Applied Deep Learning for the Prediction of Fluid Flows and Multiphysics Modeling

George El Haber

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES Paris

Applied Deep Learning for the Prediction of Fluid Flows and Multiphysics Modeling

Application de l'Apprentissage Profond à la Prédiction et à la Modélisation des Écoulements de Fluides Multiphysiques

Soutenue par
George EL HABER
Le 11 décembre 2023

Dirigée par
Elie HACHEM

École doctorale n°364
Sciences Fondamentales et Appliquées

Spécialité
Mathématiques Numériques, Calcul Intensif et Données

Composition du jury :

Gianluigi ROZZA Professeur, SISSA	*Président*
Chady GHNATIOS Professeur, Arts et Métiers	*Rapporteur*
Vincent MOUREAU Directeur de recherche, CORIA	*Rapporteur*
Aurélien LARCHER Chargé de recherche, Mines Paris	*Examinateur*
David RYCKELYNCK Professeur, Mines Paris	*Co-directeur de thèse*
Elie HACHEM Professeur, Mines Paris	*Directeur de thèse*

MINES PARIS | PSL

To my father,

Acknowledgements

While the list of individuals who I have to acknowledge may be extensive, it is essential for me to express my gratitude to those who have contributed to the success of this work, whether academically or morally.

Starting with my thesis director and supervisor, Prof. Elie Hachem, your support, trust, and guidance have been invaluable throughout this journey. I am grateful for the responsibility you entrusted to me, the insights you offered, the discussions we had, and the criticism that has continually refined the quality of my work.

I would also like to extend my gratitude to Prof. David Ryckelynck, Dr. Jonathan Viquerat, and Dr. Aurélien Larcher for their insightful comments. Additionally, I extend my thanks to every professor and member of the CEMEF team, specially to those in the CFL group, as well as to my colleagues, friends, and all who contributed to making these three years a truly enriching and enjoyable experience.

Not only for your support throughout these three years but also for your presence during every obstacle I've ever encountered and for shaping me into the person I am today, I want to express my deepest thanks and gratitude to my family— my father, my mother, and my two sisters. Also, I want to thank my friends, those whom I consider more like family than friends for always being there for me.

Finally, I want to thank Cynthia, the person who has shared in all the ups and downs with me throughout these past years, and who has been present in every single detail of this journey. Thank you for turning bad days into good ones and good days into even better ones!

Thank you all,
George

Publications

Research Articles

- A. Patil, J. Viquerat, A. Larcher, G. El Haber, E. Hachem; Robust deep learning for emulating turbulent viscosities. Physics of Fluids 1 October 2021; 33 (10): 105118. https://doi.org/10.1063/5.0064458

- G. El Haber, J. Viquerat, A. Larcher, D. Ryckelynck, J. Alves, A. Patil, E. Hachem; Deep learning model to assist multiphysics conjugate problems. Physics of Fluids 1 January 2022; 34 (1): 015131. https://doi.org/10.1063/5.0077723

- G. El Haber, J. Viquerat, A. Larcher, J. Alves, F. Costes, E. Perchat, E. Hachem; Deep learning model for two-fluid flows. Physics of Fluids 1 February 2023; 35 (2): 022103. https://doi.org/10.1063/5.0134421.

- G. El Haber *et al.*, Tackling the curse of dimensionality in Deep learning Models. *preprint to be submitted in the end of 2023.*

Oral Presentations

- *Coupling of DL with Computational Mechanics*, BigSims Seminar, France, January 04, 2022.

- *Incorporating Deep Learning Models for Reducing the Computation Burden of CFD Simulations*, Colloque doctorants EDSFA, France, May 16, 2022.

- *Deep Learning Model Operating on Graph Structured Data for Assisting Multiphase Flows*, ECCOMAS Congress, Oslo, Norway, June 06, 2022.

- *Assisting Multiphysics CFD Simulations by Incorporating Deep Learning Models*, IA pour les sciences de l'ingénierie, CNRS Workshop, France, June 28, 2022.

- *Exploiting deep graphical models for multiphase flow problems*, IUTAM symposium on data-driven mechanics and surrogate modeling, Paris, France, October 26, 2022.
- *Graph Neural Network for Two-Fluid Flows*, IACM Computational Fluids Conference, Cannes, France, April 26, 2023.
- *Graph Neural Network for Two-Fluid Flow Problems with Advanced Implementation Methods*, U.S. National Congress on Computational Mechanics, Albuquerque, United States, July 24, 2023.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Abstract *The significance of fluid flows in the operation of diverse systems has been recognized since ancient times. The pursuit of a deeper comprehension of these flows, their characteristics and interactions with the environment, has resulted in a rich history marked by diverse contributions. These contributions have ultimately led to the development of today's Computational Fluid Dynamics (CFD) tools, which play an indispensable role in simulating fluid flows across various industries. Despite their current extensive abilities, several challenges still impede the ability of computational fluid dynamics to meet the continuously escalating needs of present industries. In an attempt to reduce computational costs while achieving greater accuracy, the CFD community is continuously investigating new methodologies to enhance its current standing. The success of deep learning models and data-based approaches in various domains has positioned them as primary candidates for aiding in resolving the current challenges of the CFD community. The convergence of these two distinct disciplines, each with a rich history of contributions and achievements, ignites scientific curiosity for exploring innovative solutions beyond the current paradigm. Thus, this chapter is dedicated to providing a brief history of both disciplines, highlighting present challenges encountered by current CFD tools, existing interdisciplinary and intradisciplinary approaches for tackling them, and finally, presenting the scope of interest of this thesis along with the suggested outline.*

