Deep Learning-Assisted Modelling of Turbulence in Fluids

Aakash Patil

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Mines Paris - PSL

Modélisation de la Turbulence dans les Fluides Assistée par l'Apprentissage Profond

Deep Learning Assisted Modelling of Turbulence in Fluids

Soutenance le

Aakash PATIL

Le 1 Février, 2023

École doctorale n°364

Sciences Fondamentales et Appliquées

Spécialité

Mathématiques Numériques, Calcul Intensif, et Données

Composition du jury :

Gianluigi ROZZA Professor, SISSA Trieste, Italy	*Président*
Ricardo VINUESA Associate Professor, KTH Stockholm, Sweden	*Rapporteur*
Paola CINNELLA Professor, Sorbonne University, France	*Rapporteur*
Jonathan VIQUERAT Researcher, Mines Paris - PSL, France	*Examinateur*
Elie Hachem Professor, Mines Paris - PSL, France	*Directeur de thèse*

MINES PARIS | PSL

Dedicated to My Mother and Father
who continue to support and sacrifice their everything for me

Acknowledgements

I am deeply grateful to all those who supported me from my high-school years, when my interest in fluid mechanics and computer sciences began, to my Ph.D. completion, where I developed expertise in turbulence and deep learning.

First and foremost, I extend my heartfelt gratitude to my parents, Shri. Vijay Patil and Sau. Aruna Patil, for their unwavering support and encouragement. Their tireless efforts and sacrifices for my education and well-being are beyond measure. As I reflect upon their struggles, my own journey from bachelors to doctorate seems comparatively effortless. I cannot thank them enough for everything they have done for me.

I would like to express my heartfelt gratitude to my advisor, Elie Hachem, for providing me with everything I needed to succeed in my research. Right from the day I was hired, Elie ensured that I had state-of-the-art hardware, computing infrastructure, and a generous conference and travel budget. Under his guidance, I learned valuable skills such as leadership, confidence, independence, and the freedom to conduct research on my own terms. He instilled in me the importance of the socio-economic impact of research, which helped me ask the right questions. Throughout the challenging times of the COVID-19 pandemic, he checked on me like an elder brother and encouraged me to take days off to rest and recharge. I am grateful for his constant support, encouragement, and insistence on work-life balance.

Jonathan, my colleague and *unofficial advisor*, played a crucial role in my Ph.D. journey. I am grateful for his guidance in writing research papers and his validation of every small step. Without his support, this Ph.D. would not have been possible. He also helped to advocate for my ideas and at times even pushed my advisor, Elie, to support them. Special thanks to Florence, Aurelien, Philippe, and Franck for interesting discussions.

Thanks to Mrunmayee, the love of my life, for supporting my unconventional *atrangi* ideas of changing the world and for ensuring that my Ph.D. meets my expectations. She has been my best friend since my teenage years, and now, as my wife, her constant company and support have been invaluable to me. Also, my younger sister Mrunalini has been a constant support throughout my journey, always there for me like an older sister. She played a key role in keeping me sane during the challenging times and accompanied me throughout the writing of my thesis, validating my flow of ideas. I am grateful for her unwavering support and encouragement.

I would like to thank my dear *kutumb* best friends, Nikhil, Shivraj, Chaitanya, Manish, Srijith, Sakshi, and Parag for their life advice and constant support throughout my journey. I am also grateful to my master's and Ph.D. friends, Mohan, Akshay, Diana, Leo, Diego, Bipin, Bastien, Pierre-Alexandre, and Puneeth, for showing me the importance of work-life balance and enjoying life. I would like to express my heartfelt gratitude to my *family* in Antibes - Ashish sirji, Megha vahini, Nisarg, and Shrikant, for their care, engaging discussions, and for the best time we had in French Riviera. They were my constant source of encouragement and support, and I cherish the memories of our time together.

I am grateful to my colleagues at Mines ParisTech - CEMEF for making the Ph.D. journey enjoyable and memorable. I had the pleasure of working alongside Jonathan, Robin, Sofia, Gulia, Luca, and later Diego and Adrien. I am also thankful to Junfeng, Hassan, George, Victor, Franco, Joe, Wassim, Ramy, Ghanniya, Sasha, Juhi, and Prasanth for their company and good times. I would always remember our dicussions on a spectrum of topics from computing to economics to finance, and ofcourse food and culture.

I am grateful to Salunkhe sir for being my long-term mentor, who introduced me to research projects, grant writing, and guided me towards a career in research from a young age. During my master's in turbulence at Lille, I would like to thank Jean Phillpe for his mentorship and encouragement, Jean Marc for honing my experimental skills, and Thomas for giving me multiple opportunities and pushing me to explore Machine Learning for studying turbulence. Frank in Poitiers provided me with experimental access and helped me develop my skills in studying turbulence. I am also thankful to Corentin in Toulouse, who gave me my first exposure to French supercomputers and deep learning infrastructure.

I am grateful for the invaluable opportunities provided by the Lille Turbulence Program, CNRS-GDR Turbulence meetings, and the overall turbulence community in France. These opportunities allowed me to share my perspectives and defend my views on important topics such as reproducibility, replicability, the future of applied artificial intelligence, and the importance of giving it a test of time.

Finally, I am grateful to the gods and goddesses of all religions for providing opportunities and taking care of my family and friends during the challenging time of the Covid-19 pandemic. Pursuing a Ph.D. during the pandemic was not easy, especially while living far away from loved ones, but I feel fortunate to have received support from somewhere or something. Or, as some of my friends would say, perhaps I was just lucky ! I am grateful for the opportunities and blessings that have come my way, and I will always be humbled by the experience.

Table of Contents

List of Figures

Written and Oral Communications

The works presented in this manuscript have led or contributed to the following written communications.

- Aakash Patil, Jonathan Viquerat, and Elie Hachem. "Autoregressive Transformers for Data-Driven Spatio-Temporal Learning of Turbulent Flows." arXiv preprint arXiv:2209.08052 (2022).

- André Fuchs, Swapnil Kharche, Aakash Patil, Jan Friedrich, Matthias Wächter, and Joachim Peinke. "An open source package to perform basic and advanced statistical analysis of turbulence data and other complex systems." Physics of Fluids 34, no. 10 (2022): 101801.

- George El Haber, Jonathan Viquerat, Aurelien Larcher, David Ryckelynck, Jose Alves, Aakash Patil, and Elie Hachem. "Deep learning model to assist multiphysics conjugate problems." Physics of Fluids 34, no. 1 (2022): 015131.

- Aakash Patil, Jonathan Viquerat, Aurélien Larcher, George El Haber, and Elie Hachem. "Robust deep learning for emulating turbulent viscosities." Physics of Fluids 33, no. 10 (2021): 105118.

- Aakash Patil. "Development of Deep Learning Methods for Inflow Turbulence Generation." arXiv e-prints (2019): arXiv-1910.

The works presented in this manuscript have led to the following oral communications.

- "Subgrid Scale Enhancements Assisted by Deep Learning", 2nd Workshop on Artificial Intelligence in Plasma Science, Aix Marseille University, July 2022

- "Deep Learning Assisted Predictive Modelling of Turbulent Flows", The Orsay Mechanics Seminar - LIMSI-FAST-LISN, University of Paris-Saclay, April 2022

- "On The Limits of Deep Learning For Super-Resolution and Reconstruction of Turbulence", CNRS GDR Navier Stokes 2.00 - University of Paris-Saclay, October 2021.
- "Towards Generalized and Robust Deep Learning For Turbulent Flows", IACM MMLDT-CSET Conference, San Diego, September 27, 2021

Disclaimer

The content described in Chapter 3 - Robust Learning for Turbulent Flows includes material reprinted from [1] with the permission of AIP Publishing, post-publication in Physics of Fluids. Reprint permission or license number 5443810560242 issued with the terms and conditions provided by AIP Publishing and Copyright Clearance Center.

The content described in Chapter 2 - Theoretical Background includes material from [2] and [3] and has been modified for giving readers a detailed overview of preliminaries required in the later chapters of this work. The original authors retain full ownership and copyright of the original material.

1 Patil, A., Viquerat, J., Larcher, A., El Haber, G., & Hachem, E. (2021). Robust deep learning for emulating turbulent viscosities. Physics of Fluids, 33(10), 105118.

2 Kharche, S. P. (2021). High Reynolds turbulence: Study of the inertial intermittency in axisymmetric jet and von Kármán cryogenic flows. Université Grenoble Alpes.

3 Higham, C. F., & Higham, D. J. (2018). Deep Learning: An Introduction for Applied Mathematicians. arXiv preprint arXiv:1801.05894.

Chapter 1

Introduction

La turbulence est un phénomène fondamental et omniprésent dans de nombreux écoulements naturels et industriels, et son étude constitue un défi de longue date en mécanique des fluides. Si la turbulence à grande échelle a été largement étudiée, la compréhension de la turbulence à petite échelle reste un défi important en raison de sa nature complexe et multiforme. La turbulence à petite échelle joue un rôle crucial dans diverses applications, telles que le mélange turbulent, la combustion et le transfert de chaleur. Ce chapitre souligne l'importance des études sur la turbulence, en particulier les défis associés à la turbulence à petite échelle, et introduit les concepts et termes clés qui sont pertinents pour la recherche présentée dans la thèse. En identifiant les lacunes de connaissances dans le domaine et en formulant les défis de recherche, cette thèse vise à contribuer aux efforts en cours pour améliorer notre compréhension et notre modélisation de la turbulence.

1.1 Fluid Mechanics and Turbulence

Whether we look inside ourselves - the flow of blood in our veins and arteries - or around us - the air and sea currents - or when we look far away - the stellar plasma, the clouds of nourishing gases of the young stars in formation - we can see the effect of a collection of the laws of nature governing fluids from various sources. These laws which we humans have tried to write through the mathematics formalisms, the equations we have deduced from our observations, experiences, and logical reasonings, form the science of Fluid Mechanics. The term mechanics refers to the study of the dynamics of a system in a medium with respect to a reference frame; the medium in this case is the fluid. The definition encompasses any perfectly deformable material medium, such as liquids, gases, and plasmas. This science, built over centuries by renowned engineers, physicists, and mathematicians such as Euler, Navier, Stokes, Reynolds, and Kolmogorov, has not yet revealed all its secrets. The Navier-Stokes equations governing fluid dynamics were established around the year 1820-1823, and since then, they have resisted the most brilliant brains, so that the proof of the existence of a general solution has still not been established, let alone its uniqueness. The year of writing this thesis marks 200 years of research work around these equations. This same spirit constitutes an unlimited field of experimental research, to try to understand, reproduce, and adapt some mechanisms in order to answer the current problems: for example to understand the roles of the ocean currents governing the thermodynamic equilibrium between the ocean and the atmosphere and the associated meteorological and ecological phenomena.

The theoretical and experimental study of fluid mechanics is based on certain hypotheses that are the basis of its formulation which is part of a more general framework: the mechanics of continuous media. In order to define a continuous medium, we must first dive into the molecular structure of fluids, and consider them at the microscopic scale. We would then see the matter as it is: in the case of a gas, we would see a great number of molecules in agitation which are separated from each other by empty spaces far exceeding the size of the molecule itself. In the case of a liquid, the molecules are much closer, as far as the repulsive forces between the molecules allow it, and the distribution of the mass in the volume would be strongly affected by the mass of the particles within their nucleus. At this scale, it is necessary to take into account the non-uniform distribution of masses and molecules to model this reality, this is the objective of statistical mechanics and Boltzmann equations. In contrast, the mechanics of continuous media is placed at the macroscopic scale, scales larger than several times the size of the molecules themselves, considering only observable variables measurable at these scales. At this scale, nature appears as a continuum in which observable variables are defined for each portions of the fluid. These fluid portions are sometimes referred to as fluid eddies associated with the swirling fluids. The levels of description and their associated formalisms are represented in the figure 1.1.

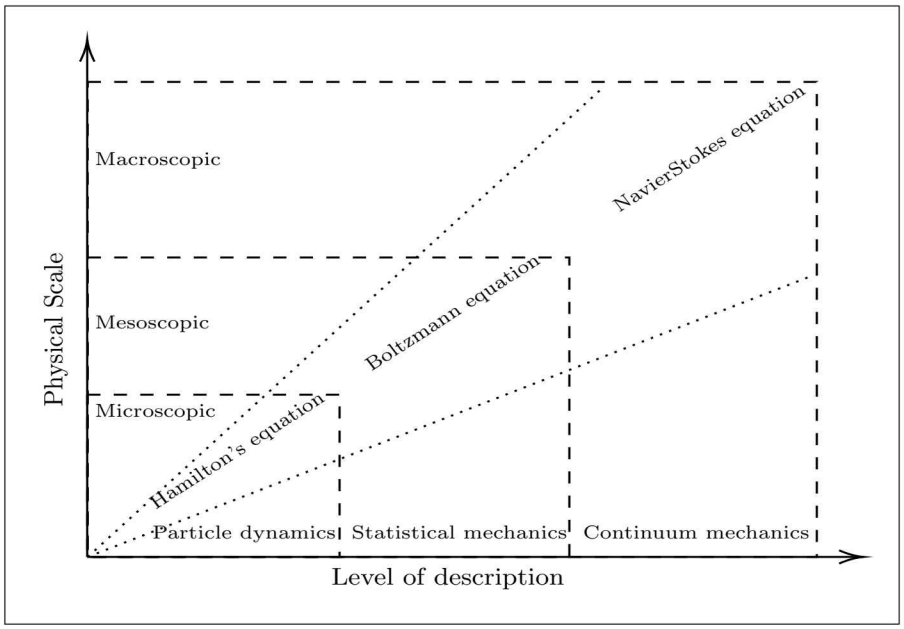


Figure 1.1: Different levels of formalisms of mechanics description and the formalisms established to study them. Reproduced from Mattila [2010]

In 1883, Osborne Reynolds experimentally studied the regimes of fluids. He found that the more the inertial effects are dominant compared to viscous effects, the less the fluid is able to absorb the fluctuations and instabilities that could appear in the fluid movement . He developed a dimensionless number, which now bears his name, with which he characterized three different regimes. For a measured Reynolds number of less than a few thousand, the flow is called laminar flow. It is characterized by a regular topography of observable flow features. It is stable, and its streamlines point in the same direction, each parallel to the other. For higher Reynolds numbers, up to ten thousand, this topography becomes less and less regular and instabilities start to be visible. In this range of Reynolds, the fluid remains able to contain the instabilities, without dissipating them. This regime is referred to as transient flow, since generally the instabilities are either absorbed by the fluid or diffused in the rest of the flow by completely modifying it. From the tens of thousands and beyond, the instabilities in the velocity field are no longer absorbed, and generate vortex structures of varying sizes, the appearance of these structures being the mark of the turbulent regime. Leonardo da Vinci, the famous scientist, artist and engineer, captured the essence of the nature of turbulence in fluids through systematic observation and combined his keen observational skills with exceptional artistic talent to catalog turbulent flow phenomena back in 1509 as depicted in figure 1.2

Without going into the mathematical details that will be discussed in the later chapters, we summarize the Reynolds number, noted by Re as :

$$Re = \frac{\text{Inertial Effects}}{\text{Viscous Effects}}$$

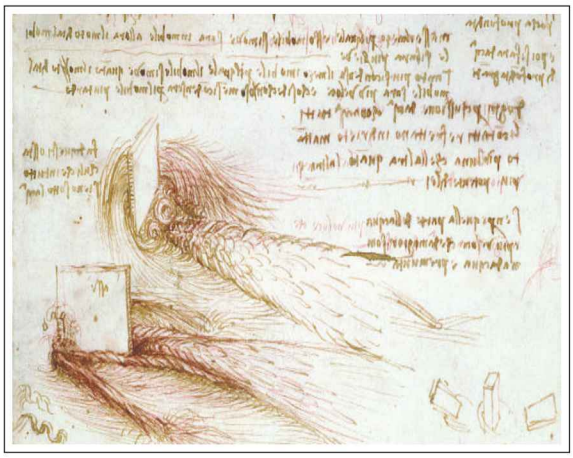


Figure 1.2: Painting by Leonardo da Vinci illustrates turbulence in his studies of wake turbulence, the development of a region of chaotic flow as water streams past an obstacle. Courtesy: Google Images

It means that when the inertial effects dominate the viscous effects, the fluid motion belongs to the high-Reynolds range. Historically, after Reynolds, it was Richardson who first conceptualized turbulence in the modern sense. In the 1920s, while laying the foundations of modern meteorology, he defined what we call today turbulence as the composition of vortex structures of different sizes, and even wrote a poem about it:

> Big whorls have little whorls
> Which feed on their velocity,
> And little whorls have lesser whorls
> And so on to viscosity

Here he introduces several fundamental notions: first of all the fact that large vortices contain potentially smaller vortices which are strengthened by the distribution of velocity, in a sense the kinetic energy, from this large structure to the smaller ones. In his words, he also introduces the transfer of energy without loss, from structures to structures, until the dissipation of this kinetic energy by molecular friction affects characterizing the viscosity of the fluid. Finally, through the idea of kinetic energy carried by each scale, Richardson allows us to consider the most natural way to study turbulence: using Fourier space which allows a representation of the energetic cascade describing the scale by scale cascade of energy, as theorized in Richardson [1922]. Figure 1.3 shows a representation of scale by scale turbulence cascade.

In 1941, Andrey Kolmogorov, one of the most fascinating minds of the 20^{th} century, deepened and perfected Richardson's theory of the energy cascade. In doing so, he laid the basis for the analytical, experimental, and numerical study of turbulent flows. He established that the geometrical constraints and the possible properties of the flow media only impact the large scales. Indeed, the boundary

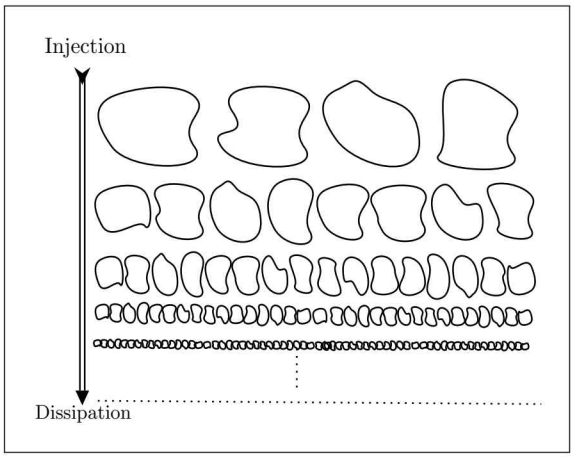


Figure 1.3: Schematic of scale-by-scale turbulence energy cascade showing larger scales. Reproduced from Richardson [1922]

conditions of the domain result in the generation of large-scale structures whose size is often of the order of magnitude imposed by the domain in which the fluid evolves. These scales break up by interacting with each other, and produce smaller scales that follows the cascade process in a range of scales where the energy conveyed is written universally with the law $E(k) \approx k^{-5/3}$. These scales constitute the inertial range of the energetic cascade. This process continues up to the so-called Kolmogorov scale noted η, universally defined by the rate of kinetic energy injection at large scales whose causes are external to the fluid, and the dissipation of the fluid under study. Thus, it is possible to know exactly the order of magnitude of the dissipative scales even before the cascade is set up. Beyond this scale, other scales are created but the energy is generated drastically as depicted in figure 1.4. Moreover, since the energy transfer within the inertial range is lossless, Kolmogorov and Richardson assume that the rate of dissipation of kinetic energy from the dissipative scales denoted ϵ is exactly the rate of injection of the same energy at the large scales. In order of magnitude, Kolmogorov [1941] writes

$$\eta = \left(\frac{\nu^3}{\epsilon}\right)^{1/4}$$

Since then, many efforts have been made to understand the physics of turbulence, and the Richardson-Kolmogorov approaches remain the reference. With the rise of the power of computers available in laboratories, and even more so those of supercomputers, the computational simulation of fluids has become an essential tool accompanying fundamental research. We primarily distinguish two types of simulations: the first one proposes to solve the Navier-Stokes equations from first principles on all spatial scales up to the Kolmogorov scale, while the other proposes that a part of the physics is not solved but modeled from relations built by experimental and empirical considerations. The first principle simulation

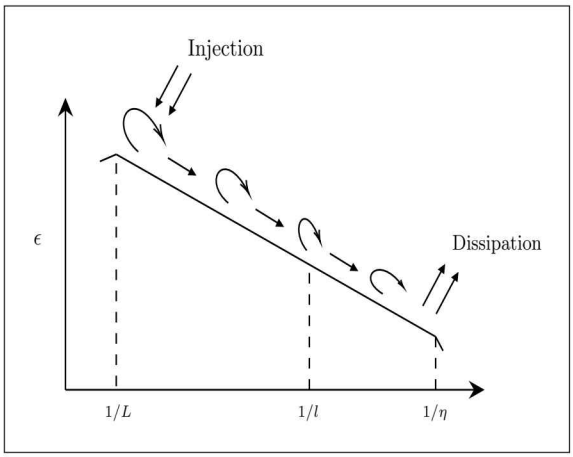


Figure 1.4: Schematic of scale-by-scale turbulence energy spectrum showing injection and dissipation at smaller and larger scales. Inspired from Kolmogorov [1941]

of turbulent flows is often referred to as a direct numerical simulation (DNS). To ensure simulation up to Kolomogorv scales, the DNS requires a very fine spatio-temporal discretization of the study domain and the number of discrete points needed is of the order of $Re^{9/4}$. Physically, this excessive number is explained by the fact that the higher the Reynolds number, the lesser the capacity of the fluid to dissipate the instabilities, thus pushing the inertial zone towards the very small scales. The dissipative scales are then even smaller, and the discrete step of the domain must necessarily be smaller than these scales. We understand that despite the technological advancements and the capabilities of the best computers, the possibility to compute an accurate simulation of a real flow, whose Reynolds is of the order of a million for the flow of air in an open area, is limited. For this reason, researchers have developed new approaches, in which a part of the physics is not solved, but modeled from relations built by experimental observations and empirical considerations. The results from such models, will therefore be less reliable than with DNS but, on the other hand, simulations with higher Reynolds can converge in reasonable times on standard computational resources. Let us summarize of the most commonly used approaches:

RANS The so-called averaged approach consists in decomposing all the variables appearing in the Navier-Stokes equations into a sum of a mean component and the fluctuation around this mean. Thus, these procedures solve the averaged Navier-Stokes equations or Reynolds Averaged Navier-Stokes (RANS), and model the amount of motion that the fluctuating field injects into the mean field. This quantity is called the Reynolds Tensor.

LES The large-scale motions are computed from the filtered unsteady Navier-Stokes equations, while the small-scale ones are modeled with *ad hoc* procedures. Then, mean flow parameters are obtained from statistics over a set of solved large-scale motions, and small-scale ones referred as sub-grid scale (SGS) modeling. This is referred to as Large Eddy Simulation (LES). This so-called filtered

approach consists in decomposing all the variables appearing in the Navier-Stokes equations into a sum of a filtered component and a residual component. Similar to the RANS models, these procedures solve the filtered fluid equations and model the interactions between the filtered, so-called resolved, and residual, unresolved, components through the stress tensor under the mesh.

DNS Any fluctuating motion in the flow fields is computed from the exact unsteady Navier-Stokes equations, that is, a DNS of a turbulent flow is a time-dependent and three-dimensional numerical solution in which the flow equations are computed as accurately as possible without any turbulence model introduced. Mean flow parameters are later obtained from statistics over a broad set of numerical solutions. DNS considers all degrees of freedom appearing in the flow without using any approximating assumption. However, as the required grid points number increases faster than the square of the Reynolds number, DNS of turbulence is at present feasible only at low or moderate Reynolds numbers.

In these RANS, LES, and similar approaches, the modeling of the fluctuating or residual terms is necessary. This modeling *closes* the system composed of the simplified equations and the conservation of mass equation from continuum mechanics. The closure of this system simply consists in providing an additional relation linking the unknown quantities, specific to the chosen model, with the quantities available in the framework. Without this closure, it is impossible to solve the system. The challenge is to establish this additional equation and it is referred as the closure problem.

While these models are developing and becoming more complex, a huge amount of simulation data considered exact has been generated by DNS and experiments conducted for decades. At the same time, data assimilation and artificial intelligence algorithms have allowed the development of new research domains, and to revisit the previous works. Supervised data-driven algorithm methods consist essentially of an optimization problem where the goal is to adjust a set of parameters W to be able to model one or several outputs Y from a corpus of data X provided to the algorithm. The reliability of this modeling is evaluated according to an objective function and to one or more additional metrics allowing to obtain a global understanding of the quality of the output provided. Artificial intelligence (AI) is based on the same principle, except that it is necessary that this optimization is general enough to be applied to new cases without modifying the previously established parameters. In this context, the optimization of these parameters is called learning or training. The learning must allow the AI to predict with an efficiency close to the one evaluated at the end of the training.

1.2 Artificial Intelligence and Deep Learning

In the twenty-first century, machine learning is becoming a part of our everyday life. The great advances in artificial intelligence make it possible to perform tasks that were a dream in the previous century. For example, it is possible to automatically create animation or living portraits from still photographs as

Source Generated images

Figure 1.5: Bringing still photographs to life using for one-shot models learned from photographs in the source column. Courtesy Zakharov, Shysheya, Burkov, and Lempitsky [2019]

Input vs Original Generated images

Figure 1.6: Model-generated completions of human-provided half-images in input column, compared with the original images. Courtesy Chen, Radford, Child, Wu, Jun, Luan, and Sutskever [2020]

shown in 1.5, and obtain automatic suggestions for missing image parts as shown in figure 1.6.

While Arthur Clarke, a science fiction author best known for *2001: A Space Odyssey*, wrote during the 1960s that sufficiently advanced technologies would be indistinguishable from magic, this is not really the case considering the current state-of-the-art knowledge: broadly, the AI methods involved are understandable in terms of linear algebra, optimization, and statistics. In the engineering sciences, the wealth of data available has also revolutionized the way we model, predict, and control dynamic systems such as turbulence, brain activity, climate, finance, and robotics. These complex systems may not always be modelable empirically mainly because of their nonlinear nature and multi-scale behavior in time and space. However, in order to measure, predict, estimate, or control these systems, it is essential to provide characterization. Inspired by biology, we understand that the knowledge of the system does not need to be complete: just think of the eagle that performs complex manoeuvres in a perturbed environment without being aware of the Navier-Stokes equations or the three-dimensional velocity field. By formalizing this reflex learning from experiments, researchers have been able to provide answers for turbulence control, optimal placement of sensors and actuators, identification of dynamic models, and dimension reduction. Data science is driven by innovations in virtual data storage, transfer capabili-

Figure 1.7: Natural-color image of atmospheric flow past the Isla Socorro off the western coast Mexico captured by Aqua satellite . The dynamics, however complex, shows structures similar to the configuration of the of the flow behind a cylinder. Image courtesy: Jeff Schmaltz LANCE/EOSDIS MODIS Team at NASA GSFC

ties, available computational hardware, and low sensor costs as introduced as the fourth paradigm of scientific discovery by Hey et al. [2009] . Today's researchers have tools to compute consistent models that generalize quantitatively or qualitatively to unmeasured portions of the state, application, or parameter space. However, the success of the methods is only proven in academic systems such as the Duffing oscillator, the Lorenz 63 system or the wake of a 2D cylinder with low Reynolds number. For scaling up to more complex systems, one must first reduce the number of degrees of freedom and counter the curse of dimension. The goal is to extract coherent structures in the system, which are low ranked in trends.

In fluid mechanics, nature confirms the existence of patterns and structures, for example as illustrated in figure 1.7 with satellite captured natural-color image of cloud vortices behind Isla Socorro, a volcanic island in the Pacific Ocean where the flow of clouds in the wake of a island is linked to the flow behind a low Reynolds cylinder. The movie of the flow can then be rewritten - rather than scrolling images defined by millions or even billions of pixels, it is a matter of scrolling a few images characterized by modes whose contribution changes over time. The existence of particular structures in the data is the entry point to the use of data science and associated data-driven tools. The extraction of such modes echoes the efforts of mathematicians to simplify a system by transforming the coordinates. The history of these transformations is rich, with analytical developments such as spectral decomposition, Fourier transformation or generalized functions. The consideration of data gives a new impetus to new methods, for example principal component analysis or proper orthogonal decomposition. This non-intrusive reduction method allows to extract the decorelated energy contributions of maximum variance from a point cloud. By restricting to the first few high-energy contributions, most of the variability is restored while the

low-energy contributions are forgotten, thus providing a low-rank approximation of the system. Another example concerns the Koopman operator, introduced in 1931, which allows to reformulate a finite-dimensional nonlinear system into an infinite-dimensional linear system. If the linearity property is desirable, the infinite dimension of the space of observables makes the treatment of application difficult. Today, the richness of the data changes the situation, as it allows the definition of finite approximations of the operator for many applications in control. The dynamic mode decomposition is an example, massively used nowadays for post-processing of physical data or video processing. With these few examples, data science shows high potential for expanding the understanding of physical laws and thereby better engineering systems. However, we must not let these tools become black boxes and keep a constant link with physics. The models learned must be robust, generalizable, and interpretable. Although debatable, an interpretable and generalizable model would mathematically translate into a low-dimensional and robust model. Despite the growing potential of deep learning for complex dynamical systems, humans should always apply the tools intelligently and with due diligence by making things as simple as possible, but not simpler.

There are several forms of artificial intelligence methods, the most known of which is the artificial neural network. There are several types of neural networks : the classical neural networks consist in a succession of nonlinear transformations of the inputs made by computational units called neurons, in order to predict the expected outputs. The outputs of each neuron are weighted by parameters called weights which are optimized during the learning process to minimize the errors made during the prediction. There are convolutional neural networks that, contrary to the previous category, can be applied to data with a spatial dimension for example an image or a volume. These architectures are constantly improving, and their applications are becoming more and more varied. In view of the excellent results of these methods, their increasing popularity and accessibility, many researchers have worked on the coupling of data-driven methods and AI with the study of turbulence, in particular to develop RANS or LES models augmented by data. The general idea is the following: data from DNS or experiments contain all the information needed to improve existing models, which are intrinsically deficient in the physics of turbulence. Researchers substitute the assumptions made during the establishment of the models with an injection of information from accurate data. Some works try to adjust the parameters of the pre-existing models, others the modeled fields themselves, and more rarely, to predict the field in question without modeling it.

1.3 Thesis Organization and Contributions

The thesis is organized as follows:

Chapter 1: *Introduction*
This chapter provides the research background and challenges formulated based on the knowledge gap by shortly reviewing the importance of turbulence studies, especially the challenging problem of understanding and modeling small-scale turbulence, and introduces the reader to key terms associated with the thesis.

Chapter 2: *Theoretical Background*
This chapter presents a comprehensive review of the background on turbulence in fluids and deep learning fundamentals. Additionally, literature based on studies of fluid mechanics and turbulence with deep learning formalisms is provided. The chapter also highlights the type of deep learning model for spatio-temporal problems, typically seen in science and engineering research.

Chapter 3: *Robust Learning for Turbulent Flows*
This chapter presents a robust strategy using patch-based training to learn turbulent viscosity from flow velocities and demonstrates its efficient use on a popular turbulence model. Training datasets are generated for flow past obstacles at high Reynolds numbers and used to train an auto-encoder type convolutional neural network with local patch inputs.

Chapter 4: *Learning Subgrid-scale Turbulence*
This chapter presents the learning of spatial subgrid scales and their enhancements in the context of turbulent flows. Wall-bounded turbulence flow at a high-Reynolds number is studied for enhancement of subgrid-scale turbulence.

Chapter 5: *Auto-Regressive Learning of Spatio-Temporal Turbulence*
This chapter presents a convolutional encoder-decoder-based transformer model developed to auto-regressively train on spatiotemporal data of turbulent flows. The demonstration of model applicability is performed on turbulent wake flow past an obstacle and environmental flow past surface mounted obstacle.

Chapter 6: *Conclusion and Perspectives*
This chapter presents the major conclusions of the thesis, key contributions to the knowledge, and limitations. Lastly, the thesis concludes by discussing potential future projects and prospectives.

Bibliography

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020.

Anthony JG Hey, Stewart Tansley, Kristin Michele Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

Andrey Nikolaevich Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. *Cr Acad. Sci. URSS*, 30:301–305, 1941.

Keijo Mattila. Implementation techniques for the lattice boltzmann method. *Jyväskylä studies in computing*, (117), 2010.

Lewis F Richardson. *Weather prediction by numerical process*. University Press, 1922.

Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9459–9468, 2019.

Chapter 2

Theoretical Background

La turbulence dans les fluides est un phénomène complexe qui a été largement étudié en mécanique des fluides. Ce chapitre présente un examen complet du contexte de la turbulence et des principes fondamentaux de l'apprentissage profond, qui sont de plus en plus appliqués aux études sur la turbulence. La littérature sur la mécanique des fluides et la turbulence avec des formalismes d'apprentissage profond est également discutée, donnant un aperçu des progrès et des défis dans ce domaine. Le chapitre met également en évidence les types de modèles d'apprentissage profond qui sont généralement utilisés pour les problèmes spatio-temporels dans la recherche en sciences et en ingénierie. Comprendre le potentiel de l'apprentissage profond dans les études sur la turbulence peut conduire à de nouvelles connaissances et à de meilleures prédictions dans une variété d'applications de la dynamique des fluides.

2.1 Turbulence in Fluids

2.1.1 Navier-Stokes to Kolmogorov

The Navier-Stokes equations govern the physics of all viscous, incompressible fluid flows, both laminar and turbulent are written as:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2}. \tag{2.1}$$

For simplicity of this section, the motion of a fluid at position $\boldsymbol{x}$ in the flow field and at time t is given by :

$$\rho\left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u}\right] = -\nabla P + \mu\nabla^2\boldsymbol{u}, \tag{2.2}$$

where ρ is the fluid density, $\boldsymbol{u}$ is the flow velocity vector field, ∇ is the gradient operator, P is the pressure, μ is the dynamic viscosity of the fluid and ∇^2 is the Laplacian operator. The Navier-Stokes equation is a time-dependent, non-linear, second-order partial differential equation that models fluid flow. Analytical solutions for this equation are generally unattainable except for the most straightforward flow fields. Equation 2.2 expresses four unknowns, which are the three velocity components and the pressure, yet it only represents three equations (one for each component). The continuity equation, which represents the conservation of mass in incompressible fluid flow, provides the fourth equation and can be expressed as:

$$\begin{aligned} \frac{\partial u_i}{\partial x_i} &= 0 \\ \nabla\cdot\boldsymbol{u} &= 0. \end{aligned} \tag{2.3}$$

Let us introduce a characteristic velocity U and length scale L of the fluid flow. In equation 2.2, the term $(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}$ embodies the effects of inertia by means of advection, for which a typical time scale is $\tau_{\text{iner}} \sim \frac{L}{U}$. The term $\mu\nabla^2\boldsymbol{u}$ describes the diffusion of momentum (*i.e.* viscous/frictional effects) as a consequence of finite viscosity of the fluid, for which a typical time scale is $\tau_{visc} \sim \frac{L^2\rho}{\mu}$. Using the inertial and viscous time scales, a fluid flow can be characterized with a dimensionless parameter called the Reynolds number Re, given by:

$$\frac{1}{Re} \sim \frac{\tau_{\text{iner}}}{\tau_{\text{visc}}} \sim \frac{\mu}{\rho U L} \sim \frac{\nu}{UL}, \tag{2.4}$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity of the fluid.

The Navier-Stokes equation's non-linearity arises due to the convective acceleration, making its solution challenging. When the Reynolds number is very low, the nonlinear terms become negligible, leading to laminar flow. However, increasing the Reynolds number results in a chaotic motion of fluid, leading to numerous bifurcations and instabilities in the flow. Notably, the solution of the Navier-Stokes equation is among the seven millennium problems proclaimed by

the Clay Mathematics Institute Fefferman [2000]. Due to its sensitivity to initial and boundary conditions, such as vanishing tangential velocity at confining walls, the Navier-Stokes equation is both non-predictable and non-deterministic.

The study of turbulence has a long history, but the development of theoretical models to describe it was impeded by the requirement for a statistical probabilistic approach. Only after centuries of effort were relatively simple models established. One of the most notable definitions of turbulence was proposed by Richardson Richardson [1922]. His description posits that turbulent flow consists of eddies or scales of varying sizes r. The Richardson cascade reflects the idea that at steady state, the energy is injected into the fluid at large scales ($r \approx L$) and is transferred towards smaller scales until the dissipation takes place at the molecular scales. Let us consider a turbulent flow of characteristic velocity U and characteristic length scale L. From the dimensional analysis it follows that the rate of injection of energy $E_I \approx \frac{U^3}{L}$. The Richardson cascade states that this rate of injection of energy must be equal to the rate of transfer of energy across scales which is eventually equal to the mean dissipation rate $\langle\epsilon\rangle$ at molecular scales, where $\langle\cdot\rangle$ denotes the ensemble average of the desired quantity. Therefore, the Richardson cascade implies that $\langle\epsilon\rangle \approx \frac{U^3}{L}$.

During the 1930s, Taylor Taylor [1935] pioneered the application of statistical methods to analyze homogeneous isotropic turbulence, which laid the foundation for understanding turbulence. Subsequently, Kolmogorov Kolmogorov [1941] proposed the K41 theory in 1941, which presents three hypotheses to theoretically describe statistically stationary and homogeneous turbulence. These hypotheses apply to turbulent flows with extremely high Reynolds numbers, defined as $Re = \frac{UL}{\nu}$.

According to Kolmogorov's hypothesis of local isotropy, the statistical behavior of the small-scale turbulent eddies ($r \ll L$) in a turbulent flow with a sufficiently high Reynolds number is isotropic. Kolmogorov's first similarity hypothesis asserts that the statistical properties of small scales in a turbulent flow are solely determined by the kinematic viscosity of the fluid and the mean dissipation rate. Based on these two parameters $\langle\epsilon\rangle$ and ν, the following Kolmogorov quantities are defined:

$$\eta = \left[\frac{\nu^3}{\langle\epsilon\rangle}\right]^{1/4}, \tag{2.5}$$

$$u_\eta = [\langle\epsilon\rangle\nu]^{1/4}, \tag{2.6}$$

$$\tau_\eta = \left[\frac{\nu}{\langle\epsilon\rangle}\right]^{1/2}, \tag{2.7}$$

where η, τ_η and u_η are the Kolmogorov scales corresponding to the space, time and velocity respectively that correspond to the smallest dissipation scales in the turbulent flow. For each scale r in the turbulent cascade, we can attribute a local Reynolds number such as: $Re_r = \frac{U_r r}{\nu}$, where U_r is the characteristic velocity

of scale r. Using the characteristic length and velocity from equation (2.5) and equation (2.5), we obtain the local Reynolds number of unity. This signifies that the energy cascade continues until the smallest scale at which the local Reynolds number (*i.e.* $Re_\eta = \frac{u_\eta \eta}{\nu}$) is unity (*i.e.* inertial forces are equal to viscous forces) where the viscous dissipation is totally dominant.

Kolmogorov's second similarity hypothesis states that in a turbulent flow of high Reynolds number, the statistics of scale r in the range $\eta \ll r \ll L$ have universal form independent of viscosity, which is only determined from $\langle\epsilon\rangle$. Based on these hypotheses, we can establish the common notion of scales in the turbulent flow. The energy is injected into the flow at large scales (say up to $r \approx L$). The transfer of energy occurs in the range of scales $\eta \ll r \ll L$ which is referred as the inertial range because of the dominant inertial effects of the large eddies. After the inertial range, the energy cascade continues in the dissipation range where the viscous effects are dominant at small scales. By definition, the averaged dissipation rate is determined by:

$$\langle\epsilon\rangle = \frac{\nu}{2} \sum_{i,j=1}^{3} \left\langle \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)^2 \right\rangle, \tag{2.8}$$

where u is the fluctuating component of velocity (later used in the turbulent channel flow subgrid scale learning in Chapter 4). Expanding the above equation for the case of homogeneous isotropic turbulence, the average dissipation rate is given by :

$$\langle\epsilon\rangle = 15\nu \left\langle \left(\frac{\partial u}{\partial x} \right)^2 \right\rangle. \tag{2.9}$$

Let us define the longitudinal velocity increment at scale r which is given by: $u_r = u(x+r) - u(x)$. The structure function of order p is defined as $S_p(r) = \langle u_r^p \rangle$. From the Kolmogorov's second similarity hypothesis it follows that:

$$S_2(r) \propto \langle\epsilon\rangle^{\frac{2}{3}} r^{\frac{2}{3}}, \tag{2.10}$$

where the equality can be imposed using the universal constant $C_2 = 2.0 - 2.4$ which was mentioned by Sreenivasan [1995]. From the second similarity hypothesis of Kolmogorov, it follows that the energy spectrum in the inertial range is given by:

$$E(k) = C\langle\epsilon\rangle^{\frac{2}{3}} k^{-\frac{2}{3}-1} = C\langle\epsilon\rangle^{\frac{2}{3}} k^{-\frac{5}{3}}, \tag{2.11}$$

where $k = \frac{2\pi}{r}$ is the wavenumber in the longitudinal direction and $C = 0.25C_2$ is the universal constant as described by Frisch and Kolmogorov. The above equation is a mere consequence of the Fourier transform of equation (2.10). Applying the same phenomenology of Kolmogorov to higher order structure functions gives:

$$S_p(r) = C_p(\langle\epsilon\rangle \cdot r)^{\frac{p}{3}} \sim r^{\zeta_p}, \tag{2.12}$$

where C_p depends on the nature of turbulent flow and ζ_p is the scaling exponent associated with p^{th} order structure function. When the structure functions behave according to equation (2.12), it is refered as a scale invariance property of the flow. Equation (2.12) implies that the distribution of velocity increments at a scale r in the inertial range has the same form *i.e.* it is self-similar.

Kolmogorov's hypothesis suggests that ζ_p varies linearly with respect to p such that:

$$\zeta_p = \frac{p}{3}. \tag{2.13}$$

According to K41 model given by the above equation, it implies that the higher order moments of the distribution of velocity increments, such as the flatness, remains constant across scales. To have a more accurate estimation of ζ_p, a large inertial range needs to exist where the power law holds i.e. $\frac{L}{\eta} >> 1$.

In the turbulent flow, the dissipation is non-uniformly distributed in space and scale. Typically, the statistics of turbulent flow are measured at a single point. Thus, Taylor hypothesis is used to calculate the structure functions in order to estimate the dissipation rate in the homogeneous isotropic turbulence. De Karman and Howarth [1938] worked out the derivation of the mean dissipation rate from the structure function given by:

$$S_3(r) = -\frac{4}{5}\langle\epsilon\rangle r + 6\nu\frac{dS_2(r)}{dr}. \tag{2.14}$$

The negative sign in the equation (2.14) is the consequence of the transfer of average energy mainly (or predominantly) towards the smallest scales, *i.e.* an irreversible process. Because of the negligible viscous effects in the inertial range, the contribution of the term $6\nu\frac{dS_2(r)}{dr}$ may be neglected. Therefore, neglecting the viscous contribution in equation (2.14), we get the popular relation by Kolmogorov [1941]:

$$S_3(r) \approx -\frac{4}{5}\langle\epsilon\rangle r. \tag{2.15}$$

Equation (2.15) is populalry known as Kolmogorov four-fifths law (where $C_3 = -\frac{4}{5}$), which is one of the most important and exact result, though it requires statistical stationarity, homogeneity, and isotropy of the turbulent flows. Reader is referred to the works of Landau and Lifshitz [2013], Pope [2001], Tennekes et al. [1972] and Frisch and Kolmogorov for a detailed course on turbulence.

