac{1}{|\Delta_{I_\beta}^\beta|}A_{I_\beta}^{\alpha\beta+}\Big(A_{I_\alpha}^{\beta\alpha+}u_I^\beta + A_{I_\alpha+1}^{\beta\alpha-}u_{I+2h_\alpha}^\beta\Big)\\
&-\bar{\varphi}_{I-2h_\beta}^\alpha \sum_{\beta=1}^{d} \frac{1}{|\Delta_{I_\beta-1}^\beta|}A_{I_\beta}^{\alpha\beta-}\Big(A_{I_\alpha-\delta_{\alpha\beta}}^{\beta\alpha+}u_{I-2h_\beta}^\beta + A_{I_\alpha-\delta_{\alpha\beta}+1}^{\beta\alpha-}u_{I-2h_\beta+2h_\alpha}^\beta\Big)\\
&-\sum_{\beta=1}^{d} \bar{\varphi}_I^\beta \frac{1}{|\Delta_{I_\alpha}^\alpha|}\Big(A_{I_\alpha}^{\beta\alpha+}u_I^\beta + A_{I_\alpha+1}^{\beta\alpha-}u_{I+2h_\alpha}^\beta\Big)A_{I_\beta}^{\alpha\beta+}\\
&-\sum_{\beta=1}^{d} \bar{\varphi}_{I-2h_\beta}^\beta \frac{1}{|\Delta_{I_\alpha-\delta_{\alpha\beta}}^\alpha|}\Big(A_{I_\alpha-\delta_{\alpha\beta}}^{\beta\alpha+}u_{I-2h_\beta}^\beta + A_{I_\alpha-\delta_{\alpha\beta}+1}^{\beta\alpha-}u_{I-2h_\beta+2h_\alpha}^\beta\Big)A_{I_\beta}^{\alpha\beta-}\\
&+\bar{\varphi}_{I+2h_\beta}^\alpha \sum_{\beta=1}^{d} \frac{1}{|\Delta_{I_\beta+1}^\beta|}A_{I_\beta+1}^{\alpha\beta+}\Big(A_{I_\alpha+\delta_{\alpha\beta}}^{\beta\alpha+}u_{I+2h_\beta}^\beta + A_{I_\alpha+\delta_{\alpha\beta}+1}^{\beta\alpha-}u_{I+2h_\beta+2h_\alpha}^\beta\Big)\\
&+\bar{\varphi}_I^\alpha \sum_{\beta=1}^{d} \frac{1}{|\Delta_{I_\beta}^\beta|}A_{I_\beta+1}^{\alpha\beta-}\Big(A_{I_\alpha}^{\beta\alpha+}u_I^\beta + A_{I_\alpha+1}^{\beta\alpha-}u_{I+2h_\alpha}^\beta\Big)\\
&+\sum_{\beta=1}^{d} \bar{\varphi}_{I+2h_\alpha}^\beta \frac{1}{|\Delta_{I_\alpha+1}^\alpha|}\Big(A_{I_\alpha+1}^{\beta\alpha+}u_{I+2h_\alpha}^\beta + A_{I_\alpha+2}^{\beta\alpha-}u_{I+4h_\alpha}^\beta\Big)A_{I_\beta+\delta_{\alpha\beta}}^{\alpha\beta+}\\
&+\sum_{\beta=1}^{d} \bar{\varphi}_{I-2h_\beta+2h_\alpha}^\beta \frac{1}{|\Delta_{I_\alpha-\delta_{\alpha\beta}+1}^\alpha|}\Big(A_{I_\alpha-\delta_{\alpha\beta}+1}^{\beta\alpha+}u_{I-2h_\beta+2h_\alpha}^\beta + A_{I_\alpha-\delta_{\alpha\beta}+2}^{\beta\alpha-}u_{I-2h_\beta+4h_\alpha}^\beta\Big)A_{I_\beta+\delta_{\alpha\beta}}^{\alpha\beta-}.\end{aligned} \tag{D.12}$$

D.3 EDDY-VISCOSITY MODELS

Six classical eddy-viscosity models are implemented, all unified under the invariant-based framework of Trias et al. [181].

This appendix lists the eddy-viscosity closure models implemented in the software, following the unified invariant-based framework of Trias et al. [181].

D.3.1 *General framework*

All eddy-viscosity models express the sub-grid stress as

$$m^\alpha(u) = -\sum_{\beta=1}^{d} \partial_\beta(2\nu_t S_{\alpha\beta}), \tag{D.13}$$

where $S_{\alpha\beta} = (\partial_\beta u^\alpha + \partial_\alpha u^\beta)/2$ is the resolved strain-rate tensor and $\nu_t \geq 0$ is the eddy viscosity.

On the staggered grid, $S_{\alpha\beta}$ is naturally defined at the staggered tensor points: the diagonal components $\bar{S}_{\alpha\alpha}$ live at the pressure points, while the off-diagonal components $\bar{S}_{\alpha\beta}$ ($\alpha \neq \beta$) live at the edge midpoints. The eddy viscosity ν_t is computed at the pressure points from the velocity gradient $A_{\alpha\beta} = \delta_\beta u^\alpha$, which is interpolated from the staggered tensor points to the pressure points. To evaluate (D.13), ν_t must then be interpolated back to the staggered tensor points where $S_{\alpha\beta}$ is defined. For the diagonal components, ν_t is already at the correct location. For the off-diagonal components, ν_t is interpolated using a four-point linear average of the surrounding pressure points.

D.3.2 *Invariants of the velocity gradient tensor*

All models below are expressed in terms of the velocity gradient tensor $A = \nabla u$ at the pressure points and its symmetric and anti-symmetric parts,

$$S = \frac{A + A^\mathsf{T}}{2}, \qquad W = \frac{A - A^\mathsf{T}}{2}. \tag{D.14}$$

Following Trias et al. [181], we define the invariants

$$\begin{gathered} Q_A = -\frac{\mathrm{tr}(A^2)}{2}, \quad Q_S = -\frac{\mathrm{tr}(S^2)}{2}, \quad Q_W = -\frac{\mathrm{tr}(W^2)}{2}, \\ R_S = \frac{\mathrm{tr}(S^3)}{3}, \quad R_A = \frac{\mathrm{tr}(A^3)}{3}, \end{gathered} \tag{D.15}$$

and the mixed invariant

$$V^2 = 4\left(\mathrm{tr}(S^2W^2) - 2Q_SQ_W\right). \tag{D.16}$$

The five independent invariants of A are Q_S, Q_W, R_S, R_A, and V^2. Note that $Q_A = Q_S + Q_W$ and that $-2Q_S = S_{\alpha\beta}S_{\alpha\beta} \geq 0$, so $Q_S \leq 0$.

D.3.3 *Implemented models*

All models take the form $\nu_t = (C\Delta)^2\,\mathcal{D}$, where C is a dimensionless model constant and Δ is the filter width. The quantity $\mathcal{D}$ has dimensions of inverse time and differs between models. The proposed values of C are taken from the original references or from Trias et al. [181].

SMAGORINSKY. The Smagorinsky model [101, 168] defines the eddy viscosity as

$$\nu_t = (C_\mathrm{S}\Delta)^2\sqrt{2S_{\alpha\beta}S_{\alpha\beta}} = (C_s\Delta)^2\sqrt{-4Q_S}, \tag{D.17}$$

with proposed constant $C_\mathrm{S} = 0.17$. This is the simplest and most widely used eddy-viscosity model. Its main limitation is that it does not vanish at solid walls or in laminar flow regions, requiring ad-hoc damping functions or dynamic procedures.

VREMAN. The Vreman model [195] defines

$$\nu_t = C_\mathrm{Vr}^2\sqrt{\frac{B_\beta}{\mathrm{tr}(AA^T)/2}}, \tag{D.18}$$

where $\beta_{ij} = \Delta_m^2 A_{im}A_{jm}$ accounts for non-isotropic grid and $B_\beta = \beta_{11}\beta_{22} - \beta_{12}^2 + \beta_{11}\beta_{33} - \beta_{13}^2 + \beta_{22}\beta_{33} - \beta_{23}^2$. The proposed constant is $C_\mathrm{Vr} = \sqrt{2.5C_\mathrm{S}^2}$. The denominator $\mathrm{tr}(AA^\mathsf{T})/2 \geq 0$ vanishes only when $A = 0$, so the model is well-defined wherever the velocity gradient is nonzero. The viscosity is set to zero when $A = 0$.

QR (VERSTAPPEN). Verstappen's minimum-dissipation QR model [181, 192] defines

$$\nu_t = -(C_\mathrm{Ve}\Delta)^2\frac{|R_S|}{Q_S}, \tag{D.19}$$

with proposed constant $C_\mathrm{Ve} = \sqrt{3/2}/\pi \approx 0.390$ [192] or $C = 0.527$ [181]. This model vanishes in two-component flows (where $R_S = 0$), which is the correct physical behavior near solid walls and in regions of axisymmetric strain.

WALE. The wall-adapting local eddy-viscosity (WALE) model [130, 181] defines

$$\nu_t = (C_\mathrm{W}\Delta)^2\frac{\left(\mathcal{S}_{ij}^d\mathcal{S}_{ij}^d\right)^{3/2}}{\left(S_{ij}S_{ij}\right)^{5/2} + \left(\mathcal{S}_{ij}^d\mathcal{S}_{ij}^d\right)^{5/4}}, \tag{D.20}$$

where $\mathcal{S}^d$ is the traceless symmetric part of A^2: $\mathcal{S}^d := (AA + A^T A^T)/2 - (Q_A/3)I$ with proposed constant $C_{\mathrm{W}} = \sqrt{2.5 C_{\mathrm{S}}}$, where C_{S} is the Smagorinsky constant. The WALE model is designed to produce the correct y^+ scaling of the eddy viscosity near solid walls, without requiring explicit damping functions.