Abstract *L'importance des écoulements de fluides dans le fonctionnement de divers systèmes est reconnue depuis l'antiquité. La quête d'une compréhension plus profonde de ces écoulements, de leurs caractéristiques et de leurs interactions avec l'environnement, a abouti à une histoire riche marquée par des contributions diverses. Ces contributions ont finalement conduit au développement des outils actuels de la Dynamique des Fluides Numérique (CFD), qui jouent un rôle indispensable dans la simulation des écoulements de fluides dans diverses industries. Malgré leurs capacités actuelles étendues, plusieurs défis entravent toujours la capacité de la dy-*

namique des fluides numérique à répondre aux besoins continuellement croissants des industries actuelles. Dans le but de réduire les coûts de calcul tout en garantissant une plus grande précision, la communauté de la CFD cherche continuellement de nouvelles méthodologies pour améliorer sa situation actuelle. Le succès des modèles d'apprentissage en profondeur et des approches basées sur les données dans divers domaines les positionne comme des candidats principaux pour aider à résoudre les défis actuels de la communauté de la CFD. La convergence de ces deux disciplines distinctes, chacune avec une riche histoire de contributions et de réalisations, suscite la curiosité scientifique pour explorer des solutions innovantes au-delà du paradigme actuel. Ainsi, ce chapitre est consacré à fournir un bref historique des deux disciplines, en soulignant les défis actuels rencontrés par les outils actuels de la CFD, les approches interdisciplinaires et intradisciplinaires existantes pour les aborder, et enfin, en présentant le champ d'intérêt de cette thèse ainsi que le plan suggéré.

Contents

1.1 Evolution of Fluid Understanding

From ancient times, the importance of fluids for the functioning of various systems was noticed. Fluid flows and their interaction with their surroundings is the focus point in many phenomena. For instance, ocean currents, atmosphere weather, water in rivers, melting of metals, blood in the human body, lift and drag forces in aeronautics and automotive, and irrigation systems are just a few examples that highlight the role of fluids in our everyday life. The journey of understanding these fluids, to endeavor their powers and optimize various systems, stems back to ancient Romans, with Archimedes initiating the field of hydrostatics to understand waterworks, such as canals and bathhouses [1]. A few centuries later, holding the same objective, Leonardo Da Vinci observed in his famous nine parts collection [2], "The Motion and Measurement of Water," plenty of natural phenomena and thoroughly depicted them, while Isaac Newton contributed to quantifying and predicting the fluid flow phenomena through his famous order laws and principles [3].

The impulse to attain a more profound comprehension and unravel the complexities of these physical phenomenon drove the search for mathematical models to ease their understanding. The modeling of fluid flow problems began with the efforts of Daniel Bernoulli, who harnessed the principle of energy conservation and employed it for fluid motion [4], resulting in the well-known Bernoulli equation. Leonhard Euler focused on conservation laws and later introduced the currently known Euler's Equations that depict a more general form of the latter model, allowing for extending its spectrum of applicability [5]. Finally, expanding upon prior research and encompassing the effect of viscous transport on the latter equations led both the Frenchman Claude Louis Marie Henry Navier and the Irishman George Gabriel Stokes to the formulation of the currently most vital system of equations for understanding the behavior of fluids, the famous Navier-Stokes equations [6].

Although the formulation of the Navier-Stokes equations held a significant intrinsic value in the theoretical and physical insights they provided regarding fluid flows, their complexity, and non-linearity restricted their analytical solution to only simplified problems for specific situations, with no general proof achieved for the existence and smoothness of solutions for all possible fluid flows [7]. This initially distilled choices of further understanding flows to very few and rendered experiments as the primary approach for studying fluid dynamics. Although notable results were found, especially by Osborne Reynolds's experiments in turbulent flows [8], the need for a more convenient, inexpensive, and reproducible approach remained an urge. The emergence of numerical methods provided a breakthrough perspective for solving the complex system of differential equations, with initial attempts for numerical solutions of flow around a circular cylinder by both Thom and Kawaguti, converging to the same solution [9, 10]. However, the vast time scale required for acquiring the desired solution kept these methods infeasible until the development of a new set of

numerical methods by Los Alamos National Laboratory, [11], and novel computational algorithms by the Imperial College of London, [11], which paved the way for Computational Fluid Dynamics as the go-to choice for studying fluid flow problems. Figure 1.1 summarizes the key contributions in understanding fluid flows until their spread as main tools in the market.

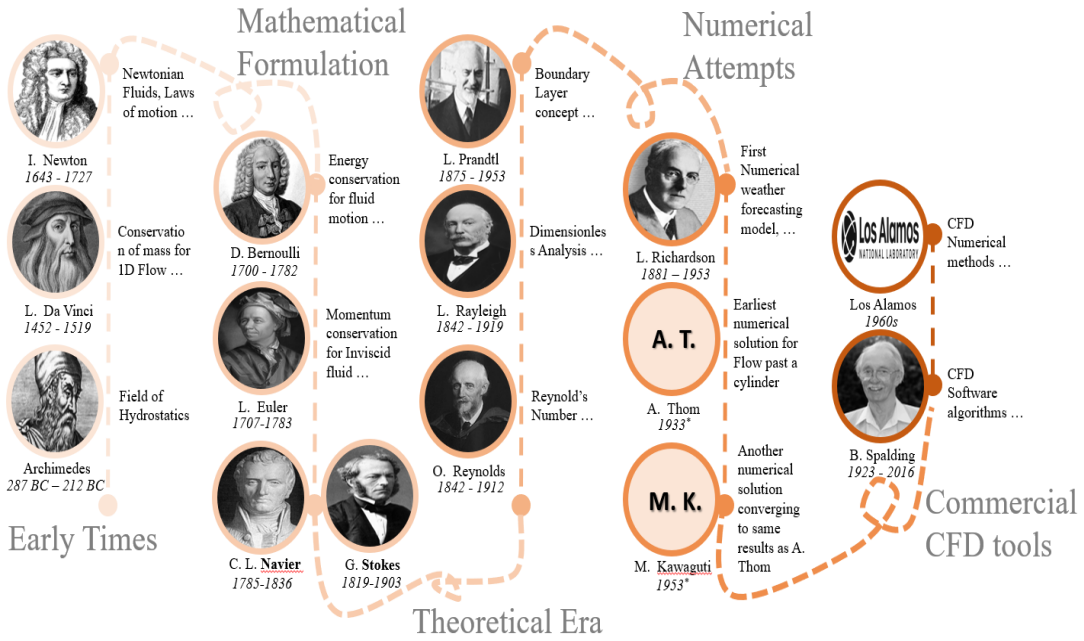


Figure 1.1: Evolution of Fluid Mechanics

From the day of their release, CFD tools has played a prominent role in understanding fluid dynamics. CFD simulators have become an indispensable tool for predicting fluid flows, analyzing them, and optimizing them in many fields [12], including aerospace, automotive, energy, environmental, biomedical, and architectural. The contribution of Computational Fluid Dynamics tools in enhancing efficiency, ensuring safety, and optimizing performance in various systems while simultaneously reducing production costs and environmental impacts has motivated scientists and industries to conduct extensive research on new algorithms and methods daily, aiming to attain more accurate and efficient CFD tools. This is obvious by investigating the number of published articles related to this field, as shown in Figure 1.2.