2.1.2 The closure problem and modelling

Theoretically, the Navier-Stokes equation is deterministic, which means that if proper initial and boundary conditions are specified, equation (2.1) can very well describe the evolution of dependent variables. But the non-linearity of this equation makes it highly sensitive such that many realizations are possible for an infinitesimal change in flow conditions. Also, to resolve all the scales in a

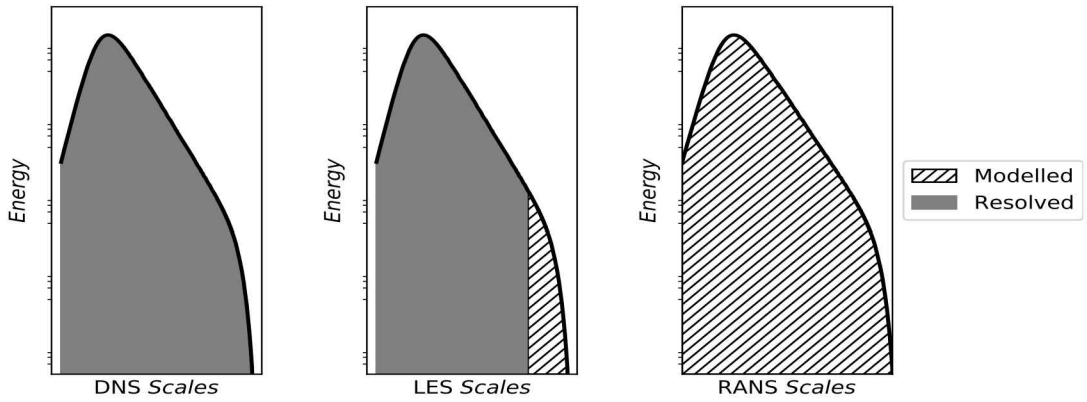


Figure 2.1: Representation of resolved and modelled scales of turbulence for DNS, LES, and RANS approaches.

turbulent flow, equation (2.1), must be discretized with very large number of computational points, which again increases as the Reynolds number increases. Therefore, numerically solving full Navier-Stokes equation for practical turbulent flows with realistic Reynolds numbers remains infeasible.

One way to avoid the problem of solving the full NS equations is by using the statistical approach which can be achieved with the Reynold's decomposition, wherein a stochastic field, for example u, is decomposed into its mean $\bar{u}$ and fluctuating component u' such that:

$$u = \bar{u} + u'.$$

Using the same decomposition method for the equation (2.1), and taking an average over time, we obtain the Reynolds Averaged Navier-Stokes (RANS) equation, which govern the mean flow:

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0,$$

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{u}_i}{\partial x_j^2} - \frac{\partial}{\partial x_j}\overline{(u_i' u_j')}. \tag{2.16}$$

The RANS equations are similar to the Navier-Stokes equations, except for the last term $\overline{(u_i' u_j')}$ which is called as the Reynold stress, and is a direct consequence of the non-linearity in the system. The RANS system is an unclosed set of equations, as it has more number of independent variables than the number of equations. This is the *closure problem* in turbulence. RANS modeling consists in closing this set of equations by obtaining substitutive equations for $\overline{(u_i' u_j')}$ in the terms of mean flow quantities, by removing any reference to the fluctuating components, that would help to obtain the equations containing *only* the mean flow quantities.

Similarly, another way to avoid the problem of solving the full Navier-Stokes equations is by spatio-temporally filtering the equations (2.1). Filtering operation

involves decomposing the velocity into sum of filtered and residual components which means,

$$u_i(x_i,t) = \underbrace{\widetilde{u_i}(x_i,t)}_{filtered} + \underbrace{u'(x_i,t)}_{residual}. \tag{2.17}$$

Note that filtered components are also mentioned as resolved components and the residual components are also mentioned as subgrid-scale (SGS) components. For LES, the continuity and momentum conservation equations are obtained by filtering the Navier-Stokes equations and written as:

$$\frac{\partial \widetilde{u_i}}{\partial x_i} = 0, \tag{2.18}$$

$$\frac{\partial\left(\widetilde{u_i}\right)}{\partial t} + \frac{\partial\left(\widetilde{u_i}\widetilde{u_j}\right)}{\partial x_j} = -\frac{\partial \widetilde{p}}{\partial x_i} + \nu\frac{\partial^2 \widetilde{u_i}}{\partial x_j^2} - \frac{\partial \tau_{ij}}{\partial x_j}. \tag{2.19}$$

For clarity, the τ_{ij} can be expanded to be written as:

$$\tau_{ij} \;=\; \underbrace{(\widetilde{\widetilde{u_i}\widetilde{u}_j} - \widetilde{u}_i\widetilde{u}_j)}_{L_{ij}} + \underbrace{(\widetilde{\widetilde{u_i}u'_j} + \widetilde{u'_i\widetilde{u}_j})}_{C_{ij}} + \underbrace{\widetilde{u'_iu'_j}}_{R_{ij}} \tag{2.20}$$

The L_{ij}, C_{ij}, R_{ij} denote the large scale interaction term, cross stress tensor term, and Reynold's stress tensor term respectively. Formulation τ_{ij} forms the backbone of LES modeling and all the efforts of LES community are towards proper modelling of the τ_{ij} in terms of physical and numerical parameters. There exists a zoo of LES models for τ_{ij} based on various functional and implicit modeling approaches. Perhaps the most simple and widely used LES models was proposed by Smagorinsky [1963]. The Smagorinsky model, a type of linear eddy-viscosity functional model, is the simplest and most widely used LES model in which the anisotropic part of residual stress tensor is written as:

$$\tau_{ij} = -2\nu_t\widetilde{S_{ij}}, \tag{2.21}$$

$$\text{with } \nu_t = l_{S_{ij}}^2\widetilde{S} = (C_S\Delta)^2\widetilde{S_{ij}}, \tag{2.22}$$

where $\widetilde{S_{ij}}$ is the filtered strain-rate, and l_s and C_S are Smagorinsky lenghthscale and coefficient, respectively. Based on the physics of interest and complexities taken into account, several other popular models include the Dynamic Smagorinsky model by Germano et al. [1991], the Wall-Adapting Local Eddy-viscosity (WALE) model by Nicoud and Ducros [1999], the SIGMA model by Nicoud et al. [2011], etc. The reader is referred to the books by Pope [2001] and Sagaut [2006] for a detailed overview of the LES methods and practices.

Though LES is now becoming more affordable for real engineering problems, the challenges for LES in turbomachinery have always been with the high Reynolds number flows. As demonstrated by Löhner and Baum [2013], for an

Table 2.1: An estimate of number of grid points(N_p) and timesteps(N_t) requirements for various methods. Adapted from Löhner and Baum [2013]

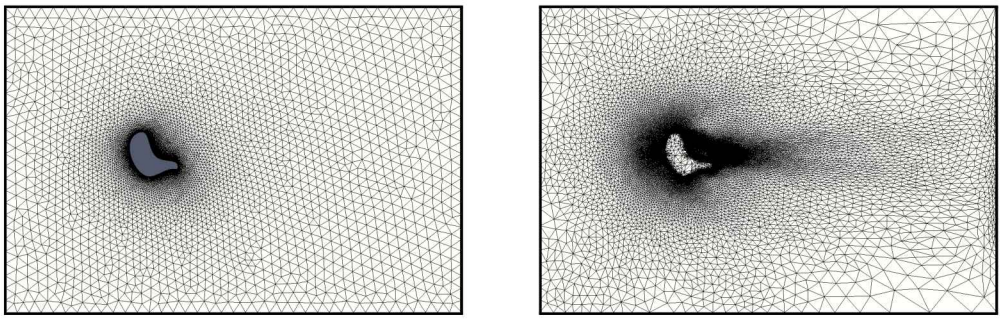
Re	**RANS** N_p	**RANS** N_t	**LES** N_p	**LES** N_t	**DNS** N_p	**DNS** N_t
10^6	$10^{6.4}$	$10^{3.2}$	$10^{8.4}$	$10^{4.2}$	$10^{13.6}$	$10^{6.8}$
10^7	$10^{6.8}$	$10^{3.4}$	$10^{8.8}$	$10^{4.4}$	$10^{15.2}$	$10^{7.6}$
10^8	$10^{7.2}$	$10^{3.6}$	$10^{9.2}$	$10^{4.6}$	$10^{16.8}$	$10^{8.4}$
10^9	$10^{7.6}$	$10^{3.8}$	$10^{9.6}$	$10^{4.8}$	$10^{18.4}$	$10^{9.2}$

LES run of the range of $Re \approx 10^6$, about 10^5 timesteps(N_t) are required, which, assuming at 0.4 sec/step for 10^8 of total number of grid points(N_p), implies 1.11 thousand hours or 46.5 days. This time is even higher for Direct Numerical Simulation (DNS). Table 2.1 provides an estimation of N_t and N_p for various methods and associated Reynolds numbers. With the advanced High-Performance Computing (HPC) architectures and massively parallel LES solvers, and assuming one can gain a factor of 10 in the simulation time, the resulting time of 4.65 days would still be a significantly high time considering its use in digital-twins. It could imply that even with an unlimited number of processors/cores, LES and DNS runs for realistic Reynolds-numbers will require days or weeks of execution as pointed out by Löhner and Baum [2013].

2.1.3 Numerical Treatments

For the majority of this thesis, Navier-Stokes equations are cast into a stabilized finite element formulation and solved using a variational multiscale (VMS) solver implemented in CimLib by Hachem et al. [2013]. In Chapter 3 and Chapter 5 , the Navier-Stokes equations solver used is equipped with a remeshing technique by Jannoun et al. [2015], which is able to track both (*i*) the fluid/solid interfaces, and (*ii*) the areas of strong gradients. Classically, the resolution of Navier-Stokes equations around solid obstacles relies on body-fitted methods, where the mesh boundary follows the geometry of the obstacle. These methods require the generation of a full mesh (domain and obstacle) for each computation, which can be both time and memory-consuming for example, see figure 2.2a). To overcome this issue, immersed methods based on a unified formulation were introduced that propose to immerse a boundary mesh of the obstacle in a background mesh for example, see figure 2.2b.

The body-fitted mesh requires mesh generation when the studied geometry is changed. The immersed method requires a background mesh for the domain and a mesh for the obstacle. The Cimlib solver is equipped with a mesh adaptation technique to refine the elements around the solid interface. The first step is to compute the signed distance function (level set) Bruchon et al. [2009] of the given geometry to each node of the background mesh. The signed distance function



(a) Body-fitted mesh (b) Immersed mesh

Figure 2.2: Comparison of two different methods for interface representation. Courtesy: Chen [2022]

is used to localize the interface of the immersed structure, but it is also used to initialize the desirable properties on both sides of the latter, and given as:

$$\alpha(\mathbf{x}) = \pm d(\mathbf{x}, \Gamma_{im}), \forall \mathbf{x} \in \Omega, \tag{2.23}$$

with the following sign convention: $\alpha \geq 0$ inside the solid domain defined by the interface Γ_{im}, and $\alpha \leq 0$ outside this domain. Using this function, the fluid-solid interface Γ_{im} is easily identified as the zero iso-value of the function α:

$$\Gamma_{im} = \{\mathbf{x} \in \Omega, \alpha(\mathbf{x}) = 0\}. \tag{2.24}$$

Indeed, for the elements crossed by the level-set functions, fluid-solid mixtures are used to determine the element's effective properties. To do so, a Heaviside function $H(\alpha)$ is defined as follows:

$$H(\alpha) = \begin{cases} 1 \text{ if } \alpha > 0, \\ 0 \text{ if } \alpha < 0, \end{cases} \tag{2.25}$$

With the Heaviside equation, one can easily write the unified governing equations for both the fluid phase and solid phase:

$$\begin{cases} \rho^* \left(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}\right) - \nabla \cdot \left(2\eta \boldsymbol{\varepsilon}(\mathbf{v}) + \boldsymbol{\tau} - p\mathbf{I}\right) = 0, \\ \nabla \cdot \mathbf{v} = 0, \end{cases} \tag{2.26}$$

with the mixed quantities defined by:

$$\begin{aligned} \rho^* &= H(\alpha)\rho_s + (1 - H(\alpha))\rho_f, \\ \boldsymbol{\tau} &= H(\alpha)\boldsymbol{\tau}_s. \end{aligned}$$

The subscripts f and s refer respectively to the fluid and to the solid. In the latter equality, $\boldsymbol{\tau}$ acts as a Lagrange multiplier that yields $\boldsymbol{\varepsilon}(\mathbf{v}) = 0$ in the solid Hachem et al. [2013]. The boundary conditions with the no-slip conditions on the solid surface extended to nodes inside the obstacles of the modified governing equations are the same throughout Chapters 3 and 5, except for the channel

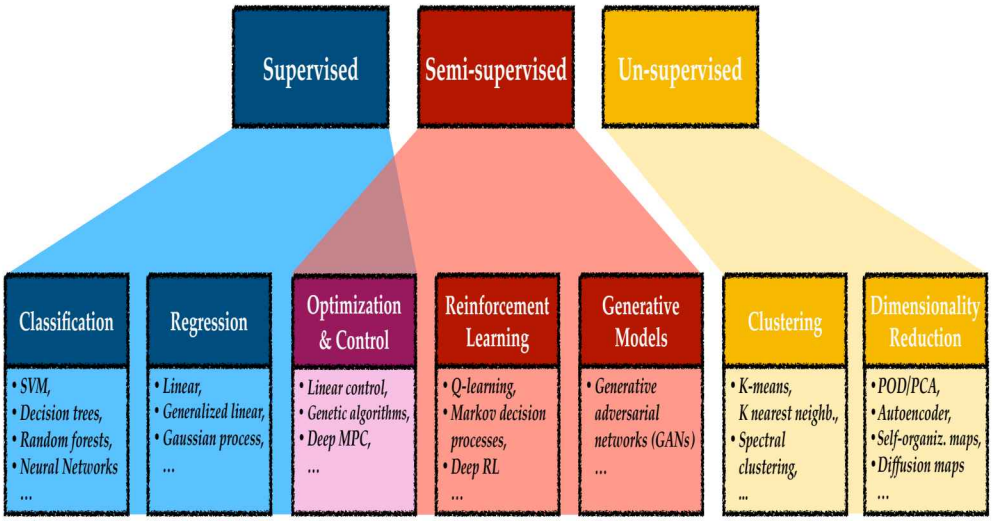


Figure 2.3: General view of machine learning and algorithms mainly used. Courtesy: Brunton et al. [2020]

flow in Chapter 4. Additionally, each chapter separately presents the numerical treatments for the governing equations and data generation.

For Chapter 4, which deals with learning subgrid-scale turbulence, the numerical treatment of Navier Stokes equations is slightly different. As opposed to the methods described before, where the data comes from Direct Numerical Simulation of turbulent channel flow in which the incompressible flow is integrated in the form of evolution equations for the wall-normal vorticity and the Laplacian of the wall-normal velocity as in Kim et al. [1987], and the spatial discretization is de-aliased Fourier in the two wall-parallel directions. Chebychev polynomials in y are used and time stepping is third-order semi-implicit Runge-Kutta by Spalart et al. [1991] using spectral methods for the Navier Stokes equations with one infinite and two periodic directions.

2.2 Deep Learning for Spatio-Temporal Data

The term "deep learning" encompasses a powerful suite of tools that have gained widespread popularity across diverse application domains, ranging from image recognition, speech recognition, and natural language processing, to targeted advertising and drug discovery. It is a rapidly-evolving and data-intensive field, with continuous advancements driven by the specific requirements of various application areas and the capabilities of new high-performance computing architectures. The field has witnessed significant growth, with sophisticated open-source software packages made available by prominent technology companies. At the core of this deep learning revolution lie fundamental concepts from applied and computational mathematics, including calculus, approximation theory, optimization, and linear algebra. Deep learning leverages hierarchical layers of hidden variables

to construct nonlinear, high-dimensional predictors, and training such architectures involves stochastic gradient descent for parameter regularization with the ultimate goal of minimizing out-of-sample predictive mean squared error. The prediction of spatio-temporal flows poses a formidable challenge, given the complex interactions and nonlinearities inherent in dynamic spatio-temporal data. Deep learning, with its hierarchical layers of hidden variables, is capable of capturing these intricate interactions and nonlinearities, paving the way for advanced predictive modeling in spatio-temporal domains.

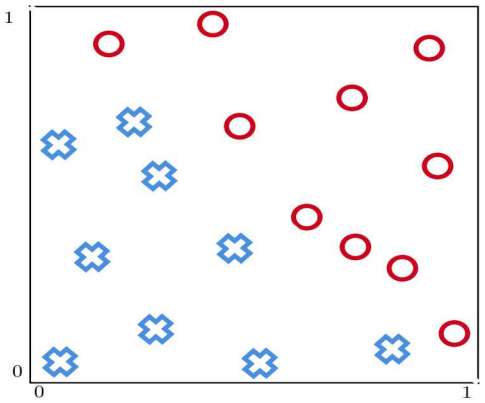
2.2.1 Deep Learning Preliminaries

This section is tailored for readers who are new to the field of deep learning. If the reader is already familiar with terms such as "training", "layers", "loss function", etc., it is recommended to skip this section. The deep learning literature is replete with specialized jargon that must be comprehended before delving into the intricacies of the subject. In order to provide a concise and coherent overview within the context of this thesis, let us elucidate these concepts through a simplified example problem. Consider a set of points depicted in Figure 2.4a, which represents labeled data, wherein certain points are categorized as A denoted by circles, and the remaining points are categorized as B denoted by crosses. For instance, this data could pertain to the locations of air injectors on a body, with category A indicating successful outcomes of air injections. Can we utilize this data to categorize a newly proposed air injector location? The task at hand is to construct a transformation that maps any point in $\mathbb{R}^2$ to either a circle or a square.

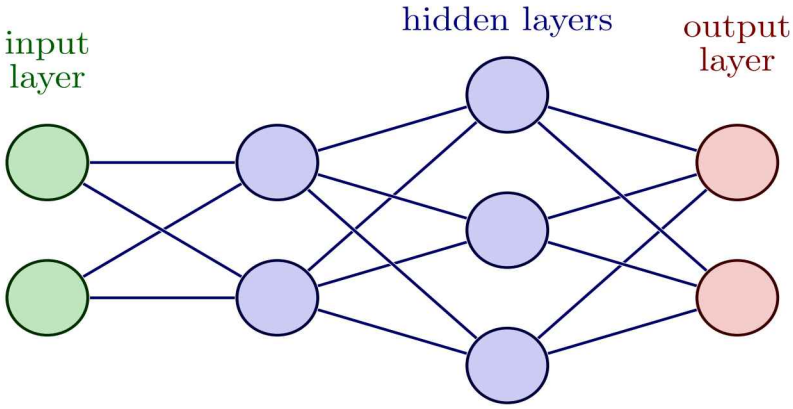
In this example, we will attempt to devise an artificial neural network approach utilizing repeated application of a simple nonlinear function. We will proceed by setting up multiple layers of neurons in the artificial neural network, as illustrated in Figure 2.4b. This network architecture will be applied to the problem delineated in Figure 2.4a. Specifically, in the network depicted in Figure 2.4b, the first (input) layer comprises two circles, corresponding to the two components of our input data points. The second layer is composed of two solid circles, signifying the utilization of two neurons. The arrows from the first layer to the second layer denote that both components of the input data are made available to the neurons in the second layer. In each layer, each neuron produces a single real number, which is then transmitted to every neuron in the subsequent layer. In the next layer, each neuron computes its own weighted combination of these values, incorporates its own bias (often defined layer-wise), and applies the sigmoid function. If the real numbers generated by the neurons in one layer are collated into a vector a, then the vector of outputs from the subsequent layer takes on the form

$$\sigma(Wa + b). \tag{2.27}$$

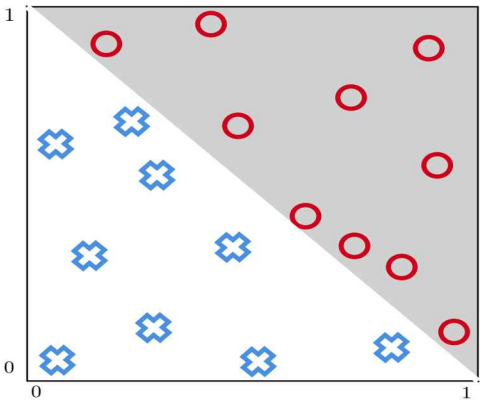
Here, W is a matrix and b is a vector. We say that W contains the *weights* and b



(a)



(b)



(c)

Figure 2.4: (a) Labeled data points in $\mathbb{R}^2$. Circles denote points in category A. Crosses denote points in category B. (b) Artifical neural network with four layers. (c) Visualization of output from the trained artificial neural network applied to the labeled data points.

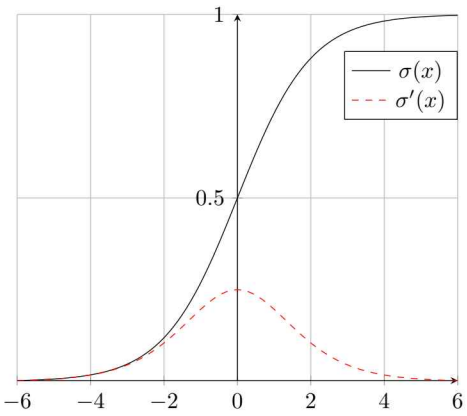


Figure 2.5: A sigmoid function and it's derivative.

contains the *biases*. The dimensions of W are such that the number of columns matches the number of neurons that produced the vector a at the previous layer, while the number of rows matches the number of neurons at the current layer. Similarly, the number of components in b matches the number of neurons at the current layer. To emphasize the role of the i^{th} neuron in (2.27), we can isolate the i^{th} component as :

$$\sigma\left(\sum_j w_{ij}a_j + b_i\right),$$

where the sum runs over all entries in a. The σ is a *activation* function, a sigmoid function in this case, on which our network is based, where:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \tag{2.28}$$

which is illustrated in the figure 2.8 over the interval $-10 \leq x \leq 10$. The $\sigma(x)$ function can be regarded as a smoothed version of a step function, which mimics the behavior of a neuron in the brain, firing (i.e., giving output equal to one) if the input is large enough, and remaining inactive (i.e., giving output equal to zero) otherwise. The sigmoid also has the convenient property that its derivative takes the simple form:

$$\sigma'(x) = \sigma(x)\left(1-\sigma(x)\right), \tag{2.29}$$

The steepness and location of the transition in the sigmoid function can be modified by scaling and shifting the argument, also known as *weighting* and *biasing* the input in the context of neural networks. The sigmoid function (2.28), as shown in figure 2.8, and the rectified linear unit (2.30) are commonly used as activation functions. The rectified linear unit, abbreviated as $ReLU$, is defined as:

$$\sigma(x) = \begin{cases} 0, & \text{for } x \leq 0, \\ x, & \text{for } x > 0, \end{cases} \tag{2.30}$$

Alternatives include the *step function*:

$$\begin{cases} 0, & \text{for } x \leq 0, \\ 1, & \text{for } x > 0. \end{cases}$$

Various other activation functions are also commonly used, and their performance may vary depending on the specific application. One issue that can arise with some activation functions is *saturation*, which refers to the production of very small derivatives that can result in reduced gradient updates. For instance, the step function and rectified linear unit have completely flat portions, which can lead to this saturation issue. To mitigate this, a modified version called the *leaky rectified linear unit* can be used, as exemplified by:

$$f(x) = \begin{cases} 0.01x, & \text{for } x \leq 0, \\ x, & \text{for } x > 0, \end{cases}$$

is sometimes preferred, in order to force a nonzero derivative for negative inputs. To keep our notations manageable, we need to interpret the sigmoid function in a vectorized sense. For $z \in \mathbb{R}^m$, $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ is defined by applying the sigmoid function in the obvious componen-wise manner, so that:

$$(\sigma(z))_i = \sigma(z_i).$$

Since the input data has the form $x \in \mathbb{R}^2$, the weights and biases for layer two may be represented by a matrix $W^{[2]} \in \mathbb{R}^{2\times 2}$ and a vector $b^{[2]} \in \mathbb{R}^2$, respectively. The output from layer two then has the form:

$$\sigma(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2.$$

Layer three has three neurons, each receiving input in $\mathbb{R}^2$. So the weights and biases for layer three may be represented by a matrix $W^{[3]} \in \mathbb{R}^{3\times 2}$ and a vector $b^{[3]} \in \mathbb{R}^3$, respectively. The output from layer three then has the form:

$$\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) \in \mathbb{R}^3.$$

The fourth (output) layer has two neurons, each receiving input in $\mathbb{R}^3$. So the weights and biases for this layer may be represented by a matrix $W^{[4]} \in \mathbb{R}^{2\times 3}$ and a vector $b^{[4]} \in \mathbb{R}^{[2]}$, respectively. The output from layer four, and hence from the overall network, has the form:

$$F(x) = \sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma(W^{[2]}x + b^{[2]}) + b^{[3]}\right) + b^{[4]}\right) \in \mathbb{R}^2. \tag{2.31}$$

The expression (2.31) defines a function $F : \mathbb{R}^2 \to \mathbb{R}^2$ in terms of its 23 parameters - the entries in the weight matrices and bias vectors. Since the aim of this task is to produce a classifier based on the data in figure 2.4a, we do this

by optimizing over the parameters. We will require $F(x)$ to be close to $[1,0]^T$ for data points in category A and close to $[0,1]^T$ for data points in category B. Then, given a new point $x \in \mathbb{R}^2$, it would be reasonable to classify it according to the largest component of $F(x)$; that is, category A if $F_1(x) > F_2(x)$ and category B if $F_1(x) < F_2(x)$. This requirement on F may be specified through a *loss function*. Denoting the ten data points in figure 2.4a by $\{x^{\{i\}}\}_{i=1}^{10}$, we use $y(x^{\{i\}})$ for the target output; that is,

$$y(x^{\{i\}}) = \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \text{if } x^{\{i\}} \text{ is in category A,} \\ \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \text{if } x^{\{i\}} \text{ is in category B.} \end{cases} \tag{2.32}$$

Our loss function then takes the form:

$$\mathcal{L}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2. \tag{2.33}$$

It is important to note that the Loss function is dependent on the weights and biases, while the data points are considered fixed. The specific form of the Loss function, as shown in (2.33), where the discrepancy is measured by averaging the squared Euclidean norm over the data points, is commonly known as a *quadratic loss function*. In the context of optimization, the Loss function serves as our *objective function*, and the process of selecting weights and biases that minimize this loss function is referred to as *training* the network. It is important to highlight that, in principle, rescaling a loss function does not alter the fundamental optimization problem. Nevertheless, one potential approach is to modify the loss function in a way that promotes the use of small weights in the optimization process. For example, (2.37) could be extended to

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} \tfrac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2 + \frac{\lambda}{N}\sum_{l=2}^{L} \| W^{[l]} \|_2^2. \tag{2.34}$$

Here $\lambda > 0$ is the regularization parameter. A rationale behind the cost function given in (2.34) is that large weights in the neural network may result in neurons that are overly sensitive to their inputs, which can potentially reduce their reliability when confronted with new data. It is worth noting that this argument does not typically extend to the biases, as they are usually not included in regularization terms aimed at promoting weight regularization.

Subsequently, an optimization toolbox or library is employed to minimize the loss function (2.33) over the 23 parameters that define $W^{[2]}$, $W^{[3]}$, $W^{[4]}$, $b^{[2]}$, $b^{[3]}$, and $b^{[4]}$, using the data presented in Figure 2.4a. The resulting trained network is then utilized to generate the boundary shown in Figure 2.4c, where points in the shaded region are classified as category A, while points in the unshaded region are classified as category B based on the condition $F_1(x) > F_2(x)$.

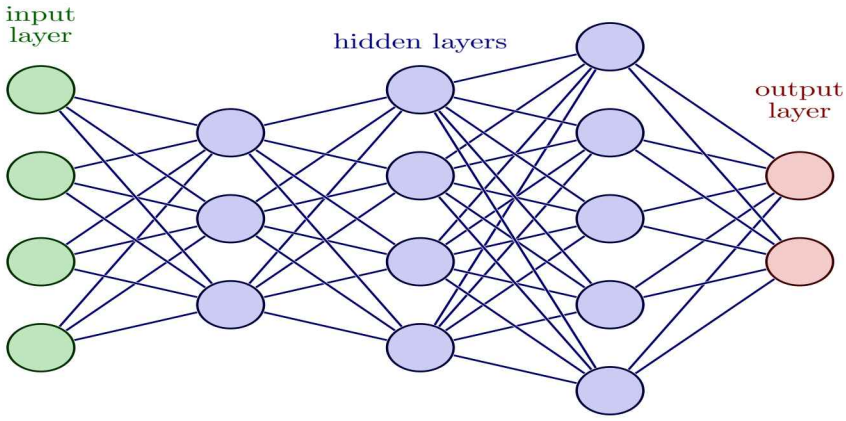


Figure 2.6: A network with five layers .

The example depicted in Figure 2.4c may appear small-scale in terms of the size of the data and network, but the underlying optimization problem of minimizing a non-convex objective function over 23 variables is inherently challenging. Exhaustively searching over a 23-dimensional parameter space is impractical, and finding the global minimum of a non-convex function is not guaranteed. Experimentation with different data point arrangements in Figure 2.4c and initial weight and bias guesses reveals that finding an acceptable solution is not always possible, thus emphasizing the significance of focusing on the optimization problem.

Let's now introduce a comprehensive notation that enables the definition of a generalized network. The concept of neurons, which are represented by the sigmoid activation function and operate in layers, was introduced in the example of a four-layer network. At each general layer, every neuron receives the same input, which is a real value from every neuron in the previous layer, and produces a single real value that is passed on to every neuron in the next layer. In the input layer, there is no notion of a "previous layer," and each neuron receives the input vector. Similarly, in the output layer, there is no concept of a "next layer", and these neurons provide the overall output of the network. The layers between the input and output layers are commonly referred to as "hidden layers." The term "hidden layers" does not carry any specific meaning; it simply indicates that these neurons perform intermediate calculations. Formally, this also implies that deep learning, a loosely-defined term, often involves the utilization of multiple hidden layers in a network. We suppose that the network has L layers, with layers 1 and L being the input and output layers, respectively. Suppose that layer l, for $l = 1, 2, 3, \ldots, L$ contains n_l neurons. So n_1 is the dimension of the input data. Overall, the network maps from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$. We use $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ to denote the matrix of weights at layer l. More precisely, $w^{[l]}_{jk}$ is the weight that neuron j at layer l applies to the output from neuron k at layer $l-1$. Similarly, $b^{[l]} \in \mathbb{R}^{n_l}$ is the vector of biases for layer l, so neuron j at layer l uses the bias $b^{[l]}_j$.

In Fig 2.11 we give an example with $L = 5$ layers. Here, $n_1 = 4$, $n_2 = 3$,

$n_3 = 4$, $n_4 = 5$ and $n_5 = 2$, so $W^{[2]} \in \mathbb{R}^{3\times4}$, $W^{[3]} \in \mathbb{R}^{4\times3}$, $W^{[4]} \in \mathbb{R}^{5\times4}$, $W^{[5]} \in \mathbb{R}^{2\times5}$, $b^{[2]} \in \mathbb{R}^3$, $b^{[3]} \in \mathbb{R}^4$, $b^{[4]} \in \mathbb{R}^5$ and $b^{[5]} \in \mathbb{R}^2$.

Given an input $x \in \mathbb{R}^{n_1}$, we may then neatly summarize the action of the network by letting $a_j^{[l]}$ denote the output, or *activation*, from neuron j at layer l. So, we have

$$a^{[1]} = x \in \mathbb{R}^{n_1}, \tag{2.35}$$

$$a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \quad \text{for } l = 2, 3, \ldots, L. \tag{2.36}$$

It should be clear that (2.35) and (2.36) amount to an algorithm for feeding the input forward through the network in order to produce an output $a^{[L]} \in \mathbb{R}^{n_L}$. Now suppose we have N pieces of data, or *training points*, in $\mathbb{R}^{n_1}$, $\{x^{\{i\}}\}_{i=1}^N$, for which there are given target outputs $\{y(x^{\{i\}})\}_{i=1}^N$ in $\mathbb{R}^{n_L}$. Generalizing (2.33), the quadratic loss function that we wish to minimize has the form

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N} \tfrac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2, \tag{2.37}$$

where, to keep notation under control, we have not explicitly indicated that Loss is a function of all the weights and biases. Moreover, training a network corresponds to choosing the parameters, that is, the weights and biases, that minimize the loss function. The weights and biases take the form of matrices and vectors, but at this stage it is convenient to imagine them stored as a single vector that we call p. The example in figure 2.4b has a total of 23 weights and biases. So, in that case, $p \in \mathbb{R}^{23}$. Generally, we will suppose $p \in \mathbb{R}^s$, and write the loss function in (2.37) as Loss(p) to emphasize its dependence on the parameters. So Loss : $\mathbb{R}^s \to \mathbb{R}$.

We now introduce a classical method in optimization that is often referred to as *steepest descent* or *gradient descent*. The method proceeds iteratively, computing a sequence of vectors in $\mathbb{R}^s$ with the aim of converging to a vector that minimizes the loss function. Suppose that our current vector is p. How should we choose a perturbation, Δp, so that the next vector, $p + \Delta p$, represents an improvement? If Δp is small, then ignoring terms of order $\| \Delta p \|^2$, a Taylor series expansion gives

$$\text{Loss}(p + \Delta p) \approx \text{Loss}(p) + \sum_{r=1}^{s} \frac{\partial\, \text{Loss}(p)}{\partial\, p_r}\Delta p_r. \tag{2.38}$$

Here $\partial\, \text{Loss}(p)/\partial\, p_r$ denotes the partial derivative of the loss function with respect to the rth parameter. For convenience, we will let $\nabla\text{Loss}(p) \in \mathbb{R}^s$ denote the vector of partial derivatives, known as the *gradient*, so that

$$(\nabla\text{Loss}(p))_r = \frac{\partial\, \text{Loss}(p)}{\partial\, p_r}.$$

Then (2.38) becomes

$$\text{Loss}(p + \Delta p) \approx \text{Loss}(p) + \nabla\text{Loss}(p)^T\Delta p. \tag{2.39}$$

Our aim is to reduce the value of the loss function. The relation (2.39) motivates the idea of choosing Δp to make $\nabla \text{Loss}(p)^T \Delta p$ as negative as possible. We can address this problem via the Cauchy–Schwarz inequality, which states that for any $f, g \in \mathbb{R}^s$, we have $|f^T g| \leq \| f \|_2 \| g \|_2$. So the most negative that $f^T g$ can be is $-\| f \|_2 \| g \|_2$, which happens when $f = -g$. Hence, based on (2.39), we should choose Δp to lie in the direction $-\nabla \text{Loss}(p)$. Keeping in mind that (2.39) is an approximation that is relevant only for small Δp, we will limit ourselves to a small step in that direction. This leads to the update

$$p \to p - \eta \nabla \text{Loss}(p). \tag{2.40}$$

Here η is small stepsize that, in this context, is known as the *learning rate*. We choose an initial vector and iterate with (2.40) until some stopping criterion has been met, or until the number of iterations has exceeded the computational budget. Our loss function (2.37) involves a sum of individual terms that runs over the training data. It follows that the partial derivative $\nabla \text{Loss}(p)$ is a sum over the training data of individual partial derivatives. More precisely, let

$$C_{x^{\{i\}}} = \tfrac{1}{2} \| y(x^{\{i\}}) - a^{[L]}(x^{\{i\}}) \|_2^2. \tag{2.41}$$

Then, from (2.37),

$$\nabla \text{Loss}(p) = \frac{1}{N} \sum_{i=1}^{N} \nabla C_{x^{\{i\}}}(p). \tag{2.42}$$

When dealing with a significant number of parameters and a large dataset, computing the gradient vector (2.42) at every iteration of the steepest descent method (2.40) can become computationally prohibitive. As a more cost-effective alternative, the mean of the individual gradients over all training points can be replaced by the gradient computed at a single randomly chosen training point. This approach results in the simplest form of the *stochastic gradient* method. A single step may be summarized as

1. Choose an integer i uniformly at random from $\{1, 2, 3, \ldots, N\}$.

2. Update
$$p \to p - \eta \nabla C_{x^{\{i\}}}(p). \tag{2.43}$$

The stochastic gradient method, in essence, selects one randomly chosen training point at each step to represent the entire training set. As the iterations progress, the method encounters more training points. While this approach results in a significant reduction in loss-per-iteration, there is no guarantee that the overall loss function will be reduced, as we have replaced the mean gradient with a single sample (2.43), even when using a small learning rate η.

The stochastic gradient method that is presented in (2.43) is just one of many possible variations. In this particular version, the index i was chosen by sampling with replacement, meaning that a training point can be chosen more than once and has an equal probability of being chosen again in the next step. However,

an alternative approach is to sample without replacement, where each of the N training points is cycled through in a random order. Performing N steps in this manner, refered to as completing an *epoch*, may be summarized as follows:

1. Shuffle the integers $\{1, 2, 3, \ldots, N\}$ into a new order, $\{k_1, k_2, k_3, \ldots, k_N\}$.
2. for $i = 1$ upto N, update

$$p \to p - \eta \nabla C_{x^{\{k_i\}}}(p). \tag{2.44}$$

When viewing the stochastic gradient method as an approximation of the mean over all training points in (2.42) using a single sample, it is reasonable to consider a compromise by using a small sample average. This can be achieved by taking steps of the following form for some $m \ll N$:

1. Choose m integers, $k_1, k_2, \ldots, k_m$, uniformly at random from $\{1, 2, 3, \ldots, N\}$.
2. Update

$$p \to p - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla C_{x^{\{k_i\}}}(p). \tag{2.45}$$

In this iteration, the set $\{x^{\{k_i\}}\}_{i=1}^{n}$ is known as a *mini-batch*. An alternative to the stochastic gradient method with replacement is to sample *without replacement*, where the training set is randomly split into K distinct mini-batches and cycled through, assuming $N = Km$ for some K. This choice of mini-batch size and randomization strategy is often influenced by the requirements of high performance computing architectures, as the stochastic gradient method is typically implemented in large-scale computations. Additionally, it is possible to dynamically vary these choices, such as mini-batch size and learning rate, during training to potentially accelerate convergence.

In the context of deep learning applications, the learning rate η (also known as step size) is often kept constant, or a strategy for reducing the step size is employed. Determining appropriate learning rates or hyper-parameters for the reduction schedule is typically done empirically through numerical experiments and observation of the progression of the loss function. One drawback of stochastic gradient descent (SGD) is that the descent in the objective function f is not guaranteed at every iteration, and it can be slow. Moreover, as the iterates converge to a solution, the variance of the gradient estimate g^k tends to approach zero. To address these issues, modifications of SGD such as coordinate descent (CD) and momentum-based methods are used. In CD, each step updates a single component of the gradient at the current point and then updates the corresponding component of the variable vector in the negative gradient direction. Momentum-based versions of SGD, or accelerated algorithms, were originally proposed by Nesterov [2003]. These methods incorporate momentum, which combines new gradient information with the previous search direction. Empirically, momentum-based methods often exhibit better convergence properties for deep

learning networks. The key idea is that the gradient only influences changes in the "velocity" of the update, analogous to the reduction in kinetic energy that allows for "slowing down" of movements near the minima. The momentum parameter is typically chosen empirically using techniques such as cross-validation. Nesterov's momentum method, also known as Nesterov acceleration, calculates the gradient at the point predicted by the momentum, which can be thought of as a look-ahead strategy. Another popular modification to SGD is the AdaGrad method, which adaptively scales each of the learning parameters at each iteration based on their historical gradient information. The PRMSprop method extends the idea of AdaGrad by placing more weight on recent values of the squared gradient to scale the update direction. Finally, the Adam method, which is one of the most well-known optimization algorithms, combines both PRMSprop and momentum methods, incorporating adaptive learning rates and momentum to achieve faster convergence in many cases.

Let's apply the stochastic gradient method to effectively train an artificial neural network. This entails a shift from the general vector of parameters p to the specific entries in the weight matrices and bias vectors. Our objective is to compute partial derivatives of the loss function with respect to each $w_{jk}^{[l]}$ and $b_j^{[l]}$. As we have observed, the key idea behind the stochastic gradient method is to leverage the inherent structure of the loss function. Since (2.37) is a linear combination of individual terms that operate over the training data, the same holds true for its partial derivatives. Consequently, the primary focus is on accurately computing these individual partial derivatives. Hence, for a fixed training point $C_{x^{\{i\}}}$ in (2.41) is regarded as a function of the weights and biases. So the dependence on $x^{\{i\}}$ may be dropped and simply written as

$$C = \tfrac{1}{2}\|y - a^{[L]}\|_2^2. \tag{2.46}$$

As previously noted in equation (2.36), the final output $a^{[L]}$ of the artificial neural network fully determines the dependence of the loss function C on the weights and biases. To facilitate the derivation of meaningful expressions for the partial derivatives of the loss function, it is advantageous to introduce two additional sets of variables. First let

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}, \qquad \text{for } l = 2, 3, \ldots, L. \tag{2.47}$$

The term *weighted input* for neuron j at layer l is denoted as $z_j^{[l]}$. The fundamental relation (2.36) that propagates information through the network may then be written as

$$a^{[l]} = \sigma\left(z^{[l]}\right), \qquad \text{for } l = 2, 3, \ldots, L. \tag{2.48}$$

Second, let $\delta^{[l]} \in \mathbb{R}^{n_l}$ be defined by

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \qquad \text{for } 1 \le j \le n_l \quad \text{and} \quad 2 \le l \le L. \tag{2.49}$$

This expression, which is often called the *error* in the j^{th} neuron at layer l, is an intermediate quantity that serves as a useful tool for analysis and computation. However, it should be noted that the term "error" in this context can be ambiguous. Specifically, at a general hidden layer, it is unclear how much blame should be attributed to each neuron for discrepancies in the final output. Additionally, at the output layer L, the expression (2.49) does not directly quantify the magnitude of these discrepancies. The idea of referring to $\delta_j^{[l]}$ in (2.49) as an error seems to have arisen because the loss function can only be at a minimum if all partial derivatives are zero, so $\delta_j^{[l]} = 0$ is a useful goal. It may be more helpful to keep in mind that $\delta_j^{[l]}$ measures the sensitivity of the loss function to the weighted input for neuron j at layer l. At this stage, it is also necessary to define the Hadamard, or componentwise, product of two vectors. If x and y are vectors in $\mathbb{R}^n$, then the Hadamard product, denoted as $x \circ y$, is a vector in $\mathbb{R}^n$ defined as $(x \circ y)_i = x_i y_i$, where the components of the resulting vector are obtained by pairwise multiplication of the corresponding components. With this notation, the following results are a consequence of the chain rule. We have:

$$\begin{aligned}
\delta^{[L]} &= \sigma'(z^{[L]}) \circ (a^L - y), && (2.50)\\
\delta^{[l]} &= \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}, && \text{for } 2 \le l \le L-1, \quad (2.51)\\
\frac{\partial C}{\partial b_j^{[l]}} &= \delta_j^{[l]}, && \text{for } 2 \le l \le L, \quad (2.52)\\
\frac{\partial C}{\partial w_{jk}^{[l]}} &= \delta_j^{[l]} a_k^{[l-1]}. && \text{for } 2 \le l \le L. \quad (2.53)
\end{aligned}$$

The relation (2.48) with $l = L$ shows that $z_j^{[L]}$ and $a_j^{[L]}$ are connected by $a^{[L]} = \sigma\left(z^{[L]}\right)$, and hence

$$\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \sigma'(z_j^{[L]}).$$

Also, from (2.46),

$$\frac{\partial C}{\partial a_j^{[L]}} = \frac{\partial}{\partial a_j^{[L]}} \frac{1}{2} \sum_{k=1}^{n_L} (y_k - a_k^{[L]})^2 = -(y_j - a_j^{[L]}).$$

So, using the chain rule,

$$\delta_j^{[L]} = \frac{\partial C}{\partial z_j^{[L]}} = \frac{\partial C}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = (a_j^{[L]} - y_j)\sigma'(z_j^{[L]}),$$

which is the componentwise form of (2.50).