σ-MODEL (NICOUD). The σ-model [131, 181] is defined in terms of the singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$ of the velocity gradient tensor A:

$$\nu_t = (C_\sigma \Delta)^2 \frac{\sigma_3(\sigma_1 - \sigma_2)(\sigma_2 - \sigma_3)}{\sigma_1^2}, \tag{D.21}$$

with proposed constant $C_\sigma = 1.35$. This model vanishes for two-component and axisymmetric flows, and automatically produces the correct near-wall scaling without damping functions.

S3PQR (TRIAS). The S3PQR model [181] is a family of models parameterized by an exponent p:

$$\nu_t = (C_{\mathrm{Tr}} \Delta)^2 P_{AA}^{\,p}\, Q_{AA}^{-(p+1)}\, R_{AA}^{(p+5/2)/3}, \tag{D.22}$$

where

$$P_{AA} = 2(Q_W - Q_S) = \mathrm{tr}(AA^{\mathsf{T}}), \quad Q_{AA} = V^2 + Q_A^2, \quad R_{AA} = R_A^2. \tag{D.23}$$

The choice of p recovers or approximates several known models. Trias et al. propose $p \in \{-5/2, -1, 0\}$ as representative members of the family.

BIBLIOGRAPHY

[1] Syver Døving Agdestein and Benjamin Sanderse. "Discretize First, Filter next – a New Closure Model Approach." In: *8th European Congress on Computational Methods in Applied Sciences and Engineering*. CIMNE, 2022. DOI: 10.23967/eccomas.2022.094.

[2] Syver Døving Agdestein and Benjamin Sanderse. "Discretize First, Filter next: Learning Divergence-Consistent Closure Models for Large-Eddy Simulation." In: *Journal of Computational Physics* 522 (Feb. 2025), p. 113577. DOI: 10.1016/j.jcp.2024.113577.

[3] Syver Døving Agdestein and Benjamin Sanderse. *A Differentiable Software Suite for Accelerated Simulation of Turbulent Flows*. Apr. 2026. DOI: 10.48550/arXiv.2604.18536. arXiv: 2604.18536 [math].

[4] Syver Døving Agdestein and Benjamin Sanderse. *Comparison of Data-Driven Symmetry-Preserving Closure Models for Large-Eddy Simulation*. Mar. 2026. DOI: 10.48550/arXiv.2603.05325. arXiv: 2603.05325 [math].

[5] Syver Døving Agdestein, Roel Verstappen, and Benjamin Sanderse. "Exact Expressions for the Unresolved Stress in a Finite-Volume Based Large-Eddy Simulation." In: *Journal of Computational Physics* 556 (July 2026), p. 114810. DOI: 10.1016/j.jcp.2026.114810.

[6] H. Jane Bae and Adrian Lozano-Duran. "Towards Exact Subgrid-Scale Models for Explicitly Filtered Large-Eddy Simulation of Wall-Bounded Flows." In: *Annual Research Briefs* 2017 (Dec. 2017).

[7] H. Jane Bae and Adrian Lozano-Duran. *Numerical and Modeling Error Assessment of Large-Eddy Simulation Using Direct-Numerical-Simulation-Aided Large-Eddy Simulation*. Aug. 2022. DOI: 10.48550/arXiv.2208.02354. arXiv: 2208.02354 [physics].

[8] A. Báez Vidal, O. Lehmkuhl, F.X. Trias, and C.D. Pérez-Segarra. "On the Properties of Discrete Spatial Filters for CFD." In: *Journal of Computational Physics* 326 (Dec. 2016), pp. 474–498. DOI: 10.1016/j.jcp.2016.09.002.

[9] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. "Learning Data Driven Discretizations for Partial Differential Equations." In: *Proceedings of the National Academy of Sciences* 116.31 (July 2019), pp. 15344–15349. DOI: 10.1073/pnas.1814058116. arXiv: 1808.04930 [cond-mat].

[10] J. Bardina, J. Ferziger, and W. Reynolds. "Improved Subgrid-Scale Models for Large-Eddy Simulation." In: *13th Fluid and Plasma-Dynamics Conference*. Snowmass,CO,U.S.A.: American Institute of Aeronautics and Astronautics, July 1980. DOI: 10.2514/6.1980-1357.

[11] A.R. Barron. "Universal Approximation Bounds for Superpositions of a Sigmoidal Function." In: *IEEE Transactions on Information Theory* 39.3 (May 1993), pp. 930–945. DOI: 10.1109/18.256500.

[12] Andrea Beck, David Flad, and Claus-Dieter Munz. "Deep Neural Networks for Data-Driven LES Closure Models." In: *Journal of Computational Physics* 398 (Dec. 2019), p. 108910. DOI: 10.1016/j.jcp.2019.108910.

[13] Andrea Beck and Marius Kurz. "A Perspective on Machine Learning Methods in Turbulence Modeling." In: *GAMM-Mitteilungen* 44.1 (Mar. 2021), e202100002. DOI: 10.1002/gamm.202100002.

[14] Andrea Beck and Marius Kurz. "Toward Discretization-Consistent Closure Schemes for Large Eddy Simulation Using Reinforcement Learning." In: *Physics of Fluids* 35.12 (Dec. 2023), p. 125122. DOI: 10.1063/5.0176223. arXiv: 2309.06260 [physics].

[15] Mark Benjamin and Gianluca Iaccarino. *Neural Network–Based Closure Models for Large–Eddy Simulations with Explicit Filtering*. SSRN Scholarly Paper. Rochester, NY, May 2023. DOI: 10.2139/ssrn.4462714. Social Science Research Network: 4462714.

[16] S. Berrone and D. Oberto. "An Invariances-Preserving Vector Basis Neural Network for the Closure of Reynolds-averaged Navier–Stokes Equations by the Divergence of the Reynolds Stress Tensor." In: *Physics of Fluids* 34.9 (Sept. 2022), p. 095136. DOI: 10.1063/5.0104605.

[17] Luigi C. Berselli. *Mathematics of Large Eddy Simulation of Turbulent Flows*. Scientific Computation. Berlin: Springer-Verlag, 2006. ISBN: 978-3-540-26316-6.

[18] Tim Besard, Valentin Churavy, Alan Edelman, and Bjorn De Sutter. "Rapid Software Prototyping for Heterogeneous and Distributed Platforms." In: *Advances in Engineering Software* 132 (June 2019), pp. 29–46. DOI: 10.1016/j.advengsoft.2019.02.002.

[19] Tim Besard, Christophe Foket, and Bjorn De Sutter. "Effective Extensible Programming: Unleashing Julia on GPUs." In: *IEEE Transactions on Parallel and Distributed Systems* 30.4 (Apr. 2019), pp. 827–841. DOI: 10.1109/TPDS.2018.2872064.

[20] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. "Julia: A Fresh Approach to Numerical Computing." In: *SIAM Review* (Feb. 2017). DOI: 10.1137/141000671.

[21] George W. Bluman and Sukeyuki Kumei. *Symmetries and Differential Equations*. Vol. 81. Applied Mathematical Sciences. New York, NY: Springer, 1989. ISBN: 978-1-4757-4309-8. DOI: 10.1007/978-1-4757-4307-4.

[22] J. P. Boris, F. F. Grinstein, E. S. Oran, and R. L. Kolbe. "New Insights into Large Eddy Simulation." In: *Fluid Dynamics Research* 10.4 (Dec. 1992), pp. 199–228. DOI: 10.1016/0169-5983(92)90023-P.

[23] Jay P. Boris. "On Large Eddy Simulation Using Subgrid Turbulence Models Comment 1." In: *Whither Turbulence? Turbulence at the Crossroads*. Ed. by J. L. Lumley. Berlin, Heidelberg: Springer, 1990, pp. 344–353. ISBN: 978-3-540-47032-8. DOI: 10.1007/3-540-52535-1_53.

[24] Rikhi Bose and Arunabha M. Roy. "Invariance Embedded Physics-Infused Deep Neural Network-Based Sub-Grid Scale Models for Turbulent Flows." In: *Engineering Applications of Artificial Intelligence* 128 (Feb. 2024), p. 107483. DOI: 10.1016/j.engappai.2023.107483.

[25] Leon Bottou. "Stochastic Gradient Learning in Neural Networks." In: *Proceedings of Neuro-Nımes*. Vol. 91. Nimes, 1991, p. 12.

[26] Bernardo P. Brener, Matheus A. Cruz, Matheus S. S. Macedo, and Roney L. Thompson. "A Highly Accurate Strategy for Data-Driven Turbulence Modeling." In: *Computational and Applied Mathematics* 43.1 (Feb. 2024), p. 59. DOI: 10.1007/s40314-023-02547-9.

[27] Guillaume A. Brès, Peter Jordan, Vincent Jaunet, Maxime Le Rallic, André V. G. Cavalieri, Aaron Towne, Sanjiva K. Lele, Tim Colonius, and Oliver T. Schmidt. "Importance of the Nozzle-Exit Boundary-Layer State in Subsonic Turbulent Jets." In: *Journal of Fluid Mechanics* 851 (Sept. 2018), pp. 83–124. DOI: 10.1017/jfm.2018.476.