1.2 The Rise of Deep Learning

After a period marked by numerous substantial contributions, the ease for introducing novelty within the CFD field began to decrease. On the other hand, artifi-

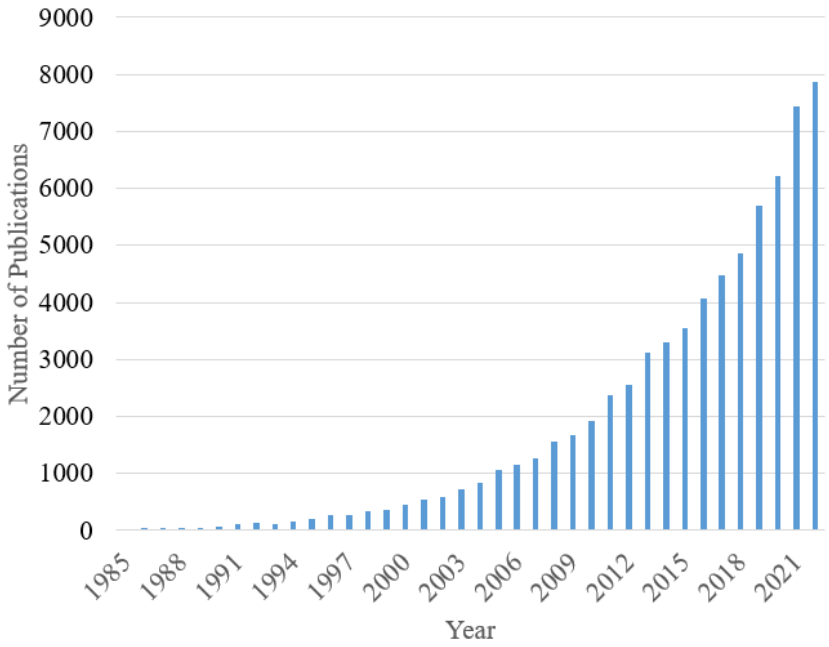


Figure 1.2: The number of articles, including both peer-reviewed and preprints, mentioning Computational Fluid Dynamics in either their title or their abstract. Results obtained from https://app.dimensions.ai/

cial intelligence, in general, and deep learning, in particular, began to demonstrate promising potential across various fields. Although deep learning is considered a relatively new field, its history runs back to the early days and has witnessed multiple peaks, valleys, and competitors before reaching its current position. The current deep and complex models of today, with their high capacities and potentials, stem back to a simple model introduced by Franck Rosenblatt under the name of the Perceptron, [13]. This computational model, which evolved from the early theoretical ideas of McCulloch and Pitts for mimicking the human brain nerve network by artificial neurons [14], consisted solely of a single output layer of perceptrons that could learn linear logical operators and classify simple shapes. The ambitions of Rosenblatt toward artificial intelligence and the birth of machine learning fueled great interest in this field. However, this momentum was shortly criticized by the subsequent findings of Marvin Minsky and Seymour Papert, [15]. Their research shed light on the limitations of the perceptron model and its mathematical inability even to model simple non-linear logical functions such as the exclusive OR, XOR, and the necessity of having at least two layers for modeling such functions with the incapability of current learning algorithms to train multilayer models.

These pivotal findings of Minsky and Papert triggered a first winter in machine

learning research, with interest in achieving artificial intelligence dimmed greatly. This winter lasted until Geoffrey Hinton, later named the godfather of deep learning, proposed a solution for mitigating the difficulties in training multilayer perceptions [16]. Hinton and two other colleagues, David Rumelhart and Ronald Williams, proposed employing the backpropagation algorithm, previously derived in the early 1960s [17], and later modernized by Linnainmaa [18], for training neural networks.

The introduction of backpropagation for training neural networks and the mathematical proof of their ability to act as universal approximations, [19], brought some warmth to the AI winter and motivated various developments during that period. Mainly, LeCun *et al.*, [20], defined the concept of convolutional neural networks trained by backpropagation for classifying handwritten digits from the MNIST dataset. Their LeNet-5 model was an extension of a previous model known as NeoCogNitron [21]. NeoCogNitron initially introduced the feature extraction and pooling concept but did not incorporate backpropagation for learning its parameters. The first practical application of Neural networks came through the LeNet-5 model, with only 60K learnable parameters, where it was nationally deployed for processing around 20 % of handwritten checks in the United States around 1998 [22]. Another significant advancement emerged from the research of Schmidhuber and Hochreiter, which enhanced the practicality and effectiveness of recurrent neural networks, [23], by introducing the Long-short-term memory variant that addressed the vanishing gradient problem and allowed them to capture long-range dependencies [24].

However, these developments in neural networks often went unnoticed and were overshadowed by Support Vector Machines (SVM) [25], an alternate machine learning approach developed by Cortes and Vapnik that showed faster training and simpler interpretability [26]. Although superior accuracy was attainable by relying on neural networks if trained on larger datasets, computational constraints and lack of data availability back then hindered such training. This impracticality of neural network training was further compounded by the vanishing gradients problem in deep networks with numerous layers [27], thus again diverting attention over neural network-related research.