To describe (2.51), we use the chain rule to convert from $z_j^{[l]}$ to $\{z_k^{[l+1]}\}_{k=1}^{n_{l+1}}$. Applying the chain rule, and using the definition (2.49),

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}}. \quad (2.54)$$

Now, from (2.47) we know that $z_k^{[l+1]}$ and $z_j^{[l]}$ are connected via

$$z_k^{[l+1]} = \sum_{s=1}^{n_l} w_{ks}^{[l+1]} \sigma\left(z_s^{[l]}\right) + b_k^{[l+1]}.$$

Hence,

$$\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = w_{kj}^{[l+1]} \sigma'\left(z_j^{[l]}\right).$$

In (2.54) this gives

$$\delta_j^{[l]} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} w_{kj}^{[l+1]} \sigma'\left(z_j^{[l]}\right),$$

which may be rearranged as

$$\delta_j^{[l]} = \sigma'\left(z_j^{[l]}\right) \left((W^{[l+1]})^T \delta^{[l+1]}\right)_j.$$

This is the componentwise form of (2.51). To show (2.52), we note from (2.47) and (2.48) that $z_j^{[l]}$ is connected to $b_j^{[l]}$ by

$$z_j^{[l]} = \left(W^{[l]} \sigma\left(z^{[l-1]}\right)\right)_j + b_j^{[l]}.$$

Since $z^{[l-1]}$ does not depend on $b_j^{[l]}$, we find that

$$\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = 1.$$

Then, from the chain rule,

$$\frac{\partial C}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} = \delta_j^{[l]},$$

using the definition (2.49). This gives (2.52). Finally, to obtain (2.53) we start with the componentwise version of (2.47),

$$z_j^{[l]} = \sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]},$$

which gives

$$\frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = a_k^{[l-1]}, \qquad \text{independently of } j, \tag{2.55}$$

and

$$\frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = 0, \qquad \text{for } s \neq j. \tag{2.56}$$

In other words, (2.55) and (2.56) follow because the j^{th} neuron at layer l uses the weights from only the j^{th} row of $W^{[l]}$, and applies these weights linearly. Then, from the chain rule, (2.55) and (2.56) give

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \sum_{s=1}^{n_l} \frac{\partial C}{\partial z_s^{[l]}} \frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} a_k^{[l-1]} = \delta_j^{[l]} a_k^{[l-1]},$$

where the last step used the definition of $\delta_j^{[l]}$ in (2.49).

There are many aspects here that deserve our attention. We recall from (2.35), (2.47) and (2.48) that the output $a^{[L]}$ can be evaluated from a *forward pass* through the network, computing $a^{[1]}$, $z^{[2]}$, $a^{[2]}$, $z^{[3]}$, ..., $a^{[L]}$ in order. Having done this, we see from (2.50) that $\delta^{[L]}$ is immediately available. Then, from (2.51), $\delta^{[L-1]}$, $\delta^{[L-2]}$, ..., $\delta^{[2]}$ may be computed in a *backward pass*. From (2.52) and (2.53), we then have access to the partial derivatives. Computing gradients in this way is known as *back propagation*.

To gain further understanding of the back propagation formulas (2.52) and (2.53), it is useful to recall the fundamental definition of a partial derivative. The quantity $\partial C/\partial w_{jk}^{[l]}$ measures how C changes when we make a small perturbation to $w_{jk}^{[l]}$. For illustration, figure 2.11 highlights the weight $w_{43}^{[3]}$. It is clear that a change in this weight has no effect on the output of previous layers. So to work out $\partial C/\partial w_{43}^{[3]}$ we do not need to know about partial derivatives at previous layers. It should, however, be possible to express $\partial C/\partial w_{43}^{[3]}$ in terms of partial derivatives at subsequent layers. More precisely, the activation feeding into the 4^{th} neuron on layer 3 is $z_4^{[3]}$, and, by definition, $\delta_4^{[3]}$ measures the sensitivity of C with respect to this input. Feeding in to this neuron we have $w_{43}^{[3]} a_3^{[2]} + \text{constant}$, so it makes sense that

$$\frac{\partial C}{\partial w_{43}^{[3]}} = \delta_4^{[3]} a_3^{[2]}.$$

Similarly, in terms of the bias, $b_4^{[3]} + \text{constant}$ is feeding in to the neuron, which explains why

$$\frac{\partial C}{\partial b_4^{[3]}} = \delta_4^{[3]} \times 1.$$

We may avoid the componentwise product notation in (2.50) and (2.51) by introducing diagonal matrices. Let $D^{[l]} \in \mathbb{R}^{n_l \times n_l}$ denote the diagonal matrix with (i,i) entry given by $\sigma'(z_i^{[l]})$. Then we see that $\delta^{[L]} = D^{[L]}(a^{[L]} - y)$ and $\delta^{[l]} = D^{[l]}(W^{[l+1]})^T \delta^{[l+1]}$. We could expand this out as

$$\delta^{[l]} = D^{[l]}(W^{[l+1]})^T D^{[l+1]}(W^{[l+2]})^T \cdots D^{[L-1]}(W^{[L]})^T D^{[L]}(a^{[L]} - y).$$

We also recall from (2.29) that $\sigma'(z)$ is trivial to compute. The relation (2.52) shows that $\delta^{[l]}$ corresponds precisely to the gradient of the loss function with respect to the biases at layer l. If we regard $\partial C/\partial w_{jk}^{[l]}$ as defining the (j,k) component in a matrix of partial derivatives at layer l, then (2.53) shows this matrix to be the *outer product* $\delta^{[l]} {a^{[l-1]}}^T \in \mathbb{R}^{n_l \times n_{l-1}}$.

It follows that the update in the basic gradient descent method (2.40) has the form

$$\begin{aligned} W^{[l]} &\rightarrow W^{[l]} - \eta \frac{1}{N} \sum_{i=1}^{N} \delta^{[l]}_{\{x^i\}} a^{[l-1]^T}_{\{x^i\}} \\ b^{[l]} &\rightarrow b^{[l]} - \eta \frac{1}{N} \sum_{i=1}^{N} \delta^{[l]}_{\{x^i\}}. \end{aligned}$$

Combining the concepts discussed above, we can outline the pseudocode for an algorithm that trains a neural network using a fixed number, n_{iter}, of stochastic gradient iterations. For simplicity, we consider the basic version (2.43) where single samples are chosen with replacement. The algorithm involves performing a forward pass through the network for each training point in order to evaluate the activations, weighted inputs, and overall output $a^{[L]}$. Then a backward pass is performed to compute the errors and updates.

For counter $= 1$ upto n_{iter}

Choose an integer k uniformly at random from $\{1, 2, 3, \ldots, N\}$

$x^{\{k\}}$ is current training data point

$a^{[1]} = x^{\{k\}}$

For $l = 2$ upto L

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = \sigma\left(z^{[l]}\right)$

$D^{[l]} = \text{diag}\left(\sigma'(z^{[l]})\right)$

end

$\delta^{[L]} = D^{[L]}\left(a^{[L]} - y(x^{\{k\}})\right)$

For $l = L - 1$ downto 2

$\delta^{[l]} = D^{[l]}(W^{[l+1]})^T \delta^{[l+1]}$

end

For $l = L$ downto 2

$W^{[l]} \rightarrow W^{[l]} - \eta\, \delta^{[l]} a^{[l-1]^T}$

$b^{[l]} \rightarrow b^{[l]} - \eta\, \delta^{[l]}$

end

end

The phenomenon of *overfitting* arises when a trained neural network achieves high accuracy on the given data, but fails to generalize well to new, unseen data. Essentially, this indicates that the training process has overly focused on irrelevant and unrepresentative "noise" present in the given data. Various approaches have been proposed to combat overfitting, and often multiple techniques are used in combination. One effective technique is to split the given data into two distinct groups.

- *Training data* is used in the definition of the loss function that defines the optimization problem. Hence this data drives the process that iteratively updates the weights.

- *Validation data*, on the other hand, does not play a role in the optimization process itself, as it does not affect the weight update procedure at each step. Its purpose is solely to assess the performance of the current network. During the optimization process, the loss function corresponding to the validation data can be evaluated, providing a measure of how effectively the current set of trained weights performs on unseen data.

From an intuitive standpoint, overfitting can be understood as a situation where the optimization process continues to minimize the loss function with respect to the training data, resulting in a better fit to that data. However, the loss function for the validation error may no longer decrease, indicating that the performance on unseen data is not improving. Hence, it is reasonable to stop the training process when no further improvement is observed on the validation data.

Deep learning procedure comprises following two key steps:

1. Training phase: pair the input with expected output, until a sufficiently close match has been found. Gauss' original least squares procedure is a common example.

2. Validation and test phase: Evaluating the effectiveness of the deep learning model in making predictions on unseen data, considering factors such as data size, target value, input characteristics, and other model properties, including mean-error for numeric predictors and classification-errors for classifiers.

Often, the validation phase is split into two parts.

2.a Initially, assess the accuracy of all approaches in making predictions on unseen data (also known as validation) to estimate their out-of-sample performance.

2.b Subsequently, compare the models and determine the most effective approach based on the performance observed on the validation data (also known as verification).

Step 2.b. can be skipped if there is no need to select an appropriate model from several rivaling approaches. In such cases, the researcher can simply partition the dataset into a training set and a test set for evaluation purposes.

For the sake of brevity, certain topics and details have been excluded. Those interested in delving deeper into the subject matter are encouraged to consult various literature sources. An excellent starting point is the work by Higham and Higham [2019] and Nielsen [2015], which offer a hands-on tutorial-style account of deep learning methods. The survey LeCun et al. [2015] provides an intuitive and accessible overview of the key concepts behind deep learning and highlights recent success stories. A more in-depth review of the prize-winning performances of deep learning tools can be found in Schmidhuber [2015], which also traces the evolution of ideas through more than 800 references. The article Wang and Raj [2017] provides an overview of the historical background of deep learning and tracks the development of key concepts. For an extensive examination of the current state-of-the-art, we suggest the book Goodfellow et al. [2016]. The book does an exceptional job of introducing fundamental concepts from various fields, such as computer science/discrete mathematics, applied/computational mathematics, and probability/statistics/inference, before integrating them into the deep learning context. The article Bottou et al. [2018] provides a comprehensive overview of optimization tasks that arise in machine learning. It not only summarizes the current theory underlying the stochastic gradient method but also discusses various alternative techniques. The authors emphasize that optimization tools should be interpreted and judged with care when working within this inherently statistical framework. In addition to training issues, Mallat [2016] provides a mathematical framework for understanding the cascade of linear and nonlinear transformations used by deep networks.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a special class of artificial neural networks which have become a standard tool in computer vision applications and regarded as one of the most important contributions in deep learning. These systems are able to recognize patterns in segments of data, such as images or recorded speech, and further process these results in order to produce decisions. The study of CNNs was greatly inspired by the Nobel prize work of Hubel and Wiesel [1962] , who studied the visual neural cortex of cats. Hubbel and Wiesel determined that neurons were organized hierarchically, such that simple neurons first detected the presence of small visual features (S-cells), and deeper neurons detected the presence of more complex objects (C-cells) Hubel and Wiesel [1962]. After years of research, these insights led to the first Convolutional Neural Network proposed in 1998 by LeCun et al. [1998]. Their neural network, called LeNet-5, managed to recognize handwritten digits with remarkable accuracy even under the presence of noise and other effects. Today, the CNNs holding the world record in hand-written digit recognition reaches 99.79% accuracy based

on similar principles Aizenberg and Gonzalez [2018]. A schematic representation of the structure employed in the original LeNet-5 network can be found depicted in 2.7. Here, it can be seen that the network first filters an image looking for small patterns, such as edges or parts of a digit. The results obtained by each filter are then passed to deeper layers. The operation called sub-sampling in the LeNet-5 network corresponds to a max-pooling operation. Even though more advanced pooling operators have been developed, these operators allow CNNs to consider various alternatives for detecting an event, and only retain the strongest detected signals. For example, in vehicle detection systems, the first convolutional filters can try to detect the presence of various types of wheels, such that a max-pooling operator can decide whether a wheel was present in the picture or not. If a wheel appeared in the image, one of the filters will raise a strong signal, and otherwise all the filters will yield zero. The previous process of convolutions and sub-sampling is repeated until the last layers of the CNNs are reached. Here, fully-connected neurons are employed in order to detect high level concepts, such as the true presence of a hand-written digit. Despite being an early prototype, modern CNNs have remained significantly close to the concepts introduced in the LeNet-5 network. Several modelling advances have been introduced to improve the efficiency of these systems however. One of the most important inventions has been the use of Dropout regularization Srivastava et al. [2014], where a random subset of the connections present in a CNN is suppressed each iteration, such that the system does not become over-reliant on small details. Dropout regularization is considered as one of the strongest techniques developed in Machine Learning, since it allows the user to introduce a strong form of regularization without further tuning hyper-parameters. Another important innovation in CNN's includes Szegedy's Inception modules by Szegedy et al. [2015], which allow CNNs to consider various convolutional filters and a sub-sampling operator at the same time. The optimizer is thus responsible for selecting the most appropriate operations between these alternatives, and a suitable architecture for a problem can be developed automatically.

The general framework described in previous subsection does not scale well in the case of spatial data. Consider a color image made up of 200 by 200 pixels, each with a red, green and blue component. This corresponds to an input vector of dimension $n_1 = 200 \times 200 \times 3 = 120,000$, and hence a weight matrix $W^{[2]}$ at level 2 that has $120,000$ columns. If we allow general weights and biases, then this approach is clearly infeasible. CNNs get around this issue by constraining the values that are allowed in the weight matrices and bias vectors. Rather than a single full-sized linear transformation, CNNs repeatedly apply a small-scale linear kernel, or filter, across portions of their input data. In effect, the weight matrices used by CNNs are extremely sparse and highly structured.

To understand why this approach might be useful, consider premultiplying an

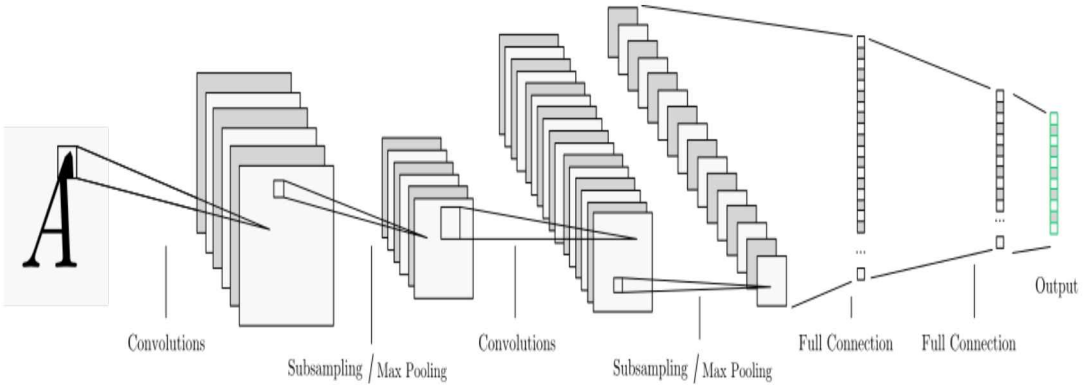


Figure 2.7: Schematic representation of a traditional Convolutional Neural Network equivalent to the LeNet-5 architecture. Courtesy: LeCun et al. [1998].

input vector in $\mathbb{R}^6$ by the matrix

$$\begin{bmatrix} 1 & -1 & & & & \\ & 1 & -1 & & & \\ & & 1 & -1 & & \\ & & & 1 & -1 & \\ & & & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{5\times 6}. \tag{2.57}$$

This produces a vector in $\mathbb{R}^5$ made up of differences between neighboring values. In this case we are using a filter $[1, -1]$ and a *stride* of length one—the filter advances by one place after each use. Appropriate generalizations of this matrix to the case of input vectors arising from 2D images can be used to detect *edges* in an image—returning a large absolute value when there is an abrupt change in neighboring pixel values. Moving a filter across an image can also reveal other features, for example, particular types of curves or blotches of the same color. So, having specified a filter size and stride length, we can allow the training process to learn the weights in the filter as a means to extract useful structure. The word "convolutional" arises because the linear transformations involved may be written in the form of a convolution. In the 1D case, the convolution of the vector $x \in \mathbb{R}^p$ with the filter $g_{1-p}, g_{2-p}, \ldots, g_{p-2}, g_{p-1}$ has kth component given by

$$y_k = \sum_{n=1}^{p} x_n g_{k-n}.$$

The example in (2.57) corresponds to a filter with $g_0 = 1$, $g_{-1} = -1$ and all other

$g_k = 0$. In the case

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} a & b & c & d & & & & & & \\ & & a & b & c & d & & & & \\ & & & & a & b & c & d & & \\ & & & & & & a & b & c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ 0 \end{bmatrix}, \tag{2.58}$$

we are applying a filter with four weights, a, b, c, and d, using a stride length of two. Because the dimension of the input vector x is not compatible with the filter length, we have *padded* with an extra zero value. In practice, image data is typically regarded as a three dimensional tensor: each pixel has two spatial coordinates and a red/green/blue value. With this viewpoint, the filter takes the form of a small tensor that is successsively applied to patches of the input tensor and the corresponding convolution operation is multi-dimensional. From a computational perspective, a key benefit of CNNs is that the matrix-vector products involved in the forward and backward passes through the network can be computed extremely efficiently using fast transform techniques.

A convolutional layer is often followed by a *pooling* layer, which reduces dimension by mapping small regions of pixels into single numbers. For example, when these small regions are taken to be squares of four neigboring pixels in a 2D image, a *max pooling* or *average pooling* layer replaces each set of four by their maximum or average value, respectively.

The application of CNNs to CFD has yielded several interesting results. For example, in the work of Tompson et al. [2017], Convolutional Neural Networks were employed to simulate raising smoke plumes. More applications of CNNs in CFD will appear in the future, since these systems have an inherent ability to detect macro patterns and apply corrections. However, it is important to remember that the predictive capabilities of CNNs should be harnessed within appropriate physical frameworks, such that their work can be simplified. These systems should never be employed to replace physical constraints which could be easily hard-coded. As a rule of thumb, every additional prediction expected from a Deep Learning system increases the amount of training data required substantially

2.2.3 Recurrent Neural Networks to Transformers

Classically, the temporal problems were handled using Long Short Term Memory (LSTM) neural networks. These networks employ an architecture closely resembling an electronic circuit, as it can be seen in 2.10. Despite the complexity of

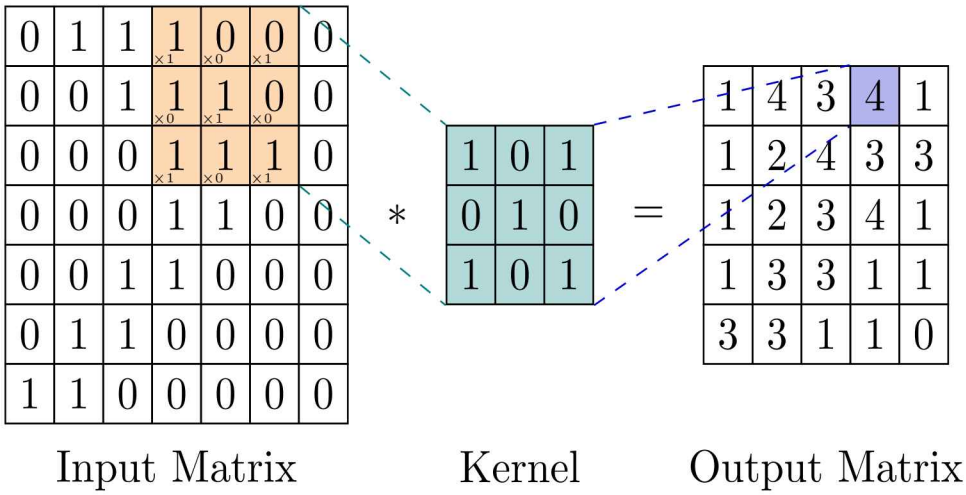


Figure 2.8: Convolution operation on spatial data where the convolution operation essentially performs dot products between the filters and local regions of the input

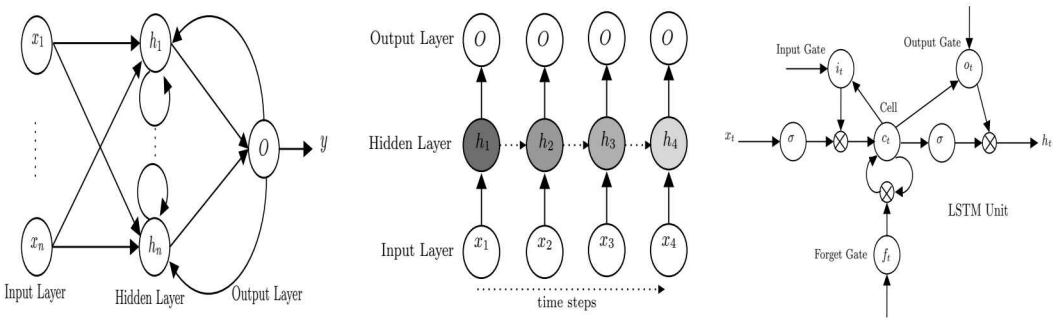


Figure 2.9: (Left) An example of Recurrent Neural Network (Center) Illustrated example of the vanishing gradient problem with an unfolded RNN where gradients get smaller at each time step as described by Lipton et al. [2015]. (Right) Inside an LSTM unit where the gated units input, output, and forget gate are present.

these systems, their use has massified due to their impressive ability to perform complex tasks, such as performing accurate language translations. One of the key features in LSTM networks is their ability to decide whether any given inputs should be transformed, remembered or forgotten. Thanks to these long-term dependencies, LSTM's can manage to navigate through sentences, character by character, and perform accurate translations. Language translation formally corresponds to a branch of Natural Language Processing (NLP), which attempts to build AI systems capable of processing human language. One important discovery made in this field is that there exist simpler units which can achieve a similar performance to LSTM units. These units are called Gate Recurrent Units (GRU). Here, it can be noted that GRU's present far less parameters than LSTM units indeed. This in turn implies that larger systems can be built, such that the overall system performance is increased for the same total number of parameters Chung et al. [2014] . Another important advance made in NLP is the use of bi-directional LSTM networks, which allow computers to process sentences from start to finish, and vice-versa. This has proven useful while performing translations between languages which invert the sentence order, such as English vs. Spanish. Beyond their success in language translation tasks, LSTM networks present a remarkable ability to learn complex data patterns.

Transformer models in deep learning were developed to address natural language processing problems, where sentence completion and translation are performed using a word by word embedding Vaswani et al. [2017]. The sentence-completing NLP tasks can be understood as a temporal learning problem, with a time series of words or sentence tensors measured over a time duration. These models have achieved remarkable performances in a variety of other tasks, including learning image patches as sequences, image completion and reconstruction Vaswani et al. [2017]; Dai et al. [2021]; Wu et al. [2021]. As a result, the transformer models have been challenging the classic long short-term memory (LSTM) models, the *de facto* RNNs, and replacing them with a state-of-the-art approach in a variety of temporal learning tasks.

Like RNNs, transformers are designed to handle sequential input data. However, unlike the latter, they do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence, and self-attention itself identifies/learns the weights of attention. In the case of spatio-temporal data, the attention can be applied to the spatial as well as the temporal sequence to attend to or pay attention to. The vanilla transformers in their original form are pure sequence to sequence models, as they learn a target output sequence from an input sequence, *i.e.* they perform transformation at the sequence level. Their limitations, such as disrupting temporal coherence and failing to capture long-term dependencies, were reached for sentence completion of language generation tasks, where difficulties were noted while generating texts with a model which learns sequences without the knowledge of full-sequences Bahdanau et al. [2014]; Yu et al. [2017]; Guo et al. [2018]. Several studies were performed, such as that of Dai *et al.* Dai et al. [2019], to address this inability to

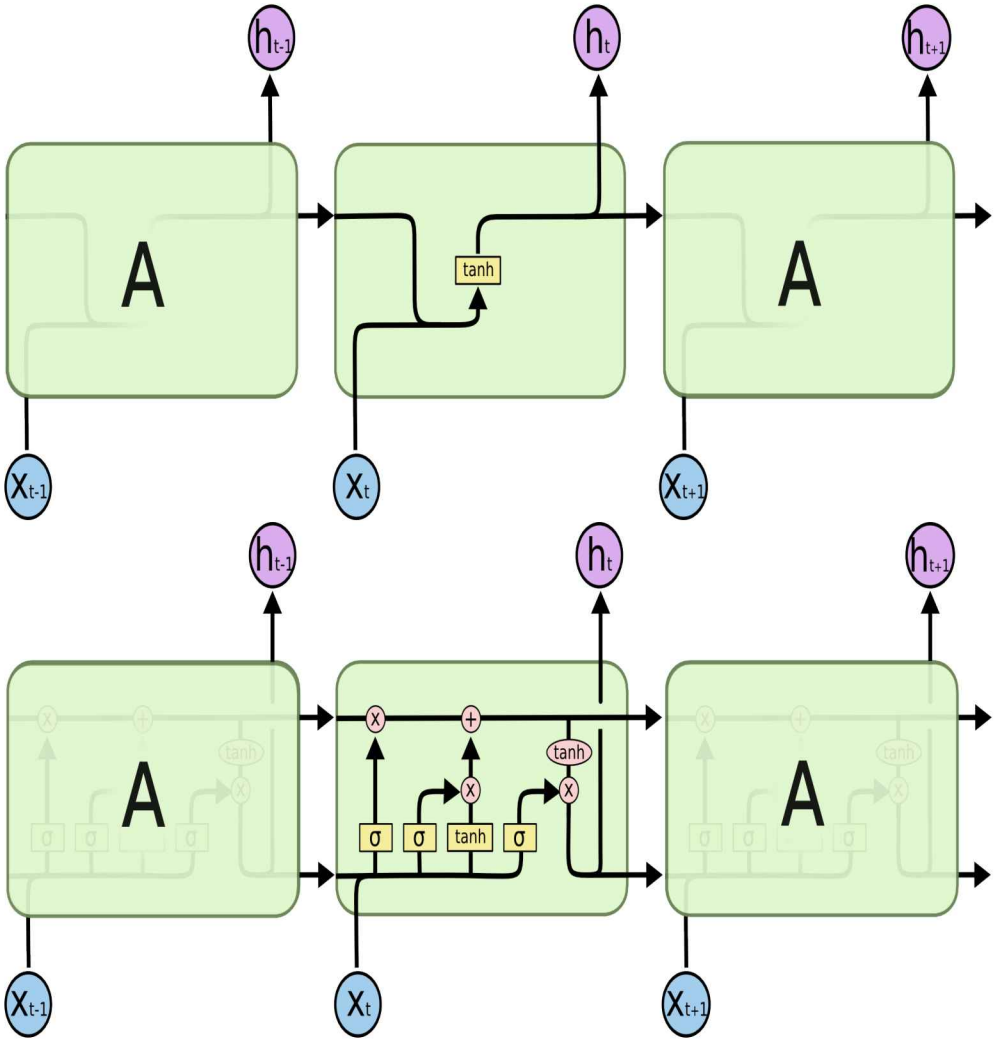


Figure 2.10: Schematic representations of a Long Short Term Memory (LSTM) neural network. Courtesy: Olah [2015]

capture long-term dependencies by attending to memories from previously learned parameters, yet at the expense of computing costs. To deal with some of these issues, autoregressive transformers were proposed by Katharopoulos et al. [2020] for sentence and image completion tasks. Although not explicitly stated in some works, the Generative Pre-trained Transformer (GPT) family of models Radford et al. [2018, 2019]; Brown et al. [2020] are in fact autoregressive transformers inspired by the decoder part of the original transformers. In Katharopoulos et al. [2020], Katharopoulous *et al.* showed that a self-attention layer trained in an autoregressive fashion can be seen as a recurrent neural network. Transformers can be combined with the classic convolutional encoder-decoder type models to harness their full potential when the input and target output tensors are in a spatio-temporal form. As locality is more important in learning small-scale features, this combination serves as a powerful method for a variety of computer-vision problems, including video-frame prediction. The self-attention mechanism on convolutional layers not only *attends* or focuses on a sequence of significance, but it also improves the representation of spatially-relevant regions by focusing on important features and suppressing less-important ones Woo et al. [2018].

When a transformer block is applied after a convolutional layer, the model learns to emphasize meaningful features along the channel sequence and spatial dimensions. The input sequences are first appended channel-wise to the input layer and subsequent convolutional operations are performed in the encoder. In the convolutional layers, the intermediate feature maps $\mathbb{F} \in \mathbb{R}^{C\times H\times W}$ of a given layer are passed through the self-attention convolutional transformer layer conv_α, which simultaneously attends to spatial representation and the positional embeddings of the input sequence channels. In conv_α, let $x, y \in \mathbb{R}^C$ be the input and output intermediate feature tensors, with C representing the number of intermediate channels. When $i, j \in \mathbb{R}^{H\times W}$ are indices of the spatial nodes, a classical convolution operation is performed such that:

$$y_i = \sum_{j\in N(i)} W_{i\to j} x_j, \tag{2.59}$$

where $N(i)$ represents the spatial nodes in a local neighborhood defined by a kernel of size $k \times k$ centered at node i, $i \to j$ represents the relative spatial relationship from i to j, and $W_{i\to j} \in \mathbb{R}^{C\times C}$ is the weight matrix. On the other hand, self-attention for intermediate convolutional features has three weight matrices $W_q, W_k, W_v \in \mathbb{R}^{C\times C}$ to compute query, key and value respectively. For each convolution window, the self-attention is given as:

$$\begin{aligned} y_i &= \sum_{j\in N(i)} \alpha_{i\to j} W_v x_j, \\ \alpha_{i\to j} &= \frac{e^{(W_q x_i)^T W_k x_j}}{\sum_{z\in N(i)} e^{(W_q x_i)^T W_k x_z}} = \frac{W_{qk} x_i[j]}{\sum_{z\in N(i)} W_{qk} x_i[z]}, \end{aligned} \tag{2.60}$$

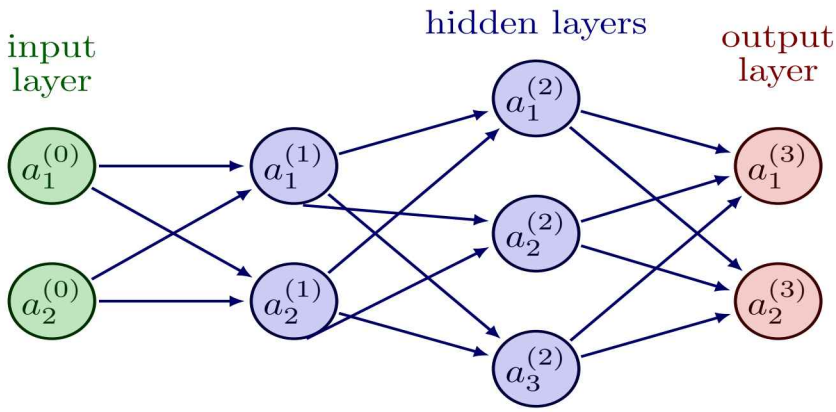


Figure 2.11: An example encoder-decoder network architecture

where the self-attention $\alpha_{i \to j} \in (0, 1)$ is a scalar that controls the contribution of values in spatial nodes, with $W_{qk} \in \mathbb{R}^{C \times k^2}$, and $[j]$ means j^{th} element of the tensor. α is usually normalized by a softmax operation such that $\sum_j \alpha_{i \to j} = 1$.

2.2.4 Encoder-Decoder Architecture

The idea behind *autoencoders* by McClelland et al. [1987] is, perhaps surprisingly, to produce an overall network whose output matches its input. More precisely, one network, known as the *encoder*, corresponds to a map F that takes an input vector, $x \in \mathbb{R}^s$, and produces a lower dimensional output vector $F(x) \in \mathbb{R}^t$. So $t \ll s$. Then a second network, known as the *decoder*, corresponds to a map G that takes us back to the same dimension as x; that is, $G(F(x)) \in \mathbb{R}^s$. We could then aim to minimize the sum of the squared error $\|x - G(F(x))\|_2^2$ over a set of training data. Note that this technique does not require the use of labelled data—in the case of images we are attempting to reproduce each picture without knowing what it depicts. Intuitively, a good encoder is a tool for dimension reduction. It extracts the key features. Similarly, a good decoder can reconstruct the data from those key features.

In turbulent flow problems, the state-space is high-dimensional, owing to the complex spatio-temporal dynamics involved. However, low dimensional features can be extracted, making relevant the use of dimensionality reduction techniques. Reconstruction and prediction problems are therefore equivalent to the estimation of the reduced or latent state, thus making the use of encoder-decoder based architecture a natural choice. The encoder-decoder architecture comprises an encoder that takes input tensors and maps them to a high-dimensional representation by learning which parts of the input tensors are important and converts or passes them to abstract low-dimensional representation. With the addition of a decoder after this encoder, this high-dimensional representation is converted to target output tensors. By chaining the encoder and decoder together, their

weight matrices jointly learn the output tensors from input tensors, thus helping to learn the small-scale features. The decoder is comprised of successive upsamplings followed by convolutions, and brings the latent space representation back to the original spatial dimensions of the target output.

2.3 Examples of Deep Learning Applied to Turbulence

During the past decade, the coupling of CFD algorithms with deep learning methods, and especially neural networks, have progressed rapidly. Such couplings offer new perspectives and opportunities to assist the existing CFD solvers, by leveraging the power of deep learning and provide an additional probe into our understanding of the modeling of turbulent flows. Industrial applications of such couplings include increase efficiency and accuracy for the simulation of complex flows, but also faster design cycles as well as empowered real-time digital twins. One of the early works on the use of neural networks in fluid dynamics was reported in Milano and Koumoutsakos [2002], where near-wall velocity fields were predicted by comparing the equivalency of prediction from neural networks with proper orthogonal decomposition-based reconstruction. Several works, such as Yarlanki et al. [2012]; Cheung et al. [2011]; Kato and Obayashi [2014], have used velocity field data to predict model parameters and their probability distributions to quantify and reduce modeling errors. A proper framework for using machine learning methods in the area of fluid dynamics was laid down since the works of Parish and Duraisamy [2016] and Xiao et al. [2016] in which the authors have demonstrated the use of these methods for turbulence modeling in the form of estimation of model uncertainties using machine learning. Overall, a paradigm for data-driven predictive modeling of turbulent flows by systematic implementation of machine learning and inverse modeling was described in Wang et al. [2017]; Singh et al. [2017b,c]; Singh and Duraisamy [2016]; Singh et al. [2017a]. Moreover, the direct prediction of Reynolds stresses for RANS and prediction of deconvoluted direct numerical simulation have been proposed in Ling and Templeton [2015]; Ling et al. [2016]; Vollant et al. [2017]; Maulik and San [2017]. Similar works involving re-generating turbulence statistics as well as super-resolution have been demonstrated in Fukami et al. [2019b]; Mohan et al. [2019]; Beck et al. [2019]; Kim and Lee [2019]; Fukami et al. [2019a, 2020] whereas Zhao et al. [2019]; Taghizadeh et al. [2020] investigated the coupling of RANS with machine learning optimization. Deep learning has also been utilized in CFD for a variety of related tasks such as drag prediction as described in Viquerat and Hachem [2020] and flow-reconstruction along with uncertainty estimation in Chen et al. [2021]. The use of machine learning in the turbulence modeling community has been summarized in the recent reviews by Duraisamy *et al.* Duraisamy et al. [2019] and Zang *et al.* Zhang et al. [2019].

The Spalart-Allmaras turbulence model has also been subject to machine

learning-based investigations in several works. In 2015, Tracey *et al.* Tracey et al. [2015] demonstrated one of the first works on the SA model using neural networks to predict a part of the RANS closure model (namely the source terms of the eddy viscosity transport). Their study features hand-picking of features from the SA eddy viscosity transport equation in order to predict its source term. Later on, Singh *et al.* Singh et al. [2017c] followed a similar procedure by exploiting hand-picking of input features deploying neural networks for the prediction of source terms of SA eddy viscosity transport, and later presented both a priori and a posteriori analysis. More recently, Liang et al. [2019]; Maulik et al. [2020] proposed to predict the eddy viscosity from input features consisting of data from Navier-Stokes and transport equations, while Pal [2020] used neural networks for predicting subgrid-scale viscosity in the geophysical applications.

Preliminary work on subgrid-scale modeling of turbulent flows using neural networks was done by Maulik et al. [2019c]. The authors proposed a methodology for the subgrid-scale modeling of Kraichnan turbulence with a homogenous two-dimensional case. Their input data to the neural network consisted of information on a nine-point stencil derived from the vorticity-stream function formulation. With these input data, they used a fully-connected multilayer preceptron(MLP) type neural network to predict the subgrid-scale quantity, which was a subgrid forcing term in their case. In the subsequent works Maulik et al. [2019b], used the same MLP-type neural networks to predict the probabilities of switching between a functional and structural subgrid-scale modeling approach. They also proposed a hybrid subgrid-scale term which was a combination of functional and structural subgrid-scale models multiplied by their respective probabilities. Similarly, in Maulik et al. [2019a], the authors proposed a method to switch between explicit and implicit subgrid-scale formulation based on the predictions provided by the neural network. On the choice of input-output parameters to the neural networks, Wang et al. [2018b] have tested and compared the relative significance of various flow variables. Xie et al. [2019] have used the MLP-type neural networks to predict subgrid-scale stress and energy in case of an isotropic & three-dimensional compressible turbulence. Bode et al. [2019a] used an enhanced super-resolution generative adversarial network (ESR-GAN by Wang et al. [2018a]) architecture to reconstruct DNS-like velocity fields from filtered velocity fields in case of a homogenous isotropic turbulence DNS. In subsequent work, Bode et al. [2019b] made use of the same ESR-GAN but with cost-functions inspired by physical laws, such as mass conservation, and trained with a very high-resolution DNS to obtain subfilter terms for momentum and scalar mixing. Recently, Kim and Lee [2020] have predicted turbulent heat transfer using the convolutional neural networks. They have also investigated the interpretation of learned convolutional kernels and the filters which marks an important step towards using deep learning to understand the physics of fluids.

The computer graphics community has also made use of deep learning methods to speed up simulations for generating realistic animations of fluids such as water and smoke to be used in applications such as computer gaming and movies.

For example, Tompson et al. [2017] used an incompressible Euler's equation with a customized Convolutional Neural Network (CNN) to predict velocity updates within a finite difference method solver. Chu and Thuerey [2017] proposes double CNN networks to synthesize high-resolution flow simulation based on reusable space-time regions. Xie et al. [2018] developed deep learning models in the context of fluid flow animation, where physical consistency is less critical. Steffen Wiewel [2019] proposed a method for the data-driven inference of temporal evolutions of physical functions with deep learning. However, fluid animation emphases on the realism through various camera angles of the simulation rather than the physical consistency of the predictions or physics metrics and diagnostics of relevance to scientists. Such approaches, though not directly applicable to numerical simulation in the traditional context, are inspirational in terms of the complex learning architectures.

Bibliography

Igor Aizenberg and Alexander Gonzalez. Image recognition using mlmvn and frequency domain features. In *2018 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Andrea Beck, David Flad, and Claus-Dieter Munz. Deep neural networks for data-driven les closure models. *Journal of Computational Physics*, 398:108910, 2019.

Mathis Bode, Michael Gauding, Konstantin Kleinheinz, and Heinz Pitsch. Deep learning at scale for subgrid modeling in turbulent flows. *arXiv preprint arXiv:1910.00928*, 2019a.

Mathis Bode, Michael Gauding, Zeyu Lian, Dominik Denker, Marco Davidovic, Konstantin Kleinheinz, Jenia Jitsev, and Heinz Pitsch. Using physics-informed super-resolution generative adversarial networks for subgrid modeling in turbulent reactive flows. *arXiv preprint arXiv:1911.11380*, 2019b.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Julien Bruchon, Hugues Digonnet, and Thierry Coupez. Using a signed distance function for the simulation of metal forming processes: Formulation of the contact condition and mesh adaptation. from a lagrangian approach to an eulerian approach. *International journal for numerical methods in engineering*, 78(8): 980–1008, 2009.

Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual review of fluid mechanics*, 52:477–508, 2020.

Junfeng Chen. Reseaux de neurones convolutifs pour la prediction de flux constant autour d'obstacles 2d. *Ph.D. Thesis*, 2022.

Junfeng Chen, Jonathan Viquerat, Frederic Heymes, and Elie Hachem. A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2d obstacles. *arXiv preprint arXiv:2104.03619*, 2021.

Sai Hung Cheung, Todd A. Oliver, Ernesto E. Prudencio, Serge Prudhomme, and Robert D. Moser. Bayesian uncertainty analysis with applications to turbulence modeling. *Reliability Engineering System Safety*, 96(9):1137–1149, 2011. ISSN 0951-8320. doi: https://doi.org/10.1016/j.ress.2010.09.013. Quantification of Margins and Uncertainties.

Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Transactions on Graphics (TOG)*, 36(4):69, 2017.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

Theodore De Karman and Leslie Howarth. On the statistical theory of isotropic turbulence. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 164(917):192–215, 1938.

Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annu. Rev. Fluid Mech.*, 51:357–377, 2019.

Charles L Fefferman. Existence and smoothness of the navier-stokes equation. *The millennium prize problems*, 57:67, 2000.

Uriel Frisch and Andreĭ Kolmogorov. *Turbulence: the legacy of AN Kolmogorov*.

Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. *J. Fluid Mech.*, 870:106–120, 2019a.

Kai Fukami, Yusuke Nabae, Ken Kawai, and Koji Fukagata. Synthetic turbulent inflow generator using machine learning. *Physical Review Fluids*, 4(6):064603, 2019b.

Kai Fukami, Koji Fukagata, and Kunihiko Taira. Machine learning based spatio-temporal super resolution reconstruction of turbulent flows. *arXiv preprint arXiv:2004.11566*, 2020.

Massimo Germano, Ugo Piomelli, Parviz Moin, and William H Cabot. A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A: Fluid Dynamics*, 3(7): 1760–1765, 1991.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Elie Hachem, Stephanie Feghali, Ramon Codina, and Thierry Coupez. Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation. *International journal for numerical methods in engineering*, 94(9):805–825, 2013.

Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *Siam review*, 61(4):860–891, 2019.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

Ghina Jannoun, Elie Hachem, Jerôme Veysset, and Thierry Coupez. Anisotropic meshing with time-stepping control for unsteady convection-dominated problems. *Applied Mathematical Modelling*, 39(7):1899–1916, 2015.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

Hiroshi Kato and Shigeru Obayashi. *Data Assimilation for Turbulent Flows*, chapter AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan 2014. doi: 10.2514/6.2014-1177. 0.

John Kim, Parviz Moin, and Robert Moser. Turbulence statistics in fully developed channel flow at low reynolds number. *Journal of fluid mechanics*, 177: 133–166, 1987.

Junhyuk Kim and Changhoon Lee. Deep unsupervised learning of turbulence for inflow generation at various reynolds numbers. *arXiv preprint arXiv:1908.10515*, 2019.

Junhyuk Kim and Changhoon Lee. Prediction of turbulent heat transfer using convolutional neural networks. *Journal of Fluid Mechanics*, 882, 2020.

Andrey Nikolaevich Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. *Cr Acad. Sci. URSS*, 30:301–305, 1941.

Lev Davidovich Landau and Evgenii Mikhailovich Lifshitz. *Fluid Mechanics: Landau and Lifshitz: Course of Theoretical Physics, Volume 6*, volume 6. Elsevier, 2013.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

SUN Liang, AN Wei, LIU Xuejun, and LYU Hongqiang. On developing data-driven turbulence model for dg solution of rans. *Chinese Journal of Aeronautics*, 32(8):1869–1884, 2019.

Julia Ling and J Templeton. Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty. *Phys. Fluids*, 27(8):085103, 2015.

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

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Rainald Löhner and Joseph D Baum. Handling tens of thousands of cores with industrial/legacy codes: Approaches, implementation and timings. *Computers & Fluids*, 85:53–62, 2013.

Stéphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203, 2016.

Romit Maulik and Omer San. A neural network approach for the blind deconvolution of turbulent flows. *Journal of Fluid Mechanics*, 831:151–181, 2017.

Romit Maulik, Omer San, and Jamey D Jacob. Connecting implicit and explicit large eddy simulations of two-dimensional turbulence through machine learning. *arXiv preprint arXiv:1901.09329*, 2019a.