[28] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering*. 2nd ed. 2022. ISBN: 978-1-009-11563-6.

[29] Dhawal Buaria and Katepalli R. Sreenivasan. "Forecasting Small-Scale Dynamics of Fluid Turbulence Using Deep Neural Networks." In: *Proceedings of the National Academy of Sciences* 120.30 (July 2023), e2305765120. DOI: 10.1073/pnas.2305765120.

[30] E. Buckingham. "On Physically Similar Systems; Illustrations of the Use of Dimensional Equations." In: *Physical Review* 4.4 (Oct. 1914), pp. 345–376. DOI: 10.1103/PhysRev.4.345.

[31] J. M. Burgers. "Hydrodynamics. — Application of a Model System to Illustrate Some Points of the Statistical Theory of Free Turbulence." In: *Selected Papers of J. M. Burgers*. Ed. by F. T. M. Nieuwstadt and J. A. Steketee. Dordrecht: Springer Netherlands, 1995, pp. 390–400. ISBN: 978-94-011-0195-0. DOI: 10.1007/978-94-011-0195-0_12.

[32] Nianzheng Cao, Shiyi Chen, and Gary D. Doolen. "Statistics and Structures of Pressure in Isotropic Turbulence." In: *Physics of Fluids* 11.8 (Aug. 1999), pp. 2235–2250. DOI: 10.1063/1.870085.

[33] Gary J. Chandler and Rich R. Kerswell. "Invariant Recurrent Solutions Embedded in a Turbulent Two-Dimensional Kolmogorov Flow." In: *Journal of Fluid Mechanics* 722 (May 2013), pp. 554–595. DOI: 10.1017/jfm.2013.122.

[34] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. *Neural Ordinary Differential Equations*. Dec. 2019. DOI: 10.48550/arXiv.1806.07366. arXiv: 1806.07366 [cs].

[35] Tianping Chen and Hong Chen. "Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems." In: *IEEE Transactions on Neural Networks* 6.4 (July 1995), pp. 911–917. DOI: 10.1109/72.392253.

[36] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. *cuDNN: Efficient Primitives for Deep Learning*. Dec. 2014. DOI: 10.48550/arXiv.1410.0759. arXiv: 1410.0759 [cs].

[37] Alexandre Joel Chorin. "Numerical Solution of the Navier-Stokes Equations." In: *Mathematics of Computation* 22.104 (1968), pp. 745–762. DOI: 10.1090/S0025-5718-1968-0242392-2.

[38] Fotini Katopodes Chow and Parviz Moin. "A Further Study of Numerical Errors in Large-Eddy Simulations." In: *Journal of Computational Physics* 184.2 (Jan. 2003), pp. 366–380. DOI: 10.1016/S0021-9991(02)00020-7.

[39] Valentin Churavy et al. *JuliaGPU/KernelAbstractions.Jl: V0.9.10*. Zenodo. Oct. 2023. DOI: 10.5281/ZENODO.8436610.

[40] Paola Cinnella. *Data-Driven Turbulence Modeling*. Apr. 2024. DOI: 10.48550/arXiv.2404.09074. arXiv: 2404.09074 [physics].

[41] Robert A. Clark, Joel H. Ferziger, and W. C. Reynolds. "Evaluation of Subgrid-Scale Models Using an Accurately Simulated Turbulent Flow." In: *Journal of Fluid Mechanics* 91.1 (Mar. 1979), pp. 1–16. DOI: 10.1017/S002211207900001X.

[42] Taco Cohen and Max Welling. "Group Equivariant Convolutional Networks." In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, June 2016, pp. 2990–2999.

[43] Taco S. Cohen and Max Welling. *Steerable CNNs*. Dec. 2016. DOI: 10.48550/arXiv.1612.08498. arXiv: 1612.08498 [cs].

[44] Alex Connolly, Yu Cheng, Robin Walters, Rui Wang, Rose Yu, and Pierre Gentine. *Deep Learning Turbulence Closures Generalize Best With Physics-based Methods*. Feb. 2025. DOI: 10.22541/essoar.173869578.80400701/v1.

[45] G. Cybenko. "Approximation by Superpositions of a Sigmoidal Function." In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274.

[46] Thibault Dairay, Eric Lamballais, Sylvain Laizet, and John Christos Vassilicos. "Numerical Dissipation vs. Subgrid-Scale Modelling for Large Eddy Simulation." In: *Journal of Computational Physics* 337 (May 2017), pp. 252–274. DOI: 10.1016/j.jcp.2017.02.035.

[47] Simon Danisch and Julius Krumbiegel. "Makie.Jl: Flexible High-Performance Data Visualization for Julia." In: *Journal of Open Source Software* 6.65 (Sept. 2021), p. 3349. DOI: 10.21105/joss.03349.

[48] J. W. Deardorff. "On the Magnitude of the Subgrid Scale Eddy Coefficient." In: *Journal of Computational Physics* 7.1 (Feb. 1971), pp. 120–133. DOI: 10.1016/0021-9991(71)90053-2.

[49] Filippo Maria Denaro. "What Does Finite Volume-based Implicit Filtering Really Resolve in Large-Eddy Simulations?" In: *Journal of Computational Physics* 230.10 (May 2011), pp. 3849–3883. DOI: 10.1016/j.jcp.2011.02.011.

[50] Filippo Maria Denaro. "An Inconsistence-Free Integral-Based Dynamic One- and Two-Parameter Mixed Model." In: *Journal of Turbulence* 19.9 (Sept. 2018), pp. 754–797. DOI: 10.1080/14685248.2018.1513136.

[51] Jack Dongarra, John Gunnels, Harun Bayraktar, Azzam Haidar, and Dan Ernst. *Hardware Trends Impacting Floating-Point Computations In Scientific Applications*. Dec. 2024. DOI: 10.48550/arXiv.2411.12090. arXiv: 2411.12090 [math].

[52] Karthik Duraisamy. "Perspectives on Machine Learning-Augmented Reynolds-averaged and Large Eddy Simulation Models of Turbulence." In: *Physical Review Fluids* 6.5 (May 2021), p. 050504. DOI: 10.1103/PhysRevFluids.6.050504.

[53] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. "Turbulence Modeling in the Age of Data." In: *Annual Review of Fluid Mechanics* 51.1 (Jan. 2019), pp. 357–377. DOI: 10.1146/annurev-fluid-010518-040547.

[54] Joel H. Ferziger, Milovan Perić, and Robert L. Street. *Computational Methods for Fluid Dynamics*. 4th ed. 2020. Springer eBooks Engineering. Cham: Springer, 2020. ISBN: 978-3-319-99691-2. DOI: 10.1007/978-3-319-99693-6.

[55] David Flad, Andrea D. Beck, Gregor Gassner, and Claus-dieter Munz. "A Discontinuous Galerkin Spectral Element Method for the Direct Numerical Simulation of Aeroacoustics." In: *20th AIAA/CEAS Aeroacoustics Conference*. Atlanta, GA: American Institute of Aeronautics and Astronautics, June 2014. ISBN: 978-1-62410-285-1. DOI: 10.2514/6.2014-2740.

[56] Uriel Frisch. *Turbulence: The Legacy of A. N. Kolmogorov*. Cambridge: Cambridge University Press, 1995.

[57] Christer Fureby and Fernando F. Grinstein. "Large Eddy Simulation of High-Reynolds-Number Free and Wall-Bounded Flows." In: *Journal of Computational Physics* 181.1 (Sept. 2002), pp. 68–97. DOI: 10.1006/jcph.2002.7119.

[58] Timothy Gallagher, Ayaboe Edoh, and Venke Sankaran. "Residual and Solution Filtering for Explicitly-Filtered Large-Eddy Simulations." In: *AIAA Scitech 2019 Forum*. San Diego, California: American Institute of Aeronautics and Astronautics, Jan. 2019. ISBN: 978-1-62410-578-4. DOI: 10.2514/6.2019-1645.

[59] Thomas B. Gatski, M. Yousuff Hussaini, and John L. Lumley, eds. *Simulation and Modeling of Turbulent Flows*. Oxford University Press, Sept. 1996. ISBN: 978-0-19-510643-5. DOI: 10.1093/oso/9780195106435.001.0001.

[60] M. Germano. "Differential Filters for the Large Eddy Numerical Simulation of Turbulent Flows." In: *The Physics of Fluids* 29.6 (June 1986), pp. 1755–1757. DOI: 10.1063/1.865649.

[61] Massimo Germano, Ugo Piomelli, Parviz Moin, and William H. Cabot. "A Dynamic Subgrid-Scale Eddy Viscosity Model." In: *Physics of Fluids A: Fluid Dynamics* 3.7 (July 1991), pp. 1760–1765. DOI: 10.1063/1.857955.

[62] Bernard J Geurts and Darryl D Holm. "Commutator Errors in Large-Eddy Simulation." In: *Journal of Physics A: Mathematical and General* 39.9 (Feb. 2006), p. 2213. DOI: 10.1088/0305-4470/39/9/015.

[63] Bernard J. Geurts and Fedderik Van Der Bos. "Numerically Induced High-Pass Dynamics in Large-Eddy Simulation." In: *Physics of Fluids* 17.12 (Dec. 2005), p. 125103. DOI: 10.1063/1.2140022.

[64] S. Ghosal and P. Moin. "The Basic Equations for the Large Eddy Simulation of Turbulent Flows in Complex Geometry." In: *Journal of Computational Physics* 118.1 (Apr. 1995), pp. 24–37. DOI: 10.1006/jcph.1995.1077.