Another significant breakthrough accomplished by Hinton and two of his students regarding the vanishing gradient problem sooner restored this fading attention. Hinton *et al.* tied this problem to the poor initialization of trainable parameters and proposed a layer-by-layer pretraining of the model's parameters [28]. The combination of this work, significant advancements in computational speed, and the adoption of graphical processing units all contributed to deep neural networks regaining competitiveness with other machine learning algorithms, including SVMs. Understanding the reasons behind the early obstacles encountered by deep learning models and addressing them by allocating greater consideration to the choice of activation functions and trainable parameters initialization, [29, 30], rapidly led to

proving their supreme performance over other machine learning algorithms by highlighting their power and achieving a state of art results starting with the ImageNet competition [31]. Deep learning models have, since then, consistently dominated the research landscape, driven by substantial interest and rational ambitions from major corporations like Google and Microsoft, with multiple potentials still uncovered. Figure 1.3 summarizes the key contributions that have shaped the history of deep learning and led to its current flourishing.

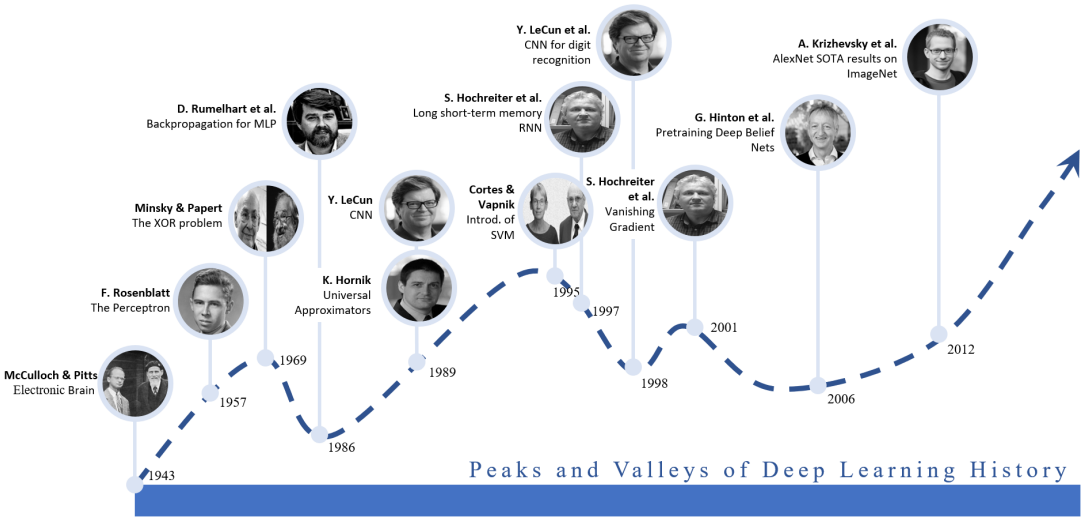


Figure 1.3: Evolution of Deep Learning

Although deep learning is an evolving field whose future is still hard to determine, the multitude of disciplines in which deep learning techniques have proven to be successful throughout the last decade and their transformative force in reshaping industries ascend it as an interdisciplinary technique with promising capacity worth investigating for addressing challenges in unexplored domains.

1.3 Challenges in CFD

Despite its long history and extensive contributions, several challenges still impede the ability of computational fluid dynamics to meet the continuously escalating needs of current industries. The significance of CFD for the industry lies in its ability to provide a virtual laboratory for analyzing fluid phenomena. This allows for optimizing various designs and enhancing the performance of many products, all while avoiding the need for expensive and time-consuming physical prototyping. However, this role is consistently tested by the ongoing requirement of achieving accurate simulations at an accelerated pace for even more intricate processes.

The significance of meeting these demands and accelerating numerical simulation techniques resides in the substantial time savings achieved. These savings extend to the various phases of system realization, beginning with the initial forecasts and extending to the diverse parametric studies employed in system conception and optimization. Additionally, it presents an opportunity to capture finer physics still unattainable with the largest available supercomputers. Thus, striving to cut huge computational costs and enhance its solvers, the CFD community began by investigating interdisciplinary approaches before expanding its search toward new disciplines that offer fresh, innovative solutions from out of the current box.

1.4 Interdisciplinary Approaches

Prior to incorporating deep learning techniques into the pursuit of addressing the complex challenges within the Computational Fluid Dynamics (CFD) community, a multitude of diverse approaches have been proffered and proven beneficial across a spectrum of scenarios. For example, in an attempt to close the gap of uncertainty between a numerical model prediction of a complex system and actual data, data assimilation techniques have been suggested [32, 33]. These techniques help reduce the uncertainty present in a model prediction by merging the strengths of numerical models and realistic observations. Data assimilation relies on comparing a short-term forecast of a numerical model to a newly received observation. The resulting error is employed for calibrating and refining the model state. These techniques were successful in various fields, such as oceanography and meteorology, where they have been used for operational ocean monitoring, [34], or large-scale weather forecasting [35]. Despite its advantages, data assimilation may encounter many challenges, mainly when access to observations is constrained, thereby diminishing the prospect of improving numerical model predictions. Similarly, when the problem being investigated resides in a high-dimensional space, the feasibility of the assimilation process is also questioned.

On another track, Model order reduction (MOR) is another discipline that focuses more on reducing the computational cost and complexity of CFD simulations rather than enhancing its accuracy, thus allowing for efficient exploration of variations in large parametric spaces [36]. MOR techniques aim to substitute a computationally expensive high-fidelity model with a faster and more efficient reduced-order model while maintaining a certain level of accuracy. Various numerical approaches have spanned the realm of Reduced order models, such as the Proper Orthogonal Decomposition - Galerkin approach [37], the proper generalized decomposition approach [38], or the reduced basis method [39]. Finally, although these approaches have been extensively and successfully employed for various CFD applications [40–42], their feasible applicability is not guaranteed for all problems, especially for the

ones with nonlinear solution manifolds such as advection-dominated problems, wave propagation problems, or elliptic problems with high parameter sensitivity [43–45]. These problems are characterized by maximum projection error, also known as the Kolmogorov width [46], that slowly decays when increasing the dimension of the approximation space. The required large number of approximation bases for attaining a sufficiently low Kolmogorov width hinders the fast computation of the approximation solution and thus motivates further research in this field. Various interdisciplinary approaches have already been suggested for adapting MOR techniques to these non-reducible problems, such as online adaptation of the ROM with new information for limiting the extrapolation error [47], interpolating between different precomputed ROMs [48], or relying on a dictionary of basis vectors [49], or a dictionary of several local ROMs [50]. However, various current traits of these methods still encourage deploying more efforts to explore their potential for improvement.