Romit Maulik, Omer San, Jamey D Jacob, and Christopher Crick. Sub-grid scale model classification and blending through deep learning. *Journal of Fluid Mechanics*, 870:784–812, 2019b.

Romit Maulik, Omer San, Adil Rasheed, and Prakash Vedula. Subgrid modelling for two-dimensional turbulence using neural networks. *Journal of Fluid Mechanics*, 858:122–144, 2019c.

Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. A turbulent eddy-viscosity surrogate modeling framework for reynolds-averaged navier-stokes simulations. *Computers & Fluids*, page 104777, 2020.

James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, volume 2. MIT press, 1987.

Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence. *arXiv preprint arXiv:1903.00033*, 2019.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

Franck Nicoud and Frédéric Ducros. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. *Flow, turbulence and Combustion*, 62 (3):183–200, 1999.

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. *Physics of Fluids*, 23(8):085106, 2011.

Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

Christopher Olah. Understanding lstm networks. 2015.

Anikesh Pal. Deep learning emulation of subgrid-scale processes in turbulent shear flows. *Geophysical Research Letters*, 47(12):e2020GL087005, 2020.

Eric J. Parish and Karthik Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics*, 305:758–774, 2016.

Stephen B Pope. Turbulent flows, 2001.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Lewis F Richardson. *Weather prediction by numerical process*. University Press, 1922.

Pierre Sagaut. *Large eddy simulation for incompressible flows: an introduction*. Springer Science & Business Media, 2006.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Anand Pratap Singh and Karthik Duraisamy. Using field inversion to quantify functional errors in turbulence closures. *Physics of Fluids*, 28(4):045110, 2016.

Anand Pratap Singh, Karthikeyan Duraisamy, and Ze Jia Zhang. Augmentation of turbulence models using field inversion and machine learning. In *55th AIAA Aerospace Sciences Meeting*, page 0993, 2017a.

Anand Pratap Singh, Racheet Matai, Asitav Mishra, Karthikeyan Duraisamy, and Paul A Durbin. Data-driven augmentation of turbulence models for adverse pressure gradient flows. In *23rd AIAA Computational Fluid Dynamics Conference*, page 3626, 2017b.

Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal*, pages 1–13, 2017c.

Joseph Smagorinsky. General circulation experiments with the primitive equations: I. the basic experiment. *Monthly weather review*, 91(3):99–164, 1963.

Philippe R Spalart, Robert D Moser, and Michael M Rogers. Spectral methods for the navier-stokes equations with one infinite and two periodic directions. *Journal of Computational Physics*, 96(2):297–324, 1991.

Katepalli R Sreenivasan. On the universality of the kolmogorov constant. *Physics of Fluids*, 7(11):2778–2784, 1995.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Nils Thuerey Steffen Wiewel, Moritz Becher. Latent-space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum*, 38, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Salar Taghizadeh, Freddie D Witherden, and Sharath S Girimaji. Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations. *arXiv preprint arXiv:2004.03031*, 2020.

Geoffrey Ingram Taylor. Statistical theory of turbulenc. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 151(873): 421–444, 1935.

Hendrik Tennekes, John Leask Lumley, Jonh L Lumley, et al. *A first course in turbulence*. MIT press, 1972.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

Brendan D Tracey, Karthikeyan Duraisamy, and Juan J Alonso. A machine learning strategy to assist turbulence model development. In *53rd AIAA aerospace sciences meeting*, page 1287, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jonathan Viquerat and Elie Hachem. A supervised neural network for drag prediction of arbitrary 2d shapes in laminar flows at low reynolds number. *Computers & Fluids*, page 104645, 2020.

Antoine Vollant, Guillaume Balarac, and C Corre. Subgrid-scale scalar flux modelling based on optimal estimation theory and machine-learning procedures. *J. Turbul.*, 18(9):854–878, 2017.

Haohan Wang and Bhiksha Raj. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.

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

Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018a.

Zhuo Wang, Kun Luo, Dong Li, Junhua Tan, and Jianren Fan. Investigations of data-driven closure for subgrid-scale stress in large-eddy simulation. *Physics of Fluids*, 30(12):125101, 2018b.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021.

H Xiao, J-L Wu, J-X Wang, R Sun, and CJ Roy. Quantifying and reducing model-form uncertainties in reynolds-averaged navier–stokes simulations: A data-driven, physics-informed bayesian approach. *Journal of Computational Physics*, 324:115–136, 2016.

Chenyue Xie, Jianchun Wang, Hui Li, Minping Wan, and Shiyi Chen. Artificial neural network mixed model for large eddy simulation of compressible isotropic turbulence. *Physics of Fluids*, 31(8):085112, 2019.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

S. Yarlanki, B. Rajendran, and H. Hamann. Estimation of turbulence closure coefficients for data centers using machine learning algorithms. In *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pages 38–42, May 2012.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Xinlei Zhang, Jinlong Wu, Olivier Coutier-Delgosha, and Heng Xiao. Recent progress in augmenting turbulence models with physics-informed machine learning. *Journal of Hydrodynamics*, 31(6):1153–1158, 2019.

Yaomin Zhao, Harshal D Akolekar, Jack Weatheritt, Vittorio Michelassi, and Richard D Sandberg. Turbulence model development using cfd-driven machine learning. *arXiv preprint arXiv:1902.09075*, 2019.

Chapter 3

Robust Learning for Turbulent Flows

Des modèles les plus simples aux réseaux neuronaux profonds complexes, la modélisation de la turbulence à l'aide de techniques d'apprentissage automatique présente encore de nombreux défis. Dans ce contexte, la présente contribution propose une stratégie robuste utilisant l'apprentissage par patch pour apprendre la viscosité turbulente à partir des vitesses d'écoulement et démontre son utilisation efficace sur le modèle de turbulence Spalart-Allmaras. Des ensembles de données d'entraînement sont générés pour l'écoulement à travers des obstacles bidimensionnels à des nombres de Reynolds élevés et utilisés pour entraîner un réseau neuronal convolutionnel de type auto-encodeur avec des entrées de patchs locaux. Par rapport à une technique d'apprentissage standard, l'apprentissage par patchs permet non seulement d'améliorer la précision, mais aussi de réduire les coûts de calcul nécessaires à l'apprentissage.

3.1 Introduction

Computational fluid dynamics (CFD) is an essential asset for research and industrial applications. Despite advances in computational power over the years, industrial CFD tools still largely rely on the Reynolds Averaged Navier-Stokes (RANS) turbulence models due to cost-savings and lesser time-to-solution offered by RANS when compared with intensive Large Eddy Simulations (LES) and Direct Numerical Simulations (DNS), especially for flows at high-Reynolds numbers. Among the variety of one-equation to many equations RANS models, the Spalart Allmaras (SA) turbulence model Spalart and Allmaras [1992], that solves for the kinematic eddy turbulent viscosity, and has not been derived from the existing turbulent kinetic energy-based RANS models. In this sense, it can be described as a proper one-equation model which does not require knowledge of a specific problem and additional advantages include numerical stability as well as reliability for convergence of results. Due to these reasons, the SA model has been widely used, documented, and serves as a benchmark turbulence model for many CFD applications Ferziger et al. [2002]; Spalart [2000].

The Spalart-Allmaras turbulence model has also been subject to machine learning-based investigations in several works. In 2015, Tracey *et al.* Tracey et al. [2015] demonstrated one of the first works on the SA model using neural networks to predict a part of the RANS closure model (namely the source terms of the eddy viscosity transport). Their study features hand-picking of features from the SA eddy viscosity transport equation in order to predict its source term. Later on, Singh *et al.* Singh et al. [2017] followed a similar procedure by exploiting hand-picking of input features deploying neural networks for the prediction of source terms of SA eddy viscosity transport, and later presented both a priori and a posteriori analysis. More recently, Liang et al. [2019]; Maulik et al. [2020] proposed to predict the eddy viscosity from input features consisting of data from Navier-Stokes and transport equations, while Pal [2020] used neural networks for predicting subgrid-scale viscosity in the geophysical applications.

Although convolutional neural networks are traditionally trained using full-scale inputs, patch-wise deep learning models have been successfully applied in the past in the computer-vision community. In particular, Long *et al.* proposed a work on object detection Long et al. [2015] where data was spatially divided into patches of information, which were then fed to the model with a primary intention of reducing memory consumption during training. This study proved efficient not only in terms of error reduction, but also in memory consumption during the training. Also, it has been demonstrated in the image classification tasks in Farabet et al. [2012]; Pinheiro and Collobert [2014] that the patch-wise training can correct class imbalance as well as assist in the spatial correlation of dense patches. More than saving training memory, patch-wise training offers a great opportunity for deep learning research in CFD, primarily because the spatial data can be divided into small patches of neighboring nodes to achieve global as well as local learning, independent of the size of the domain. It also

offers a way for CFD data augmentation by increasing the quantity of the same data with flipping and rotation of patches.

In the present contribution, we aim at learning the SA turbulent viscosity from the velocity field using a convolutional neural network trained following a patch-based approach. The proposed novelties are (i) the blind-learning of SA eddy viscosity from input velocities, without any hand-picking, manual feature selection, non-dimensionalization, or tailored losses, and (ii) a novel patch-based learning strategy with an auto-encoder type convolutional neural network, provided as a way towards generalized deep learning in turbulence modeling. The remaining of the paper is organized as follows: first, the problem setup and its governing equations are described, and the methods used for the dataset generation are covered; then, the selected network architecture is presented, and the patch-based learning procedure is described thoroughly; finally, the interests of the proposed approach are assessed, and results are discussed and compared to baseline solutions.

3.2 Problem setup & data generation

3.2.1 Governing equations

The evolution of the velocity $\boldsymbol{u}$ and pressure p in an incompressible fluid flow with given positive constant density ρ and dynamic viscosity μ is governed by the Navier-Stokes equations:

$$\begin{cases} \rho \left(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}\right) - \nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f}, \\ \nabla \cdot \boldsymbol{u} = 0, \end{cases} \tag{3.1}$$

where $\boldsymbol{\sigma} = 2\mu\, \boldsymbol{\varepsilon}(\boldsymbol{u}) - p\, \mathbf{I}_d$ is the Cauchy stress tensor for a Newtonian fluid, $\boldsymbol{\varepsilon}(\boldsymbol{u})$ the strain-rate tensor, and $\mathbf{I}_d$ the d-dimensional identity tensor. Equations (5.6) are supplemented with adequate boundary and initial conditions, to be specified. Reynolds-Averaged Navier-Stokes (RANS) equations are then obtained by applying the Reynolds decomposition to the system (5.6), such that velocity and pressure are expressed as the sum of a mean-field and a fluctuation. Applying a time averaging operator to the resulting expressions yields a forcing term under the form of the divergence of the so-called Reynolds stress tensor. The latter consists of correlations of velocity fluctuations and accounts for the effect of the turbulent fluctuations on the averaged flow. In the Boussinesq approximation, first-order closure of the system of averaged equations amounts to a mean gradient hypothesis: turbulence is therefore modelled as an additional diffusivity called eddy viscosity μ_t. The eddy viscosity μ_t itself proceeds from a model involving one or more turbulent scales, each of which is the solution of a nonlinear convection-diffusion-reaction equation. For additional details, the reader is referred to the works of Pope [2001] on turbulent flows.

The turbulence model chosen to compute the eddy viscosity is the one-equation

Spalart-Allmaras (SA) model Spalart and Allmaras [1992], which describes the evolution of the kinematic eddy viscosity by solving a convection-diffusion-reaction problem and serves as baseline for future testing of other models. Applying this model, the eddy viscosity μ_t in the Navier-Stokes equations is obtained by $\mu_t = \rho\,\nu_t f_{v1}$, where f_{v1} is a given damping function to enforce linear profile in the viscous sublayer. The turbulent scale ν_t is itself governed by the following nonlinear convection-diffusion-reaction equation:

$$\frac{\partial \nu_t}{\partial t} + \boldsymbol{u}\cdot\nabla\nu_t - c_{b1}(1-f_{t2})\tilde{S}\nu_t + \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left(\frac{\nu_t}{d}\right)^2 - \frac{c_{b2}}{\sigma}\nabla\nu_t\cdot\nabla\nu_t - \frac{1}{\sigma}\nabla\cdot[(\nu+\nu_t)\nabla\nu_t] = 0 \tag{3.2}$$

where d is the distance to the nearest wall boundary, $\sigma = 2/3$, and $\tilde{S}$ is the modified vorticity magnitude given as,

$$\tilde{S} = S + \frac{\nu_t}{\kappa^2 d^2} f_{v2}, \qquad S = \sqrt{2W(\boldsymbol{u}) : W(\boldsymbol{u})},$$

Here $\kappa = 0.4$ is the von Kármán constant, W is the rotation-rate tensor, f_{v2} is a damping function to enforce the logarithmic profile, with other damping functions given as:

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \qquad \chi = \frac{\nu_t}{\nu}, \qquad f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}} \qquad f_{t2} = c_{t3}e^{-c_{t4}\chi^2}$$

$$f_w = g\left[\frac{1+c_{w3}^6}{g^6 + c_{w3}^6}\right]^{\frac{1}{6}}, \qquad g = r + c_{w2}(r^6 - r), \qquad r = \frac{\nu_t}{\tilde{S}\kappa^2 d^2},$$

and model coefficients are specified as:

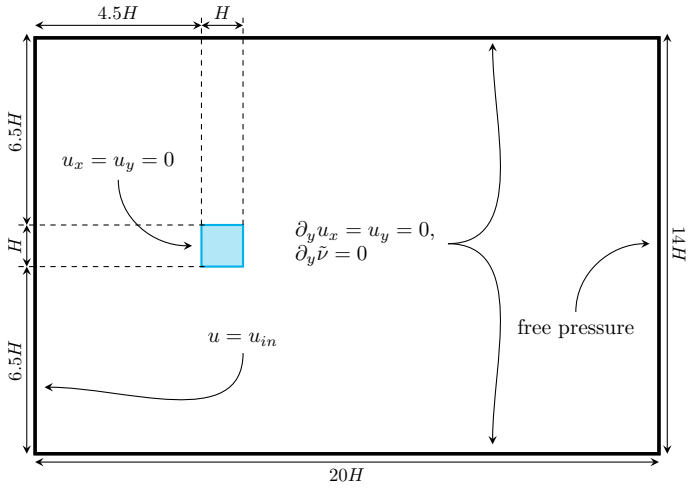
$$c_{b1} = 0.1355 \quad , \; c_{b2} = 0.622 \quad , \; c_{v1} = 7.1 \quad , \; c_{v2} = 0.7 \quad , \; c_{v3} = 0.9$$

$$c_{w1} = \frac{c_{b1}}{\kappa} + \frac{1+c_{b2}}{\sigma} \quad , \; c_{w2} = 0.3 \quad , \; c_{w3} = 2 \quad , \; c_{t3} = 1.2 \quad , \; c_t = 0.5.$$

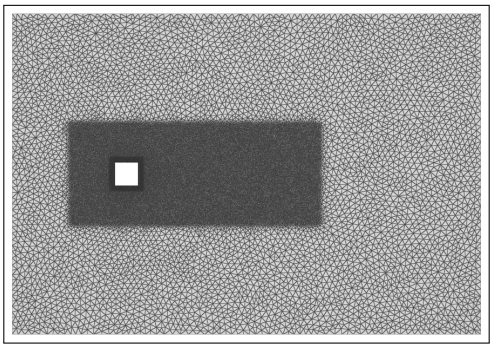
From dimensional considerations, ν_t is proportional to the product of characteristic length and velocity, and as a result proportional to the Reynolds number:

$$\nu_t \propto uL \sim f(Re) \tag{3.3}$$

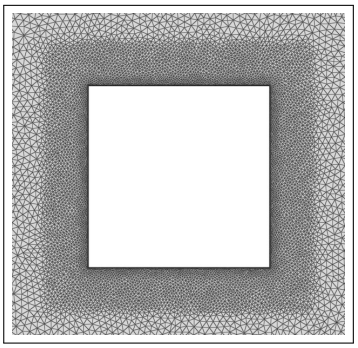
More details on the implementation of this model can be found in Guiza et al. [2020], and more details on the turbulent viscosity models can be found in Pope [2001]. Variants of the SA model exist in the literature, most of which are collected in NASA's turbulence modeling resource webpage Rumsey et al. [2010]. In this present work, the *negative Spalart-Allmaras Model* was selected due to its capability to avoid the generation of negative turbulent viscosity without the use of clipping Allmaras and Johnson [2012]. These equations were cast into a stabilized finite element formulation and solved using an in-house variational multi-scale solver CimLib CFD Hachem et al. [2013]. For additional details, the reader is referred to Hachem et al. [2013] and Guiza et al. [2020]



(a) Sketch of the considered problem (not to scale)



(b) Associated mesh



(c) Zoom on the cylinder area

Figure 3.1: **2D square cylinder configuration and mesh used for the study.** (5.3a) The cylinder lateral size is denoted H, and is centered at the origin of the domain. The dimensions of the computational domain are $[-5H, 15H] \times [-7H, 7H]$ in the streamwise x and crosswise y directions. (3.1b)-(3.1c) The mesh used for CFD computations is refined along with mesh-convergence.

3.2.2 Datasets of turbulent flow around obstacle

We consider the widely benchmarked turbulent flow past a two-dimensional (2D) square cylinder Rodi et al. [1997]; Guiza et al. [2020]. A sketch of the problem, including its dimensions, is presented in figure 3.1, along with the associated mesh. The baseline Reynolds number is set to 22×10^3, based on the inlet velocity and the cylinder diameter. The inflow boundary conditions are $\boldsymbol{u} = (V_{in}, 0)$, together with $\tilde{\nu} = 3\nu$, which corresponds to a ratio of eddy to kinematic viscosity of approximately 0.2. For the lateral boundaries, we use symmetry conditions $\partial_y u_x = u_y = 0$ and $\partial_y \tilde{\nu} = 0$. For the outflow, $\partial_x u_x = \partial_x u_y = 0$, $\partial_x \tilde{\nu} = 0$ together with $p = 0$ are prescribed. Finally, no-slip conditions $\boldsymbol{u} = 0$ and $\tilde{\nu} = 0$ are imposed at the cylinder surface.

Following the problem setup and methods, a baseline dataset (hereafter re-

Table 3.1: **Datasets generated for the present study.** Two types of obstacles and three different Reynolds numbers are considered, resulting in six different datasets, each holding 3000 snapshots of steady-state velocities and turbulent velocities.

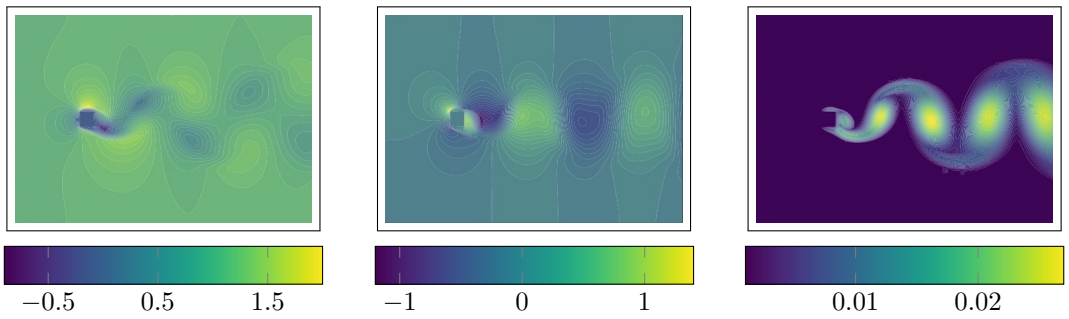
Dataset name	Re	Obstacle type
SqRe22k	22×10^3	2D square
SqRe44k	44×10^3	2D square
SqRe88k	88×10^3	2D square
CyRe22k	22×10^3	2D cylinder
CyRe44k	44×10^3	2D cylinder
CyRe88k	88×10^3	2D cylinder

ferred to as SqRe22k) composed of 3000 snapshots of steady velocities and SA turbulent viscosities is generated by skipping the transient regime and storing the established regime (*i.e.* each snapshot is captured only after the flow is established). Each snapshot is sampled on a rectilinear grid having spatial dimensions of $(N_x \times N_y) = (360 \times 300)$. The sampling on a rectilinear grid was performed to facilitate the use of CFD data coming from unstructured meshes. The same rectilinear grid was used to perform sampling on the square and circular obstacles. For points inside the obstacle, the velocities and turbulent viscosities were zeroed out, following the no-slip boundary conditions on the obstacle. In practice, it would be possible to skip the unstructured-to-structured sampling by making use of the graph neural networks, as presented in recent works Chen et al. [2021]. The dataset is deliberately not normalized to achieve robust and generalizable training. For testing purposes, additional datasets are also generated by changing the obstacle to a 2D circular cylinder, and by modifying the Reynolds number. As is summarised in table 3.1, six different datasets are obtained. Sample snapshots of velocity and turbulent viscosity from SqRe22k are shown in figure 3.2. In the following, the training subset is composed of 75% of the SqRe22k samples and 25% of the CyRe44k samples, while the remaining samples are reserved for the validation and testing subsets (each of the latter is therefore composed of 12.5% of the SqRe22k and 37.5% of the CyRe44k).

3.3 Network architecture and training procedure

3.3.1 Deep learning model

Given the input dataset $\boldsymbol{x}$ (here, the velocity snapshots from the RANS simulations) and the desired output dataset $\boldsymbol{y}$ (here, the turbulent viscosity snapshots from the RANS simulations), we desire to find the optimal set of weights and



(a) u field (b) v field (c) $\tilde{\nu}$ field

Figure 3.2: **Snapshot of velocities u (3.2a), v (3.2b), and turbulent viscosity $\tilde{\nu}$ (3.2c)** from dataset SqRe22k.

biases $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$ in a deep-learned model f such that $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{y}$. The set of free parameters $\boldsymbol{\theta}$ is optimized using Adam Kingma and Ba [2014], in order to iteratively minimize the mean squared error (MSE) loss defined as:

$$\mathcal{L} = \frac{1}{n_s} \sum_{i=1}^{n_s} (\boldsymbol{y}^i - f(\boldsymbol{x}; \boldsymbol{\theta})^i)^2, \tag{3.4}$$

where n_s is the number of samples. The full training dataset is shown repeatedly to the network after a shuffling step during the training, and each pass is referred to as an *epoch*. An early stopping criterion is used along with a reduction of learning rate if learning doesn't improve after every 100 epochs. The neural network was implemented using TensorFlow Abadi et al. [2016], and trained on an Nvidia Tesla V100 GPU.

The network architecture proposed for the present work is an *auto-encoder* structure Hinton and Salakhutdinov [2006] . Auto-encoders contain two parts: (i) a converging part that decreases the spatial dimension of the input (the encoder) and compresses the input using successive convolutions, and (ii) a diverging part that rebuilds a predicted output of the same size as input (the decoder). The encoder and decoder handle the spatial-dimensionality reduction by compressing the high-dimensional spatial data, using convolutional layers, to a low-dimensional representation called latent space. For example, a $N_y \times N_z$ feature map can be reduced to $N_y/2 \times N_z/2$ using a convolutional layer with a stride of 2. An essential aspect of this operation is that it preserves the most important features of the map. To increase robustness and generalization of the trained model, data standardization was not performed. Instead, batch normalization layers were used, which apply a transformation that maintains the mean and standard deviation of output close to 0 and 1, respectively. The proposed network architecture is shown in figure 3.3. In the literature, similar architectures (trained with full-scale inputs) were successfully exploited for studies focusing on turbulent flows Fukami et al. [2019]; Mohan et al. [2019].

The convolutional filters used in the proposed architecture incorporate a symmetric boundary condition into the padding operation. Classically, padding is

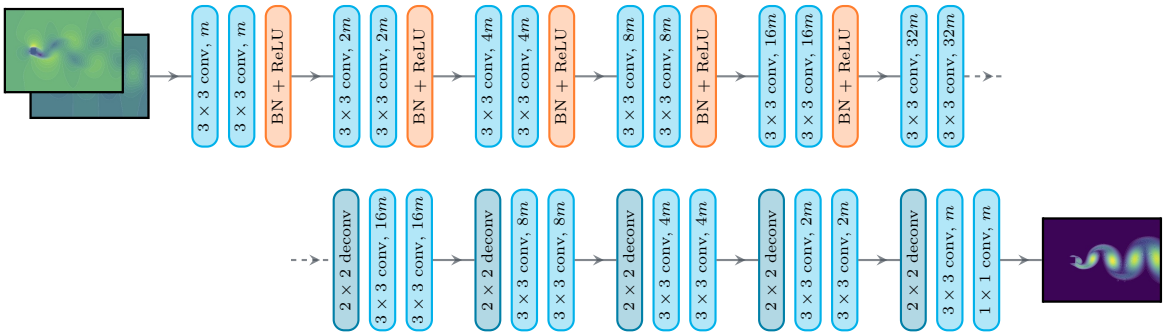
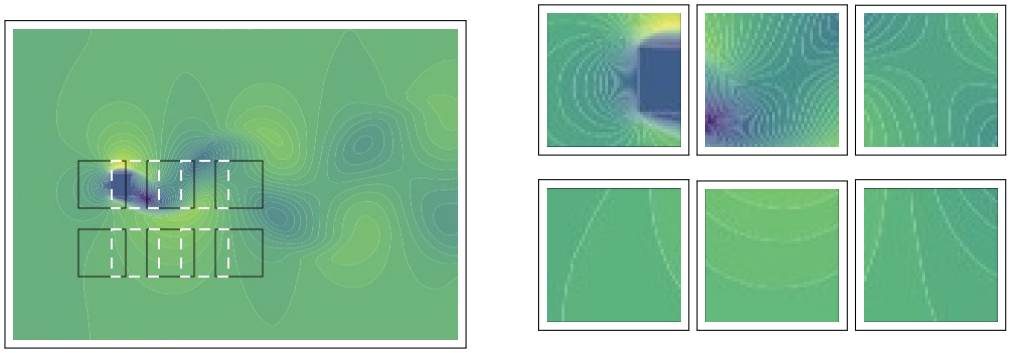


Figure 3.3: **Proposed auto-encoder network architecture.** The encoder branch is based on a convolution-convolution-batch-normalization pattern: the first convolution has a stride of $s = 1$, while the second has a stride of $s = 2$. The batch-normalization layer is followed by a rectified linear unit (ReLU) layer. At each occurrence of the pattern, the spatial dimensions are divided by two, while the number of filters, noted m, is doubled. In the decoder branch, a transposed convolution step is first applied to the input from the previous layer, while the number of filters is halved and two convolution layers are applied. At the end of the last layer, a 1×1 convolution is applied to obtain the final output.

used to preserve the spatial dimensions of the field being convoluted, but the standard zero-padding approach doesn't usually represent the expected physical behavior. Indeed, padding with zeros everywhere would violate the representation of existing boundary conditions, for example, the notion of wall-boundaries would have lesser significance if a region is padded with zeros on all the sides in a channel flow Patil and Lapyere [2019] . To preserve the boundary conditions after multiple successive convolutions, a boundary condition formulation was implemented such that the walls could be padded with zeros if required, while the periodic sides could be padded with adequate values from the periodic cells. The ReLU function was used as an activation function, which is known to be an effective tool for stabilizing the weight update in the machine learning process Nair and Hinton [2010].

3.3.2 Patch-based training procedure

We remind the goal of the present work, which is to train a deep learning model to infer the turbulent viscosity $\tilde{\nu}$ at every grid point from the velocities (u, v) at the same position. As underlined earlier, no data-preprocessing tasks such as normalization or standardization were used, and the input-output fields were used "as is" from the RANS simulation output. Similar to splitting between training and validation dataset as described in section 3.2.2, we use a mixture of the SqRe22k and CyRe44k datasets. The first stage of patch-based learning consists of dividing each snapshot of the dataset into smaller $n \times n$ overlapping patches with stride s, as is shown in figure 3.4. In this case, the number of patches



(a) Original snapshot (b) Extracted patches

Figure 3.4: **Patch extraction from u field**. Patches in figure (3.4b) are obtained from the original snapshot (3.4a) For better clarity of the figure, overlapping is only applied in the horizontal direction, and different colors are used to differentiate overlapping patches. Similarly, patches at the same corresponding locations are taken for v and $\tilde{\nu}$ fields.

obtained can be doubled by considering an up-down flipping transformation on the same snapshot.

For a baseline comparison, the proposed network is also trained conventionally over the full-spatial field dimensions, without using patch-based learning (this training method is hereafter referred to as M1). In this context, the batch size is 32, and the learning rate is 0.001. When a sufficient accuracy level is reached and no more improvement is observed, the training is terminated using the early-stopping criterion. A decent accuracy after convergence is obtained for both training and validation subsets, with a mean-squared error of 1×10^{-6}, as presented on the learning curve in figure 3.5. Total training time is 0.85 hours on a Tesla V100 GPU card, for 28 million degrees of freedom.

For patch-based training, patches from the different samples are randomly shuffled together and presented to the network in batches of size 32, with a learning rate equal to 0.001 (this training method is hereafter referred to as M2). Baseline values for the patch size n and the stride s are chosen to be 50 and 75, respectively, but their respective impact on the training performance is evaluated in section 3.4. Similarly, the impact of batch size is assessed in the following section. The model is trained for 850 epochs, after which the accuracy stops improving, resulting in a final MSE error of the order of 1×10^{-7}, *i.e.*one order of magnitude lower than that of method M1. Total training time is 2.38 hours for 1.7 million degrees of freedom. Although this represents about 3 times the training time of method M1, it must be noticed that the final M2 accuracy is significantly lower than that of M1, as is visible in figure 3.5. More, the final generalization level is also superior, evidenced by the negligible gap between validation and training curves. As the patch-based approach grounds the learning in a local velocity-to-turbulent-viscosity inference, it is argued that the trained network is able to re-use local mappings from one snapshot to another, leading

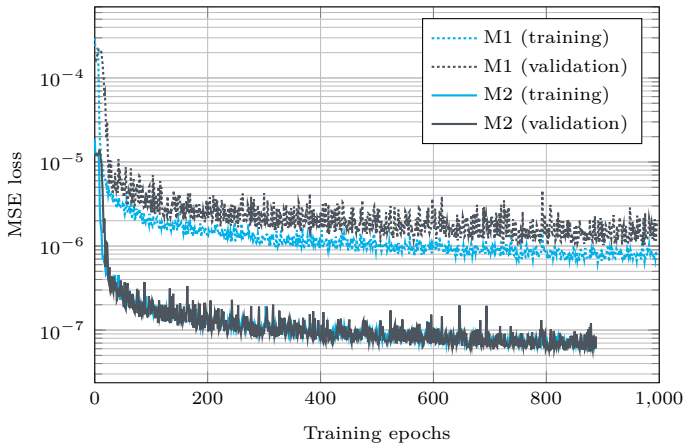


Figure 3.5: **Training and validation loss history** for the M1 and M2 training methods. The patch-based technique (M2) yields lower error and better generalization than the baseline M1, as evidenced by the negligible gap between validation and training curves. M1 training was performed for 1000 epochs, and the M2 training was stopped after 850 epochs when error stopped improving.

to improved generalization capabilities compared to a monolithic snapshot-to-snapshot inference.

3.4 Results and discussion

In this section, the benefits induced by the patch-based training procedure are compared with that of the regular M1 training method on predictive tasks. To this end, predictions of both models are evaluated against reference solutions obtained from the CFD solver. In the remaining of this section, training data consists of 75% of samples from the SqRe22k dataset and 25% of samples from the CyRe44k dataset. Such a mixing of datasets is used to assess the generalization capabilities of the two methods, as both datasets present similar flow features, but with different obstacles. First, comparisons are made on out-of-training samples from the SqRe22k dataset using baseline training parameters. Then, predictions obtained with snapshots from different datasets (SqRe44k, SqRe88k, CyRe22k, and CyRe88k) are evaluated against their references. Finally, a parametric study considering the impact of batch size b, the patch size n, and the stride size s on the final performance is proposed. Overall, comparisons are made on the basis of (i) contour plots of predicted and expected $\tilde{\nu}$, (ii) 1D plots of $\tilde{\nu}$ along streamwise and spanwise lines at different locations in the domain, as shown in figure 3.6, and (iii) scatter and density plots of the predicted $\tilde{\nu}$ against reference values.

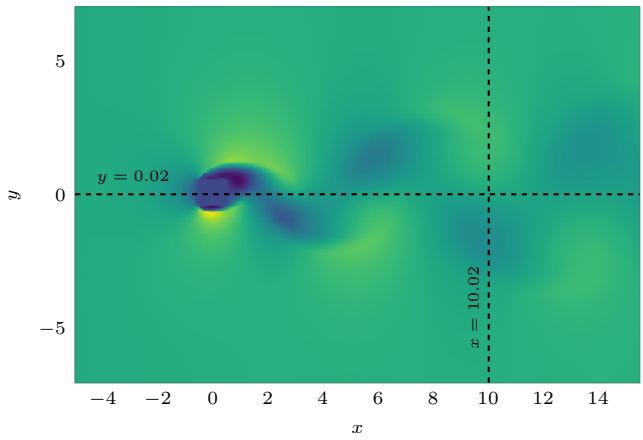


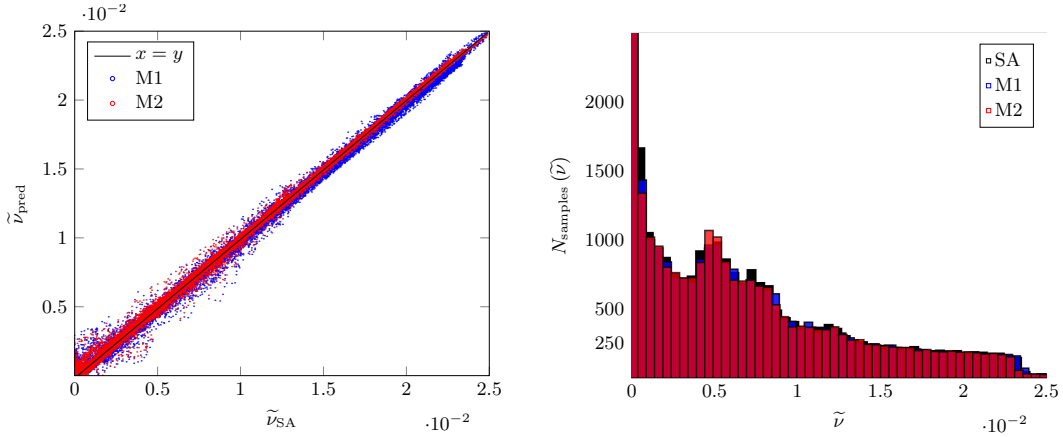
Figure 3.6: **Locations of the probe lines used for comparison to CFD reference**.

3.4.1 Comparison on out-of-training snapshot

In this section and the following, baseline training parameters are used, *i.e.*batch size is equal to 32, patch size n is equal to 50, and stride size s is equal to 75. As stated above, the training data consists of 75% of samples from the SqRe22k dataset and 25% of samples from the CyRe44k dataset. M1 and M2 models' predictive capabilities are compared on an out-of-training snapshot from the SqRe22k dataset, as shown in figure 3.7. As can be observed on the scatter plot (figure 3.7a), both M1 and M2 methods are in good accordance with the reference regarding the predicted $\tilde{\nu}$. Still, the M2 prediction presents an average relative deviation of 2.25% on the entire sample, against 5.04% for M1. More, its maximum relative deviation is also lower, with 36.44% for M2, against 76.23% for M1. To illustrate, the error fields obtained with M1 and M2 predictions are shown on the same snapshot in figure 3.8.

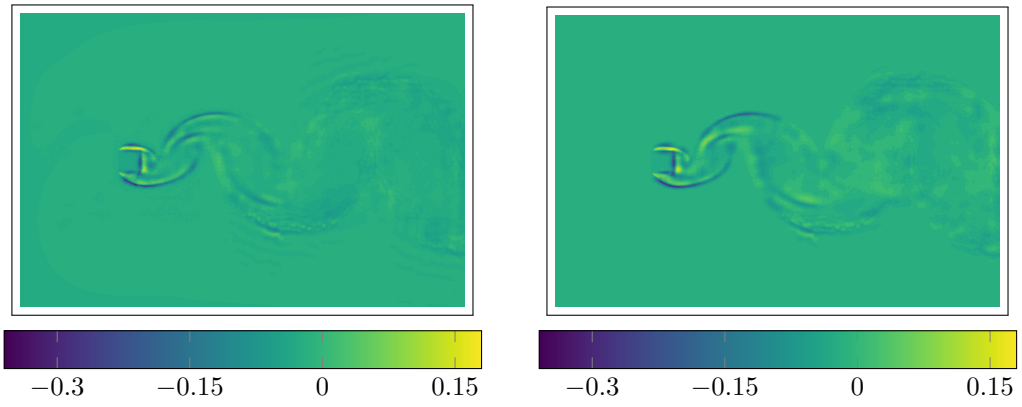
3.4.2 Comparison on out-of training datasets

In this section, models M1 and M2 (trained on a mixed dataset composed of samples from SqRe22k and CyRe44k) are used to make predictions on snapshots from datasets SqRe44k, SqRe88k, CyRe22k, and CyRe88k, which were not used for training. M1 and M2 predictions for one snapshot of each dataset are compared against CFD reference on stream-wise and span-wise 1D plots of $\tilde{\nu}$, at the locations presented in figure 3.6. Results are shown in figure 3.9. As can be observed, the patch-based trained model consistently outperforms the M1 model, while presenting an excellent agreement with reference data. On the $x = 10.02$ line, which represents full developed wake region, performances of M1 and M2 models are close on SqRe44k and SqRe88k datasets, but M1 significantly overestimates the $\tilde{\nu}$ values on the CyRe22k and CyRe88k datasets, indicating that model M1 is unable to fully leverage the diversity of the training dataset, and only learns



(a) (b)

Figure 3.7: **Scatter plot and histogram** of predicted and expected $\tilde{\nu}$ for an out-of-training snapshot of SqRe22k. (3.7a) The plot is a superposition of two scatter plots, namely SA against M1 and SA against M2. (3.7b): The histogram compares the occurence of truth and predictions on a step-type filled histogram.



(a) Relative error for prediction from M1

(b) Relative error for prediction from M2

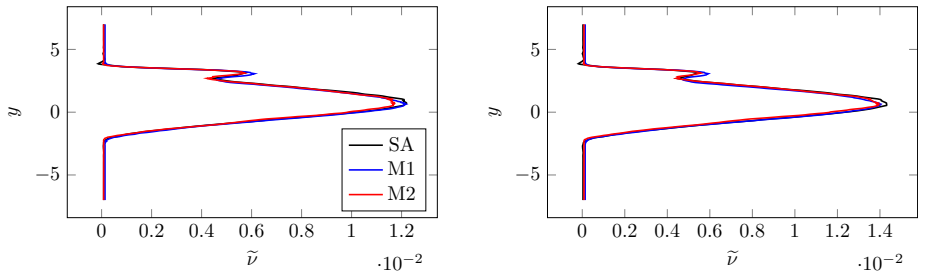
Figure 3.8: **Contour plots of relative errors** obtained from the same snapshot input from dataset SqRe22k, using methods M1 (3.8a) and M2 (3.8b). In both cases, maximal error levels are observed in the vicinity of the obstacle.

full-scale velocity-turbulent viscosity patterns. Conversely, the M2 model here proves its ability to learn local feature mapping from velocity field to turbulent viscosity field and accurately reconstructs it, independently of the obstacle-type and Reynolds number. Similarly, on the $y = 0.02$ line, which passes through the obstacle boundaries as well as the wake regions, M1 and M2 models show similar performances on datasets with a square obstacle, while M1 largely deviates from the reference data on snapshots coming from datasets with a cylindrical obstacle. Contrarily, the M2 model again provides accurate predictions. The latter results are further emphasized on the contour plots of figure 3.10, where M1 predictions on cylindrical obstacles present inaccurate features and saturated fields in the turbulent area downstream of the obstacle. This again indicates the inability of training procedures on full-scale samples to infer proper mapping from velocity fields to turbulent viscosity fields at the local scale, which is not the case of patch-based training. The Reynolds numbers are of similar orders in magnitudes which explains the capabilities of M1 and M2 to extrapolate on Re values outside of their training datasets. Hence, the extrapolation capabilities of the M2 model could be assessed even at higher Reynolds number.

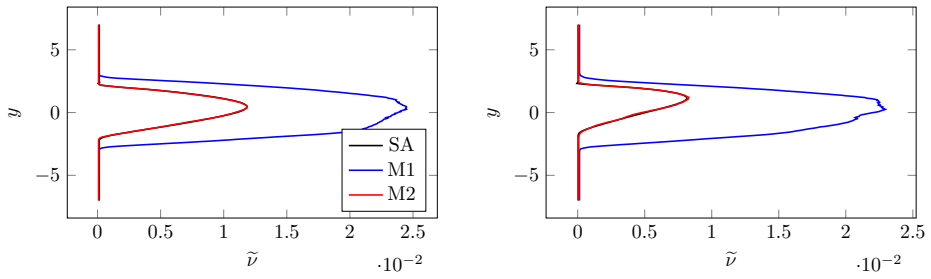
3.4.3 Parametric study

A parametric study is performed to explore the impact of the batch size b, the patch size n and the stride size s on the MSE error $\mathcal{L}_{MSE}$ (as defined in equation (5.5)) computed on validation data. To this end, the performances of various (n, s) pairs with relation $s = 1.5 \times n$ are first compared in terms of final validation performance and training time. To select the best performance of each pair, early stopping is used during training, and the average validation error over the last 50 epochs, noted $\overline{\mathcal{L}_{MSE}}$, is retained. As shown in table 3.2a, the pairs $(100, 150)$ and $(50, 75)$ yield close performances in terms of final MSE error. Although the $(100, 150)$ is slightly better in accuracy and training time, the $(50, 75)$ pair is preferred for its larger amount of patches per snapshot. The larger errors of the $(20, 30)$, $(10, 15)$, $(6, 9)$, and $(2, 3)$ pairs can be attributed to the low number of points per patch making it difficult to train the model with the same hyper-parameters, while the $(200, 300)$ pair prevents the efficient learning of local features, and is likely to present the same flaws as method M1.

In a second time, we consider the impact of varying stride size s for the previously-select n value, equal to 50. Results are presented in table 3.2b. As can be seen, no significant difference is observed for stride values ranging from 30 to 300, indicating that in this context, the amount of patches per snapshot (and thereby total samples) is not a limitation. Finally, the effect of varying batch size is assessed for $(n, s) = (50, 75)$. As shown in table 3.2c, small batch sizes 8, 16, and 32 yield close error levels, while larger batch sizes are associated with errors larger by roughly one order of magnitude. Although $b = 8$ is slightly lower than the other values, $b = 32$ is retained as the best accuracy/training time ratio.