[65] Sandip Ghosal. "An Analysis of Numerical Errors in Large-Eddy Simulations of Turbulence." In: *Journal of Computational Physics* 125.1 (Apr. 1996), pp. 187–206. DOI: 10.1006/jcph.1996.0088.

[66] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 978-0-262-03561-3.

[67] Yifei Guan, Adam Subel, Ashesh Chattopadhyay, and Pedram Hassanzadeh. "Learning Physics-Constrained Subgrid-Scale Closures in the Small-Data Regime for Stable and Accurate LES." In: *Physica D: Nonlinear Phenomena* 443 (Jan. 2023), p. 133568. DOI: 10.1016/j.physd.2022.133568.

[68] Ernst Hairer, Michel Roche, and Christian Lubich. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Vol. 1409. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer, 1989. ISBN: 978-3-540-51860-0. DOI: 10.1007/BFb0093947.

[69] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II*. Vol. 14. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer, 1996. ISBN: 978-3-642-05220-0. DOI: 10.1007/978-3-642-05221-7.

[70] Per Christian Hansen. *Discrete Inverse Problems: Insight and Algorithms*. Fundamentals of Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 2010. ISBN: 978-0-89871-696-2.

[71] Francis H. Harlow and J. Eddie Welch. "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface." In: *The Physics of Fluids* 8.12 (Dec. 1965), pp. 2182–2189. DOI: 10.1063/1.1761178.

[72] Nicholas J. Higham and Theo Mary. "Mixed Precision Algorithms in Numerical Linear Algebra." In: *Acta Numerica* 31 (May 2022), pp. 347–414. DOI: 10.1017/S0962492922000022.

[73] Florian Hindenlang, Gregor J. Gassner, Christoph Altmann, Andrea Beck, Marc Staudenmaier, and Claus-Dieter Munz. "Explicit Discontinuous Galerkin Methods for Unsteady Problems." In: *Computers & Fluids* 61 (May 2012), pp. 86–93. DOI: 10.1016/j.compfluid.2012.03.006.

[74] Rik Hoekstra and Wouter Edeling. “Reduced Subgrid Scale Terms in Three-Dimensional Turbulence.” In: *Computer Methods in Applied Mechanics and Engineering* 449 (Feb. 2026), p. 118506. DOI: 10.1016/j.cma.2025.118506.

[75] Philipp Holl and Nils Thuerey. “PhiFlow: Differentiable Simulations for PyTorch, TensorFlow and Jax.” In: *Proceedings of the 41st International Conference on Machine Learning*. PMLR, July 2024, pp. 18515–18546.

[76] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer Feedforward Networks Are Universal Approximators.” In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[77] “IEEE Standard for Floating-Point Arithmetic.” In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (July 2019), pp. 1–84. DOI: 10.1109/IEEESTD.2019.8766229.

[78] Michael Innes. *Don’t Unroll Adjoint: Differentiating SSA-Form Programs*. Mar. 2019. DOI: 10.48550/arXiv.1810.07951. arXiv: 1810.07951 [cs].

[79] Anna Ivagnes, Toby Van Gastelen, Syver Døving Agdestein, Benjamin Sanderse, Giovanni Stabile, and Gianluigi Rozza. “A New Data-Driven Energy-Stable Evolve-Filter-Relax Model for Turbulent Flow Simulation.” In: *Computer Methods in Applied Mechanics and Engineering* 450 (Mar. 2026), p. 118654. DOI: 10.1016/j.cma.2025.118654.

[80] Bahrul Jalaali and Kie Okabayashi. *A Priori Assessment of Rotational Invariance in Multiscale CNN-Based Subgrid-Scale Model for Wall-Bounded Turbulent Flows*. Nov. 2025. DOI: 10.48550/arXiv.2511.16995. arXiv: 2511.16995 [physics].

[81] Antony Jameson. “The Construction of Discretely Conservative Finite Volume Schemes That Also Globally Conserve Energy or Entropy.” In: *Journal of Scientific Computing* 34.2 (Feb. 2008), pp. 152–187. DOI: 10.1007/s10915-007-9171-7.

[82] Jinhee Jeong and Fazle Hussain. “On the Identification of a Vortex.” In: *Journal of Fluid Mechanics* 285 (Feb. 1995), pp. 69–94. DOI: 10.1017/S0022112095000462.

[83] Martin Karp et al. “Effects of Lower Floating-Point Precision on Scale-Resolving Numerical Simulations of Turbulence.” In: *Journal of Computational Physics* 549 (Mar. 2026), p. 114600. DOI: 10.1016/j.jcp.2025.114600.

[84] Grzegorz Kaszuba, Tomasz Krakowski, Bartosz Ziegler, Andrzej Jaszkiewicz, and Piotr Sankowski. "Implicit Modeling of Equivariant Tensor Basis with Euclidean Turbulence Closure Neural Network." In: *Physics of Fluids* 37.2 (Feb. 2025). DOI: 10.1063/5.0249490.

[85] Tom Kimpson, E. Adam Paxton, Matthew Chantry, and Tim Palmer. "Climate-Change Modelling at Reduced Floating-Point Precision with Stochastic Rounding." In: *Quarterly Journal of the Royal Meteorological Society* 149.752 (2023), pp. 843–855. DOI: 10.1002/qj.4435.

[86] Ryan N. King, Peter E. Hamlington, and Werner J. A. Dahm. "Autonomic Closure for Turbulence Simulations." In: *Physical Review E* 93.3 (Mar. 2016), p. 031301. DOI: 10.1103/PhysRevE.93.031301.

[87] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs].

[88] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. Feb. 2017. DOI: 10.48550/arXiv.1609.02907. arXiv: 1609.02907 [cs].

[89] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. "Machine Learning–Accelerated Computational Fluid Dynamics." In: *Proceedings of the National Academy of Sciences* 118.21 (May 2021), e2101784118. DOI: 10.1073/pnas.2101784118.

[90] Georg Kohl, Li-Wei Chen, and Nils Thuerey. *Benchmarking Autoregressive Conditional Diffusion Models for Turbulent Flow Simulation*. Dec. 2024. DOI: 10.48550/arXiv.2309.01745. arXiv: 2309.01745 [cs].

[91] M. Kurz and A. Beck. "Investigating Model-Data Inconsistency in Data-Informed Turbulence Closure Terms." In: *14th WCCM-ECCOMAS Congress*. CIMNE, 2021. DOI: 10.23967/wccm-eccomas.2020.115.

[92] Marius Kurz and Andrea Beck. "A Machine Learning Framework for LES Closure Terms." In: *ETNA - Electronic Transactions on Numerical Analysis* 56 (2022), pp. 117–137. DOI: 10.1553/etna_vol56s117. arXiv: 2010.03030 [physics].

[93] Marius Kurz, Andrea Beck, and Benjamin Sanderse. *Harnessing Equivariance: Modeling Turbulence with Graph Neural Networks*. Apr. 2025. DOI: 10.48550/arXiv.2504.07741. arXiv: 2504.07741 [physics].

[94] Marius Kurz, Philipp Offenhäuser, Dominic Viola, Michael Resch, and Andrea Beck. "Relexi — A Scalable Open Source Reinforcement Learning Framework for High-Performance Computing." In: *Software Impacts* 14 (Dec. 2022), p. 100422. DOI: 10.1016/j.simpa.2022.100422.

[95] Maxime W. Lafarge, Erik J. Bekkers, Josien P.W. Pluim, Remco Duits, and Mitko Veta. "Roto-Translation Equivariant Convolutional Networks: Application to Histopathology Image Analysis." In: *Medical Image Analysis* 68 (Feb. 2021), p. 101849. DOI: 10.1016/j.media.2020.101849.

[96] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[97] Sanjiva K. Lele. "Compact Finite Difference Schemes with Spectral-like Resolution." In: *Journal of Computational Physics* 103.1 (Nov. 1992), pp. 16–42. DOI: 10.1016/0021-9991(92)90324-R.

[98] A. Leonard. "Energy Cascade in Large-Eddy Simulations of Turbulent Fluid Flows." In: *Advances in Geophysics*. Ed. by F. N. Frenkiel and R. E. Munn. Vol. 18. Turbulent Diffusion in Environmental Pollution. Elsevier, Jan. 1975, pp. 237–248. DOI: 10.1016/S0065-2687(08)60464-1.

[99] B. P. Leonard. "A Stable and Accurate Convective Modelling Procedure Based on Quadratic Upstream Interpolation." In: *Computer Methods in Applied Mechanics and Engineering* 19.1 (June 1979), pp. 59–98. DOI: 10.1016/0045-7825(79)90034-3.

[100] Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink. "A Public Turbulence Database Cluster and Applications to Study Lagrangian Evolution of Velocity Increments in Turbulence." In: *Journal of Turbulence* 9 (Jan. 2008), N31. DOI: 10.1080/14685240802376389.

[101] D Lilly. *The Representation of Small-Scale Turbulence in Numerical Simulation Experiments*. Tech. rep. UCAR/NCAR, Nov. 1966, 986 KB. DOI: 10.5065/D62R3PMM.

[102] Douglas K. Lilly. "On the Computational Stability of Numerical Solutions of Time-Dependent Non-Linear Geophysical Fluid Dynamics Problems." In: *Monthly Weather Review* 93.1 (Jan. 1965), pp. 11–25. DOI: 10.1175/1520-0493(1965)093<0011:OTCSON>2.3.CO;2.

[103] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. "Reynolds Averaged Turbulence Modelling Using Deep Neural Networks with Embedded Invariance." In: *Journal of Fluid Mechanics* 807 (Nov. 2016), pp. 155–166. DOI: 10.1017/jfm.2016.615.