In the complex realm of turbulence flows, achieving the comprehensive resolution of scales while enforcing a mesh size smaller than the tiniest eddy scale proves to be a daunting computational challenge, even for relatively modest Reynolds numbers [51]. Consequently, various methods have been developed within turbulence modeling to address this issue. Among these methods, Large Eddy Simulation (LES) emerges as a sub-grid scale modeling technique that attempts to balance between accuracy and efficiency [52]. LES focuses on resolving larger scales only while modeling the influence of smaller unresolved scales. Despite this, LES remains out of reach for large-scale engineering simulations, compelling industrial applications to adopt the Reynolds-averaged Navier-Stokes (RANS) family of models [53]. These RANS models return time-averaged fields by representing the effects of turbulence through an artificial eddy viscosity [54–56]. However, despite their computational speed, RANS methods reside at the lower end of the accuracy spectrum. Consequently, various approaches have been proposed recently to hybridize them with superior methods, such as [57], while continuously searching to bridge the gap between computational efficiency and precise simulation results.

Finally, In addition to the preceding methodologies, a multitude of other techniques are also employed to further enhance and accelerate CFD models and have been comprehensively discussed in the literature. For instance, advances in parallel computing have significantly impacted CFD models by dividing a complex problem into smaller sub-problems that can be solved concurrently on multiple processors or cores. This distribution of workload has significantly contributed to attaining faster, higher resolution, and more integral simulations [58]. The potential of parallel computing was further realized with the emergence of Graphical Processing units and their recent adaptation in CFD tools, thus allowing for dramatic speedups compared to traditional central processing units (CPUs) [59]. Also, integrating adaptive mesh refinement techniques to optimize the mesh resolution in critical areas and coars-

ening it in regions with minor variations or less important physical impact allowed for even more efficient computational resource allocation. These techniques proved to be of significant importance in an extensive range of simulations, particularly in scenarios with complex geometries or in scenarios with multiple fluids evolving, thus allowing for achieving accurate results while further minimizing computational costs [60].

These mentioned methodologies and approaches certainly do not constitute an exhaustive list of all the developments employed to address the diverse challenges in CFD. However, it underscores the many existing difficulties and sheds light on the substantial scope for further improvement in currently employed techniques. This has encouraged the CFD community to explore other contemporary domains, particularly deep learning methodologies, in an attempt to complement current techniques and overcome some of the remaining obstacles.

1.5 Intradisciplinary Approaches

After achieving remarkable results on the ImageNet dataset and surpassing state-of-the-art models in terms of accuracy, deep learning models earned recognition and attracted more attention for further exploration. The considerable attention and investment from giant corporations, such as Google and Microsoft, intensified the research in Deep Learning and led it to gain considerable momentum, as shown in Figure 1.4.

The fruit of this surge initially impacted domains that traditionally lacked well-established mathematical models and often relied on conventional machine-learning approaches that demanded manual feature engineering [61]. As a primary example and due to the initial popularity of convolutional neural networks, the field of computer vision witnessed multiple revolutionary developments, especially in tasks related to object detection [62], image classification [63], and semantic segmentation [64]. This rendered deep learning techniques a standard approach for obtaining a performant and robust solution in computer vision and thus commercialized it for various scenarios like automated driving, product inspection, abnormality detection in medical imaging, facial recognition, and many others. Similar transformative effects were also witnessed in the fields of natural language processing [65], speech recognition [66], and many more. The positive impact of deep learning on these domains brought experts from diverse backgrounds together to leverage its capabilities. This led to the exploration of various model architectures, based on the needs of each domain, tailored to tackle specific challenges and operate over different data types. As an example, autoencoder-based models, specializing in dimensionality reduction and feature extraction, returned significant results in denoising tasks [67]. Meanwhile, transformer networks, known for their effectiveness on data with long-

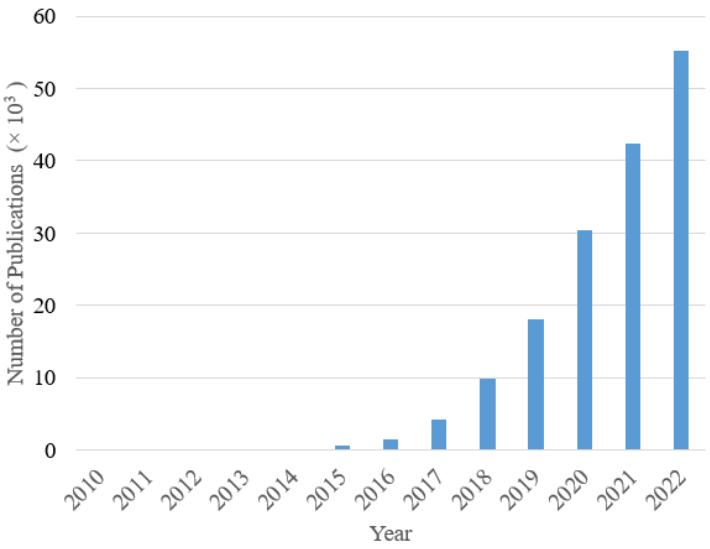


Figure 1.4: The number of articles, including both peer-reviewed and preprints, mentioning Deep Learning in either their title or their abstract. Results obtained from https://app.dimensions.ai/

range dependencies, found great utility in chatbots and image captioning [68, 69]. Lastly, graph neural networks expanded the capabilities of traditional convolutional neural networks to handle graph-structured data and found valuable applications in social network analysis and customer recommendation systems [70, 71]. These architectures represent just a fraction of the diverse landscape of deep learning and highlight its ability to evolve and adapt to various tasks.