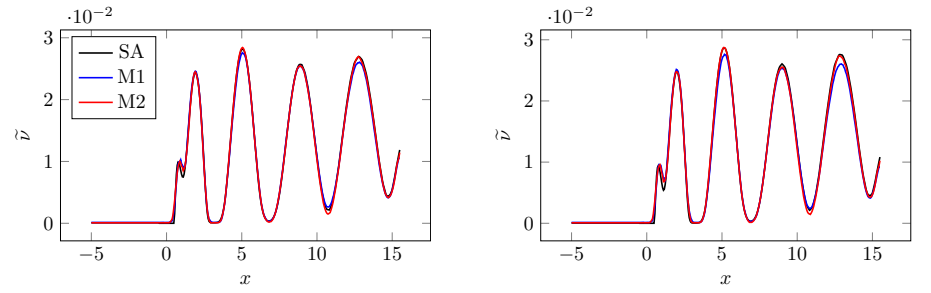


(a) SqRe44k (b) SqRe88k

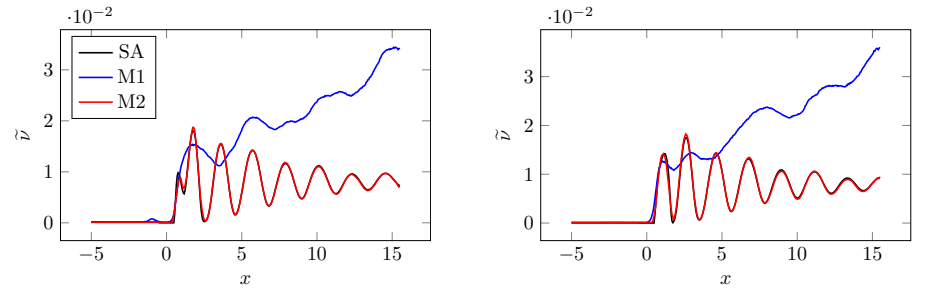


(c) CyRe22k (d) CyRe88k

(a)-(d): $\widetilde{\nu}$ in the span-wise direction at $x = 10.02$



(e) SqRe44k (f) SqRe88k



(g) CyRe22k (h) CyRe88k

(e)-(h): $\widetilde{\nu}$ in the stream-wise direction at $y = 0.02$

Figure 3.9: **Line plots along** $x = 10.2$ **and along** $y = 0.1$ comparing prediction accuracies of M1 and M2 on out-of-training samples from datasets SqRe44k (3.9a)-(3.9e), SqRe88k (3.9b)-(3.9f), CyRe22k (3.9c)-(3.9g) and CyRe88k (3.9d)-(3.9h). M1 and M2 perform similarly on datasets with a square obstacle, even on higher Re values. Yet, M1 consistently fails at predicting accurate $\tilde{\nu}$ on samples with cylindrical obstacle, while M2 presents an almost-perfect fit with CFD reference. The small deviation observed for M2 at the top of the square cylinder can be likely attributed to the unstructured-to-structured data sampling, and its study is deferred to a future work.

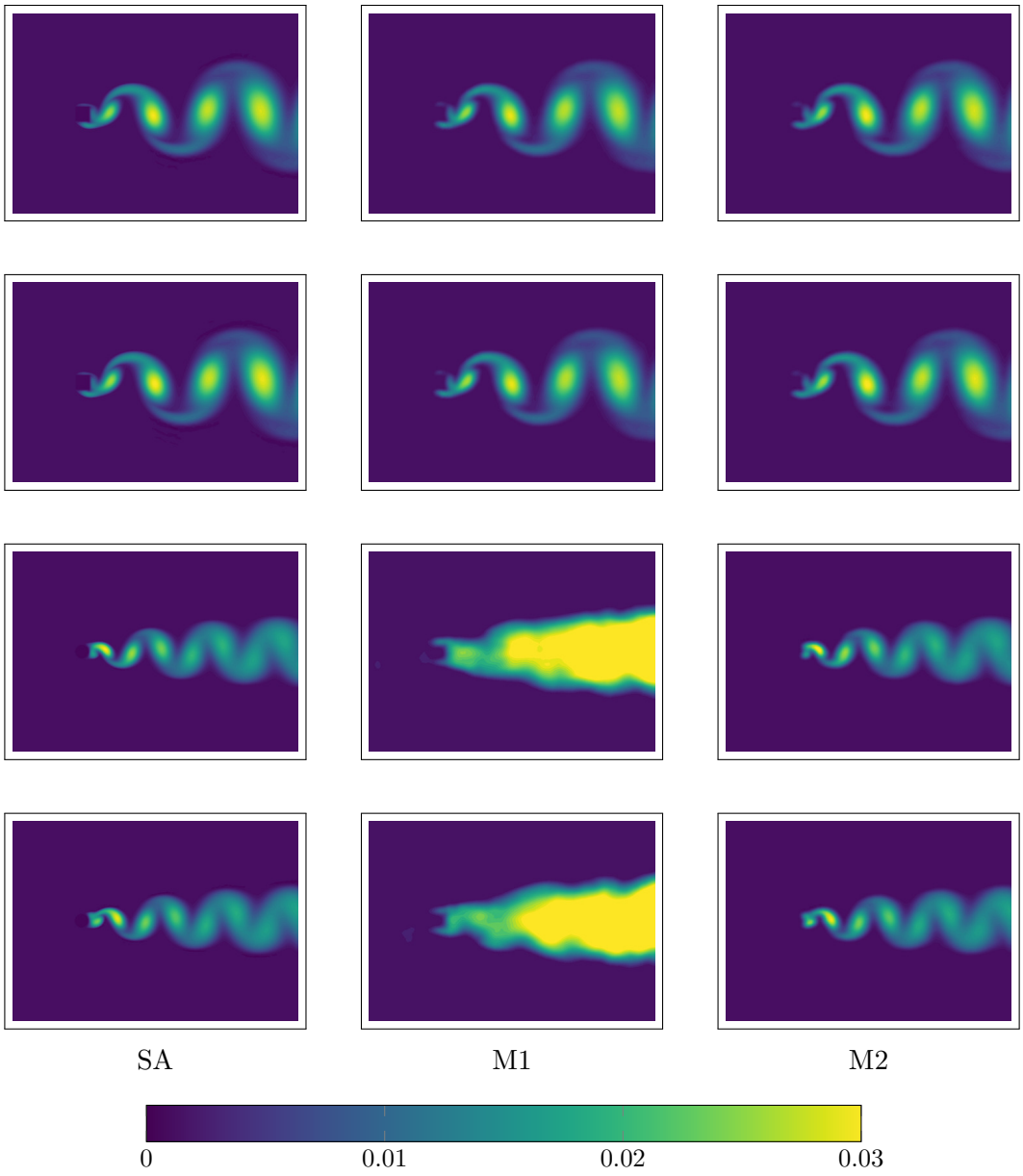


Figure 3.10: **Comparison of M1 and M2 $\tilde{\nu}$ predictions against CFD reference on snapshots from different out-of-training datasets**, namely SqRe44k (top row), SqRe88k (second row), CyRe22k (third row), and CyRe88k (bottom row). While M1 and M2 perform similarly on snapshots with square obstacle even at high Re numbers, M1 predictions on cylindrical obstacle are significantly saturated in the wake region, showing that the model was unable to learn features from the related samples in the training dataset. Conversely, M2 predictions are in line with the SA reference.

(a) Model performance for various (n, s) pairs

Pairs (n, s)	$\overline{\mathcal{L}_{MSE}}$	**Training time (hours)**	**Patches per snapshot**
200,300	1.08×10^{-6}	1.06	1
100,150	3.09×10^{-7}	1.85	4
50,75	3.36×10^{-7}	2.38	20
20,30	1.77×10^{-6}	3.34	120
10,15	1.94×10^{-6}	7.09	480
6,9	1.98×10^{-6}	62.26	1320
2,3	7.85×10^{-6}	128.5	12000

(b) Model performance for varying stride s, with $n = 50$

Pairs (n, s)	$\overline{\mathcal{L}_{MSE}}$	**Training time (hours)**	**Patches per snapshot**
50,300	3.98×10^{-7}	1.93	2
50,150	4.06×10^{-7}	2.34	6
50,75	3.36×10^{-7}	2.38	20
50,30	3.57×10^{-7}	10.2	99

(c) Model performance for varying batch size b, with $(n, s) = (50, 75)$

Batch size	$\overline{\mathcal{L}_{MSE}}$	**Training time (hours)**
256	3.10×10^{-6}	0.86
128	2.34×10^{-6}	1.07
64	1.48×10^{-6}	1.40
32	3.36×10^{-7}	2.38
16	4.09×10^{-7}	3.37
8	2.85×10^{-7}	9.80

Table 3.2: **Model performance for varying** (n, s, b) **parameters.** (3.2a) Model performance for various (n, s) pairs, with the constraint $s = 1.5 \times n$. Best validation performance is obtained for $(100, 150)$, but the close performance of $(50, 75)$ and its larger amount of generated snapshots make it a more versatile candidate. (3.2b) Model performance for varying stride size s, with $n = 50$. Best performance is obtained for $s = 75$, although other stride values present closer performance levels. (3.2c) Model performance for varying batch size b, with $(n, s) = (50, 75)$. Although best performance was obtained for $b = 8$, batch sizes of 16 and 32 made no significant different in validation error. Hence, faster training was privileged, and $b = 32$ was retained.

3.5 Conclusions

In this article, we have demonstrated the deployment of a robust deep learning model for predicting Spalart-Allmaras eddy viscosities. The method of patch-based training works by dividing the full-scale samples into patches, in order to let the model learn multiple local feature mappings, instead of learning monolithic full-scale features. Applied to an auto-encoder architecture, it was observed that patch-based training led to training and validation errors one order of magnitude lower than standard full-scale training, and was able able to efficiently learn local mappings from multiple datasets with different features, which was not the case of full-scale training method. For practical CFD purposes, a local patch-based model would be of great importance so that any input fluid domain, either full or in parts by region of interest, can be split into patches and passed to the model to predict the quantities of interest. Hence, patch-based training holds an important potential to improve the usability of trained models in the coupling with CFD solvers. Deploying a trained model to solve for turbulent viscosity inside a CFD solver is regarded as a future extension of the present work.

Bibliography

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

Steven R Allmaras and Forrester T Johnson. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. In *Seventh international conference on computational fluid dynamics (ICCFD7)*, pages 1–11, 2012.

Junfeng Chen, Elie Hachem, and Jonathan Viquerat. Graph neural networks for laminar flow prediction around random 2d shapes. *arXiv preprint arXiv:2107.11529*, 2021.

Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

Joel H Ferziger, Milovan Perić, and Robert L Street. *Computational methods for fluid dynamics*, volume 3. Springer, 2002.

Kai Fukami, Yusuke Nabae, Ken Kawai, and Koji Fukagata. Synthetic turbulent inflow generator using machine learning. *Physical Review Fluids*, 4(6):064603, 2019.

G Guiza, A Larcher, A Goetz, L Billon, P Meliga, and Elie Hachem. Anisotropic boundary layer mesh generation for reliable 3d unsteady rans simulations. *Finite Elements in Analysis and Design*, 170:103345, 2020.

Elie Hachem, Stephanie Feghali, Ramon Codina, and Thierry Coupez. Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation. *International journal for numerical methods in engineering*, 94(9):805–825, 2013.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

SUN Liang, AN Wei, LIU Xuejun, and LYU Hongqiang. On developing data-driven turbulence model for dg solution of rans. *Chinese Journal of Aeronautics*, 32(8):1869–1884, 2019.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. A turbulent eddy-viscosity surrogate modeling framework for reynolds-averaged navier-stokes simulations. *Computers & Fluids*, page 104777, 2020.

Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence. *arXiv preprint arXiv:1903.00033*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, pages 285–319, 2010.

Anikesh Pal. Deep learning emulation of subgrid-scale processes in turbulent shear flows. *Geophysical Research Letters*, 47(12):e2020GL087005, 2020.

Aakash Vijay Patil and Corentin Lapyere. Development of deep learning methods for inflow turbulence generation. *arXiv preprint arXiv:1910.06810*, 2019.

Pedro Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *International conference on machine learning*, pages 82–90. PMLR, 2014.

Stephen B Pope. Turbulent flows, 2001.

W Rodi, JH Ferziger, M Breuer, and M Pourquie. Status of large eddy simulation: results of a workshop. *Transactions-American Society of Mechanical Engineers Journal of Fluids Engineering*, 119:248–262, 1997.

Chris Rumsey, Brian Smith, and George Huang. Description of a website resource for turbulence modeling verification and validation. In *40th Fluid Dynamics Conference and Exhibit*, page 4742, 2010.

Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal*, pages 1–13, 2017.

Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*, page 439, 1992.

Philippe R Spalart. Strategies for turbulence modelling and simulations. *International journal of heat and fluid flow*, 21(3):252–263, 2000.

Brendan D Tracey, Karthikeyan Duraisamy, and Juan J Alonso. A machine learning strategy to assist turbulence model development. In *53rd AIAA aerospace sciences meeting*, page 1287, 2015.

Chapter 4

Learning Subgrid-scale Turbulence

Le présent chapitre se situe à l'intersection de la vision par ordinateur, de l'apprentissage profond et de la turbulence informatique et expérimentale. L'objectif principal était d'explorer l'état de l'art des méthodes d'apprentissage profond pour estimer la turbulence à l'échelle de la maille à partir des grandes échelles mesurées ou des échelles résolues par la maille. La principale contribution de cette étude est d'effectuer un apprentissage par transfert en ré-entraînant un modèle formé pour des problèmes non physiques afin de réapprendre de nouvelles caractéristiques physiques. Les champs à gros grains sont calculés à partir des ensembles de données de référence pour faire les prédictions *a priori* en utilisant le modèle d'apprentissage profond formé sur l'ensemble de données DNS. Des reconstructions à l'échelle de la maille sont effectuées et comparées aux ensembles de données DNS et PIV de référence. Dans les deux ensembles de données, des ensembles de données d'écoulement de canal turbulent délimité par des murs sont utilisés. Les reconstructions à l'échelle de la maille le long de diverses directions de raffinement et de divers niveaux de grossissement sont comparées à la référence en mesurant des statistiques physiques telles que l'intensité de la turbulence, la dissipation, l'enstrophie et les spectres d'énergie.

4.1 Introduction

Turbulence is multiscale in nature, with a very wide range of scales coexisting and interacting. Turbulence is governed by the Navier-Stokes equation in most practical problems, and is still very difficult to predict and model. In order to understand its detailed physics, spatio-temporal resolved information is necessary. Unfortunately, none of the current facilities, even within academic research, can provide this information over a sufficiently wide spatial range and different flow conditions. Despite advances in computational resources, direct numerical simulation (DNS) is limited to flows with low to moderate Reynolds numbers or simple geometries. Complex fluid flows encountered in many engineering and physical applications are computationally demanding to resolve using DNS, hence large eddy simulation (LES) and Reynolds Averaged Navier-Stokes (RANS) modeling are commonly used frameworks that can provide accurate predictions by considering the interaction between the grid-resolved and subgrid scales. On the other hand, experimental measurement techniques such as Particle Image Velocimetry (PIV) and Hot-wire Anemometry (HWA) have limited measurement capacity for spatio-temporal resolved velocities. High-frequency tomography PIV has progressed over the last decade, but is still limited to small volumes and low velocities. HWAs provide time-resolved measurements, but combining a large number of HWAs remains challenging and disruptive.

Works involving re-generating turbulence statistics as well as super-resolution has been the focus of multiple recent contributions Fukami et al. [2019b]; Mohan et al. [2019]; Beck et al. [2019]; Kim and Lee [2019]; Fukami et al. [2019a, 2020]. Some investigations have considered machine learning methods and super-resolution data reconstruction techniques for reproducing turbulent flows Fukami et al. [2021]; Jiang et al. [2020]. Once trained on high-quality data, these methods have been shown capable of reproducing the underlying flow field using remarkably coarse measurements. Previous studies have also considered purely statistical data fusion methods for flow reconstruction. In Van Nguyen et al. [2015], a "model-free" maximum a posteriori (MAP) algorithm was proposed for fusing low-temporal-high-spatial resolution data with high-temporal-low-spatial resolution data for turbulent flow reconstruction. On the experimental side, Krishna et al. [2019, 2020] showed that the gaps between PIV snapshots can be filled with simple linear assumptions on the Navier Stokes equations. In that work, simple linear regression models were used to propagate PIV snapshots forward and backward in time, with a weighted average of forward and backward predictions used to estimate the turbulent flow between snapshots.

The current chapter addresses the problem of estimating subgrid-scale information from grid-resolved information to provide a complete view of turbulent flows at both large and small scales. Available turbulence databases, both computational and experimental, are used to reconstruct subgrid-scale turbulence. First, the DNS database used for training the deep learning model is described, followed by the rationale of the deep learning model's architecture and the as-

sociated transfer learning. Later, the subgrid-scale reconstruction statistics are compared with reference to the DNS data. The trained model is also tested against a PIV dataset to demonstrate its robustness and applicability to the experimental dataset.

4.2 Turbulent channel flow datasets and preparation

One of the important concepts in the subgrid-scale reconstruction of the turbulent flow between consecutive snapshots is the model used to approximate the flow evolution. We consider the three-dimensional incompressible Navier-Stokes equations (NSE) linearized around the mean flow profile to satisfy the momentum equation and continuity constraint. Considering the dynamics of turbulent velocity fluctuations $\boldsymbol{u} = (u_1, u_2, u_3)$ around a mean flow $\boldsymbol{U} = (U(x_2), 0, 0)$, we have:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{U} \cdot \nabla \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{U} = -\nabla p + \frac{1}{\mathrm{Re}_\tau} \nabla^2 \boldsymbol{u} + (\mathrm{NL}), \tag{4.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{4.2}$$

where $\boldsymbol{U} = [U(x_2), 0, 0]$ denotes the mean flow profile in a three-dimensional coordinate system, as shown in Fig. 4.1. We use (x_1, x_2, x_3) to represent the streamwise, wall-normal, and spanwise directions, respectively. $\boldsymbol{u} = [u_1, u_2, u_3]$ denotes the turbulent velocity fluctuations, and p represents the pressure fluctuations. Re_τ denotes the friction Reynolds number.

We assume linearity as proposed by Hunt and Carruthers [1990] in such a way that the non-linear interactions can be neglected on scales that are much shorter than typical scales of interaction, which in the case of time evolution means that the interactions shorter than typical eddy turnover time can be neglected. These assumptions allow for linearized equations of motions for the subgrid-scale reconstruction of flow over shorter spatial or temporal scales. This means the non-linear terms can be neglected when the grid resolution (Δ) or time step (Δt) is less than a typical eddy length or eddy turnover time. The simplified NSE would thus read as:

$$\begin{aligned} \underbrace{\frac{\partial u_1}{\partial t} + U\frac{\partial u_1}{\partial x_1}}_{\text{advection}} &= \underbrace{\frac{1}{\mathrm{Re}_\tau}\left(\frac{\partial^2 u_1}{\partial x_1^2} + \frac{\partial^2 u_1}{\partial x_2^2}\right)}_{\text{diffusion}} - \underbrace{u_2\frac{\partial U}{\partial x_2}}_{\text{coupling}}, \\ \underbrace{\frac{\partial u_2}{\partial t} + U\frac{\partial u_2}{\partial x_1}}_{\text{advection}} &= \underbrace{\frac{1}{\mathrm{Re}_\tau}\left(\frac{\partial^2 u_2}{\partial x_1^2} + \frac{\partial^2 u_2}{\partial x_2^2}\right)}_{\text{diffusion}}. \end{aligned} \tag{4.3}$$

A direct numerical simulation database of turbulent channel flow is used to validate the subgrid-scale reconstruction method described in this chapter. To

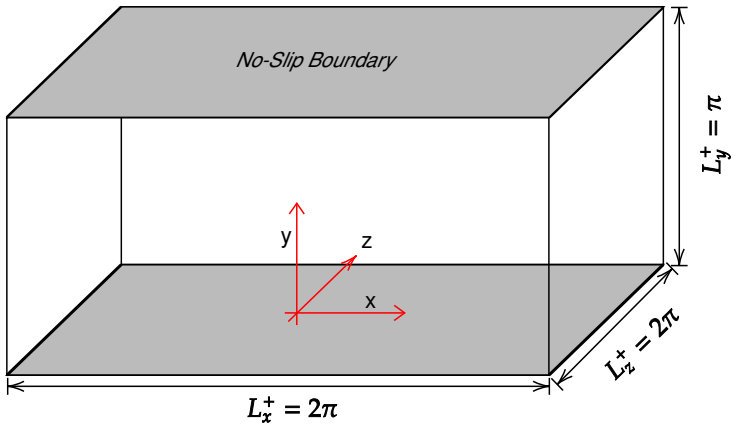


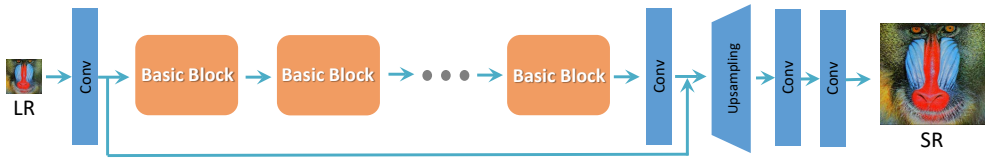
Figure 4.1: Geometry of 3D turbulent channel flow.

access fully resolved velocity fields, DNS data is chosen as it simulates flows without any turbulence modeling, and is free from noises as well as experimental uncertainties. DNS of incompressible turbulent wall-bounded flow at Reynolds number $Re_\tau \approx 1000$ based on the friction velocity is used, where $Re_\tau = u_\tau h/\nu$, and the numerical procedure described in Lozano-Durán and Jiménez [2014] is used for the simulation. The streamwise, wall-normal, and spanwise directions in the cartesian coordinates of simulation in space are denoted by x, y, z respectively. The channel domain size L_x x L_y x L_z normalized by channel half-height σ is 2π x 2π x π. Fully resolved fluctuating streamwise velocities in all three flow directions are used. The high-resolution high-fidelity DNS data is a superposition of the mean and fluctuating velocities along the symmetric directions, and is given as:

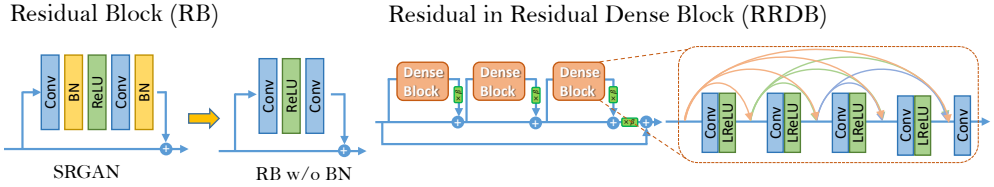
$$u = \overline{u_i^t} - u_i^{t'}, \tag{4.4}$$

where $\overline{(.)}$ denotes the mean quantity along symmetric or homogeneous directions, and $(.)'$ denotes the fluctuating quantity. In this case, we can take the mean along t and x, making the resulting velocities a function in $y - z$ planes. Hereafter, the fully resolved fluctuating velocities in a plane normal to flow direction are denoted by u'. This data contains a total $P = 512 \times 1700$ snapshots representing samples in x and t. Each snapshot has a spatial resolution of 384×512 denoting the samples in the $y - z$ plane. This is our true data, and the target of higher resolution against which we would compare our errors of subgrid-scale reconstruction.

The lower-resolution dataset is synthetically produced from the high-resolution data with spatial coarse-graining. We perform spatial coarse-graining with mean-pooling, max-pooling, and sparse sampling of the higher-resolution data. The lower resolution dataset coefficients are 4Δ, 16Δ, and 64Δ for four-times, sixteen-times, and sixty-four times coarse-grained snapshots respectively. For example, when 4Δ is spatially coarse-grained along the $y-z$ plane, a 384×512 higher resolution snapshot becomes a 96×128 lower resolution snapshot. These datasets are used to estimate high-resolution turbulent flow fields from low-resolution ones,



(a) The basic architecture from ESRGAN by Wang et al. [2018] is used where most computation is done in the low-resolution feature space. The 'basic blocks' can be designed or chosen such as residual block, dense block, or residual in residual dense blocks (RRDB) for better performance. Image courtesy: Wang et al. [2018]



(b) **Left**: Removal of batch-normalization BN layers in the residual block. **Right**: RRDB block used in ESRGAN model with suitable residual scaling parameter. Image courtesy: Wang et al. [2018]

Figure 4.2: Deep learning architecture used for learning subgrid-scale turbulence

thus demonstrating whether the subgrid-scale turbulence can be estimated from grid-resolved or large-scale measurements.

4.3 Deep Learning Architecture and Training

A deep learning model to estimate mapping between subgrid scales and large scales is required. This can be considered as whether a non-linear model $\mathbb{M}$ exists such that, given a lower-resolution field, a corresponding higher-resolution field would be predicted by:

$$u_{iHR}^{t'} = \mathbb{M}(u_{iLR}^{t'}), \tag{4.5}$$

where $u_{iHR}^{t'}$ is a higher-resolution snapshot at instant t containing the complete subgrid-scale information, and $u_{iLR}^{t'}$ is a lower-resolution snapshot at the same instant containing the large scales.

A deep learning architecture based on enhanced super-resolution generative adversarial networks (ESRGAN) Wang et al. [2018] is used to reconstruct high-resolution flow fields from low-resolution fields. Figures 4.2 show the architectures of a generator and discriminator in the ESRGAN. A generator consists of a deep convolution neural network with residual in residual dense blocks (RRDBs) Wang et al. [2019]. A low-resolution input snapshot is first fed to the generator which is passed through a convolution layer followed by a series of RRDBs, and passed through a final convolutional layer to generate a high-resolution snapshot. The original ESRGAN uses an additional discriminator to which the generated

and true snapshots are fed by passing through a series of convolutional, batch normalization, and rectified linear unit (ReLU) layers. After successful training, the generator is expected to produce a snapshot with statistics similar to the true snapshot that a discriminator cannot distinguish from the true snapshot. Readers are referred to the work of Goodfellow et al. [2020] for details on generative-adversarial networks (GANs) where two adversarial neural networks, the generator, and the discriminator, compete with each other. For the present study, we deploy transfer learning by making use of a previously trained ESRGAN model on super-resolution of images from various sources in Wang et al. [2018]. Transfer-learning is used to re-train a model for a new dataset which helps in saving training time.

This deep learning architecture aims at proposing a nonlinear model to estimate subgrid-scale information given the large-scale information. The use of deep learning models is in the context where data at all scales are available from the DNS dataset. The proposed model learns an empirical relation between large and subgrid scales, which is presented in the form of a function $\mathbb{M}(.)$ to predict subgrid scale information on new datasets and situations. The deep learning architecture aims at finding nonlinear mapping functions with a more adaptive kernel space and allows to model nonlinear phenomena.

The resulting deep learning model is trained with the Adam Kingma and Ba [2014] optimizer, to iteratively minimize the total Root Mean Square Error (RMSE) of the reconstruction loss defined by:

$$\epsilon(t) = \frac{(\int_{x_1=0}^{h}\int_{x_2=0}^{h}\left((u_1-\hat{u}_1)^2+(u_2-\hat{u}_2)^2\right)dx_1dx_2)^{1/2}}{(\int_{x_1=0}^{h}\int_{x_2=0}^{h}\left((u_1)^2+(u_2)^2\right)dx_1dx_2)^{1/2}}, \tag{4.6}$$

where $\hat{u}_1$ and $\hat{u}_2$ are the reconstructed velocity fluctuations while u_1 and u_2 are the velocity fluctuation from the DNS ground truth. Following the same metrics, we can also evaluate the RMSE with respect to the streamwise and wall-normal velocity components separately with an aim to evaluate the reconstruction errors for each component of the 2D velocity profiles:

$$\begin{aligned}\epsilon_{x_1}(t) &= \frac{(\int_{x_1=0}^{h}\int_{x_2=0}^{h}(u_1-\hat{u}_1)^2\,dx_1dx_2)^{1/2}}{(\int_{x_1=0}^{h}\int_{x_2=0}^{h}u_1^2dx_1dx_2)^{1/2}},\\ \epsilon_{x_2}(t) &= \frac{(\int_{x_1=0}^{h}\int_{x_2=0}^{h}(u_2-\hat{u}_2)^2\,dx_1dx_2)^{1/2}}{(\int_{x_1=0}^{h}\int_{x_2=0}^{h}u_2^2dx_1dx_2)^{1/2}}.\end{aligned} \tag{4.7}$$

4.4 Results and Discussions

The deep learning model is trained on 70% of the available DNS snapshots and the remaining 30% snapshots are kept for evaluation of results presented in this section. First, the results from various coarse-graining methods for generating synthetic low-resolution datasets are compared, followed by the effects of refinement direction on reconstructing the subgrid-scales. Later, successive refinement

through various scaling factors is compared. Lastly, the trained deep learning model is tested on a PIV experimental dataset of wall-bounded turbulence.

To compare the results of subgrid-scale reconstructions with DNS, several turbulence statistics are compared. Turbulence intensity, dissipation, and enstrophy are compared along with turbulent kinetic energy spectra in streamwise and spanwise directions. The turbulence intensity, also often referred to as turbulence level, is defined as:

$$I = \frac{u'}{U}, \tag{4.8}$$

where u' is the root-mean-square of the turbulent velocity fluctuations and U is the mean velocity. If the turbulent kinetic energy k is known, u' can be computed as:

$$u' = \sqrt{\frac{1}{3}\left(u_x'^2 + u_y'^2 + u_z'^2\right)} = \sqrt{\frac{2}{3}\,k}. \tag{4.9}$$

U can be computed from the three mean velocity components U_x, U_y and U_z as:

$$U = \sqrt{U_x^2 + U_y^2 + U_z^2}. \tag{4.10}$$

Turbulence dissipation, ϵ is the rate at which turbulence kinetic energy is dissipated, which is written as:

$$\epsilon \equiv \nu \overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_i'}{\partial x_k}}. \tag{4.11}$$

Finally, the enstrophy Ω is a type of potential density which is directly related to the kinetic energy in the flow that corresponds to dissipation effects in turbulent flows. For incompressible flows with $\nabla \cdot \mathbf{u} = 0$, the enstrophy can be described as the integral of the square of the vorticity ω given by:

$$\Omega \equiv \int_\Omega |\boldsymbol{\omega}|^2 \, dx. \tag{4.12}$$

4.4.1 Effects of coarse-graining

Since it is not known which pooling method for spatial coarse-graining is better than other pooling methods, the effects of these methods are examined. As described in the previous section, the lower-resolution dataset is synthetically produced from the high-resolution data with spatial coarse-graining. Spatial coarse-graining is performed with mean-pooling, max-pooling, and sparse sampling of the higher-resolution data. As shown in figure 4.3 max-pooling refers to taking the maximum value at each filter, average or mean-pooling refers to taking the average value at each filter, whereas sparse-pooling refers to sparse-sampling of the available feature space at each filter.

When max, mean, and sparse pooling methods are compared, it is found that the choice of these coarse-graining methods does not have a significant impact on

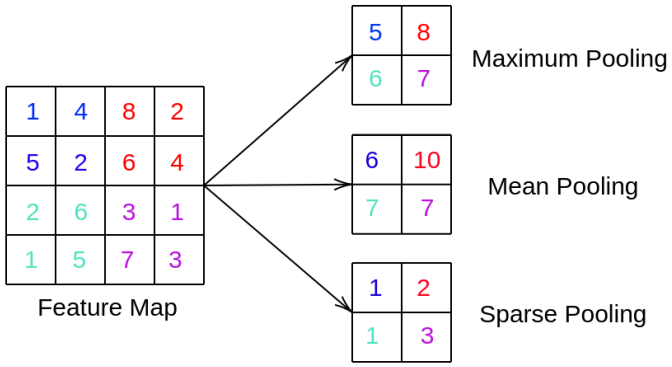


Figure 4.3: Representation of various pooling methods for spatial coarse-graining operations with a sample pool-size of 2×2 from a 4×4 feature space. Max-pooling and mean-pooling refer to taking the maximum and minimum value at each filter respectively and then arranged into a new output with a size of 2×2 feature. Sparse-pooling refers to the sparse sampling of the available feature space at each filter.

the subgrid-scale reconstruction. Figures 4.4b and 4.4c show the comparison of turbulence dissipation and enstrophy for various coarse-graining methods when measured along streamwise directions. Figure 4.4 shows a comparison of these methods by measuring turbulence intensity in a streamwise direction, where it can be noted that subgrid-scale reconstruction from all three methods have similar accuracy and are comparable to DNS. Additionally, when the spectral contributions are compared as shown in figure 4.5, it is observed that these three methods have similar accuracy for streamwise energy spectra, whereas, for the spanwise averaged spectrum of turbulent kinetic energy, it is observed that the subgrid-scale reconstruction from max-pooling has dissipated more compared to that from sparse-pooling. For further reporting, sparse pooling was chosen since it represents realistic situations for both experimental and numerical data, and it is computationally faster to perform sparse pooling.

4.4.2 Effects of Refinement Direction

Now that the effect of coarse-graining methods is established, we proceed towards investigating refinement direction and its effect on subgrid-scale reconstruction. The DNS dataset is a 3D dataset with data available along $x-y$, $y-z$, and $x-z$ 2D planar refinement directions, hence it is essential to demonstrate the deep learning model's performance for subgrid-scale reconstruction along these refinement directions. These investigations would be particularly useful later while demonstrating the trained model's applicability on the PIV dataset which is available only along two planes.

Figure 4.6 shows the comparison of turbulence intensity of DNS with the subgrid-scale reconstruction along various refinement directions. The turbulence

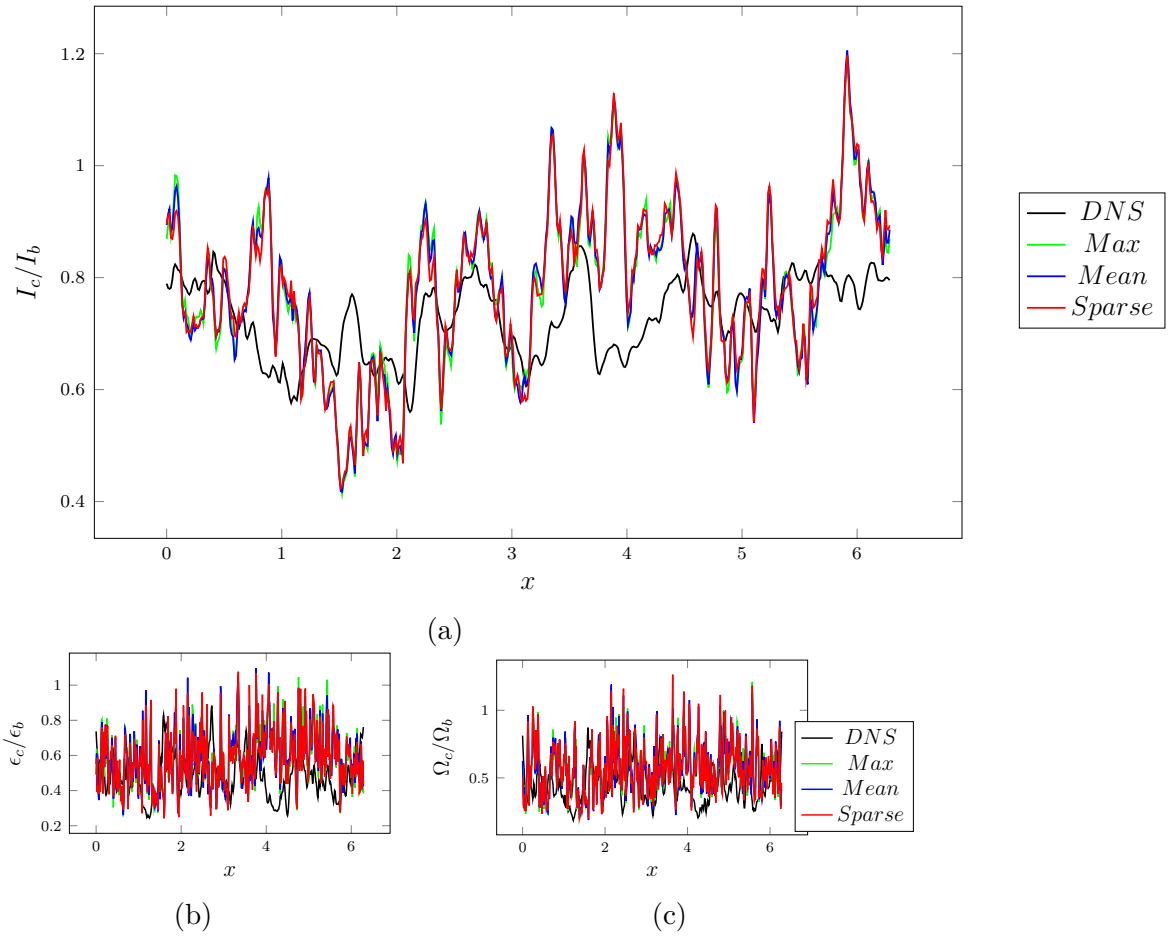
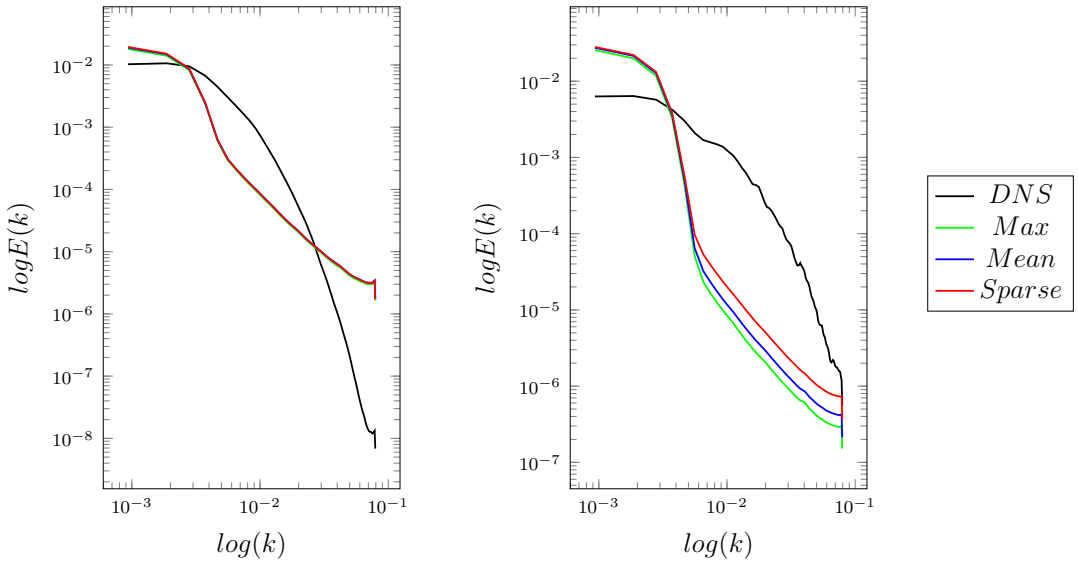


Figure 4.4: Comparison of refinements from max, mean, and sparse sampled coarse-graining methods on the (a) turbulence intensity (b) turbulent dissipation (c) turbulent enstrophy



(a) (b)

Figure 4.5: Comparison of the effect of max, mean, and sparse sampled coarse-graining methods on the (a) streamwise energy spectrum and (b) spanwise energy spectrum for the turbulent channel flow.

intensity is compared along $x-y$, $z-y$, and $x-z$ refinement directions in figure 4.6a, figure 4.6b, and figure 4.6c respectively. Similarly, turbulence dissipation and turbulence enstrophy from subgrid-scale reconstructions are compared with DNS data along the refinement directions in figure 4.8 and figure 4.7 respectively. The subgrid-scale reconstruction along all three refinement directions shows good agreement with the DNS data. From a finer evaluation, it is observed that the turbulence intensity is slightly under-predicted across all refinement directions, whose effect is then magnified in the turbulence dissipation comparison. Furthermore, streamwise and spanwise spectra are compared for these refinement directions as shown in figure 4.9 and figure 4.10. Both spectra from subgrid-scale reconstruction along all three refinement directions show good accuracy across the range of large and small scales. Overall, a good agreement in measured physical statistics is observed for subgrid-scale reconstructions compared to DNS.