[104] Mario Lino, Stathi Fotiadis, Anil A. Bharath, and Chris D. Cantwell. "Multi-Scale Rotation-Equivariant Graph Neural Networks for Unsteady Eulerian Fluid Dynamics." In: *Physics of Fluids* 34.8 (Aug. 2022). DOI: 10.1063/5.0097679.

[105] B. List, M. Lino, and N. Thuerey. "Rotational Equivariant Graph Neural Networks via Local Eigenbasis Transformations." In: *Physics of Fluids* 37.8 (Aug. 2025), p. 087178. DOI: 10.1063/5.0279499.

[106] Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. "Differentiability in Unrolled Training of Neural Physics Simulators on Transient Dynamics." In: *Computer Methods in Applied Mechanics and Engineering* 433 (Jan. 2025), p. 117441. DOI: 10.1016/j.cma.2024.117441.

[107] Björn List, Li-Wei Chen, and Nils Thuerey. "Learned Turbulence Modelling with Differentiable Fluid Solvers: Physics-Based Loss Functions and Optimisation Horizons." In: *Journal of Fluid Mechanics* 949 (Oct. 2022), A25. DOI: 10.1017/jfm.2022.738.

[108] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. May 2017. DOI: 10.48550/arXiv.1608.03983. arXiv: 1608.03983 [cs].

[109] Hao Lu and Christopher J. Rutland. "Structural Subgrid-Scale Modeling for Large-Eddy Simulation: A Review." In: *Acta Mechanica Sinica* 32.4 (Aug. 2016), pp. 567–578. DOI: 10.1007/s10409-016-0556-4.

[110] David G. Luenberger. "The Conjugate Residual Method for Constrained Minimization Problems." In: *SIAM Journal on Numerical Analysis* (July 2006). DOI: 10.1137/0707032.

[111] T.S. Lund. "The Use of Explicit Filters in Large Eddy Simulation." In: *Computers & Mathematics with Applications* 46.4 (Aug. 2003), pp. 603–616. DOI: 10.1016/S0898-1221(03)90019-8.

[112] Thomas S. Lund and E. A. Novikov. *Parameterization of Subgrid-Scale Stress by the Velocity Gradient Tensor*. Jan. 1993.

[113] T. S. Lundgren. *Linearly Forced Isotropic Turbulence*. Jan. 2003.

[114] Jonathan F. MacArt, Justin Sirignano, and Jonathan B. Freund. "Embedded Training of Neural-Network Subgrid-Scale Turbulence Models." In: *Physical Review Fluids* 6.5 (May 2021), p. 050502. DOI: 10.1103/PhysRevFluids.6.050502.

[115] Alison L. Marsden, Oleg V. Vasilyev, and Parviz Moin. "Construction of Commutative Filters for LES on Unstructured Meshes." In: *Journal of Computational Physics* 175.2 (Jan. 2002), pp. 584–603. DOI: 10.1006/jcph.2001.6958.

[116] R. Maulik and O. San. “A Neural Network Approach for the Blind Deconvolution of Turbulent Flows.” In: *Journal of Fluid Mechanics* 831 (Nov. 2017), pp. 151–181. DOI: 10.1017/jfm.2017.637.

[117] Romit Maulik and Omer San. “A Stable and Scale-Aware Dynamic Modeling Framework for Subgrid-Scale Parameterizations of Two-Dimensional Turbulence.” In: *Computers & Fluids* 158 (Nov. 2017), pp. 11–38. DOI: 10.1016/j.compfluid.2016.11.015.

[118] Romit Maulik and Omer San. “Explicit and Implicit LES Closures for Burgers Turbulence.” In: *Journal of Computational and Applied Mathematics* 327 (Jan. 2018), pp. 12–40. DOI: 10.1016/j.cam.2017.06.003.

[119] Romit Maulik, Omer San, Adil Rasheed, and Prakash Vedula. “Subgrid Modelling for Two-Dimensional Turbulence Using Neural Networks.” In: *Journal of Fluid Mechanics* 858 (Jan. 2019), pp. 122–144. DOI: 10.1017/jfm.2018.770. arXiv: 1808.02983 [physics].

[120] Ryley McConkey, Julia Balla, Jeremiah Bailey, Ali Backour, Elyssa Hofgard, Tommi Jaakkola, Abigail Bodner, and Tess Smidt. *Turbulence Teaches Equivariance to Neural Networks*. Feb. 2026. DOI: 10.48550/arXiv.2602.04695. arXiv: 2602.04695 [physics].

[121] Hugo Melchers, Daan Crommelin, Barry Koren, Vlado Menkovski, and Benjamin Sanderse. “Comparison of Neural Closure Models for Discretised PDEs.” In: *Computers & Mathematics with Applications* 143 (Aug. 2023), pp. 94–107. DOI: 10.1016/j.camwa.2023.04.030.

[122] Miguel A. Mendez, Steven L. Brunton, and Miguel A./Ianiro Mendez Andre. *Data-Driven Fluid Mechanics*. 2023. ISBN: 978-1-108-90226-7.

[123] Charles Meneveau. “Lagrangian Dynamics and Models of the Velocity Gradient Tensor in Turbulent Flows.” In: *Annual Review of Fluid Mechanics* 43.1 (Jan. 2011), pp. 219–245. DOI: 10.1146/annurev-fluid-122109-160708.

[124] Charles Meneveau and Joseph Katz. “Scale-Invariance and Turbulence Models for Large-Eddy Simulation.” In: *Annual Review of Fluid Mechanics* 32.1 (Jan. 2000), pp. 1–32. DOI: 10.1146/annurev.fluid.32.1.1.

[125] Pedro M. Milani, Julia Ling, and John K. Eaton. “On the Generality of Tensor Basis Neural Networks for Turbulent Scalar Flux Modeling.” In: *International Communications in Heat and Mass Transfer* 128 (Nov. 2021), p. 105626. DOI: 10.1016/j.icheatmasstransfer.2021.105626.

[126] Aaron Miller, Sahil Kommalapati, Robert Moser, and Petros Koumoutsakos. *Symmetry Aware Reynolds Averaged Navier Stokes Turbulence Models with Equivariant Neural Networks*. Nov. 2025. DOI: 10.48550/arXiv.2511.09769. arXiv: 2511.09769 [physics].

[127] Chin-Hoh Moeng. "A Large-Eddy-Simulation Model for the Study of Planetary Boundary-Layer Turbulence." In: *Journal of the Atmospheric Sciences* 41.13 (July 1984), pp. 2052–2062. DOI: 10.1175/1520-0469(1984)041<2052:ALESMF>2.0.CO;2.

[128] P. Moin, W. C. Reynolds, and J. H. Ferziger. *Large Eddy Simulation of Incompressible Turbulent Channel Flow*. Tech. rep. NASA-CR-152190. May 1978.

[129] Y. Morinishi, T. S. Lund, O. V. Vasilyev, and P. Moin. "Fully Conservative Higher Order Finite Difference Schemes for Incompressible Flow." In: *Journal of Computational Physics* 143.1 (June 1998), pp. 90–124. DOI: 10.1006/jcph.1998.5962.

[130] F. Nicoud and F. Ducros. "Subgrid-Scale Stress Modelling Based on the Square of the Velocity Gradient Tensor." In: *Flow, Turbulence and Combustion* 62.3 (Sept. 1999), pp. 183–200. DOI: 10.1023/A:1009995426001.

[131] Franck Nicoud, Hubert Baya Toda, Olivier Cabrit, Sanjeeb Bose, and Jungil Lee. "Using Singular Values to Build a Subgrid-Scale Model for Large Eddy Simulations." In: *Physics of Fluids* 23.8 (Aug. 2011), p. 085106. DOI: 10.1063/1.3623274.

[132] Xavier Normand and Marcel Lesieur. "Direct and Large-Eddy Simulations of Transition in the Compressible Boundary Layer." In: *Theoretical and Computational Fluid Dynamics* 3.4 (Feb. 1992), pp. 231–252. DOI: 10.1007/BF00417915.

[133] Martin Oberlack. "Invariant Modeling in Large-Eddy Simulation of Turbulence." In: *Annual Research Briefs, Center for Turbulence. Research, NASA* 3 (1997).

[134] Peter J. Olver. *Applications of Lie Groups to Differential Equations*. Vol. 107. Graduate Texts in Mathematics. New York, NY: Springer, 1986. ISBN: 978-1-4684-0276-6. DOI: 10.1007/978-1-4684-0274-2.

[135] Paolo Orlandi and R. Moreau, eds. *Fluid Flow Phenomena*. Vol. 55. Fluid Mechanics and Its Applications. Dordrecht: Springer Netherlands, 2000. ISBN: 978-1-4020-0389-9. DOI: 10.1007/978-94-011-4281-6.

[136] Steven A. Orszag and G. S. Patterson. "Numerical Simulation of Three-Dimensional Homogeneous Isotropic Turbulence." In: *Physical Review Letters* 28.2 (Jan. 1972), pp. 76–79. DOI: 10.1103/PhysRevLett.28.76.

[137] Avik Pal. *Lux: Explicit Parameterization of Deep Neural Networks in Julia*. Zenodo. Apr. 2023. DOI: 10.5281/ZENODO.7808904.

[138] Jonghwan Park and Haecheon Choi. "Toward Neural-Network-Based Large Eddy Simulation: Application to Turbulent Channel Flow." In: *Journal of Fluid Mechanics* 914 (May 2021), A16. DOI: 10.1017/jfm.2020.931.