Incorporating machine learning methods in general and deep learning techniques in particular with the scientific world was relatively delayed compared to other disciplines. The presence of mathematical models that have been developed, tested, and improved for centuries in the physical and scientific realm, along with the obscurity covering deep learning techniques due to their exclusive dependence on data, hindered most opportunities for cooperation between these two disciplines. However, although mathematical models are derived from physical laws, which themselves required a long history of observations and experimentation as detailed previously for fluid flows, these models are inherently simplifications of nature. This simplification might originate from insufficient observations, the fragility of derived models for drastic variations, the partial understanding of specific processes, or other computational challenges, as detailed previously. These limitations of current physical models, along with the proven strengths of machine learning approaches, especially regarding computation speed and approximation capacity, ignited the efforts for combining both worlds and attaining the best of both rather than solely relying on

each independently. The fruits of these efforts stem back to the pre-deep learning era, [72, 73], and have nourished the emergence of a new research field known as scientific machine learning [74].

1.5.1 Integrating Deep Learning in the Scientific World

Just as other domains have influenced the attributes of deep learning models and tailored various aspects for their specific needs, the cross-fertilization of deep learning models with the scientific world and the emergence of this new research track have given birth to various model adaptations, enabling more seamless integration. These adaptations have targeted several dimensions within the deep learning models, encompassing various areas such as the optimization of trainable parameters, their initialization, or the model architecture.

From the initial contributions to this field, Raissi *et al.* focused on addressing the sole reliance of deep learning models on data and their complete neglect of prior knowledge of physical laws [75]. They attempted to leverage prior knowledge and enhance the deep learning model capabilities by physically constraining the parameter optimizations by including the physical laws residuals in the models' loss function. Other consequent variations of this work relied on incorporating the variational form of the problem in the loss function, [76], or using the energy of the system as the basis for the construction of the loss function [77].

In an alternative approach, several research efforts were dedicated to initializing model parameters. Rather than commencing training from a random state, these approaches aimed to introduce the model to a physically informed starting point, thus accelerating its training and minimizing the dataset size needed [78]. The first suggested methodology for achieving a physics-guided initialization drew inspiration from the realm of computer vision and relied upon transfer learning [79]. In this method, the initial parameters of the model are either transferred from a pre-trained older model for a similar task, [80], or another model trained on an easy-to-collect dataset, [78]. Another methodology suggests supervising the hidden layers parameters for predicting an intermediate variable prior to training for the desired quantity of interest as depicted in [81] for modeling the temperature and flow in river networks.

Finally, various attempts contributed to leveraging deep learning architectures for its use in the scientific world. These works focused on enhancing the model's interpretability by enforcing desired physical properties into its architectural choice, thus constraining its search space and enhancing its generalization capability. One of the techniques for attaining this objective is to incorporate physically constrained neurons in the model architecture [82, 83]. These neurons are explicitly specified to mimic intermediate physical variables and thus help extract physically meaningful hidden representations. Another similar technique suggests assigning known values

from the governing equations for a certain number of the weights [84]. Moreover, system symmetries and invariances are usually desired traits for various physical systems. Encoding these characteristics into the model architecture positively impacts its robustness to multiple variations. This could be satisfied by relying on architectures that inherently possess invariant traits, such as convolutional neural networks that encode spatial invariance or recurrent neural networks that encode temporal invariance, or by adding customized layers to enforce required symmetries [85, 86]. Also, multiple architectures were proposed to favor specific physical properties or numerical schemes by construction. ResNets [87], for example, have found extensive applications in modeling various dynamical systems [88, 89]. The primary reason for their adoption is their resemblance with the Euler Integration scheme commonly used for numerically resolving ordinary differential equations [90]. The residual blocks within ResNets can be conceptualized as a mechanism for making incremental updates, thus directing the model's approximation capacity towards mastering the variations in the quantity of interest rather than exclusively concentrating on the direct transformation from the input to the output. Regarding physical properties, some models focused on integrating the Hamiltonian operator directly in their architecture. The Hamiltonian operator in physics corresponds to the system's total energy and is crucial in determining the time evolution of systems with conserved quantities. Some networks attempted to predict the Hamiltonian of the system and post-process it to attain the physical state of interest [91], while others attempted to learn the abstract phase space to easily generalize to other problems [92].

1.5.2 Expanding to the world of CFD

These various outcomes of the current coupling between the scientific and the deep learning world have triggered the curiosity of the research in the CFD community. The various emerging tailored developments under the umbrella of Scientific machine learning have revealed promising potential if they are well aligned to address the specific needs of CFD applications. In reality, these methodologies, developed by different domains independently of the specific needs of computational fluid dynamics, offered a new dimension for exploring novel solutions for addressing the current challenges. The recent growing interest of the CFD community in these data-based approaches can be noticed by inspecting the number of publications encompassing both disciplines through Figure 1.5.

For instance, in an attempt to address the computational cost of data assimilation processes for CFD applications, a deep-learning neural network was trained on the misfit between the results of a CFD air pollution model and a data assimilation model, thus implicitly providing data assimilated forecasts at a reduced cost when coupled with the CFD model [93].

Multiple efforts were also directed toward combining the potentials of both deep

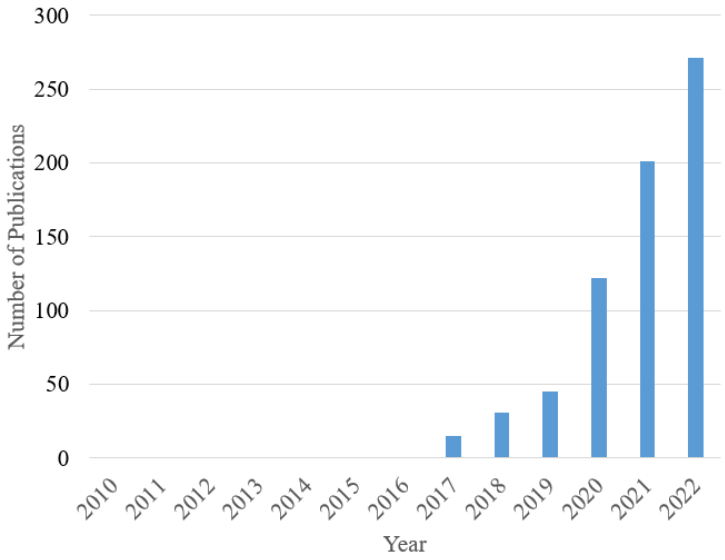


Figure 1.5: The number of articles, including both peer-reviewed and preprints, mentioning both Deep Learning and CFD in either their title or their abstract. Results obtained from https://app.dimensions.ai/