4.4.3 Investigating Successive Refinement

The quality of subgrid-scale reconstruction by the extent of coarse-graining is investigated. Successive refinements are performed until DNS-comparable spatial levels are attained. The present deep learning model is trained for 4× subgrid-scale reconstruction, which means a DNS comparable complete spatial snapshot is estimated from a 4× spatially coarse-grained snapshot. But, can a 16× spatially coarse-grained snapshot be used to predict a DNS-comparable complete spatial snapshot by using the same deep learning model? Can this be done for

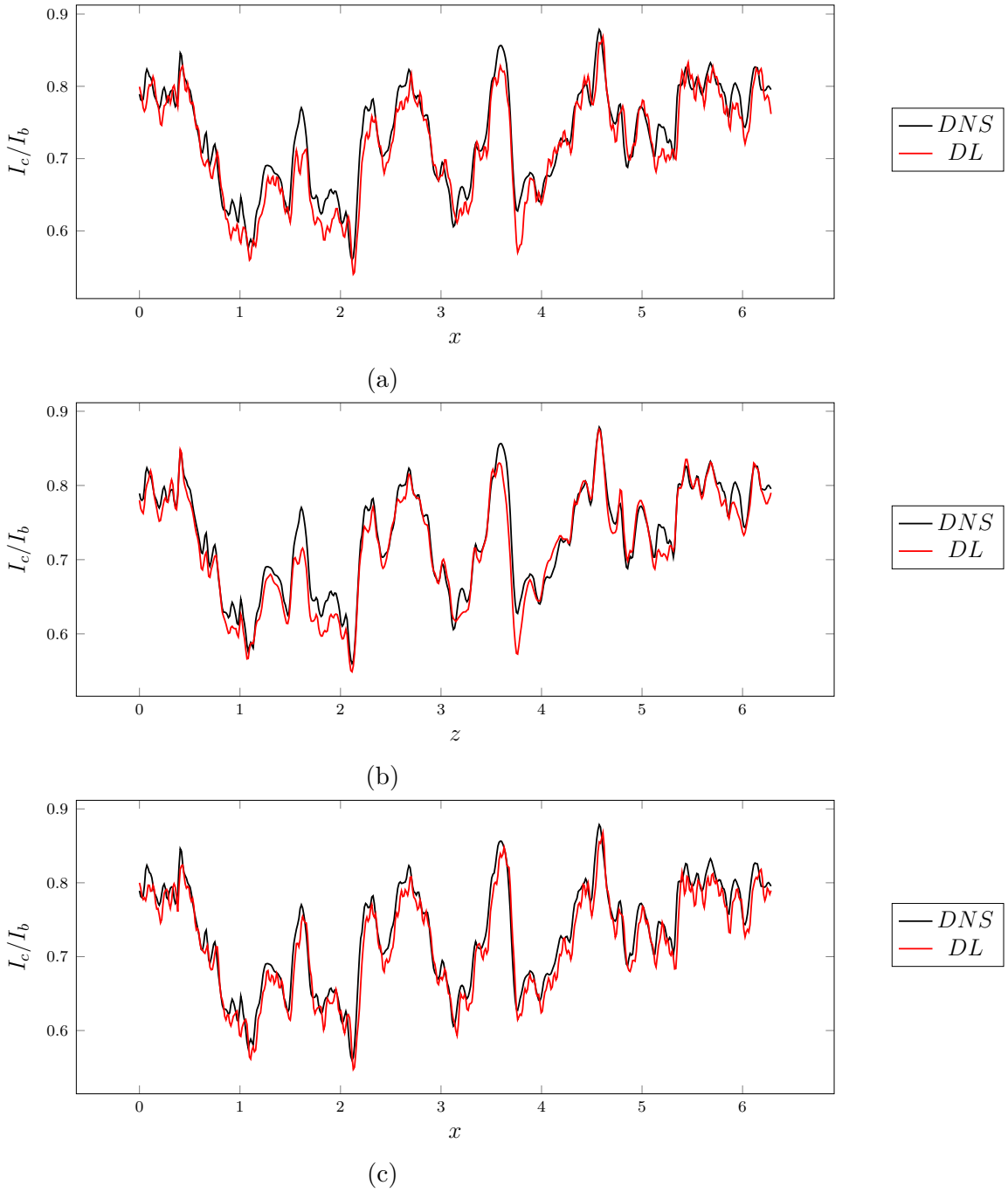


Figure 4.6: Comparison on the turbulence intensity by refinement direction in (a)XY (b)ZY (c)XZ

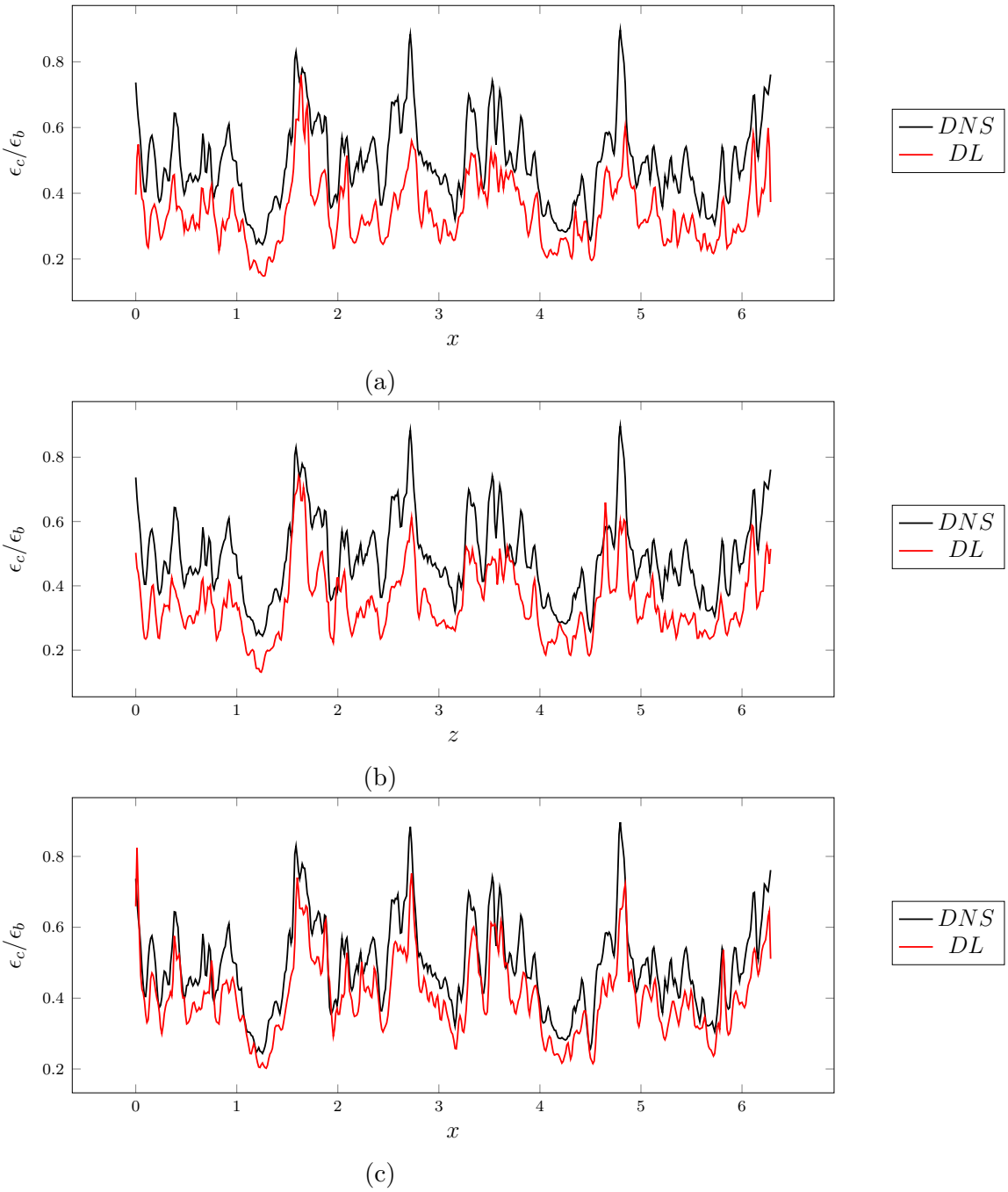


Figure 4.7: Comparison on the turbulence dissipation by refinement direction in (a)XY (b)ZY (c)XZ

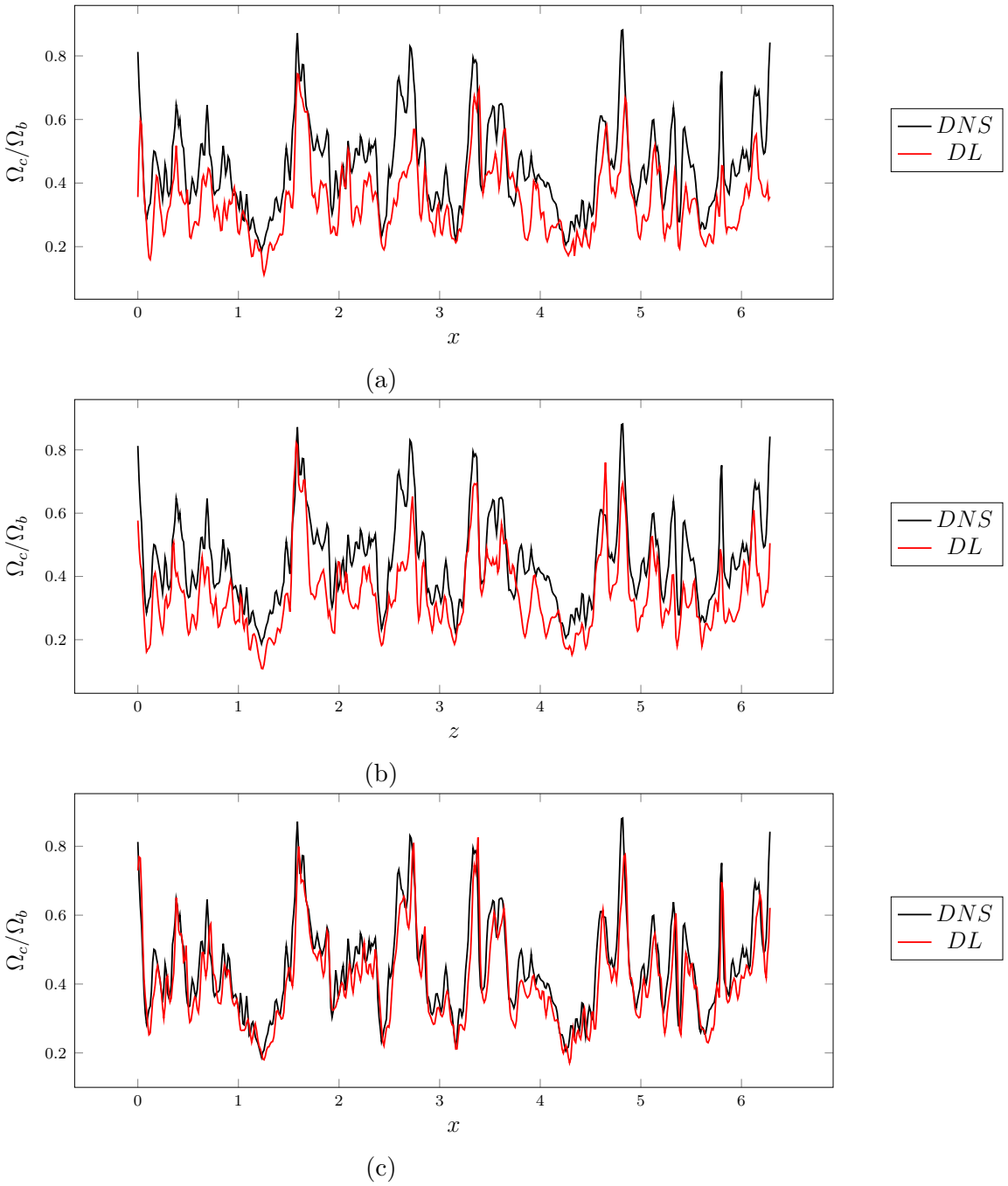


Figure 4.8: Comparison on the turbulence enstrophy by refinement direction in (a)XY (b)ZY (c)XZ

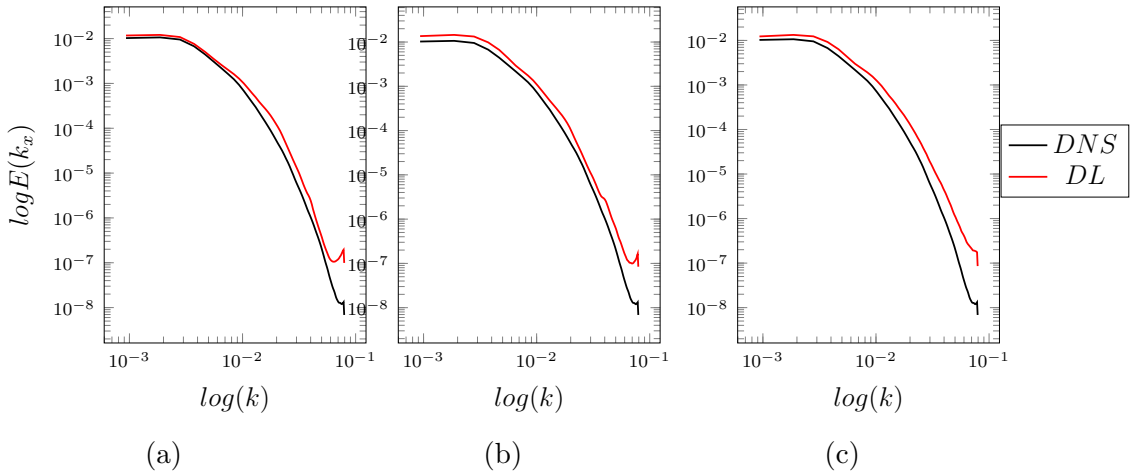


Figure 4.9: Comparison of refinement direction on the streamwise energy spectrum for the turbulent channel flow.

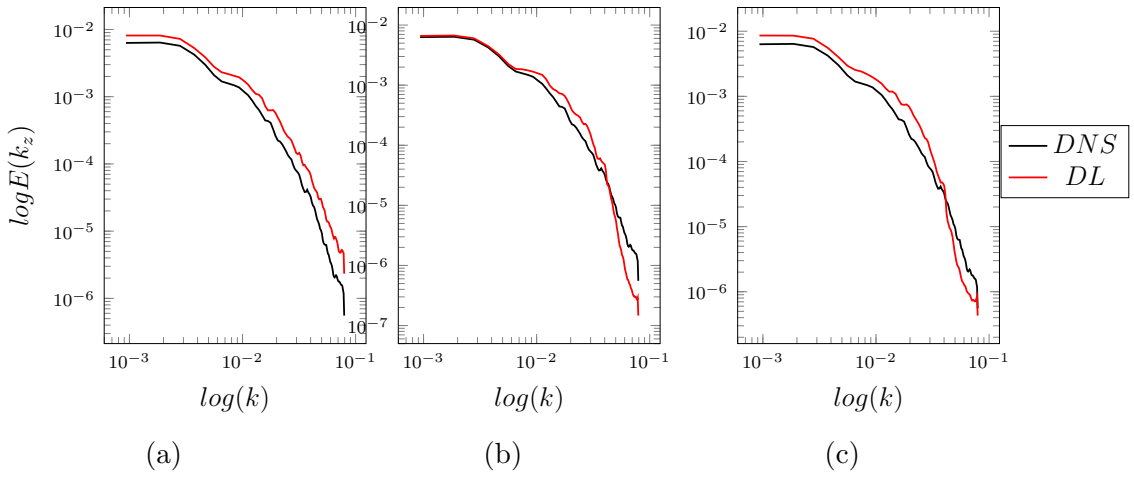


Figure 4.10: Comparison of refinement direction on the spanwise energy spectrum for the turbulent channel flow.

64× coarse-grained snapshot? Essentially, the effect of severe coarse-graining is investigated by successively refining the coarse-grained snapshot by 4× during each refinement. Thus for a DNS comparable solution of subgrid-scale reconstruction, a 64×, 16×, 4× coarse-grained snapshot would need one, two, and three successive refinements, respectively.

The subgrid-scale reconstructions for 64×, 16×, 4× coarse-grained snapshots are compared by measuring turbulence intensity as shown in figure 4.11. As expected, the 4× subgrid-scale reconstruction has better accuracy with respect to DNS, as compared to the 64× subgrid-scale reconstruction. As shown in figure 4.12 and figure 4.13, similar trends are observed for turbulence dissipation and enstrophy. It is observed that the 4× and 16× subgrid-scale reconstructions exhibit an *LES-like* behavior, whereas 64× subgrid-scale reconstruction shows a *RANS-like* behavior when turbulence intensity, dissipation, and enstrophy are compared . These results show how challenging it is for a deep learning model to get DNS-like solutions from coarse-grained data. When subgrid-scale reconstructions are investigated for streamwise and spanwise turbulence kinetic energy spectra as shown in figure 4.14, the streamwise spectra offer a distinct picture of reconstructions across scales. For streamwise spectra, 64× subgrid-scale reconstruction shows a clear and direct dissipation at larger scales, whereas 4× subgrid-scale reconstruction shows DNS comparable dissipation until smaller scales.

Additionally, root mean squared (RMS) turbulent velocities after subgrid-scale reconstructions are compared as a function of distance from the channel's wall. Subgrid-scale reconstructions for 64×, 16×, and 4× coarse-grained snapshots are compared. Turbulent velocity components along streamwise, wall-normal, and spanwise directions are compared to DNS data as shown in figure4.15. At higher y^+, i.e. away from the wall, the stream-wise velocity component shows expected agreement for all subgrid-scale reconstructions compared to DNS. Near the wall, only 4× subgrid-scale reconstructions show better agreement to that of DNS which is expected and consistent with the previous results. 64× subgrid-scale reconstructions show unexpected oscillations for all the turbulent velocity components, which could disappear with averaged statistics using more number of snapshots.

4.4.4 Application on Experimental Data

To demonstrate the robustness of the trained deep learning model for subgrid-scale reconstruction, experimental Particle Image Velocimetry (PIV) data is used. A typical PIV setup consists of flow in a wind tunnel seeded with tracer particles which get illuminated when passed across a laser sheet created in regions of interest. This tracer particle movement is captured with complementary metal–oxide–semiconductor sensors which are cameras with high shutter speeds, from which velocity fields are measured by calculating the speed and direction of tracer particles. High-resolution PIV can measure the flow at a small field-of-view but at a very high spatial resolution, but for studying a flow region with a bigger

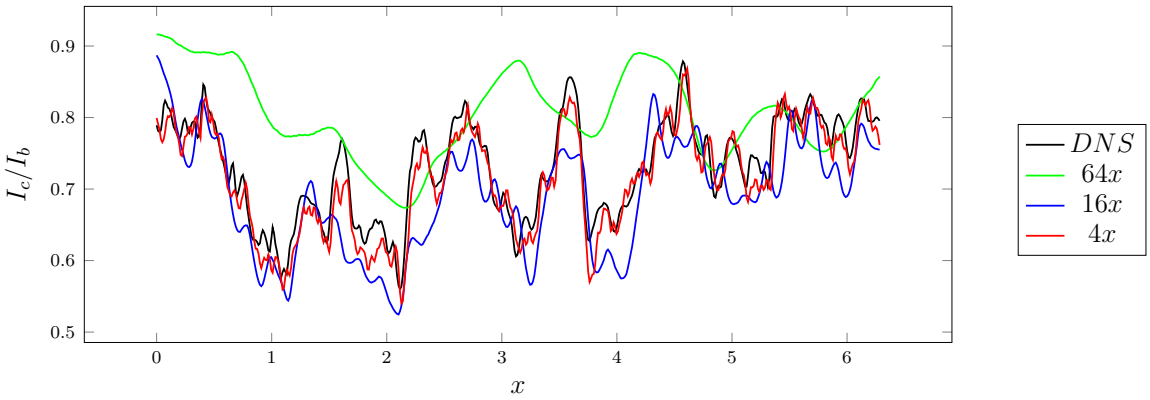


Figure 4.11: Comparison of successive refinement on the intensity of turbulence

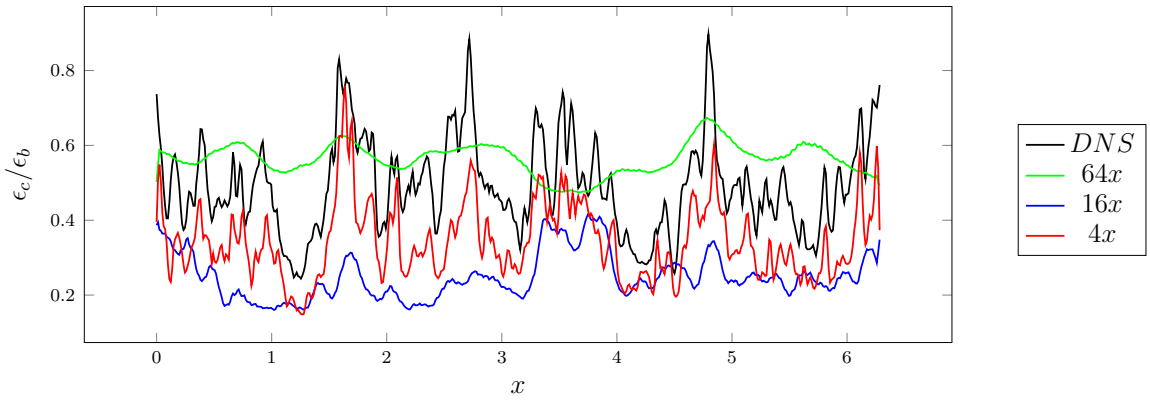


Figure 4.12: Comparison of successive refinement on the dissipation of turbulence

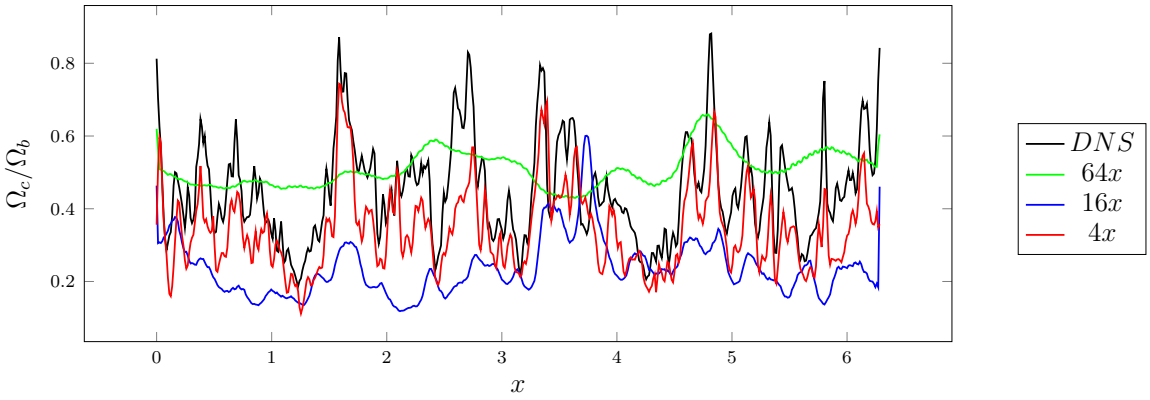


Figure 4.13: Comparison of successive refinement on the enstrophy of turbulence

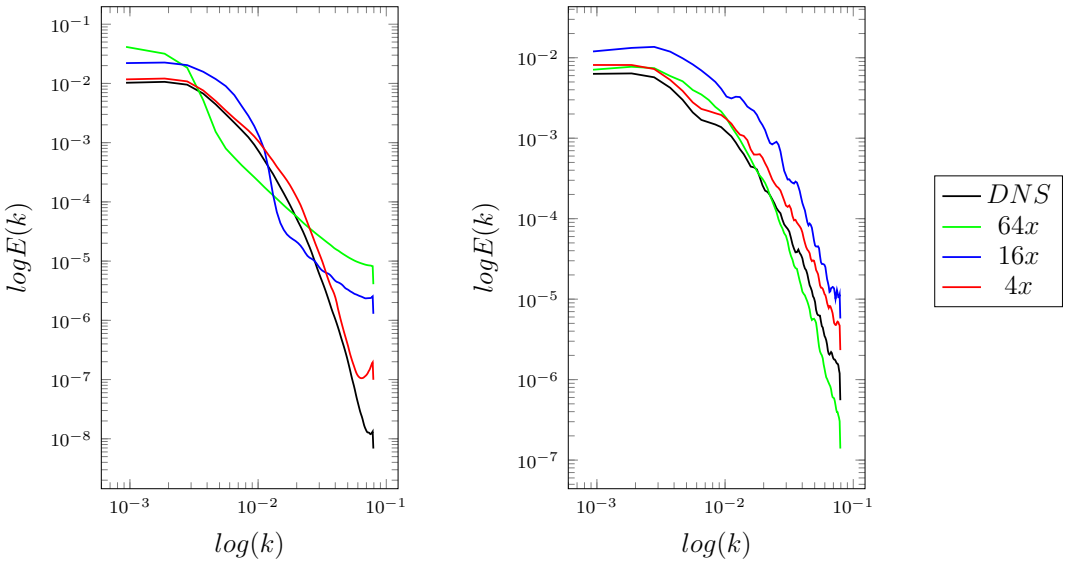


Figure 4.14: Comparison of successive refinement on the (a) streamwise (b) spanwise turbulence energy spectra

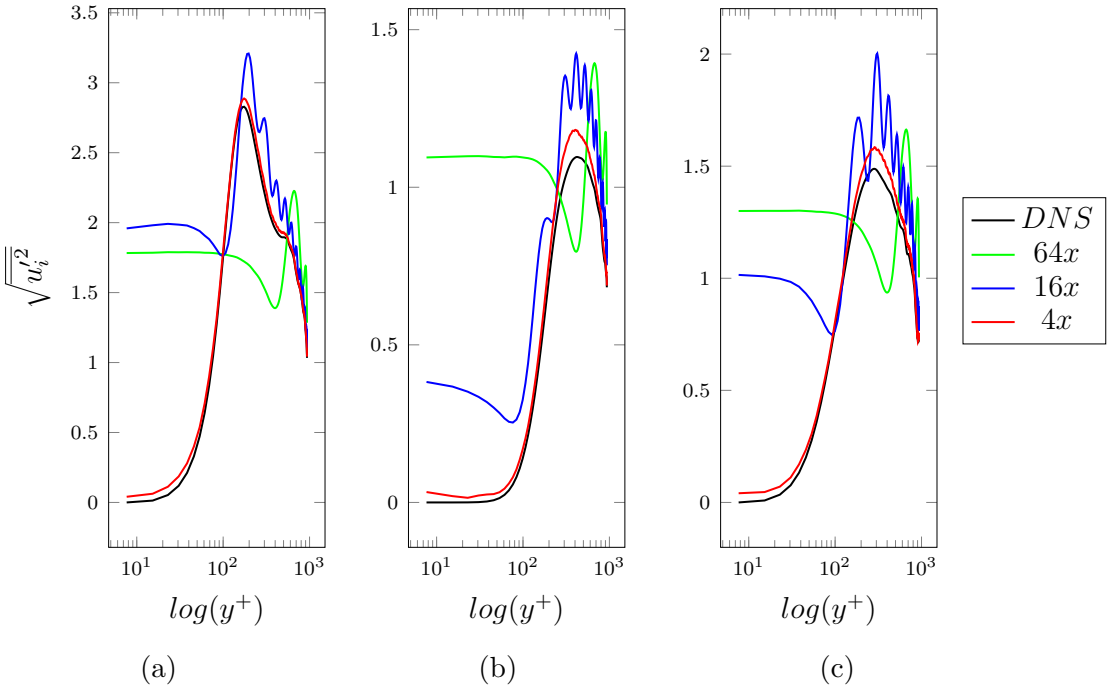
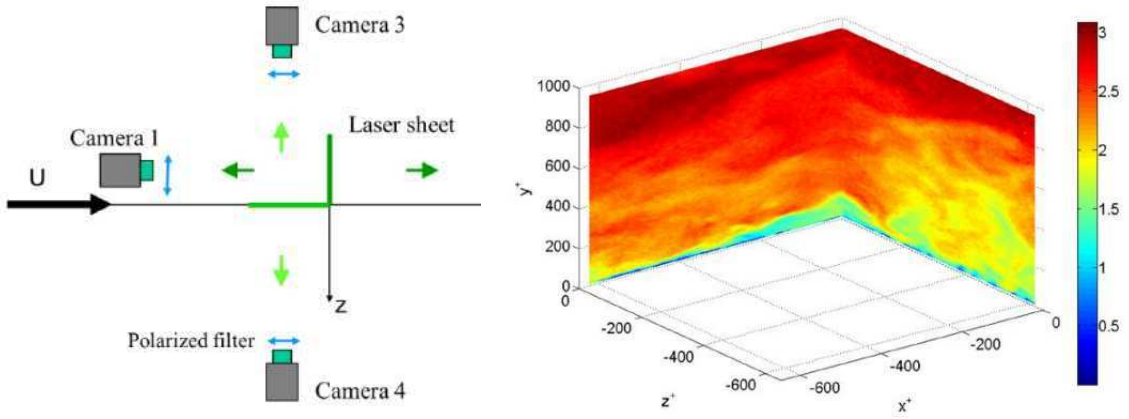


Figure 4.15: Comparison of successive refinement on the RMS of turbulent velocities in (a) streamwise (b) wall-normal (c) spanwise fluctuations as a function of distance from wall



(a) (b)

Figure 4.16: Stereo PIV setup at Lille Fluid Mechanics Laboratory (LMFL) used for studying the flat plate turbulent boundary layer (a) Top-view of the experimental set-up (b) Example of an instantaneous streamwise velocity field. Image courtesy: LMFL

field-of-view PIV resolution is usually sacrificed. This motivates the application of a trained deep learning model to reconstruct subgrid-scales or sub-pixel scales to enhance PIV data. Experimental data of flat plate turbulent boundary layer at Reynolds numbers $Re_\tau = 2300$ with two synchronized stereo-PIV perpendicular planes in the near wall region by Foucaut et al. [2016] is used. The experiment was performed in the Lille Mechanics Laboratory at the 20m long boundary layer facility, which is composed of two stereo-PIV systems acquiring data simultaneously. The two PIV planes are normal to the wall and intersect into a single line so that the set-up has a L shape as shown in figure 4.16. Each PIV system is composed of two Hamamatsu cameras with a resolution of $2k \times 2k$ capturing two fields of view of about $80mm$ in the spanwise or streamwise direction and $120mm$ in the wall-normal one.

Coarse-grained data by a factor of $4\times$ is obtained by sparse-pooling as described in previous sections. Subgrid-scale reconstruction is performed by the trained deep learning model by predicting full-field PIV snapshot from the $4\times$ coarse-grained snapshot. Figure 4.17 shows turbulence intensity measured along refinement directions in $x - y$ and $z - y$ planes. A good agreement is seen for subgrid-scale reconstruction when compared with PIV data. Similar trends are seen for turbulence dissipation and enstrophy of turbulence as shown in figure 4.18 and figure 4.19 respectively. Additionally, spectral contributions are examined by comparing streamwise and spanwise spectra as shown in figure 4.20 and figure 4.21, both of which show a good agreement across the range of scales.

Similar to the method described previously, we perform successive refinements on PIV data for $64\times$, $16\times$, and $4\times$ subgrid scale reconstruction. Figure 4.22 shows a combined plot of turbulence intensity, dissipation, and enstrophy for

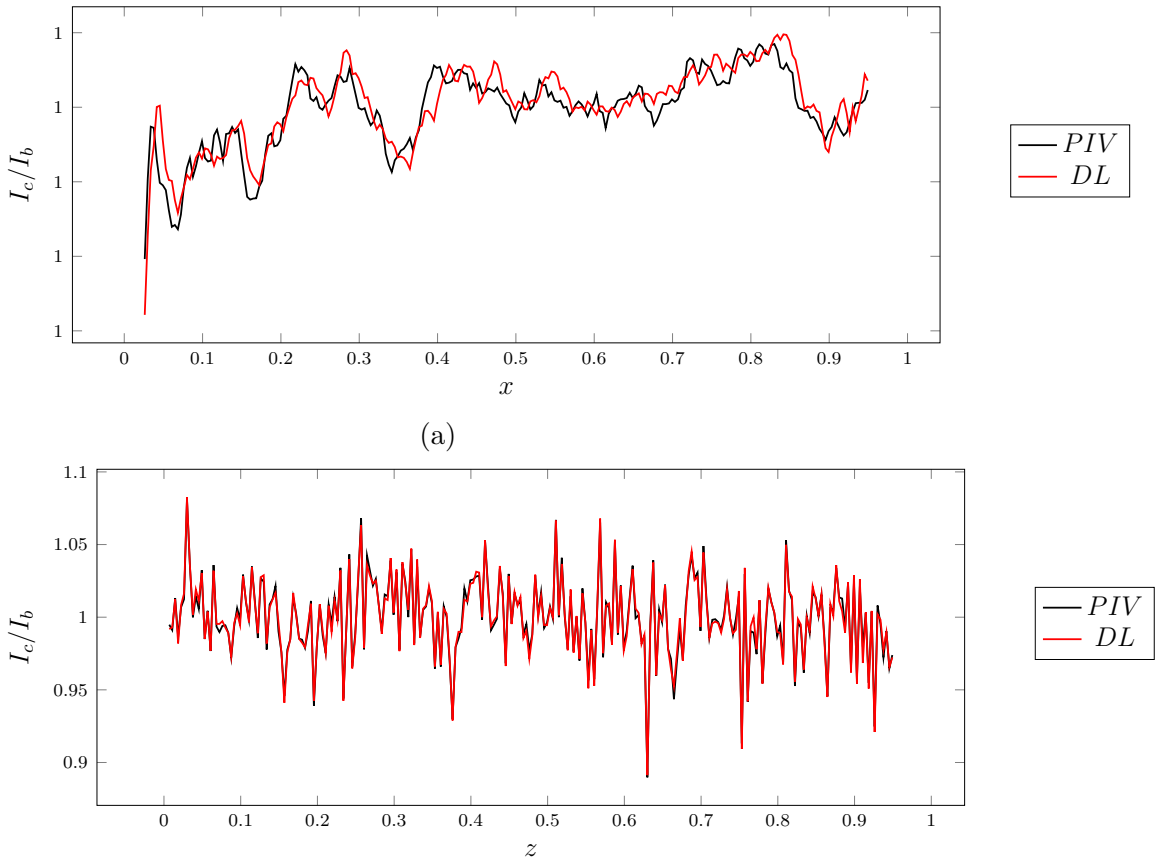


Figure 4.17: Comparison for PIV on the turbulence intensity by refinement direction in (a)XY (b)ZY

various coarse-graining levels. A good agreement is observed in subgrid-scale reconstructions against the PIV snapshot for the measured physical statistics. Similarly, streamwise and spanwise turbulent kinetic energy spectra are compared for 64×, 16×, and 4× subgrid scale reconstruction as shown in figure 4.23, which shows the expected level of good agreements with the PIV data.

4.5 Conclusions

The present study is on the intersection of computer vision, deep learning, and computational as well as experimental turbulence. The primary objective was to explore the state-of-the-art deep learning method to estimate subgrid-scale turbulence from measured large scales or grid-resolved scales. The key contribution of this study is to perform transfer learning by re-training a model trained for non-physics problems to re-learn new physical features. Coarse-grained fields are computed from the reference datasets to make the *a priori* predictions using the trained deep learning model on the DNS dataset. Subgrid-scale reconstructions

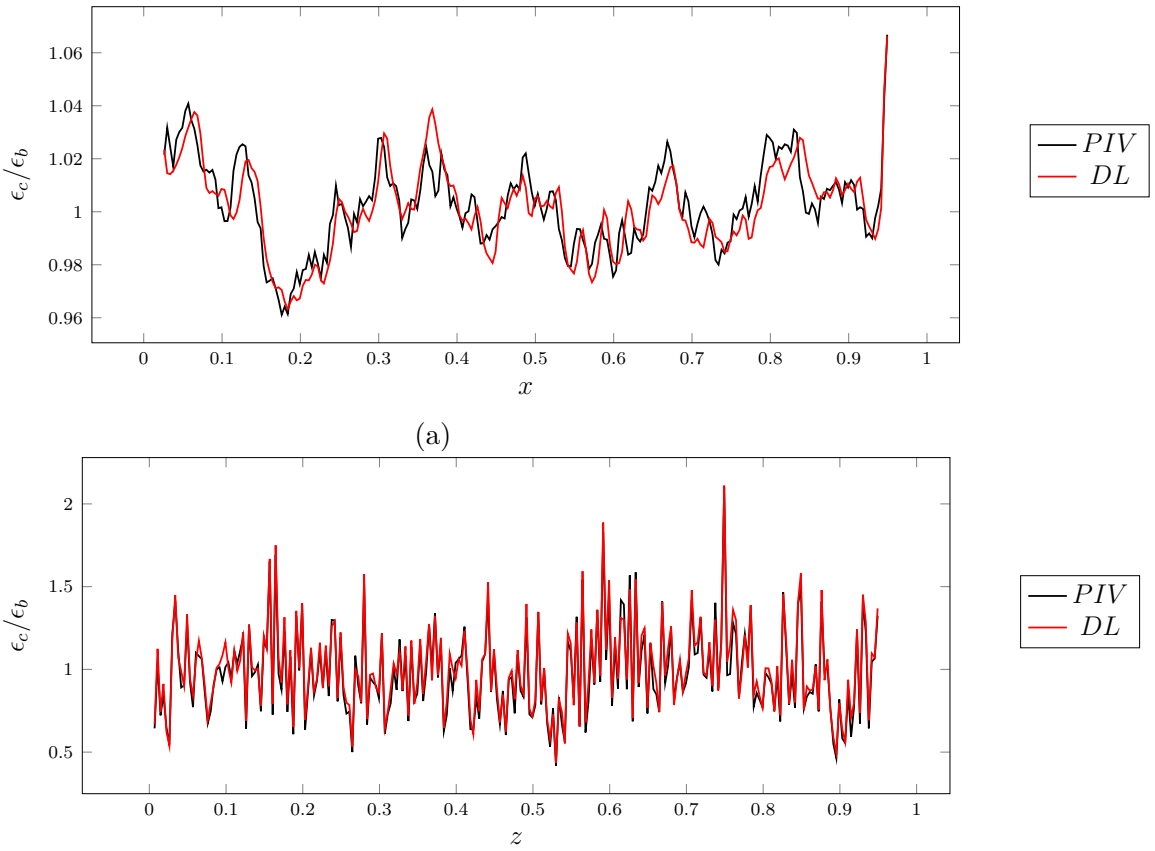


Figure 4.18: Comparison for PIV data on the turbulence dissipation by refinement direction in (a)XY (b)ZY

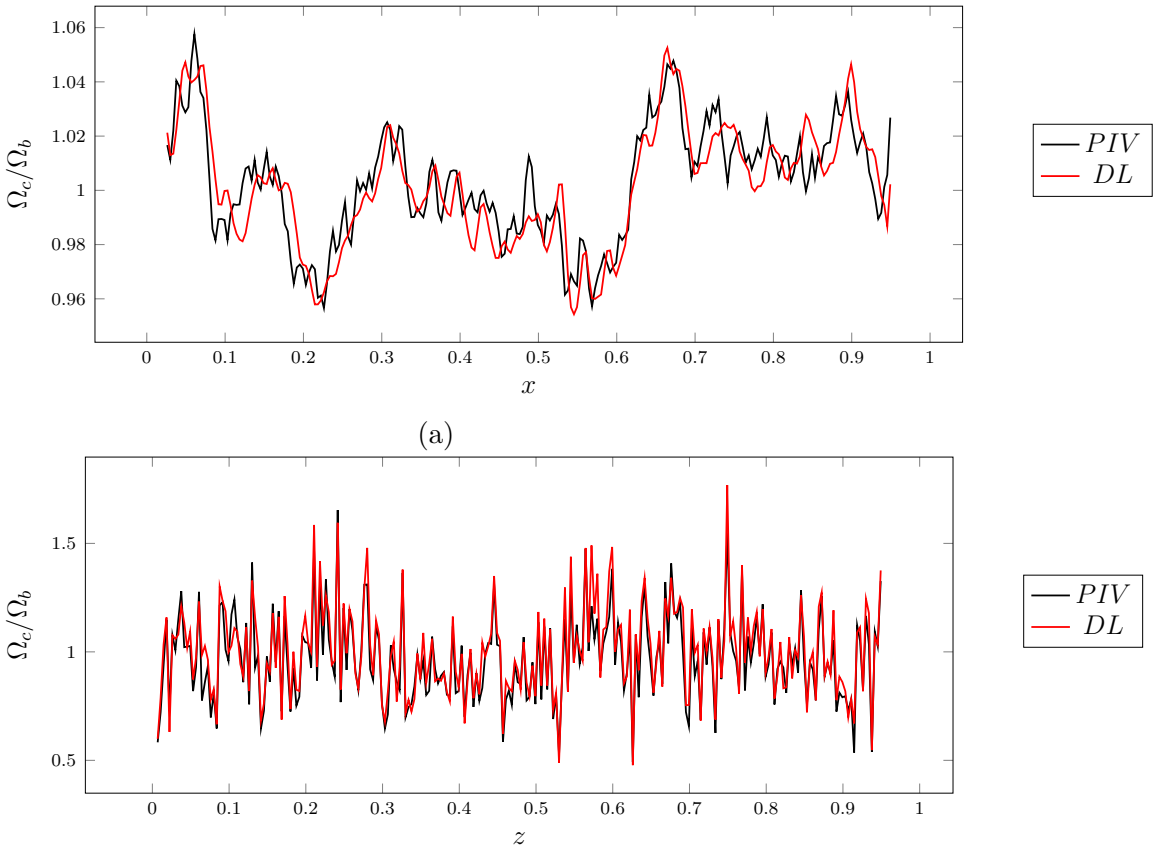


Figure 4.19: Comparison for PIV data on the turbulence enstrophy by refinement direction in (a)XY (b)ZY

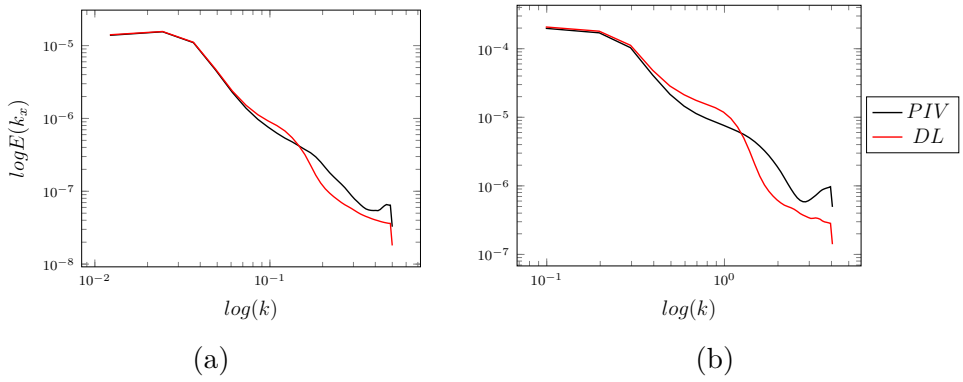
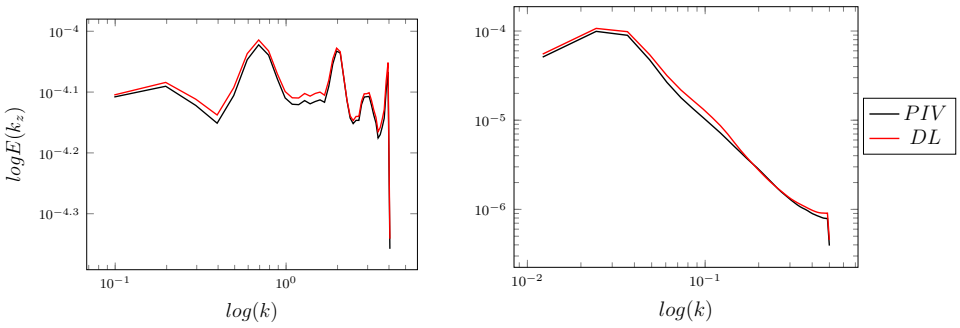


Figure 4.20: Comparison of refinement direction on the streamwise energy spectrum for the PIV data of turbulent channel flow.



(a) (b)

Figure 4.21: Comparison of refinement direction on the spanwise energy spectrum for the PIV data of turbulent channel flow.

are performed and compared with reference DNS as well as PIV datasets. In both datasets, wall-bounded turbulent channel flow datasets are used. Subgrid-scale reconstructions along various refinement directions and various coarse-graining levels are compared with the reference by measuring physical statistics like turbulence intensity, dissipation, enstrophy, and energy spectra.

PIV or HWA being limited by the hardware resolutions, experiments can only capture what is within the limits of their acquisition methods. The DNS or similar numerical simulations are limited by the grid resolutions and computing power. It is almost impossible to increase information transfer beyond this limit, but the features outside the limit can be reconstructed from features based on the global or historical context of the data, demonstrating the need for further work on deep learning to reconstruct grid-scale turbulence. Such an approach may allow going beyond what can be measured, computed, or stored. The results from this work would provide a way to design experiments or numerical simulations in such a way that deep learning facilitates the recovery of a maximum level of information. On the experimental side, time-resolved velocity fields at large field-of-view and high spatial resolution could be estimated with such a deep learning-assisted method.

Bibliography

Andrea Beck, David Flad, and Claus-Dieter Munz. Deep neural networks for data-driven les closure models. *Journal of Computational Physics*, 398:108910, 2019.

Jean-Marc Foucaut, Christophe Cuvier, Sebastien Coudert, and Michel Stanislas. 3d spatial correlation tensor from an l-shaped spiv experiment in the near wall region. In *Progress in Wall Turbulence 2*, pages 405–417. Springer, 2016.

Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. *J. Fluid Mech.*, 870:106–120, 2019a.

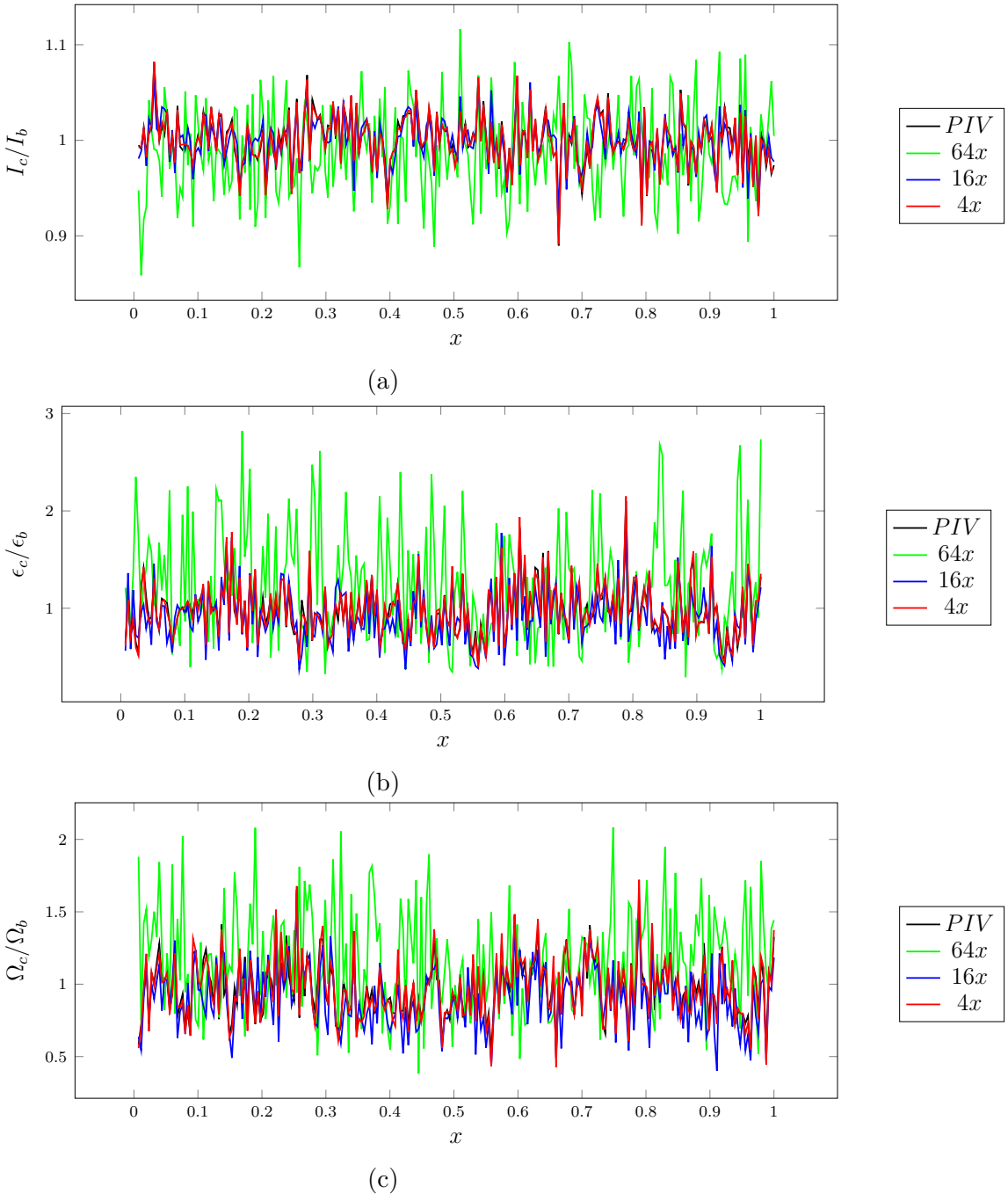


Figure 4.22: Comparison for PIV data with successive refinement on (a) the intensity of turbulence (b) dissipation of turbulence (c) enstrophy of turbulence

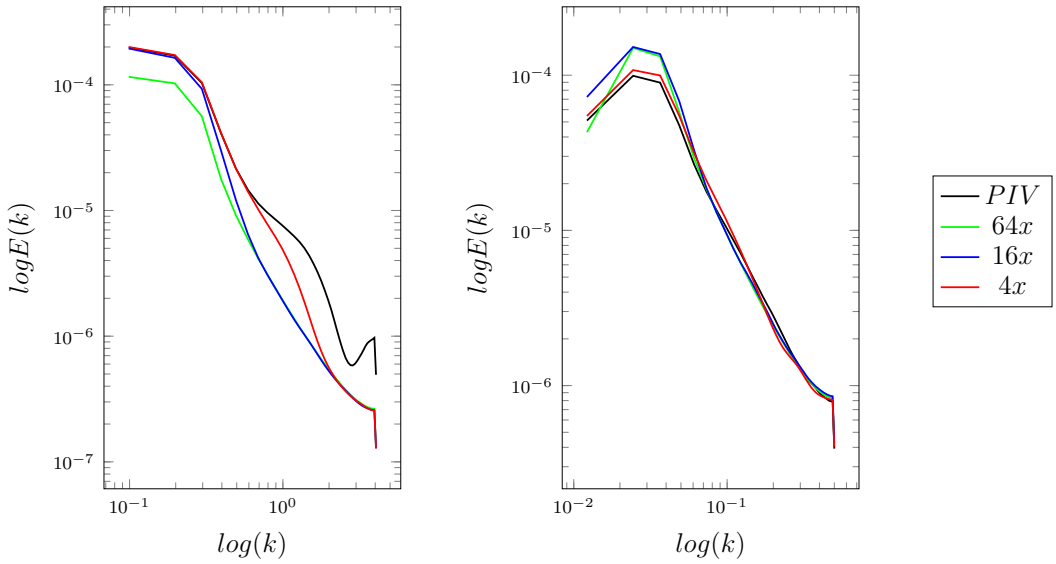


Figure 4.23: Comparison for PIV data with successive refinement on the (a) streamwise (b) spanwise turbulence energy spectra

Kai Fukami, Yusuke Nabae, Ken Kawai, and Koji Fukagata. Synthetic turbulent inflow generator using machine learning. *Physical Review Fluids*, 4(6):064603, 2019b.