[139] G. S. Patterson and Steven A. Orszag. "Spectral Calculations of Isotropic Turbulence: Efficient Removal of Aliasing Interactions." In: *The Physics of Fluids* 14.11 (Nov. 1971), pp. 2538–2541. DOI: 10.1063/1.1693365.

[140] Benjamin Peherstorfer and Karen Willcox. "Data-Driven Operator Inference for Nonintrusive Projection-Based Model Reduction." In: *Computer Methods in Applied Mechanics and Engineering* 306 (July 2016), pp. 196–215. DOI: 10.1016/j.cma.2016.03.025.

[141] J. Blair Perot. "Discrete Conservation Properties of Unstructured Mesh Schemes." In: *Annual Review of Fluid Mechanics* 43.1 (Jan. 2011), pp. 299–318. DOI: 10.1146/annurev-fluid-122109-160645.

[142] Thierry Poinsot and Denis Veynante. *Theoretical and Numerical Combustion*. 2. ed. Philadelphia, Pa: Edwards, 2005. ISBN: 978-1-930217-10-2.

[143] S. B. Pope. "A More General Effective-Viscosity Hypothesis." In: *Journal of Fluid Mechanics* 72.02 (Nov. 1975), p. 331. DOI: 10.1017/S0022112075003382.

[144] S. B. Pope. *Turbulent Flows*. Cambridge ; New York: Cambridge University Press, 2000. ISBN: 978-0-521-59125-6.

[145] Aviral Prakash, Kenneth E. Jansen, and John A. Evans. "Invariant Data-Driven Subgrid Stress Modeling in the Strain-Rate Eigenframe for Large Eddy Simulation." In: *Computer Methods in Applied Mechanics and Engineering* 399 (Sept. 2022), p. 115457. DOI: 10.1016/j.cma.2022.115457.

[146] Aviral Prakash, Kenneth E. Jansen, and John A. Evans. "Invariant Data-Driven Subgrid Stress Modeling on Anisotropic Grids for Large Eddy Simulation." In: *Computer Methods in Applied Mechanics and Engineering* 422 (Mar. 2024), p. 116807. DOI: 10.1016/j.cma.2024.116807.

[147] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. *Universal Differential Equations for Scientific Machine Learning*. Nov. 2021. DOI: 10.48550/arXiv.2001.04385. arXiv: 2001.04385 [cs].

[148] Dina Razafindralandy, Aziz Hamdouni, and Martin Oberlack. "Analysis and Development of Subgrid Turbulence Models Preserving the Symmetry Properties of the Navier–Stokes Equations." In: *European Journal of Mechanics - B/Fluids* 26.4 (July 2007), pp. 531–550. DOI: 10.1016/j.euromechflu.2006.10.003.

[149] Lewis Fry Richardson. *Weather Prediction by Numerical Process*. 2nd ed. Cambridge Mathematical Library. Cambridge: Cambridge University Press, 2007. ISBN: 978-0-521-68044-8. DOI: 10.1017/CBO9780511618291.

[150] R. S. Rogallo and P. Moin. "Numerical Simulation of Turbulent Flows." In: *Annual Review of Fluid Mechanics* 16 (Jan. 1984), pp. 99–137. DOI: 10.1146/annurev.fl.16.010184.000531.

[151] Robert S. Rogallo. "Numerical Experiments in Homogeneous Turbulence." In: *NASA Technical Memo* (1981).

[152] Pierre Sagaut, ed. *Large Eddy Simulation for Incompressible Flows: An Introduction*. Third Edition. Scientific Computation. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-26344-9. DOI: 10.1007/b137536.

[153] Omer San and Anne E. Staples. "High-Order Methods for Decaying Two-Dimensional Homogeneous Isotropic Turbulence." In: *Computers & Fluids* 63 (June 2012), pp. 105–127. DOI: 10.1016/j.compfluid.2012.04.006.

[154] B. Sanderse. "Energy-Conserving Runge–Kutta Methods for the Incompressible Navier–Stokes Equations." In: *Journal of Computational Physics* 233 (Jan. 2013), pp. 100–131. DOI: 10.1016/j.jcp.2012.07.039.

[155] B. Sanderse and F.X. Trias. "Energy-Consistent Discretization of Viscous Dissipation with Application to Natural Convection Flow." In: *Computers & Fluids* 286 (Jan. 2025), p. 106473. DOI: 10.1016/j.compfluid.2024.106473.

[156] B. (Benjamin) Sanderse. *Energy-Conserving Discretization Methods for the Incompressible Navier-Stokes Equations:Application to the Simulation of Wind-Turbine Wakes*. 2013. DOI: 10.6100/IR750543.

[157] Benjamin Sanderse, Panos Stinis, Romit Maulik, and Shady E. Ahmed. "Scientific Machine Learning for Closure Models in Multiscale Problems: A Review." In: *Foundations of Data Science* 7.1 (2025), pp. 298–337. DOI: 10.3934/fods.2024043.

[158] U. Schumann. "Subgrid Scale Model for Finite Difference Simulations of Turbulent Flows in Plane Channels and Annuli." In: *Journal of Computational Physics* 18 (Aug. 1975), pp. 376–404. DOI: 10.1016/0021-9991(75)90093-5.

[159] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units." In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, June 2016, pp. 2217–2225.

[160] Varun Shankar, Shivam Barwey, Zico Kolter, Romit Maulik, and Venkatasubramanian Viswanathan. *Importance of Equivariant and Invariant Symmetries for Fluid Flow Modeling*. May 2023. DOI: 10.48550/arXiv.2307.05486. arXiv: 2307.05486 [physics].

[161] Varun Shankar, Dibyajyoti Chakraborty, Venkatasubramanian Viswanathan, and Romit Maulik. "Differentiable Turbulence: Closure as a Partial Differential Equation Constrained Optimization." In: *Physical Review Fluids* 10.2 (Feb. 2025), p. 024605. DOI: 10.1103/PhysRevFluids.10.024605.

[162] Varun Shankar, Vedant Puri, Ramesh Balakrishnan, Romit Maulik, and Venkatasubramanian Viswanathan. *Differentiable Physics-Enabled Closure Modeling for Burgers' Turbulence*. Sept. 2022. DOI: 10.48550/arXiv.2209.11614. arXiv: 2209.11614 [physics].

[163] Wei Shyy. "A Study of Finite Difference Approximations to Steady-State, Convection-Dominated Flow Problems." In: *Journal of Computational Physics* 57.3 (Feb. 1985), pp. 415–438. DOI: 10.1016/0021-9991(85)90188-3.

[164] Maurits H. Silvis, Ronald A. Remmerswaal, and Roel Verstappen. "Physical Consistency of Subgrid-Scale Models for Large-Eddy Simulation of Incompressible Turbulent Flows." In: *Physics of Fluids* 29.1 (Jan. 2017), p. 015105. DOI: 10.1063/1.4974093.

[165] Justin Sirignano, Jonathan MacArt, and Konstantinos Spiliopoulos. "PDE-constrained Models with Neural Network Terms: Optimization and Global Convergence." In: *Journal of Computational Physics* 481 (May 2023), p. 112016. DOI: 10.1016/j.jcp.2023.112016.

[166] Justin Sirignano and Jonathan F. MacArt. *Dynamic Deep Learning LES Closures: Online Optimization With Embedded DNS*. Mar. 2023. DOI: 10.48550/arXiv.2303.02338. arXiv: 2303.02338 [physics].

[167] Justin Sirignano, Jonathan F. MacArt, and Jonathan B. Freund. "DPM: A Deep Learning PDE Augmentation Method with Application to Large-Eddy Simulation." In: *Journal of Computational Physics* 423 (Dec. 2020), p. 109811. DOI: 10.1016/j.jcp.2020.109811.

[168] J. Smagorinsky. "General Circulation Experiments with the Primitive Equations: I. The Basic Experiment." In: *Monthly Weather Review* 91.3 (Mar. 1963), pp. 99–164. DOI: 10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2.

[169] G.F. Smith. "On Isotropic Functions of Symmetric Tensors, Skew-Symmetric Tensors and Vectors." In: *International Journal of Engineering Science* 9.10 (Oct. 1971), pp. 899–916. DOI: 10.1016/0020-7225(71)90023-1.

[170] Philippe Spalart. *An Old-Fashioned Framework for Machine Learning in Turbulence Modeling*. Aug. 2023. DOI: 10.48550/arXiv.2308.00837. arXiv: 2308.00837 [physics].

[171] A. J. M. Spencer and R. S. Rivlin. "The Theory of Matrix Polynomials and Its Application to the Mechanics of Isotropic Continua." In: *Archive for Rational Mechanics and Analysis* 2.1 (Jan. 1958), pp. 309–336. DOI: 10.1007/BF00277933.

[172] A. J. M. Spencer and R. S. Rivlin. "Further Results in the Theory of Matrix Polynomials." In: *Archive for Rational Mechanics and Analysis* 4.1 (Jan. 1959), pp. 214–230. DOI: 10.1007/BF00281388.

[173] Katepalli R. Sreenivasan. "On the Universality of the Kolmogorov Constant." In: *Physics of Fluids* 7.11 (Nov. 1995), pp. 2778–2784. DOI: 10.1063/1.868656.

[174] Eric W. Stallcup and Werner J. Dahm. "Adaptive Scale-Similar Closure for Large Eddy Simulations. Part 2: Subgrid Scalar Flux Closure." In: *AIAA SCITECH 2022 Forum*. San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, Jan. 2022. ISBN: 978-1-62410-631-6. DOI: 10.2514/6.2022-0596.