learning techniques and model order reduction methods. For example, in scenarios where the quantity of interest is discretized on a huge discretization space, dimensionality reduction techniques could be employed to obtain a reduced set of variables. These techniques could either be based on traditional interdisciplinary methods, such as the Linear Proper Orthogonal decomposition (POD), or rely on a data-based deep learning approach, such as the autoencoders [94]. Next, multiple model architectures are later suggested for modeling the latent dynamics, *e.g.*, POD reduction with multi-layer perceptrons (MLP) [95], or POD with long-short-term memory models (LSTM) [96], or autoencoder with LSTM for attaining nonlinear-latent-spaces [97]. Also, in the other scenarios of non-reducible problems, where linear model order reduction techniques suffer to provide accurate solutions, multiple solutions based on integrating a deep learning model in the solution framework were suggested. Some of these solutions relied on an autoencoder, as a computationally practical approach, for computing the nonlinear manifold prior to solving the dynamical system [98, 99], while other solutions relied on a deep learning model for recommending a local ROM from a cluster of models adapted for the various regions of the nonlinear manifold [100].

Additionally, in the context of turbulence modeling, integrating deep learning techniques into the CFD community also gave rise to various contributions. Some works have suggested different architectures for directly predicting the required turbulent physical fields [101–103]. Others have taken a hybrid approach and focused

the model capacity on learning high-fidelity enriched closure models [104–106]. Furthermore, some of the research efforts focused on attaining super-resolution flows from reconstructing underresolved ones [107, 108].

1.6 Outline

This thesis is dedicated to further exploring the untapped potential resulting from the convergence of two distinct disciplines, each with a rich history of contributions and achievements. The area of interest, highlighted in Figure 1.6, broadly categorizes the scope of this work's contributions. Precisely, this thesis aims to leverage novel deep learning methodologies, integrate them with well-established CFD tools, and tailor them to align with the current requirements of the community, particularly in the context of reducing computational costs of multiphysics CFD simulations.

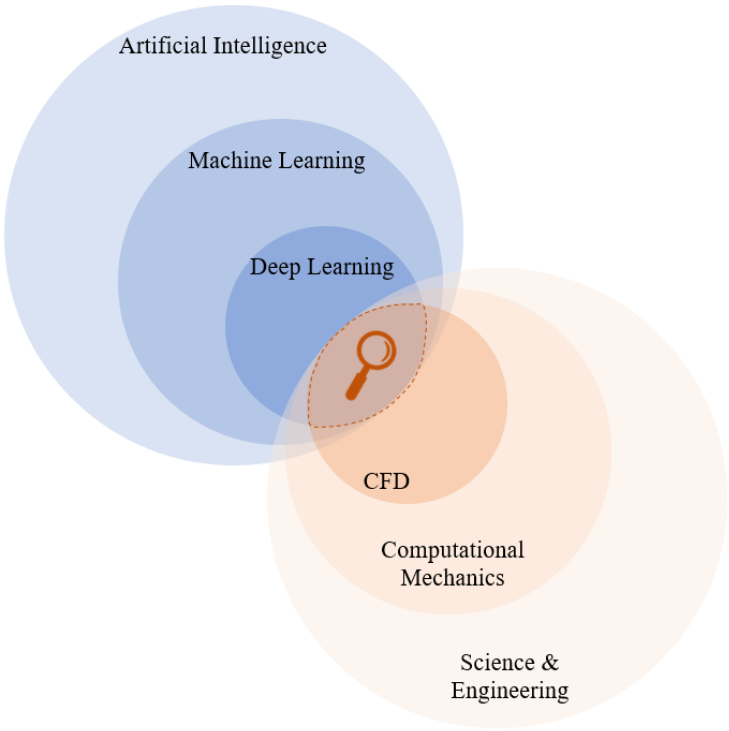


Figure 1.6: Region of focus of current thesis

The content of this manuscript is structured into three distinct stages, as illustrated in Figure 1.7. Each chapter of this thesis represents one of these stages, as will be elaborated upon later. In the first stage, the integration of deep learning models with CFD is initially examined through a convolutional neural network in

the context of a multiphysics conjugate problem. Next, in the second stage, this integration framework is further developed and adapted to address more intricate problems involving an evolving interface and necessitating a dynamic, unstructured triangular mesh. Finally, in the last stage, multiple implementation methodologies are proposed to address the curse of dimensionality of such a framework and thus expand its range of applications and facilitate its realization.

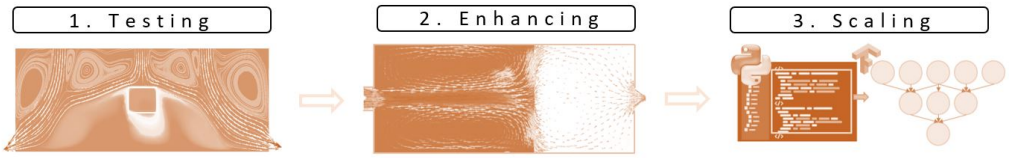


Figure 1.7: Organization of current thesis

In chapter two, a deep learning-assisted numerical solver is suggested for lifting part of the computational burden in the multiphysics CFD problem. The problem is concerned with the forced convective cooling of a workpiece placed in a confined space. Such a problem is governed by a unidirectional coupled system of partial differential equations consisting of the nonlinear Navier-Stokes equations and a scalar transport equation, the heat energy equation. Interest in addressing such a problem stems from the multitude of different processes governed by a similar set of equations. The suggested coupling framework relied on an encoder-decoder convolutional neural network trained on physical fields encoded into a structured grid for modeling the scalar energy equation while maintaining a bridge for communication and redirection with the traditional solver. The model's performance, accuracy, and ability to generalize to different cooling setups are all studied and discussed, along with all the procedure details.