Kai Fukami, Koji Fukagata, and Kunihiko Taira. Machine learning based spatio-temporal super resolution reconstruction of turbulent flows. *arXiv preprint arXiv:2004.11566*, 2020.

Kai Fukami, Koji Fukagata, and Kunihiko Taira. Machine-learning-based spatio-temporal super resolution reconstruction of turbulent flows. *Journal of Fluid Mechanics*, 909, 2021.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Julian CR Hunt and David J Carruthers. Rapid distortion theory and the 'problems' of turbulence. *Journal of Fluid Mechanics*, 212:497–532, 1990.

Chiyu Max Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Anima Anandkumar, et al. Meshfreeflownet: A physics-constrained deep continuous space-time super-resolution framework. *arXiv preprint arXiv:2005.01463*, 2020.

Junhyuk Kim and Changhoon Lee. Deep unsupervised learning of turbulence for inflow generation at various reynolds numbers. *arXiv preprint arXiv:1908.10515*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

C Vamsi Krishna, Mengying Wang, Maziar Hemati, and Mitul Luhar. Fusion of physics-based models with field measurements for turbulent flow reconstruction. In *11th International Symposium on Turbulence and Shear Flow Phenomena, TSFP 2019*, 2019.

C Vamsi Krishna, Mengying Wang, Maziar S Hemati, and Mitul Luhar. Reconstructing the time evolution of wall-bounded turbulent flows from non-time-resolved piv measurements. *Physical Review Fluids*, 5(5):054604, 2020.

Adrián Lozano-Durán and Javier Jiménez. Effect of the computational domain on direct simulations of turbulent channels up to re τ= 4200. *Physics of Fluids*, 26(1):011702, 2014.

Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence. *arXiv preprint arXiv:1903.00033*, 2019.

Linh Van Nguyen, Jean-Philippe Laval, and Pierre Chainais. A Bayesian fusion model for space-time reconstruction of finely resolved velocities in turbulent flows from low resolution measurements. *Journal of Statistical Mechanics: Theory and Experiment*, 2015(10):P10008, 2015.

Hua Wang, Dewei Su, Chuangchuang Liu, Longcun Jin, Xianfang Sun, and Xinyi Peng. Deformable non-local network for video super-resolution. *IEEE Access*, 7:177734–177744, 2019.

Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018.

Chapter 5

Auto-Regressive Learning of Spatio-Temporal Turbulence

Un modèle de transformateur basé sur un codeur-décodeur convolutif est proposé pour l'entraînement autorégressif sur des données spatio-temporelles d'écoulements turbulents. La prédiction des champs d'écoulement futurs est basée sur le champ d'écoulement précédemment prédit afin de garantir des prédictions à long terme sans divergence. Une combinaison de réseaux neuronaux convolutifs et d'architecture de transformateur est utilisée pour traiter les dimensions spatiales et temporelles des données. Pour évaluer les performances du modèle, des évaluations a priori sont effectuées et des accords significatifs sont trouvés avec les données de la vérité de terrain. Les prédictions a posteriori, qui sont générées après un nombre considérable d'étapes de simulation, présentent des variances prédites. L'apprentissage autorégressif et la prédiction des états a posteriori sont considérés comme des étapes cruciales vers le développement de modèles et de simulations de turbulence plus complexes basés sur des données. La dynamique hautement non linéaire et chaotique des flux turbulents peut être gérée par le modèle proposé, et des prédictions précises sur de longues périodes peuvent être générées. Dans l'ensemble, cette approche démontre le potentiel des techniques d'apprentissage profond pour améliorer la précision et l'efficacité de la modélisation et de la simulation des turbulences. Le modèle proposé peut être optimisé et étendu pour incorporer des conditions physiques et limites supplémentaires, ouvrant ainsi la voie à des simulations plus réalistes de la dynamique des fluides complexes.

5.1 Introduction

The main factor in turbulent flows is convection, which makes tasks such as flow control and model reduction complex and challenging. These tasks become nonlinear, high-dimensional, multi-scale, and non-convex optimization problems due to the dominance of convection over diffusion. Due to the vast amount of numerical and experimental data available for turbulent flows, data-driven approaches are now gaining popularity in the fluid mechanics community. These approaches use deep learning models to make predictions and represent a valid alternative to traditional methods. This article explores a new data-driven approach based on deep learning to estimate future fluid flow fields from previous ones. The proposed method uses a novel convolutional encoder-decoder transformer model and autoregressive training to achieve long-term spatio-temporal predictions. The approach is tested on two turbulent fluid flow cases, namely a wake-flow past a stationary obstacle and an environmental flow past a tower fixed on a surface. The results show the effectiveness of the proposed method in predicting the fluid flow fields accurately, highlighting the potential of data-driven approaches in solving challenging problems in fluid mechanics.

There are several traditional ways to address temporal estimations, such as Koopman theory and proper orthogonal decomposition, which are suitable for prediction and control Williams et al. [2015]; Brunton et al. [2016]; Rowley and Dawson [2017]. Additionally, data assimilation schemes are popular, where the model weights are updated to reflect new observations Mons et al. [2016]. In recent years, supervised learning techniques using neural networks have been applied to capture nonlinear relations between past and future states. For example, a recurrent neural network with long-short term memory was used to predict the chaotic Lorenz system, and convolutional networks were used to predict transient flows Dubois et al. [2020]; Xu and Duraisamy [2020]. There have also been attempts to approximate the full Navier-Stokes equations using deep neural networks, but prediction accuracy decreased significantly for chaotic and turbulent flows Lusch et al. [2018]; Sirignano and Spiliopoulos [2018]; Tang et al. [2021]; Sun et al. [2020]. Regarding the estimation of flow fields using deep neural networks, several studies have focused on spatial and temporal reconstruction, as well as spatial supersampling Cheng and Zhang [2021]; Yousif et al. [2021]; Schmidt et al. [2021]. Hybrid deep neural network architectures have been designed to capture the spatial-temporal features of unsteady flows Han et al. [2019], and machine learning-based reduced-order models have been proposed for three-dimensional complex flows Nakamura et al. [2021]. A deep learning framework combining long short-term memory networks and convolutional neural networks has been used to predict the temporal evolution of turbulent flames Ren et al. [2021]. However, despite the significant progress made in the acceleration of flow simulation, these models still suffer from the generalization problem and are sensitive to parameter changes Kochkov et al. [2021].

New deep learning architectures for temporal problems in unstructured and

structured data are emerging, with transformers being one of the most promising. These models make use of self-attention mechanisms to differentially weight the significance of each part of the input data Vaswani et al. [2017]; Bahdanau et al. [2014], without the need for recurrent network architecture. Inspired by neighborhood-like notions in convolutional neural networks, transformers build features of inputs using a self-attention mechanism to determine the importance of other samples in the dataset with respect to the current sample. The updated features of the inputs are simply the sum of linear transformations of all features weighted by their importance. Transformers avoid recurrence by using the self-attention mechanism, which accounts for the similarity score between elements of a sequence and the positional embedding of these elements, allowing them to account for the full sequence instead of single elements. These models have been successful in natural language processing (NLP) tasks such as translation and text summarization, and are becoming the model of choice for NLP problems, replacing classical recurrent neural network (RNN) models such as long short-term memory (LSTM) Wolf et al. [2020]; Devlin et al. [2018]; Radford et al. [2019]. Transformers have also been applied to image processing tasks using convolutional neural networks to capture relationships between different portions of an image Dosovitskiy et al. [2020]; Parmar et al. [2018]; Touvron et al. [2021]. Hybrid architectures combining convolutional layers with transformers have achieved excellent results in several computer vision tasks Dai et al. [2021]; Wu et al. [2021a]. In spatio-temporal context, transformers have been used for video-understanding tasks, capturing spatial and temporal information through the use of divided space-time attention Sharir et al. [2021]; Bertasius et al. [2021]. In fluid mechanics, attention mechanisms have enhanced the reduced-order model to extract temporal feature relationships from high-fidelity numerical solutions Wu et al. [2021b]. Recently, a similar combination of autoregressive transformers and two-dimensional homogeneous isotropic turbulence was proposed for spatio-temporal prediction of flow fields Peng et al. [2022]. However, transformers have never been used for spatio-temporal prediction of flow fields involving turbulent flows.

The present contribution is organized as follows: first, the deep learning method based on the convolutional self-attention transformer is discussed, after which focus is made on the autoregressive training procedure. The following section provides insights into the performance of the proposed approach by considering (i) a turbulent flow case with an obstacle embedded in a rectangular domain, and (ii) a surface-mounted tower in an open flow. This part is followed by a discussion and a conclusion.

5.2 Deep Learning Method

The primary focus of this contribution is to address the challenge of learning the spatio-temporal dynamics of turbulent flows, which are known for their high complexity, non-linear behavior, and high dimensionality. There are two main approaches to estimate a reference spatio-temporal field X_t: (i) reconstruction,

which involves utilizing limited measurements $\tilde{X}_t$ at a specific time t to reconstruct the full X_t field at the same time, and (ii) prediction, where a dynamical model is utilized to advance the field in time based on previous estimates. Here, spatio-temporal learning is formulated as a task with a given time-series containing N sequential snapshots $\left[x_t, x_{t+\Delta t}, ..., x_{t+(N-1)\Delta t}\right]$, in order to predict the same quantity of interest on M steps ahead in time. The input X of the deep learning model is $\left[x_t, x_{t+\Delta t}, ..., x_{t+(N-1)\Delta t}\right]$, and the output Y is $\left[x_{t+N\Delta t}, ..., x_{t+N+(M-1)\Delta t}\right]$. Each snapshot x_t can be a scalar field or a vector field containing multiple features.

Transformer models in deep learning were developed to address natural language processing problems, where sentence completion and translation are performed using a word by word embedding Vaswani et al. [2017]. The sentence-completing NLP tasks can be understood as a temporal learning problem, with a time series of words or sentence tensors measured over a time duration. These models have achieved remarkable performances in a variety of other tasks, including learning image patches as sequences, image completion and reconstruction Vaswani et al. [2017]; Dai et al. [2021]; Wu et al. [2021a]. As a result, the transformer models have been challenging the classic long short-term memory (LSTM) models, the *de facto* RNNs, and replacing them with a state-of-the-art approach in a variety of temporal learning tasks.

Like RNNs, transformers are designed to handle sequential input data. However, unlike the latter, they do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence, and self-attention itself identifies/learns the weights of attention. In the case of spatio-temporal data, the attention can be applied to the spatial as well as the temporal sequence to attend to or pay attention to. The vanilla transformers in their original form are pure sequence to sequence models, as they learn a target output sequence from an input sequence, *i.e.* they perform transformation at the sequence level. Their limitations, such as disrupting temporal coherence and failing to capture long-term dependencies, were reached for sentence completion of language generation tasks, where difficulties were noted while generating texts with a model which learns sequences without the knowledge of full-sequences Bahdanau et al. [2014]; Yu et al. [2017]; Guo et al. [2018]. Several studies were performed, such as that of Dai *et al.* Dai et al. [2019], to address this inability to capture long-term dependencies by attending to memories from previously learned parameters, yet at the expense of computing costs. To deal with some of these issues, autoregressive transformers were proposed by Katharopoulos et al. [2020] for sentence and image completion tasks. Although not explicitly stated in some works, the Generative Pre-trained Transformer (GPT) family of models Radford et al. [2018, 2019]; Brown et al. [2020] are in fact autoregressive transformers inspired by the decoder part of the original transformers. In Katharopoulos et al. [2020], Katharopoulous *et al.* showed that a self-attention layer trained in an autoregressive fashion can be seen as a recurrent neural network. Transformers can be combined with the classic convolutional encoder-decoder type models

to harness their full potential when the input and target output tensors are in a spatio-temporal form. As locality is more important in learning small-scale features, this combination serves as a powerful method for a variety of computer-vision problems, including video-frame prediction. The self-attention mechanism on convolutional layers not only *attends* or focuses on a sequence of significance, but it also improves the representation of spatially-relevant regions by focusing on important features and suppressing less-important ones Woo et al. [2018].

When a transformer block is applied after a convolutional layer, the model learns to emphasize meaningful features along the channel sequence and spatial dimensions. The input sequences are first appended channel-wise to the input layer and subsequent convolutional operations are performed in the encoder. In the convolutional layers, the intermediate feature maps $\mathbb{F} \in \mathbb{R}^{C\times H\times W}$ of a given layer are passed through the self-attention convolutional transformer layer conv_α, which simultaneously attends to spatial representation and the positional embeddings of the input sequence channels. In conv_α, let $x, y \in \mathbb{R}^C$ be the input and output intermediate feature tensors, with C representing the number of intermediate channels. When $i, j \in \mathbb{R}^{H\times W}$ are indices of the spatial nodes, a classical convolution operation is performed such that:

$$y_i = \sum_{j\in N(i)} W_{i\to j} x_j, \tag{5.1}$$

where $N(i)$ represents the spatial nodes in a local neighborhood defined by a kernel of size $k \times k$ centered at node i, $i \to j$ represents the relative spatial relationship from i to j, and $W_{i\to j} \in \mathbb{R}^{C\times C}$ is the weight matrix. On the other hand, self-attention for intermediate convolutional features has three weight matrices $W_q, W_k, W_v \in \mathbb{R}^{C\times C}$ to compute query, key and value respectively. For each convolution window, the self-attention is given as:

$$\begin{aligned} y_i &= \sum_{j\in N(i)} \alpha_{i\to j} W_v x_j, \\ \alpha_{i\to j} &= \frac{e^{(W_q x_i)^T W_k x_j}}{\sum_{z\in N(i)} e^{(W_q x_i)^T W_k x_z}} = \frac{W_{qk} x_i[j]}{\sum_{z\in N(i)} W_{qk} x_i[z]}, \end{aligned} \tag{5.2}$$

where the self-attention $\alpha_{i\to j} \in (0, 1)$ is a scalar that controls the contribution of values in spatial nodes, with $W_{qk} \in \mathbb{R}^{C\times k^2}$, and $[j]$ means j^{th} element of the tensor. α is usually normalized by a softmax operation such that $\sum_j \alpha_{i\to j} = 1$. These operations are summarized in figure 5.1.

Combining equations (5.1) and (5.2), one obtains both an input sequence dependent kernel and the learnable convolution filters providing the final output feature map $\mathbb{F}''$ by convolutional transformer layer, given as:

$$\begin{aligned} y_i &= \sum_{j\in N(i)} \text{softmax}\left(\alpha_{i\to j}\right) W_{i\to j} x_j \\ i.e.\ \mathbb{F}'' &= \text{conv}_\alpha(\mathbb{F}) \end{aligned} \tag{5.3}$$

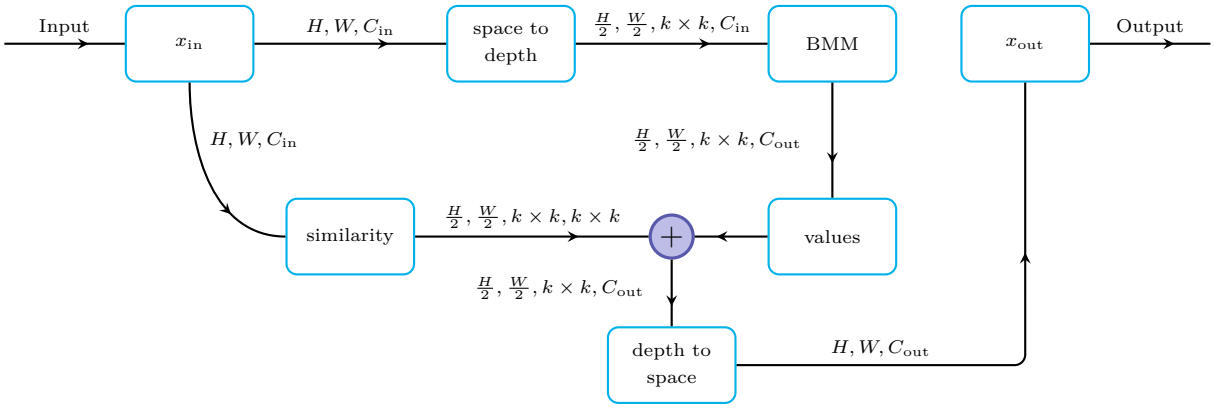


Figure 5.1: **The convolutional transformer layer** is composed of two blocks: the batched matrix multiplication (BMM) and the self-attention summation. The BMM block corresponds to $W_{i \to j} x_j$ in equation (5.2), with the batch dimension being the number of spatial locations. It performs $k \times k$ different input-dependent summations with the weights α in equation (5.2). It contains both the learnable filter and the dynamic kernel.

The current self-attention convolutional transformer layer has a 3×3 kernel and incorporates the representation of convolutional features. Combining convolutional neural networks with self-attention thus offers superior learning capabilities of spatio-temporal structures, which would benefit turbulent flows and CFD in general, where one learns spatial filters as well as temporal embeddings and dependencies. In addition to the convolutional transformer layer, the model is trained in an autoregressive fashion. Formally, autoregressive models are those which forecast future sequences from the previously forecasted sequences in a cyclical way, and thus here *auto* indicates the regression of the variable sequence against itself.

In turbulent flow problems, the high-dimensional state-space is characterized by intricate spatio-temporal dynamics, and therefore, dimensionality reduction techniques can be useful Dubois et al. [2022]. Reconstruction and prediction problems are therefore equivalent to the estimation of the reduced or latent state, thus making the use of encoder-decoder based architecture a natural choice. The encoder-decoder architecture comprises an encoder that takes input tensors and maps them to a high-dimensional representation by learning which parts of the input tensors are important and converts or passes them to abstract low-dimensional representation. With the addition of a decoder after this encoder, this high-dimensional representation is converted to target output tensors. By chaining the encoder and decoder together, their weight matrices jointly learn the output tensors from input tensors, thus helping to learn the small-scale features. The decoder is comprised of successive up-samplings followed by convolutions, and brings the latent space representation of dimension $n_z \times n_z$ back to the original spatial dimensions of the target output at time $x_{t+\Delta t}$. The figure 5.2 offers a global view of the considered deep learning architecture.

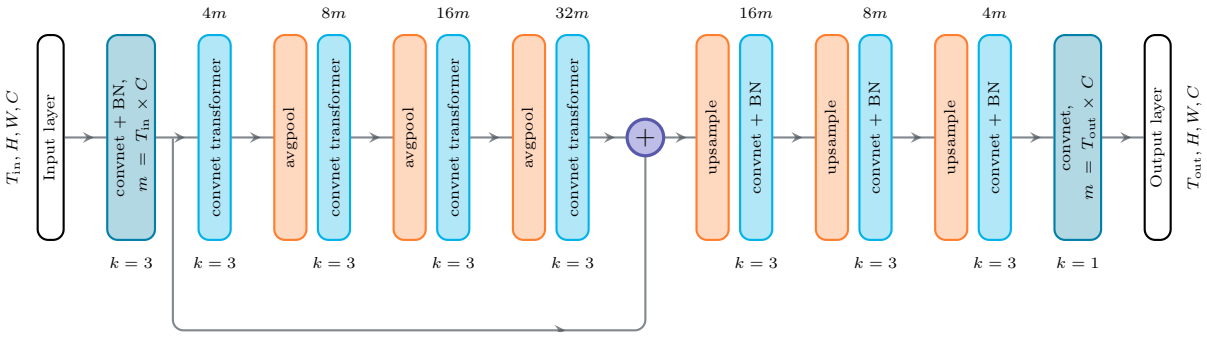


Figure 5.2: **Convolutional encoder-decoder transformer architecture** Model architecture of the convolutional encoder-decoder transformer to process low and high level features. The canonical four-stage design is utilized in addition to the convolutional transformer blocks or layers. H, W are the input resolutions for each snapshot in T_{in} sequence and T_{out} sequence.

For a trained model $\mathbb{M}$ as shown in figure 5.2, multi-step training is performed for quantity X_t in an auto-regressive manner, *i.e.* $X_{t+\Delta t}$ is predicted from previously predicted X_t, where t is some non-dimensional time. In other words, an initial condition X_t is inputted to the model to learn $\widehat{X}_{t+\Delta t}$, after what this predicted $\widehat{X}_{t+\Delta t}$ is then fed back to the model again to learn $\widehat{X}_{t+2\Delta t}$ and so on, in an autoregressive manner:

$$\begin{cases} \widehat{X}_{t+\Delta t} = \mathbb{M}(X_t), \\ \widehat{X}_{t+2\Delta t} = \mathbb{M}(\widehat{X}_{t+\Delta t}), \\ \qquad \ldots \\ \widehat{X}_{t+(n-1)\Delta t} = \mathbb{M}(\widehat{X}_{t+(n-2)\Delta t}), \end{cases} \tag{5.4}$$

where t is the time step and $X \in \mathbb{R}^{C \times H \times W}$ the input tensor snapshot at instant t. In the following, the autoregressive training sequence length is set equal to two in order to limit the computational cost.

In order to preserve meaningful values at the boundaries, the convolutional filters used in the proposed architecture incorporate a symmetric boundary condition into the padding operation. Classically, padding is used to preserve the spatial dimensions of the field being convoluted, but the standard zero-padding approach doesn't usually represent the expected physical behavior. Indeed, padding with zeros everywhere would violate the representation of existing boundary conditions, for example, the notion of wall boundaries would have lesser significance if a region is padded with zeros on all the sides in a channel flow. To preserve the boundary conditions after multiple successive convolutions, a boundary condition formulation was implemented such that the walls could be padded with zeros if required, while the other sides could be padded with adequate values from the symmetric cells. The resulting model is trained with the Adam Kingma and Ba [2014] optimizer, to iteratively minimize the total equi-weighted mean squared error (MSE) loss defined by:

$$\mathcal{L} = \frac{1}{n_s}\Bigg[\sum_{i=1}^{n_s}\left(\left(X_{t+\Delta t}\right)^i - \left(\widehat{X}_{t+\Delta t}\right)^i\right)^2 + \sum_{i=1}^{n_s}\left(\left(X_{t+2\Delta t}\right)^i - \left(\widehat{X}_{t+2\Delta t}\right)^i\right)^2 + \cdots + \sum_{i=1}^{n_s}\left(\left(X_{t+(n-1)\Delta t}\right)^i - \left(\widehat{X}_{t+(n-1)\Delta t}\right)^i\right)^2\Bigg]. \tag{5.5}$$

The ReLU function was used as an activation function, which is known to be an effective tool for stabilizing the weight update in the training process Nair and Hinton [2010]. The full training dataset is shown repeatedly to the network after a shuffling step during the training, and each pass is referred to as an epoch. An early stopping criterion is used along with a reduction of the learning rate if learning doesn't improve after every 100 epochs. The neural network was implemented using TensorFlow Abadi et al. [2016], and trained on Nvidia Tesla V100 GPUs.

5.3 Numerical simulation cases and data generation

5.3.1 Governing equations

The evolution of the velocity $\boldsymbol{u}$ and pressure p in an incompressible fluid flow with given positive constant density ρ and dynamic viscosity μ is governed by the Navier-Stokes momentum and continuity equations:

$$\begin{aligned} \rho\left(\partial_t \boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) &= \nabla\cdot\left(-p\,\mathbf{I}_d + 2\mu\,\boldsymbol{\varepsilon}(\boldsymbol{u})\right) + \boldsymbol{f}, \\ \nabla\cdot\boldsymbol{u} &= 0, \end{aligned} \tag{5.6}$$

where $\boldsymbol{\varepsilon}(\boldsymbol{u})$ is the strain-rate tensor, $\mathbf{I}_d$ is the d-dimensional identity tensor, and $\boldsymbol{f}$ is the additional forcing or source term. Equations (5.6) are supplemented with adequate boundary and initial conditions based on the physical cases. Turbulence is modeled as an additional diffusivity called eddy viscosity μ_t which itself proceeds from a model involving one or more turbulent scales, each of which is the solution of a nonlinear convection-diffusion-reaction equation. The turbulence model chosen to compute the eddy viscosity is the one-equation Spalart-Allmaras (SA) model Spalart and Allmaras [1992], which describes the evolution of the kinematic eddy viscosity like variable ν_t by solving a convection-diffusion-reaction problem.

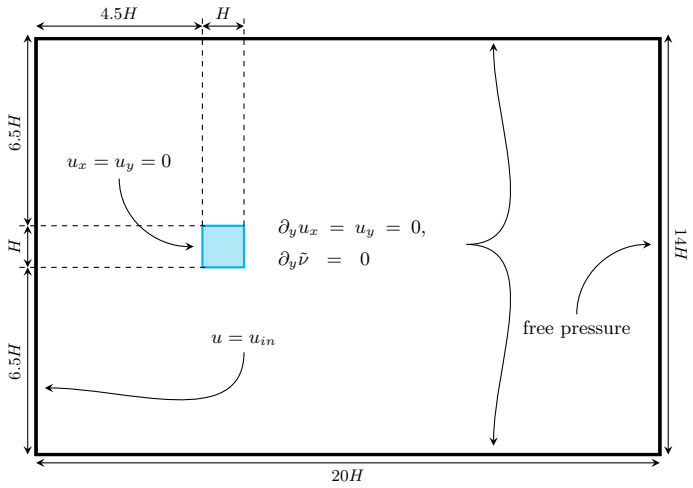
5.3.2 Case 1 : Wake-flow past a square cylinder

A widely benchmarked turbulent flow past a two-dimensional (2D) square cylinder is considered. The baseline Reynolds number is set to 22×10^3, based on

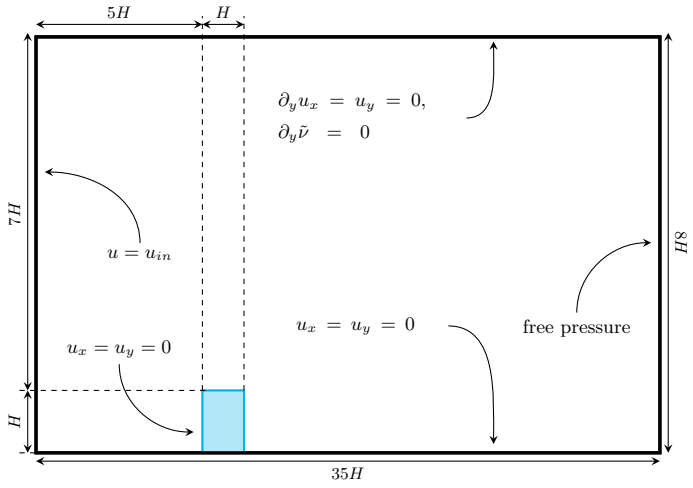
the reference velocity U_∞ and the cylinder lateral size H which is centered at the origin of the domain. The dimensions of the computational domain are $[-5H, 15H] \times [-7H, 7H]$ in the streamwise x and crosswise y directions respectively, and the domain is discretized into sufficiently large number of cells to perform a URANS or VMS simulation using an in-house finite-element flow solver Bazilevs et al. [2007]; Takizawa et al. [2018]; Hachem et al. [2013]; Guiza et al. [2020]. The inflow boundary conditions are $u = (V_{in}, 0)$, together with $\tilde{\nu} = 3\nu$, which corresponds to a ratio of eddy to kinematic viscosity of approximately 0.2. For the lateral boundaries, symmetry conditions $\partial_y u_x = u_y = 0$ and $\partial_y \tilde{\nu} = 0$ are used. For the outflow, $\partial_x u_x = \partial_x u_y = 0$, $\partial_x \tilde{\nu} = 0$ together with $p = 0$ are prescribed. Finally, no-slip conditions $u = 0$ and $\tilde{\nu} = 0$ are imposed at the cylinder surface. The time step is $\Delta t = 0.05$ seconds, and the simulation is performed for a total physical time equal to 5000 seconds. Since the wake flow is of interest, around 200 seconds are required for the flow to be established and reach periodic vortex shedding. The data corresponding to the remaining 4980 seconds is stored for training and testing purposes. The data is sampled at each $\Delta t = 0.25$ seconds, thus collecting around 1500 snapshots. In terms of non-dimensional time defined as $t^* = tU_\infty/H$, this sampling rate corresponds to $\Delta t^* = 1$. One vortex shedding cycle corresponds to a non-dimensional period $T^* = 5.23$ units, and approximately 24 shedding cycles are observed in simulation data. Given the 70/30 splitting strategy, 16 shedding cycles are observed in training data, which seems reasonable to fully characterize the dynamics of wake turbulent flow past a two-dimensional square cylinder considering its simplicity. Figure shows the sketch of the associated case.

5.3.3 Case 2 : Environmental flow over surface-mounted tower

The turbulent flow past a two-dimensional (2D) rectangular tower on the land surface is considered. The baseline Reynolds number is set to 45×10^2, based on the reference velocity U_∞ and the square tower of sides H which is placed on the surface. The dimensions of the computational domain are $[-5H, 30H] \times [-H, 7H]$ in the streamwise x and crosswise y directions respectively, and the domain is discretized into sufficiently large number of cells to perform a URANS or VMS simulation. The inflow boundary conditions are $u = (V_{in}, 0)$, together with $\tilde{\nu} = 3\nu$, which corresponds to a ratio of eddy to kinematic viscosity of approximately 0.2. For the top of the domain, the velocity component normal to the surface is set to zero. No-slip boundary conditions $u = 0$ and $\tilde{\nu} = 0$ are imposed at the tower surface, as well as the bottom surface at $y = -1$. The time step is $\Delta t = 0.01$ seconds and 300 seconds are simulated. For a statistically steady state to be reached (periodic vortex shedding to be observed), around 100 seconds are required. The data of the remaining 200 seconds (*i.e.* approximately 20×10^3 time steps) is stored for training and testing purposes. The data is sampled at each $\Delta t = 0.1$ seconds, thus collecting around 2000 snapshots. In terms of non-dimensional time



(a) Case 1 setup: Wake-flow past a square cylinder



(b) Case 2 setup: Environmental flow over a surface-mounted tower

defined as $t^* = tU_\infty/H$, this sampling at each $\Delta t = 0.1$ denotes $\Delta t^* = 1$. Figure 5.3b shows the sketch of the associated case. Initially, a free separated shear layer expands above the tower and becomes wavy, and then reattaches at the bottom surface of the domain. The shear layer flaps and vortical structures are shed from it. Approximately 18 shedding cycles are observed in simulation data. Given the 70/30 splitting strategy, 12 shedding cycles are observed in training data enough to reasonably characterize the dynamics of environmental flow over the surface-mounted obstacle.

5.4 Results and Discussion

In this section, the results are discussed as follows: first, the temporal evolutions of the quantities are compared, then the spatial measurements at various times are compared for velocity components. In a second time, temporal propagation of errors and correlation coefficients are compared along with the propagation of phase shifts. Additionally, the contour plots of quantities are also compared to provide qualitative assessments. These comparisons are performed for both the cases and for both the *a priori* and *a posteriori* simulations as illustrated in figure 5.4. To compare results, a first *a priori* simulation is performed by exploiting data samples that were not used during training. The trained model is fed snapshots at instant t, and predicts the next two snapshots at instants $t + \Delta t$ and $t+2\Delta t$, and the process is repeated by feeding the subsequent snapshots from the dataset, until the same number of snapshots is reached for comparison with the original ground truth time series. As snapshots from the dataset are utilized, this approach is termed *a priori* deep learning simulation. On the other hand, *a posteriori* simulation is performed by feeding a snapshot at instant t_0 from the same dataset not used during training, and by predicting the next two snapshots at instants $t + \Delta t$ and $t + 2\Delta t$. This predicted snapshot at instant $t + 2\Delta t$ is then injected back into the model to predict snapshots at instants $t + 3\Delta t$ and $t+4\Delta t$, and the process is similarly repeated until the same number of snapshots are obtained so as to compare with the true snapshots. This way of recycling the model predictions is termed *a posteriori* deep learning simulation. Once an equal length of time snapshots are obtained, both the *a priori* and the *a posteriori* results against the truth from the dataset can now be compared. Figure 5.5 shows the temporal evolution of the ensemble average of velocity magnitude for case 1 and case 2. For case 1, both *a priori* and *a posteriori* predictions present a good agreement with respect to the truth, whereas for case 2 the predictions, though fairly accurate, suffer from deterioration. Moreover, the long-term predictions of the model are evident from the accuracy of a posteriori predictions, giving us an indication of global long-term learning while comparing ensemble averages.

The accuracy of the predictions is further verified by comparing the values along the various streamwise and cross-streamwise locations. These locations are marked with dashed lines in figure 5.6 for both the cases. For case 1, measurements were made along streamwise directions at $x = [-2.5H, 8H, 12H, 16H]$ and

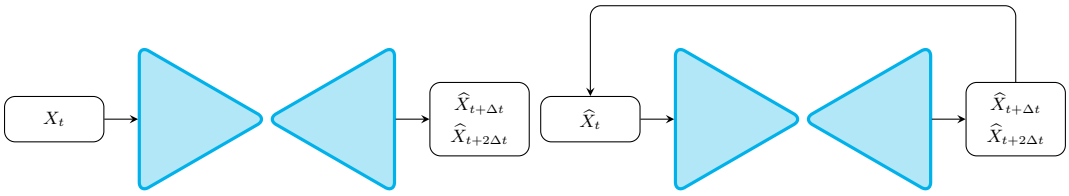
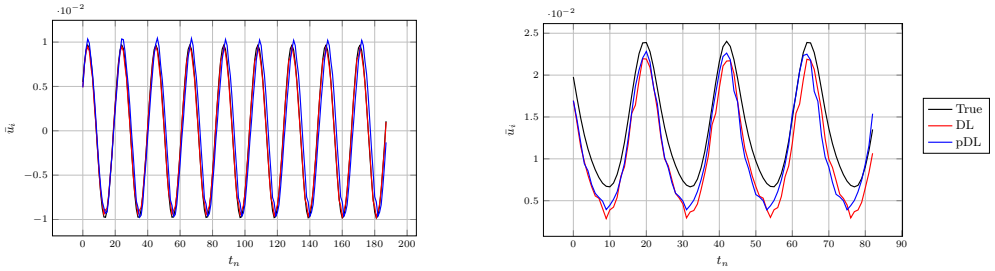


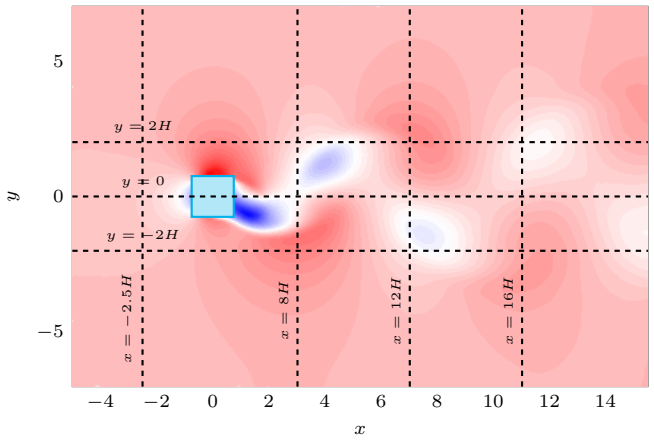
Figure 5.4: **Illustration of *a priori* and *a posteriori* simulations** Left: For *a priori* simulation, each X_t from $\{X\}$, the dataset not used in training time, is fed to the model. Right: For *a posterioi* simulation, the inputs $\widehat{X_t}$ are received from its own previous predictions, provided it was initiated with a suitable X_t.



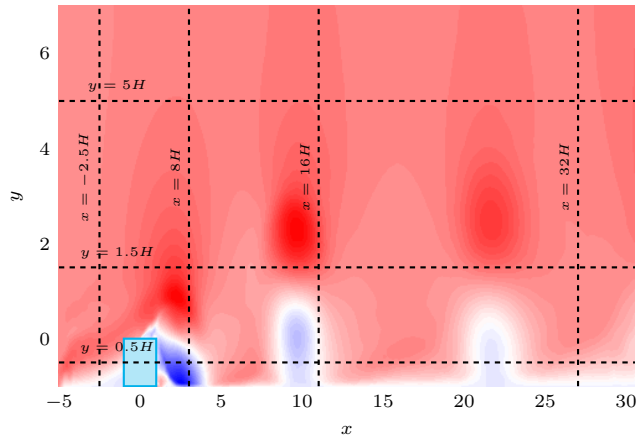
(a) *A priori* and *a posteriori* predictions compared to ground truth for case 1

(b) *A priori* and *a posteriori* predictions compared to ground truth for case 2

Figure 5.5: **Temporal evolution of the ensemble averages** for *a priori* and *a posteriori* values of velocity magnitude compared to the true values in black. Left: Ensemble mean for spatial values of velocity magnitude for case 1. Right: Ensemble mean for spatial values of velocity magnitude for case 2.



(a)



(b)

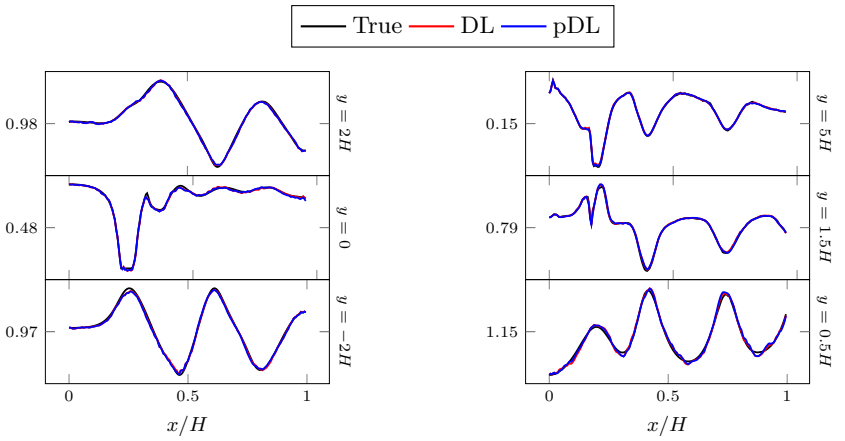
Figure 5.6: **Locations of the probe lines used for comparison to the reference quantities.** Left: Lines along streamsie and cross-streamwise directions for case 1. Right: For case 2.

cross-streamwise directions at $y = -2H, 0, 2H$, and similarly for case 2, the measurements were made at $x = [-2.5H, 8H, 16H, 32H]$ and at $y = [0.5H, 1.5H, 5H]$. As the wake-flows are topic of interest, these locations were chosen based on the region of interest away from the obstacle for both the cases. With regards to temporal evolution, the predictions were compared at a certain percentage of the total predicted snapshots. As a reminder, around 200 snapshots were predicted for case 1 and around 100 snapshots for case 2. The predictions are compared at instants $t = [2\%, 33\%, 66\%]$ to verify the quality of temporal evolution.

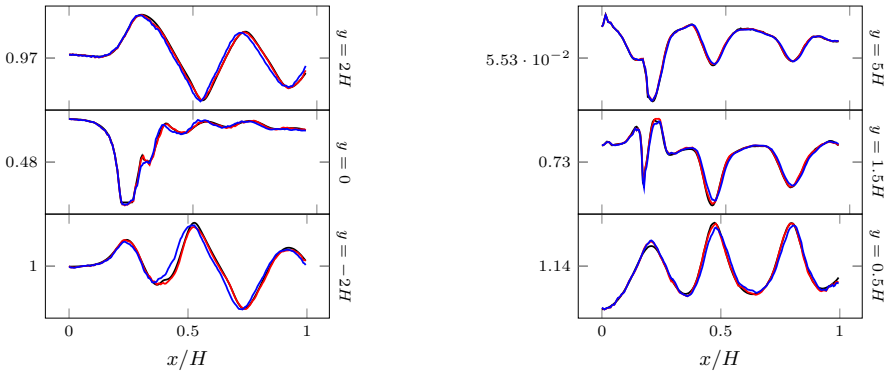
Figure 5.8 shows the evolution of temporal predictions of streamwise velocity component u_0 when measured along with cross-streamwise directions. The *a priori* predictions follow closely the reference values indicating a good agreement with the short-term predictions along with the measured spatial directions. For *a posteriori* predictions, an increasing deviation from the reference was observed as time evolves, which can be attributed to the accumulation error while making long-term predictions. Similarly, the evolution of the same quantity (u_0) when measured along streamwise directions is shown in figure 5.7. A similar trend is observed for the predictions against the reference, where the a posteriori predictions deteriorate as time evolves. It can be noted that the upstream predictions at $x = -2.5H$ are better across times, as it is not affected by the turbulent wake. Overall measurements indicate a decent agreement of both the short-term *a priori* predictions as well as long-term *a posteriori* predictions with the reference solutions.

Later, the temporal evolution of prediction error against reference by computing relative mean-squared errors of velocity magnitude for both cases is investigated. These errors are measured along the locations mentioned earlier. Figure 5.9(a) shows the evolution of error for *a priori* predictions and figure 5.9(b) shows *a posteriori* predictions for case 1 measured at streamwise locations. As could be expected, the errors accumulate for long-term posterior predictions, leading to a clear distinction when compared to *a priori* predictions. It is interesting to note that although magnitude increases over time, this evolution also follows the vortex/wake shedding cycles denoting that the trained model performs well for long-term *a posteriori* predictions. A similar trend is observed for case 2 as shown in figure 5.9(c) and figure 5.9(d), although here the magnitude of accumulated errors is higher than that of case 1. As shown in figure 5.10, a similar trend is observed when measurements of errors were performed at cross-streamwise locations.