[175] Eric W. Stallcup, Abhinav Kshitij, and Werner J. Dahm. "Adaptive Scale-Similar Closure for Large Eddy Simulations. Part 1: Subgrid Stress Closure." In: *AIAA SCITECH 2022 Forum*. San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, Jan. 2022. ISBN: 978-1-62410-631-6. DOI: 10.2514/6.2022-0595.

[176] Robin Stoffer, Caspar M. van Leeuwen, Damian Podareanu, Valeriu Codreanu, Menno A. Veerman, Martin Janssens, Oscar K. Hartogensis, and Chiel C. van Heerwaarden. "Development of a Large-Eddy Simulation Subgrid Model Based on Artificial Neural Networks: A Case Study of Turbulent Channel Flow." In: *Geoscientific Model Development* 14.6 (June 2021), pp. 3769–3788. DOI: 10.5194/gmd-14-3769-2021.

[177] Geoffrey Ingram Taylor and Albert Edward Green. "Mechanism of the Production of Small Eddies from Large Ones." In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 158.895 (Feb. 1937), pp. 499–521. DOI: 10.1098/rspa.1937.0036.

[178] R. Témam. "Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (I)." In: *Archive for Rational Mechanics and Analysis* 32.2 (Jan. 1969), pp. 135–153. DOI: 10.1007/BF00247678.

[179] Tobias Thornes, Peter Düben, and Tim Palmer. "On the Use of Scale-dependent Precision in Earth System Modelling." In: *Quarterly Journal of the Royal Meteorological Society* 143.703 (Jan. 2017), pp. 897–908. DOI: 10.1002/qj.2974.

[180] Emilio E. Torres and Werner J. Dahm. "Rational Boolean Stabilization of Subgrid Models for Large Eddy Simulation." In: *AIAA SCITECH 2023 Forum*. National Harbor, MD & Online: American Institute of Aeronautics and Astronautics, Jan. 2023. ISBN: 978-1-62410-699-6. DOI: 10.2514/6.2023-2485.

[181] F. X. Trias, D. Folch, A. Gorobets, and A. Oliva. "Building Proper Invariants for Eddy-Viscosity Subgrid-Scale Models." In: *Physics of Fluids* 27.6 (June 2015), p. 065103. DOI: 10.1063/1.4921817.

[182] F. X. Trias, M. Soria, A. Oliva, and C. D. Pérez-Segarra. "Direct Numerical Simulations of Two- and Three-Dimensional Turbulent Natural Convection Flows in a Differentially Heated Cavity of Aspect Ratio 4." In: *Journal of Fluid Mechanics* 586 (Sept. 2007), pp. 259–293. DOI: 10.1017/S0022112007006908.

[183] F.X. Trias and R.W.C.P. Verstappen. "On the Construction of Discrete Filters for Symmetry-Preserving Regularization Models." In: *Computers & Fluids* 40.1 (Jan. 2011), pp. 139–148. DOI: 10.1016/j.compfluid.2010.08.015.

[184] Ch. Tsitouras. "Runge–Kutta Pairs of Order 5(4) Satisfying Only the First Column Simplifying Assumption." In: *Computers & Mathematics with Applications* 62.2 (July 2011), pp. 770–775. DOI: 10.1016/j.camwa.2011.06.002.

[185] Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. "Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6111–6122.

[186] Harmen van der Ven. "A Family of Large Eddy Simulation (LES) Filters with Nonuniform Filter Widths." In: *Physics of Fluids* 7.5 (May 1995), pp. 1171–1172. DOI: 10.1063/1.868561.

[187] T. Van Gastelen, W. Edeling, and B. Sanderse. "Energy-Conserving Neural Network for Turbulence Closure Modeling." In: *Journal of Computational Physics* 508 (July 2024), p. 113003. DOI: 10.1016/j.jcp.2024.113003.

[188] Oleg V. Vasilyev, Thomas S. Lund, and Parviz Moin. "A General Class of Commutative Filters for LES in Complex Geometries." In: *Journal of Computational Physics* 146.1 (Oct. 1998), pp. 82–104. DOI: 10.1006/jcph.1998.6060.

[189] Arthur E. P. Veldman. ""Missing" Boundary Conditions? Discretize First, Substitute Next, and Combine Later." In: *SIAM Journal on Scientific and Statistical Computing* 11.1 (Jan. 1990), pp. 82–91. DOI: 10.1137/0911005.

[190] R.W.C.P. Verstappen and A.E.P. Veldman. "Direct Numerical Simulation of Turbulence at Lower Costs." In: *Journal of Engineering Mathematics* 32.2 (Oct. 1997), pp. 143–159. DOI: 10.1023/A:1004255329158.

[191] R.W.C.P. Verstappen and A.E.P. Veldman. "Symmetry-Preserving Discretization of Turbulent Flow." In: *Journal of Computational Physics* 187.1 (May 2003), pp. 343–368. DOI: 10.1016/S0021-9991(03)00126-8.

[192] Roel Verstappen. "When Does Eddy Viscosity Damp Subfilter Scales Sufficiently?" In: *Journal of Scientific Computing* 49.1 (Oct. 2011), pp. 94–110. DOI: 10.1007/s10915-011-9504-4.

[193] Roel Verstappen. *Merging Filtering, Modeling and Discretization to Simulate Large Eddies in Burgers' Turbulence*. SSRN Scholarly Paper. Rochester, NY, Apr. 2025. DOI: 10.2139/ssrn.5221020. Social Science Research Network: 5221020.

[194] P. Vieillefosse. "Local Interaction between Vorticity and Shear in a Perfect Incompressible Fluid." In: *Journal de Physique* 43.6 (1982), pp. 837–842. DOI: 10.1051/jphys:01982004306083700.

[195] A. W. Vreman. "An Eddy-Viscosity Subgrid-Scale Model for Turbulent Shear Flow: Algebraic Theory and Applications." In: *Physics of Fluids* 16.10 (Oct. 2004), pp. 3670–3681. DOI: 10.1063/1.1785131.

[196] A. W. Vreman. "The Projection Method for the Incompressible Navier–Stokes Equations: The Pressure near a No-Slip Wall." In: *Journal of Computational Physics* 263 (Apr. 2014), pp. 353–374. DOI: 10.1016/j.jcp.2014.01.035.

[197] A. W. Vreman and J. G. M. Kuerten. "Comparison of Direct Numerical Simulation Databases of Turbulent Channel Flow at $Re_\tau = 180$." In: *Physics of Fluids* 26.1 (Jan. 2014), p. 015102. DOI: 10.1063/1.4861064.

[198] Bert Vreman, Bernard Geurts, and Hans Kuerten. "Comparison of Numerical Schemes in Large-Eddy Simulation of the Temporal Mixing Layer." In: *International Journal for Numerical Methods in Fluids* 22.4 (1996), pp. 297–311. DOI: 10.1002/(SICI)1097-0363(19960229)22:4<297::AID-FLD361>3.0.CO;2-X.

[199] Bert Vreman, Bernard Geurts, and Hans Kuerten. "Large-Eddy Simulation of the Temporal Mixing Layer Using the Clark Model." In: *Theoretical and Computational Fluid Dynamics* 8.4 (July 1996), pp. 309–324. DOI: 10.1007/BF00639698.

[200] Bert Vreman, Bernard Geurts, and Hans Kuerten. “Large-Eddy Simulation of the Turbulent Mixing Layer.” In: *Journal of Fluid Mechanics* 339 (May 1997), pp. 357–390. DOI: 10.1017/S0022112097005429.

[201] Rui Wang, Robin Walters, and Tess E Smidt. “Relaxed Octahedral Group Convolution for Learning Symmetry Breaking in 3D Physical Systems.” In: ().

[202] Maurice Weiler and Gabriele Cesa. “General E(2)-Equivariant Steerable CNNs.” In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[203] Maurice Weiler, Patrick Forré, Erik Verlinde, and Max Welling. “Equivariant and Coordinate Independent Convolutional Networks.” In: (2023).

[204] Gabriel D. Weymouth and Bernat Font. “WaterLily.Jl: A Differentiable and Backend-Agnostic Julia Solver for Incompressible Viscous Flow around Dynamic Bodies.” In: *Computer Physics Communications* 315 (Oct. 2025), p. 109748. DOI: 10.1016/j.cpc.2025.109748.

[205] Grégoire S. Winckelmans, Alan A. Wray, Oleg V. Vasilyev, and Hervé Jeanmart. “Explicit-Filtering Large-Eddy Simulation Using the Tensor-Diffusivity Model Supplemented by a Dynamic Smagorinsky Term.” In: *Physics of Fluids* 13.5 (May 2001), pp. 1385–1403. DOI: 10.1063/1.1360192.

[206] Alan A. Wray. “Minimal Storage Time Advancement Schemes for Spectral Methods.” In: *NASA Ames Research Center, California, Report No. MS* 202 (1990).

[207] Andy Wu and Sanjiva K. Lele. “Two Neural Network Unet Architecture for Subfilter Stress Modeling.” In: *Physical Review Fluids* 10.1 (Jan. 2025), p. 014601. DOI: 10.1103/PhysRevFluids.10.014601.

[208] Maochao Xiao et al. *SmartFlow: A CFD-solver-agnostic Deep Reinforcement Learning Framework for Computational Fluid Dynamics on HPC Platforms*. Aug. 2025. DOI: 10.48550/arXiv.2508.00645. arXiv: 2508.00645 [physics].