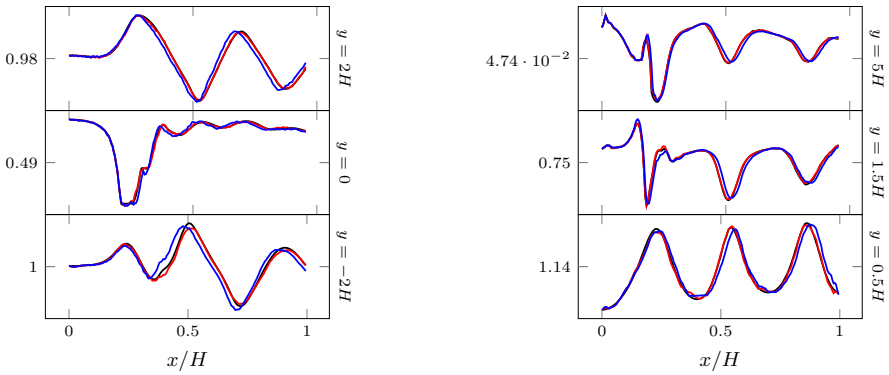
The *a posteriori* predictions are observed to experience high noise conditions caused by error propagation. Hence, extracting the correlation between these two sets (predicted vs. reference) of temporal evolution is important, in particular to assess whether the heavy noise contributions are degrading correlation values. To do so, the Pearson product-moment correlation coefficient R_{xy} of n pairs of time series data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is computed:



(a) Time $t = 0.02\,T_n$

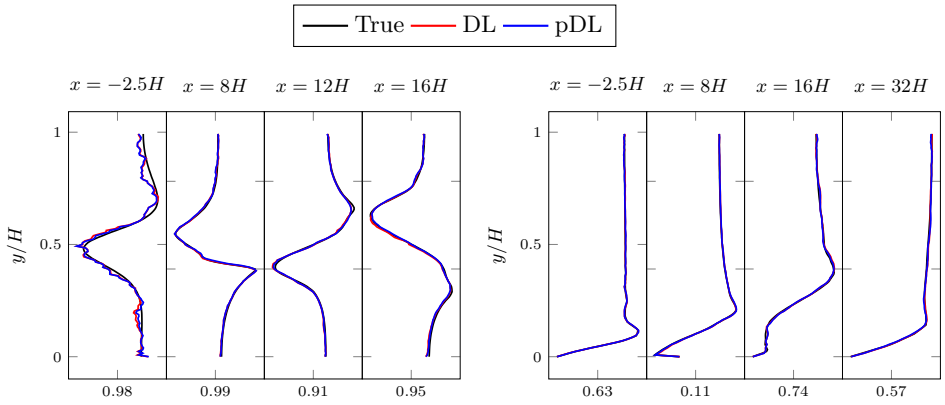


(b) Time $t = 0.33\,T_n$

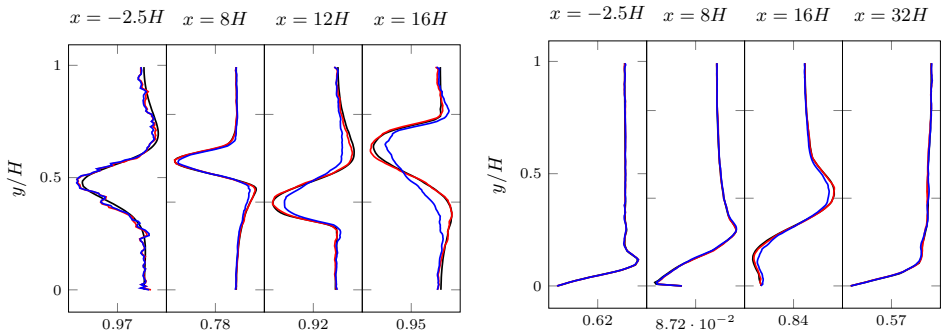


(c) Time $t = 0.66\,T_n$

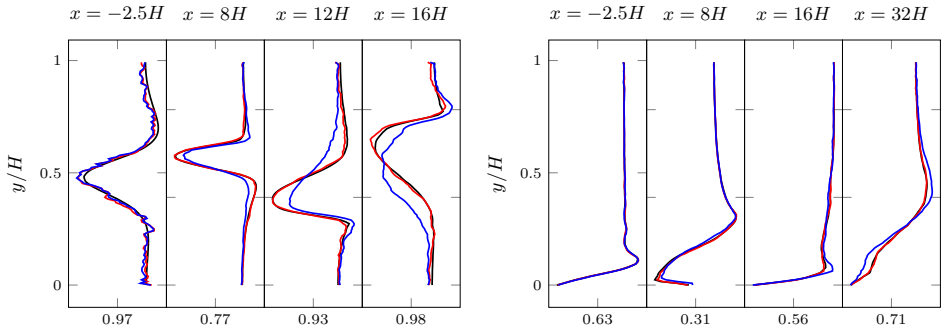
Figure 5.7: **Comparative predictions of streamwise velocity components (u_0) sampled along y-axis for case 1 (left) and case 2 (right).** Figures from top to bottom denote the predictions at increasing times, *i.e.* the top row contains instantaneous preditctions at $t = 0.02\,T_n$, the middle row at $t = 0.33\,T_n$, and the bottom row shows the predictions at $t = 0.66\,T_n$.



(a) Time $t = 0.02\,T_n$



(b) Time $t = 0.33\,T_n$



(c) Time $t = 0.66\,T_n$

Figure 5.8: **Comparative predictions of streamwise velocity components (u_0) for case 1 (left) and case 2 (right).** Figures from top to bottom, denote the predictions at increasing times, *i.e.* the top row contains instantaneous predictions at $t = 0.02\,T_n$, the middle row at $t = 0.33\,T_n$, and the bottom row shows the predictions at $t = 0.66\,T_n$.

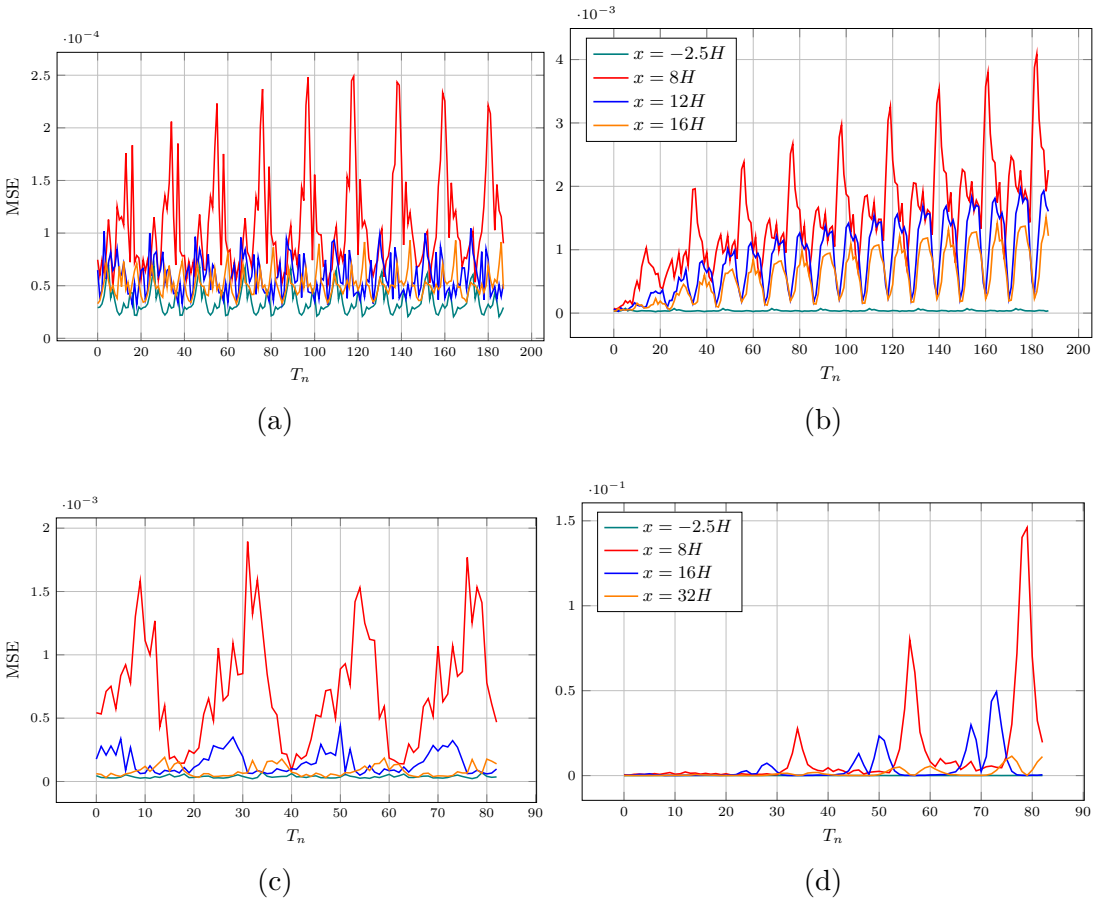


Figure 5.9: **Mean squared error propagation** for velocity magnitude with respect to the reference values. The top row shows the evolution for case 1, and the bottom row shows the evolution for case 2. On the left is the temporal evolution of a priori mean squared error. While on the right are the temporal evolution of a posteriori mean squared error. The values are shown for locations along the X-axis.

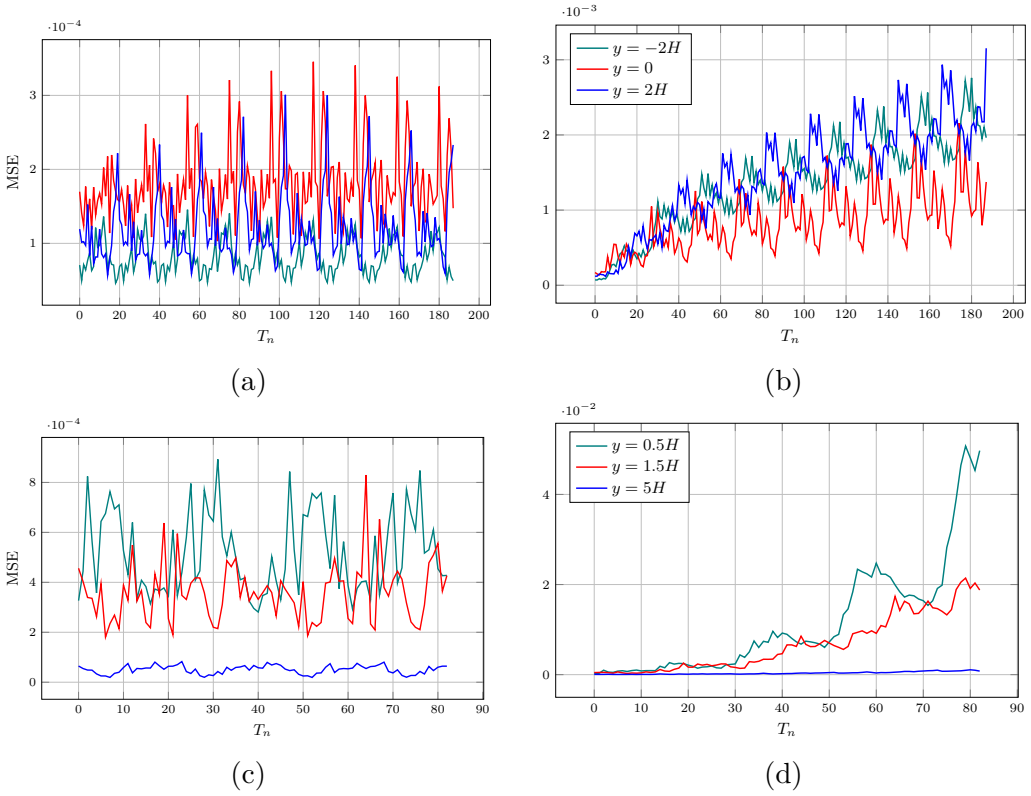


Figure 5.10: **Mean squared error propagation** for velocity magnitude with respect to the reference values. The top row shows the evolution for case 1, and the bottom row shows the evolution for case 2. On the left is the temporal evolution of *a priori* mean squared error, while on the right is the temporal evolution of *a posteriori* mean squared error. The values are shown for locations along the Y-axis.

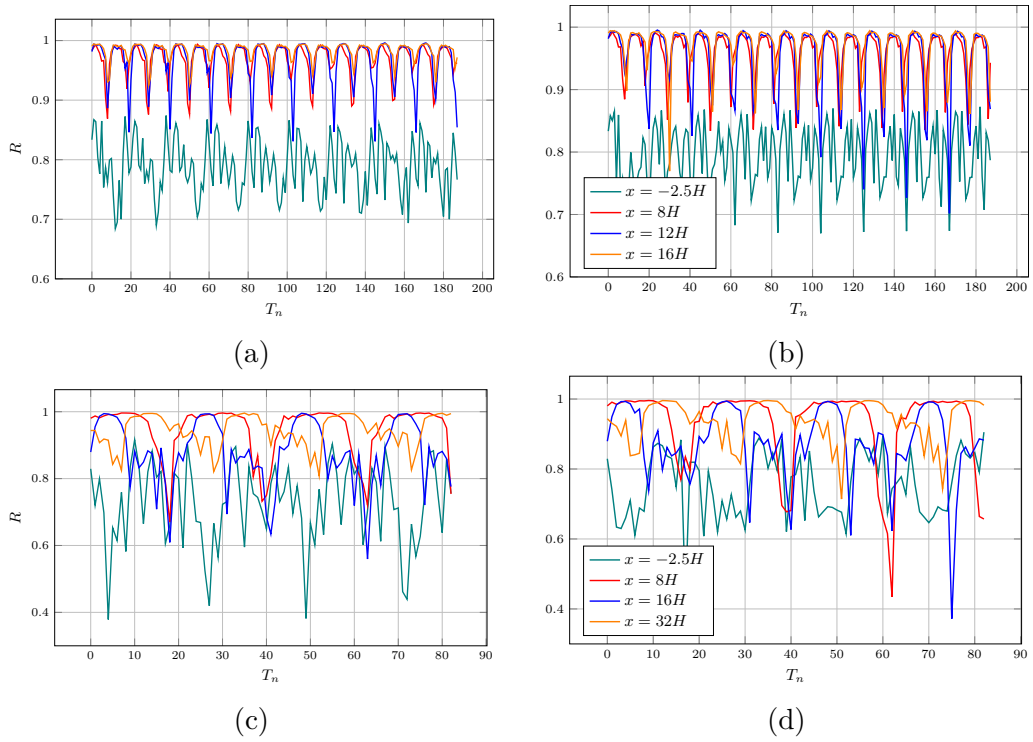


Figure 5.11: **Correlation propagation** for velocity magnitude with respect to the true values. The top row shows the evolution for case 1, and the bottom row shows the evolution for case 2. On the left is the temporal evolution of the Pearson product-moment correlation coefficient for *a priori* values with reference to true values, while on the right are the R values for *a posteriori* values with reference to true values. The values are shown for locations along the X-axis.

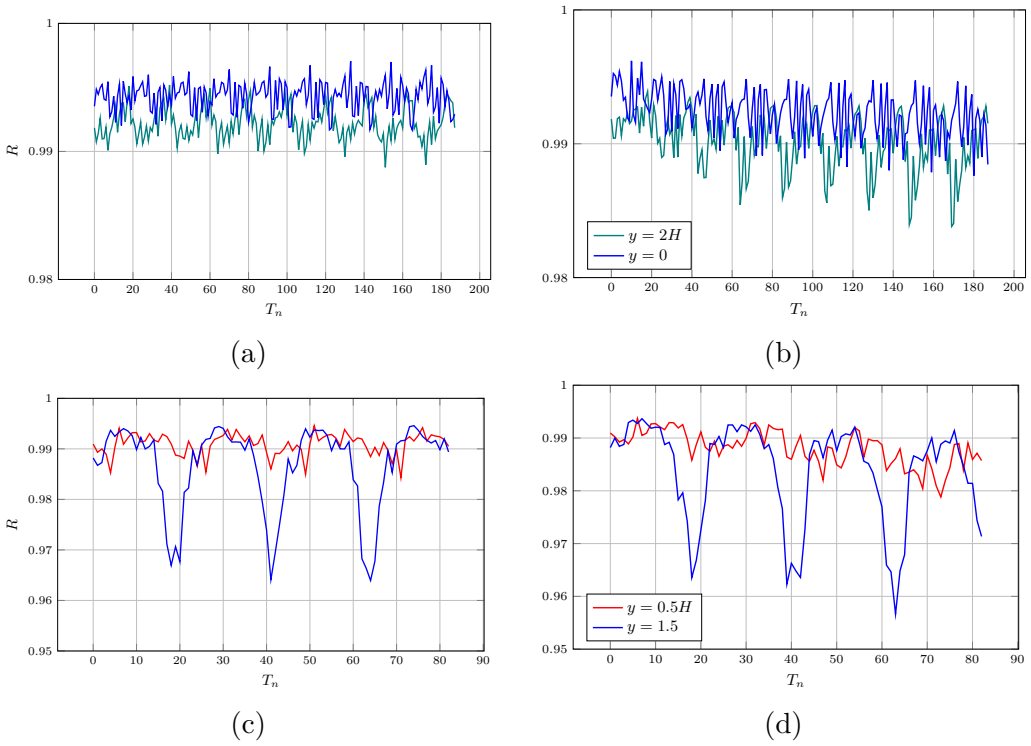


Figure 5.12: **Correlation propagation** for velocity magnitude with respect to the true values. The top row shows the evolution for case 1, and the bottom row shows the evolution for case 2. On the left is the temporal evolution of the Pearson product-moment correlation coefficient for *a priori* values with reference to true values, while on the right are the R values for *a posteriori* values with reference to true values. The values are shown for locations along the Y-axis.

$$R_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{5.7}$$

where n is sample size, x_i, y_i are the individual sample points indexed with i, and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the sample mean (analogously for $\bar{y}$). In simple terms, R_{xy} is the covariance of the two variables divided by the product of their standard deviations. For our measurements, the two variables are simply the predicted and reference snapshots at the same instants, and computation is performed for both the *a priori* and *a posteriori* predictions. Figure 5.11(b) and 5.11(d) show a gradual decrease in the correlation coefficient for the *a posteriori* predictions for case 1 and case 2 respectively. A steeper degradation of correlation is observed in the measurements at cross-streamwise locations as shown in figures 5.12(b) and 5.12(d) for both cases, while that of the *a posteriori* predictions remains stable. Since a clear trend is observed in degrading correlations for *a posteriori* predictions, the phase-shift $\varphi(t)$ were measured for the temporal evolution of velocity magnitude predictions against the reference. Measurements were done along the similar spatial directions as mentioned before, the results of which are shown in figure 5.13. The value $\varphi(t) < 0$ denotes that predictions are shifted by that value before the reference, and the $\varphi(t) > 0$ denotes predictions shifted after the reference. For case 1, it is interesting to note that the phase shift goes on increasing in magnitude as time evolves, indicating the model's stability for long-term predictions. However, any clear trend for case 2 when measured at streamwise locations was not observed.

For a qualitative assessment of results, the contours of velocity components for both cases is compared. Figure 5.14 shows the instantaneous snapshots of streamwise velocity contours for case 1 at $t = [2\%, 33\%, 66\%]$ of total predicted snapshots as mentioned earlier, and similar instantaneous snapshots for case 2 are shown in figure 5.15. For both the cases, the *a priori*, as well as *a posteriori* predictions, show a fairly accurate agreement with the reference.

5.5 Conclusions

A convolutional encoder-decoder-based transformer model has been developed to auto-regressively train on spatio-temporal data of turbulent flows. The method of auto-regressive training works by predicting future fluid flow fields from the previously predicted fluid flow field to ensure long-term predictions without diverging. The model has been validated by demonstrating its applicability to turbulent wake flow past an obstacle and environmental flow past surface mounted obstacle. The work demonstrates a promising model and method for forecasting fluid flow fields where the training data is available. The proposed model trained in an autoregressive way shows significant agreements for a priori evaluations, whereas the posterior predictions show expected deviations after a considerable number of simulation steps. The spatio-temporal complexity of predictions is compara-

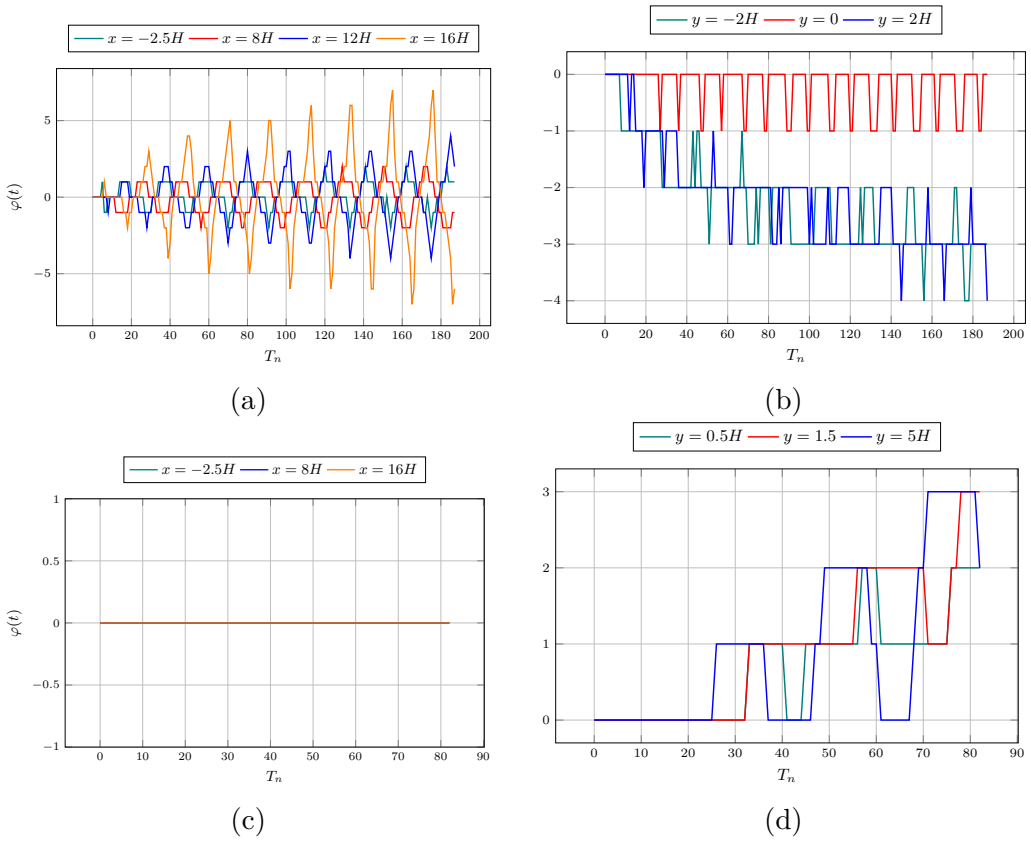


Figure 5.13: **Phase-shift evolution** for *a posteriori* values of velocity magnitude with respect to the reference values. The top row shows the evolution for case 1, and the bottom row shows the evolution for case 2. On the left are temporal evolutions measured along with streamwise locations. While on the right are the evolutions measured along with cross-streamwise locations.

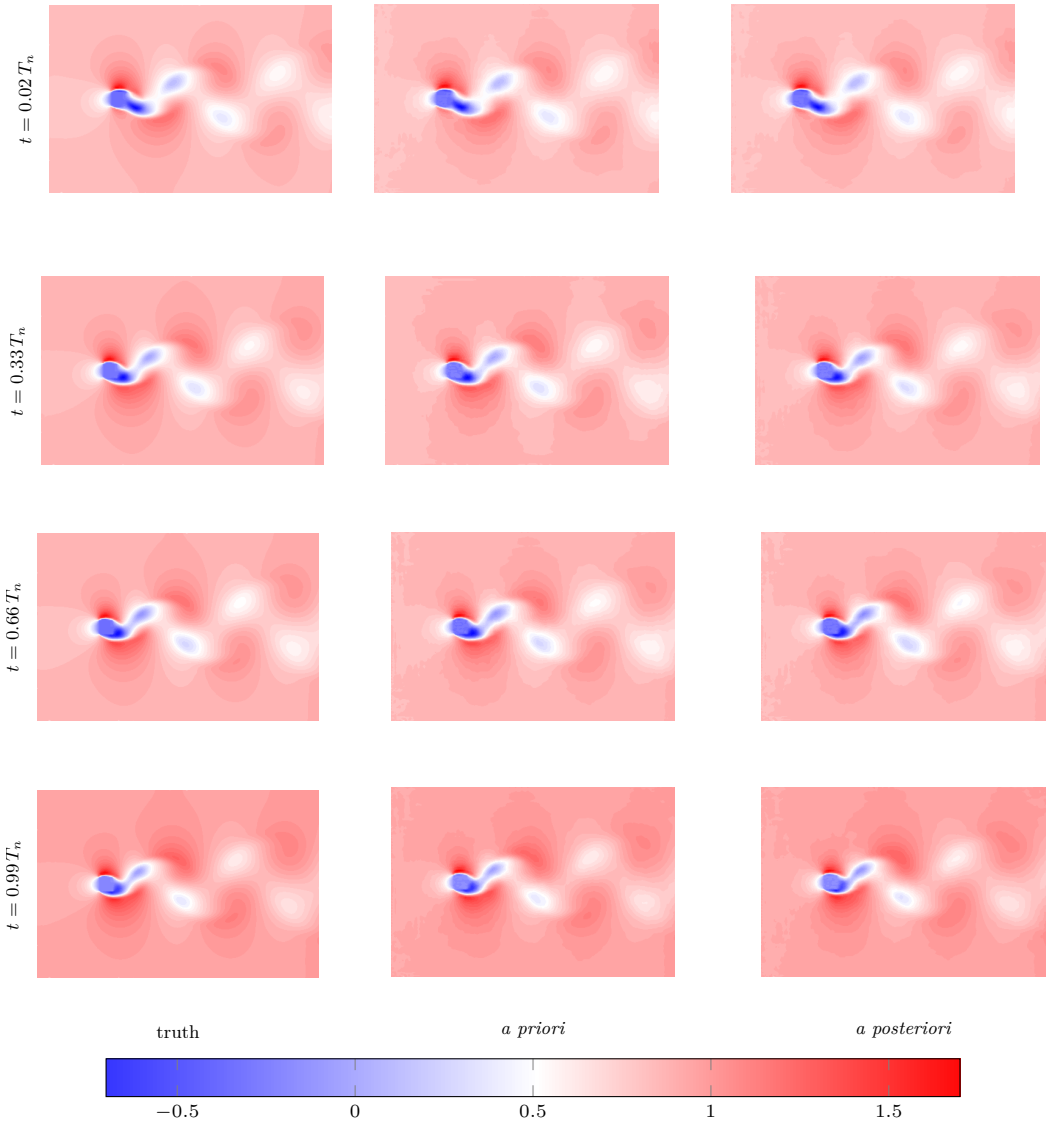


Figure 5.14: **Comparison of *a priori* and *a posteriori* prediction streamwise velocity contours** against the reference showing the temporal evolution for case 1.

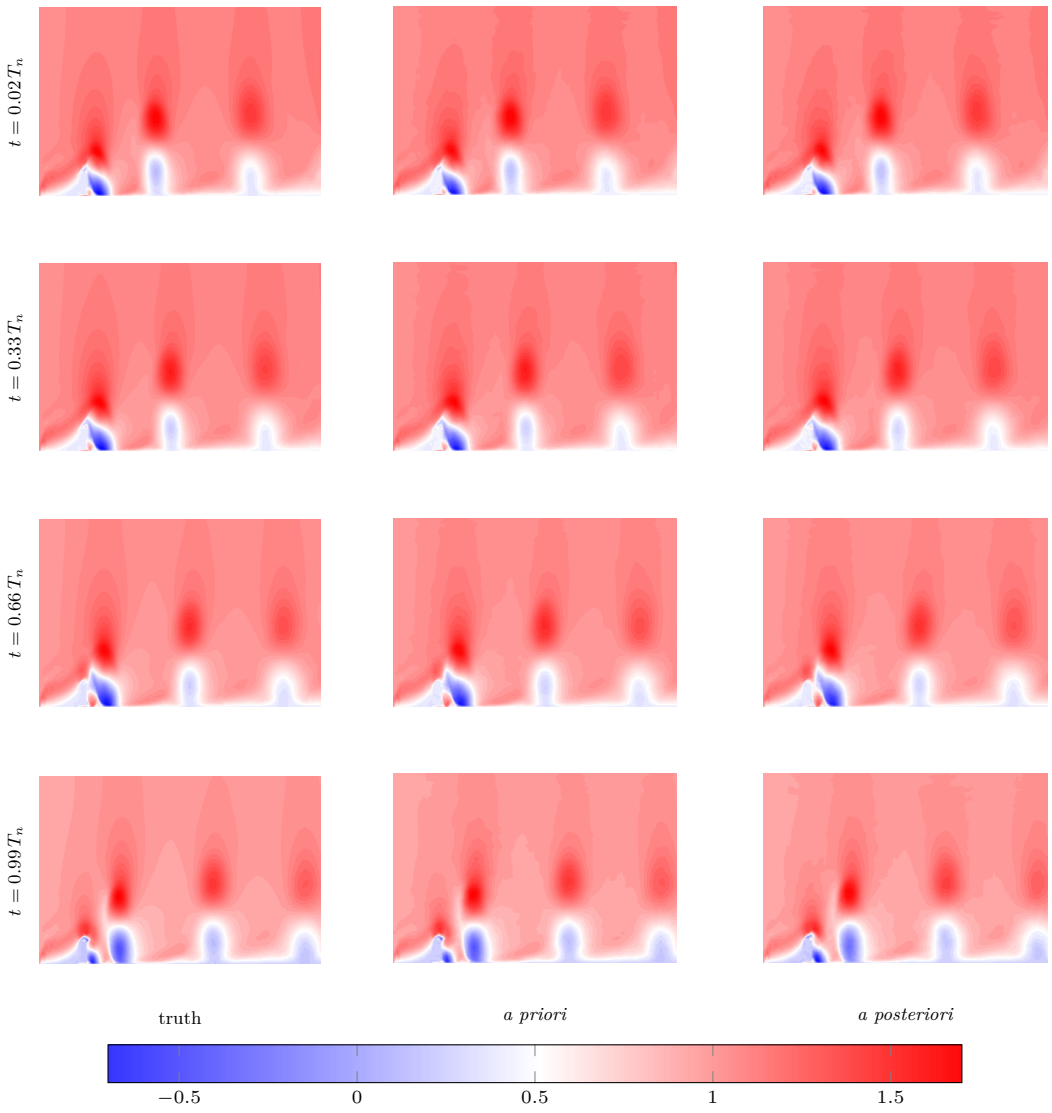


Figure 5.15: **Comparison of *a priori* and *a posteriori* prediction streamwise velocity contours** against the truth reference showing the temporal evolution for case 2.

ble to the target simulations of fully developed turbulence. The autoregressive training and prediction of *a posteriori* states is the primary step towards the development of more complex data-driven turbulence models and simulations. It is shown that the self-attention transformers incorporated within the convolutional encoder-decoder can predict up to $200\Delta t$ time steps with relatively high accuracy, and the proposed data-driven deep learning model remains stable for multiple long time scales, promising a stable and physical deep learning predictive turbulence modeling candidate. Changes to loss function can be done to achieve even longer, stable, physically realistic results. Additional experiments are needed to demonstrate the model's ability on generalizing to local mesh regions as well as longer *a posteriori* simulation steps. Further investigations on a variety of industrial and academic cases could include training for flow Reynolds numbers, turbulence intensity, and other inlet parameters. Conclusions from this work would also provide valuable insights for the development of new deep learning methods and their deployment for turbulent flows on complex geometries in industrial problems. Deploying a trained model to assist a fluid solver is regarded as a future extension of the present work.

Bibliography

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Y Bazilevs, VM Calo, JA Cottrell, TJR Hughes, A Reali, and G23614751169 Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer methods in applied mechanics and engineering*, 197(1-4):173–201, 2007.

Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding. *arXiv preprint arXiv:2102.05095*, 2(3): 4, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Steven L Brunton, Bingni W Brunton, Joshua L Proctor, and J Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PloS one*, 11(2):e0150171, 2016.

Chen Cheng and Guang-Tao Zhang. Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems. *Water*, 13(4):423, 2021.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Pierre Dubois, Thomas Gomez, Laurent Planckaert, and Laurent Perret. Data-driven predictions of the lorenz system. *Physica D: Nonlinear Phenomena*, 408: 132495, 2020.

Pierre Dubois, Thomas Gomez, Laurent Planckaert, and Laurent Perret. Machine learning for fluid flow reconstruction from limited measurements. *Journal of Computational Physics*, 448:110733, 2022.

G Guiza, A Larcher, A Goetz, L Billon, P Meliga, and Elie Hachem. Anisotropic boundary layer mesh generation for reliable 3d unsteady rans simulations. *Finite Elements in Analysis and Design*, 170:103345, 2020.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Elie Hachem, Stephanie Feghali, Ramon Codina, and Thierry Coupez. Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation. *International journal for numerical methods in engineering*, 94(9):805–825, 2013.

Renkun Han, Yixing Wang, Yang Zhang, and Gang Chen. A novel spatial-temporal prediction method for unsteady wake flows based on hybrid deep neural network. *Physics of Fluids*, 31(12):127101, 2019.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1): 1–10, 2018.

Vincent Mons, J-C Chassaing, Thomas Gomez, and Pierre Sagaut. Reconstruction of unsteady viscous flows using data assimilation schemes. *Journal of Computational Physics*, 316:255–280, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, pages 285–319, 2010.

Taichi Nakamura, Kai Fukami, Kazuto Hasegawa, Yusuke Nabae, and Koji Fukagata. Convolutional neural network and long short-term memory based reduced order surrogate for minimal turbulent channel flow. *Physics of Fluids*, 33(2): 025116, 2021.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

Wenhui Peng, Zelong Yuan, and Jianchun Wang. Attention-enhanced neural network models for turbulence simulation. *Physics of Fluids*, 34(2):025111, 2022.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Jiahao Ren, Haiou Wang, Guo Chen, Kun Luo, and Jianren Fan. Predictive models for flame evolution using machine learning: A priori assessment in turbulent flames without and with mean shear. *Physics of Fluids*, 33(5):055113, 2021.

Clarence W Rowley and Scott TM Dawson. Model reduction for flow analysis and control. *Annual Review of Fluid Mechanics*, 49:387–417, 2017.

David Schmidt, Romit Maulik, and Konstantinos Lyras. Machine learning accelerated turbulence modeling of transient flashing jets. *Physics of Fluids*, 33 (12):127104, 2021.

Gilad Sharir, Asaf Noy, and Lihi Zelnik-Manor. An image is worth 16x16 words, what is a video worth? *arXiv preprint arXiv:2103.13915*, 2021.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375: 1339–1364, 2018.

Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th aerospace sciences meeting and exhibit*, page 439, 1992.

Yifan Sun, Linan Zhang, and Hayden Schaeffer. Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data. In *Mathematical and Scientific Machine Learning*, pages 352–372. PMLR, 2020.

Kenji Takizawa, Tayfun E Tezduyar, and Yuto Otoguro. Stabilization and discontinuity-capturing parameters for space–time flow computations with finite element and isogeometric discretizations. *Computational Mechanics*, 62 (5):1169–1186, 2018.

HS Tang, L Li, M Grossberg, YJ Liu, YM Jia, SS Li, and WB Dong. An exploratory study on machine learning to couple numerical solutions of partial differential equations. *Communications in Nonlinear Science and Numerical Simulation*, 97:105729, 2021.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In

Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations, pages 38–45, 2020.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–31, 2021a.

Pin Wu, Siquan Gong, Kaikai Pan, Feng Qiu, Weibing Feng, and Christopher Pain. Reduced order model using convolutional auto-encoder with self-attention. *Physics of Fluids*, 33(7):077107, 2021b.

Jiayang Xu and Karthik Duraisamy. Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics. *Computer Methods in Applied Mechanics and Engineering*, 372:113379, 2020.

Mustafa Z Yousif, Linqi Yu, and Hee-Chang Lim. High-fidelity reconstruction of turbulent flow from spatially limited data using enhanced super-resolution generative adversarial network. *Physics of Fluids*, 33(12):125119, 2021.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Chapter 6

Conclusion and Perspectives

The use of deep learning techniques in the field of fluid mechanics, and in particular for the study of turbulent flows, is a rapidly growing research area. This thesis aimed to explore various potential approaches to deep learning for the investigation, modeling, and prediction of turbulent flows. By adopting recent literature from the computer vision community, novel techniques were developed to address the complex physics of fluids, with a particular focus on turbulent flows.

The problem and motivation for this thesis are addressed in Chapter 1, where a detailed introduction to the topic is provided. Chapter 2 builds upon this by providing a thorough theoretical background in both deep learning and the theory of turbulence in fluids. The subsequent chapters of the thesis are structured in a way that allows for a natural progression of ideas. First, a method for learning a turbulence model is presented, which is followed by subgrid-scale reconstruction of fully turbulent flow. Finally, the thesis concludes with a discussion of learning turbulent flow in both space and time, which provides a comprehensive overview of the intersection between turbulence in fluids and spatio-temporal deep learning.

Chapter 3 of this thesis focuses on the application of deep learning techniques to learn turbulence modeling. A patch-based method is proposed to enhance the robustness of the learning process, which is demonstrated on physical cases of flow past obstacles at various high Reynolds numbers. Specifically, an encoder-decoder full-convolutional deep learning architecture is proposed, which is trained using a patch-based strategy. The architecture is trained to learn the Spalart-Allmaras turbulence model without any hand-picking of input features, and the effectiveness of the proposed method is validated through various experiments. The results demonstrate the potential of deep learning for improving turbulence modeling and prediction, and highlight the importance of adopting appropriate learning strategies to achieve robust and reliable results.

In Chapter 4, the focus is on the learning of subgrid-scale turbulence from the resolved large scales. To achieve this, a successive refinement training is performed using an encoder-decoder type convolutional neural network. The goal is to investigate the relationship between the resolved and subgrid scales of turbulence, given training samples at various coarse scales. To demonstrate the applicability of this method, DNS data of plane fully-developed turbulent channel flow at a very high Reynolds number is used to set up different learning experiments. These data provide access to all scales of turbulence, which are used as a reference to evaluate different approaches. Low-resolution fields of large scales are obtained from the reference ones, and the reconstructions are compared to these. The impact of coarse-graining methods, refinement directions, as well as the effect of successive refinement, is investigated in this chapter.

Chapter 5 of the thesis focuses on spatio-temporal learning of turbulent flows. To address this, a transformer-based convolutional model is proposed, which is capable of learning the temporal snapshots of turbulent flows using an auto-regressive method. The effectiveness of this approach is demonstrated using two physical cases: turbulent flows past an obstacle in a free stream and turbulent

flows past a surface-mounted obstacle. By leveraging the autoregressive model, the approach is able to perform posteriori simulations and thereby demonstrate its efficacy in learning turbulent flows in time. Overall, Chapter 5 makes a valuable contribution to the field of turbulence and deep learning by introducing a novel technique for spatio-temporal learning of turbulent flows.

Perspectives

Many perspectives emerge from this thesis work. The following are listed as the important ones:

The global deep learning methods presented in this thesis have shown promising results, but there is still room for improvement. One potential way to improve these methods is by considering local learning with refinements in areas where the flow is complex. The patch-based method proposed earlier in this thesis can be deployed to address this issue and improve local predictions. By combining global and local learning approaches, we can achieve a more accurate and comprehensive understanding of turbulent flows.

While the deep learning models developed in this thesis are deterministic, future work could explore the potential benefits of using probabilistic models. Specifically, it may be beneficial to learn probabilistic models based on finite approximation operators that can advance the probability densities of the state being learned in both space and time. This approach may lead to a more accurate and comprehensive understanding of turbulent flows. While the models presented in this thesis were learned for fixed parameters, it would be more practical to learn models that span a range of parameters. To achieve this, generative approaches can be used to learn models that are robust to variations in Reynolds number and other flow parameters. This would improve the applicability of the learned models in practical engineering problems.

The subgrid-scale enhancements presented in this thesis were based on homogeneous coarse-grained measurements. However, exploring reconstructions with scales that are not homogeneous to the field being reconstructed, such as randomly sparse sampled measurements, could lead to improved results. Additionally, the presented models currently only use full input field measurements, but they could be extended to incorporate patches of observations.

To make the learning process physically consistent, physical laws could be incorporated into the models. One way to achieve this is by regularizing the loss function to ensure that the input-output relation learned by the model aligns with the Navier-Stokes equations. Another approach is to use Physics Informed Neural Networks, which constrain the residuals of the governing equations of interest to achieve the same goal. In this thesis, the main focus was on exploring the intersection of turbulent flows and deep learning methods, and numerical treatment was not the major concern. However, it is important to consider the numerical issues and their effect on the learning process in future work. This could involve investigating the impact of numerical schemes, discretization methods, and boundary conditions on the accuracy of the learned models, and developing new strategies to improve the numerical stability and efficiency of the learning

algorithms.

The use of deep learning models can result in a loss of interpretability due to the complexity of the network's structure and lack of a simple analytical expression, particularly in models with many hidden layers. One way to address the interpretability issue of deep learning models is to refine them using symbolic regression. By combining Graph Neural Networks with binary trees evolved by genetic algorithms, the interpretability of the models could be improved. This approach could be extended to turbulent flow cases, where the aim is to learn a physically interpretable model.

The current estimation models used in this thesis only reflect the final stage of learning, as each new model is trained from scratch without utilizing the learning history of previous models. To improve the learning efficiency for new configurations or Reynolds numbers, transfer learning could be employed to transfer knowledge from previous learning. This technique could have practical applications in learning turbulent flows.

RÉSUMÉ

Malgré plusieurs avancées dans les ressources expérimentales et informatiques, et malgré les progrès des procédures théoriques et mathématiques pour aborder la fermeture des équations de Navier-Stokes, la turbulence reste un problème non résolu même après 200 ans de recherche continue. D'autre part, l'intelligence artificielle et les technologies connexes font des progrès rapides dans plusieurs domaines de la science et de l'ingénierie, nous aidant à résoudre efficacement les problèmes de modélisation et à découvrir de nouveaux phénomènes physiques. Le présent travail tente de combiner ces deux branches et d'explorer si les machines computationnelles peuvent être utilisées pour étudier efficacement la turbulence dans les fluides, et peut-être un jour nous aider dans la découverte des lois universelles manquantes. L'apprentissage profond est utilisé pour apprendre la modélisation de la turbulence et une méthode basée sur les patchs est proposée pour un apprentissage robuste. L'apprentissage de la turbulence à l'échelle de la sous-grille à partir des grandes échelles résolues est démontré, de même que l'étude de l'effet des méthodes de raffinage grossier et des raffinements successifs. L'apprentissage spatio-temporel des flux turbulents est proposé pour apprendre les instantanés temporels et des simulations a posteriori sont effectuées.

MOTS CLÉS

Turbulence, Modélisation, Apprentissage

ABSTRACT

Despite several advancements in experimental and computational resources, and despite progress in theoretical and mathematical procedures to address the closure of Navier-Stokes equations, turbulence remains an unsolved problem even after 200 years of continuous research. On the other hand, artificial machine intelligence and related technologies are making rapid advancements in several domains of science and engineering, helping us humans to efficiently solve modeling problems and discover new physics. Present work tries to combine these two branches and explore if computational machines can be used to efficiently study turbulence in fluids, and perhaps someday help us in the discovery of the missing universal laws. Deep learning is employed to learn turbulence modeling and a patch-based method is proposed for robust learning. Learning of subgrid-scale turbulence from the resolved large scales is demonstrated along with investigation of effect of the coarse-graining methods and successive refinements. Spatio-temporal learning of turbulent flows is proposed to learn the temporal snapshots and a posteriori simulations are performed.

KEYWORDS

Turbulence, Modelling, Deep Learning