[209] P. K. Yeung and S. B. Pope. “Lagrangian Statistics from Direct Numerical Simulations of Isotropic Turbulence.” In: *Journal of Fluid Mechanics* 207 (Oct. 1989), pp. 531–586. DOI: 10.1017/S0022112089002697.

LIST OF PHD PUBLICATIONS

1. Syver Døving Agdestein and Benjamin Sanderse. "Discretize First, Filter next – a New Closure Model Approach." In: *8th European Congress on Computational Methods in Applied Sciences and Engineering*. CIMNE, 2022. DOI: 10.23967/eccomas.2022.094
2. Syver Døving Agdestein and Benjamin Sanderse. "Discretize First, Filter next: Learning Divergence-Consistent Closure Models for Large-Eddy Simulation." In: *Journal of Computational Physics* 522 (Feb. 2025), p. 113577. DOI: 10.1016/j.jcp.2024.113577
3. Syver Døving Agdestein, Roel Verstappen, and Benjamin Sanderse. "Exact Expressions for the Unresolved Stress in a Finite-Volume Based Large-Eddy Simulation." In: *Journal of Computational Physics* 556 (July 2026), p. 114810. DOI: 10.1016/j.jcp.2026.114810
4. Syver Døving Agdestein and Benjamin Sanderse. *Comparison of Data-Driven Symmetry-Preserving Closure Models for Large-Eddy Simulation*. Mar. 2026. DOI: 10.48550/arXiv.2603.05325. arXiv: 2603.05325 [math]
5. Syver Døving Agdestein and Benjamin Sanderse. *A Differentiable Software Suite for Accelerated Simulation of Turbulent Flows*. Apr. 2026. DOI: 10.48550/arXiv.2604.18536. arXiv: 2604.18536 [math]

ABOUT THE AUTHOR

Syver Døving Agdestein was born on 05-07-1996 in Bærum, Norway. He graduated from Asker upper secondary school in 2015 and went on to study mathematical engineering at INSA Toulouse in France. In 2018, he spent an exchange year at Peter the Great St. Petersburg Polytechnic University in Russia, in the department of applied mathematics and theoretical mechanics. For his end-of-studies internship, he worked for six months in the multi-disciplinary design optimization group at IRT Saint Exupéry in Toulouse. After graduating from INSA Toulouse in 2020, he spent six months as a research software engineer working on diffusion magnetic resonance imaging at Polytechnique/INRIA Saclay in Palaiseau, France. In 2021, he started his PhD in the Scientific Computing group at CWI in Amsterdam.

ACKNOWLEDGMENTS

I would like to thank my supervisors Benjamin and Roel for their expertise and thoughtful feedback throughout this project. Having different perspectives on my research, along with access to a broad network of fellow researchers, has been truly inspiring.

I am equally grateful to my colleagues in the Scientific Computing group at CWI for stimulating discussions, remarkable humor, long walks in the parks of Amsterdam, and shared trips to conferences around the world.

I also wish to thank CWI for providing an excellent research environment: generous office space, whiteboards in every room to encourage discussion, a piano for late-night cultivation, and weekly leftover food from various events. I am particularly grateful to the secretarial, IT, library, and support staff for their invaluable assistance and warmth.

Finally, I thank my family for their unwavering support and for keeping me connected to home through a steady stream of photographs from Norway.

SUMMARY

Turbulent flows are ubiquitous in nature and engineering, yet their direct numerical simulation (DNS) remains prohibitively expensive at the high Reynolds numbers encountered in practice. Large-eddy simulation (LES) reduces this cost by resolving only the large-scale motions and modeling the effect of the unresolved scales through a closure model. This thesis develops new data-driven closure models for LES of incompressible turbulence, guided by the principle that the numerical discretization should be treated as a fundamental part of the LES formulation.

The thesis is organized around three research directions. The first concerns discretization-consistent closure formulations. Starting from the linear convection equation with a non-uniform filter, intrusive and non-intrusive methods for learning the discretely filtered operator are compared, and derivative fitting is found to provide the best trade-off between accuracy and practicality. The "discretize first, filter next" approach is then extended to the incompressible Navier-Stokes equations, where a divergence-consistent face-averaging filter is proposed that ensures full model-data consistency and avoids pressure-related stability issues. It is shown that when the discretization is properly accounted for, a-priori training suffices for accurate and stable neural-network-based LES. Exact expressions for the unresolved stress tensor in discrete filtered conservation laws are further derived, revealing that this tensor is non-symmetric and non-local—properties absent from the classical continuous formulation.

The second direction investigates the incorporation of physical structure into closure models. Unconstrained neural networks, tensor basis neural networks, and group-equivariant convolutional neural networks are compared for predicting the sub-filter stress. While unconstrained networks achieve the highest pointwise accuracy, symmetry-preserving architectures better respect the physical constraints of the equations, producing more physically consistent velocity-gradient statistics. Moreover, built-in symmetry constraints reduce the space of learnable functions, which may improve generalization to unseen Reynolds numbers and flow configurations. Symmetry-preserving models also cannot introduce spurious frame-dependent forces, improving numerical stability.

The third direction addresses the software infrastructure needed for data-driven closure modeling. The open-source Julia package IncompressibleNavierStokes.jl is presented for DNS and LES on staggered Cartesian grids. The solver uses energy-conserving finite volume discretizations with hardware-agnostic kernels that run on both CPUs and GPUs, and is made fully differ-

entiable through hand-written adjoint kernels, enabling a-posteriori training of neural network closure models embedded in the solver.

The overarching conclusion is that before any LES modeling can take place, the filter, grid, numerical fluxes, and discretization scheme should be clearly defined. Closure models should be designed to act on top of the implicit dissipation already incorporated into the numerical scheme, so that sub-filter effects are not accounted for twice.

SAMENVATTING

Turbulente stromingen komen overal voor in de natuur en techniek, maar directe numerieke simulatie (DNS) ervan blijft onbetaalbaar duur bij de hoge Reynoldsgetallen die in de praktijk voorkomen. Large-eddy-simulatie (LES) verlaagt deze kosten door alleen de grootschalige bewegingen op te lossen en het effect van de onopgeloste schalen te modelleren met behulp van een sluitingsmodel. Dit proefschrift ontwikkelt nieuwe datagedreven sluitingsmodellen voor LES van incompressibele turbulentie, geleid door het principe dat de numerieke discretisatie als een fundamenteel onderdeel van de LES-formulering moet worden beschouwd.

Het proefschrift is georganiseerd rond drie onderzoeksrichtingen. De eerste betreft discretisatie-consistente sluitingsformuleringen. Uitgaande van de lineaire convectievergelijking met een niet-uniform filter vergelijken we intrusieve en niet-intrusieve methoden om de discreet gefilterde operator te leren, en concluderen dat afgeleide-fitting de beste balans biedt tussen nauwkeurigheid en praktische toepasbaarheid. Vervolgens breiden we de benadering "discretiseer eerst, filter daarna" uit naar de incompressibele Navier-Stokesvergelijkingen, waarbij we een divergentie-consistent oppervlaktegemiddeld filter voorstellen dat volledige model-data-consistentie waarborgt en drukgerelateerde stabiliteitsproblemen vermijdt. We laten zien dat wanneer de discretisatie correct wordt meegenomen, a-priori training volstaat voor nauwkeurige en stabiele neurale-netwerk-gebaseerde LES. Daarnaast leiden we exacte uitdrukkingen af voor de onopgeloste spanningstensor in discreet gefilterde behoudswetten, waaruit blijkt dat deze tensor niet-symmetrisch en niet-lokaal is—eigenschappen die afwezig zijn in de klassieke continue formulering.

De tweede richting onderzoekt het inbouwen van fysische structuur in sluitingsmodellen. Neurale netwerken zonder restricties, tensorbasis-neurale netwerken en groepsequivariante convolutionele neurale netwerken worden vergeleken voor het voorspellen van de subfilterspanning. Hoewel netwerken zonder restricties de hoogste puntsgewijze nauwkeurigheid behalen, respecteren symmetriebehoudende architecturen beter de fysische restricties van de vergelijkingen en produceren ze fysisch consistentere snelheidsgradiëntstatistieken. Bovendien verkleinen ingebouwde symmetriebeperkingen de ruimte van leerbare functies, wat de generalisatie naar onbekende Reynoldsgetallen en stromingsconfiguraties kan verbeteren. Symmetriebehoudende modellen kunnen ook geen onechte referentiekaderafhankelijke krachten introduceren, wat de numerieke stabiliteit verbetert.

De derde richting betreft de software-infrastructuur die nodig is voor datagedreven sluitingsmodellering. We presenteren IncompressibleNavier-

Stokes.jl, een open-source Julia-pakket voor DNS en LES op verspringende Cartesische roosters. De solver gebruikt energiebehoudende eindige-volume-discretisaties met hardware-onafhankelijke kernels die zowel op CPU's als GPU's draaien, en is volledig differentieerbaar gemaakt door handgeschreven adjoint-kernels, waardoor a-posteriori training van neurale-netwerksluitings-modellen in de LES-solver mogelijk wordt.

De overkoepelende conclusie is dat voordat er LES-modellering kan plaats-vinden, het filter, het rooster, de numerieke fluxen en het discretisatieschema duidelijk moeten worden gedefinieerd. Sluitingsmodellen moeten worden ontworpen om te werken bovenop de impliciete dissipatie die al in het numerieke schema is opgenomen, zodat subfiltereffecten niet dubbel worden meegeteld.