In chapter three, the coupling framework suggested in the previous chapter is upgraded to tackle a two-fluid flow multiphysics CFD problem. The system of governing equations in this problem is similar to that introduced in chapter two, except that it encompasses a bidirectional coupling between the different equations and relies on the level-set equation required for capturing the interface between the different fluids rather than the heat energy equation. Moreover, the computational cost required for resolving such an equation is negligible, and most of it is dominated by resolving the Navier-Stokes flow fields. Furthermore, the dynamic evolution of the interface with the flow fields hinders the ability to capture the induced physics using a structured discretization grid accurately. Thus, to efficiently reduce the computational burden and avoid the accumulation of error and loss of accuracy, a graph-based deep learning model that operates directly on the dynamically evolving unstructured irregular triangular mesh and thus dodges the constraints of the previous convolutional model is suggested to simulate the complex Navier-Stokes flow

fields. The methodologies proposed for successfully training such a model, along with the various ingredients involved and the obtained results, are all detailed in this chapter.

In chapter four, the problem of dimensionality during the training of the deep learning models is addressed. Although incorporating a deep learning model in the solution loop might aid in addressing various challenges, the training of these models is faced with multiple obstacles, especially with high dimensional data points. The following chapter focuses on enriching current training methodologies with advanced implementation techniques to avoid the famous out-of-memory error while securing the fidelity of the attained results.

The following manuscript is finally concluded by evaluating our proposed methods and the currently available frameworks along with the available future paths for further accompanying the numerical transformation in the industry. In particular, the work detailed in this thesis focuses on addressing the evolving requirements of Transvalor, a French company with a worldwide presence that specializes in providing simulation software for material forming and processes. Transvalor plays a crucial role in supporting diverse industries, including automotive, aerospace, energy, construction, and medical fields, as they undergo digital transformation and accompany them in their material forming operations. Transvalor equips these industries with a comprehensive suite of high-performance simulation tools comprising over seven software solutions, each tailored to cover a broad spectrum of aspects related to forming processes involving metallic solids, liquids, and polymers.

* * *

Bibliography

[1] T. G. Chondros, Archimedes life works and machines, Mechanism and Machine Theory 45 (11) (2010) 1766–1775. 4

[2] L. de Vinci, Del moto e misura dell'acqua, 1828. 4

[3] I. Newton, D. T. Whiteside, et al., The mathematical papers of isaac newton, The Mathematical Papers of Isaac Newton (2008). 4

[4] G. K. Mikhailov, Daniel bernoulli, hydrodynamica (1738), in: Landmark Writings in Western Mathematics 1640-1940, Elsevier, 2005, pp. 131–142. 4

[5] P. Constantin, On the euler equations of incompressible fluids, Bulletin of the American Mathematical Society 44 (4) (2007) 603–621. 4

[6] P. Constantin, C. Foias, Navier-stokes equations, University of Chicago Press, 2020. 4

[7] R. Temam, Navier-Stokes equations: theory and numerical analysis, Vol. 343, American Mathematical Soc., 2001. 4

[8] O. Reynolds, Xxix. an experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels, Philosophical Transactions of the Royal society of London (174) (1883) 935–982. 4

[9] A. Thom, The flow past circular cylinders at low speeds, Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character 141 (845) (1933) 651–669. 4

[10] M. Kawaguti, Numerical solution of the navier-stokes equations for the flow around a circular cylinder at reynolds number 40, Journal of the Physical Society of Japan 8 (6) (1953) 747–757. 4

[11] D. B. Spalding, Development of the eddy-break-up model of turbulent combustion, Symposium (International) on Combustion 16 (1) (1977) 1657–1663. doi:https://doi.org/10.1016/S0082-0784(77)80444-X. 5

[12] S. V. Patankar, Numerical heat transfer and fluid flow, Hemisphere Publishing Corporation, 1980. 5

[13] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychological review 65 (6) (1958) 386. 6

[14] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics 5 (1943) 115–133. 6

[15] M. Minsky, S. A. Papert, Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry, MIT press, 2017. 6

[16] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al., Learning internal representations by error propagation (1985). 7

[17] H. J. Kelley, Gradient theory of optimal flight paths, Ars Journal 30 (10) (1960) 947–954. 7

[18] S. Linnainmaa, The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors, Ph.D. thesis, Master's Thesis (in Finnish), Univ. Helsinki (1970). 7

[19] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366. 7

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural computation 1 (4) (1989) 541–551. 7

[21] K. Fukushima, Neocognitron: A hierarchical neural network capable of visual pattern recognition, Neural networks 1 (2) (1988) 119–130. 7

[22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. 7

[23] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities., Proceedings of the national academy of sciences 79 (8) (1982) 2554–2558. 7

[24] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780. 7

[25] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (1995) 273–297. 7

[26] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al., Comparison of learning algorithms for handwritten digit recognition, in: International conference on artificial neural networks, Vol. 60, Perth, Australia, 1995, pp. 53–60. 7

[27] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al., Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001). 7

[28] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554. 7

[29] A.-r. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, M. A. Picheny, Deep belief networks using discriminative features for phone recognition, in: 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, 2011, pp. 5060–5063. 7

[30] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256. 7

[31] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012). 8

[32] R. E. Kalman, A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering 82 (1) (1960) 35–45. arXiv:https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf, doi:10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552 9

[33] P. Courtier, J.-N. Thépaut, A. Hollingsworth, A strategy for operational implementation of 4d-var, Ph.D. thesis, Shinfield Park, Reading (1992 1992). 9

[34] N. Ferry, E. Rémy, P. Brasseur, C. Maes, The mercator global ocean operational analysis system: Assessment and validation of an 11-year reanalysis, Journal of Marine Systems 65 (1-4) (2007) 540–560. 9

[35] M. Bonavita, P. Lean, 4d-var for numerical weather prediction, Weather 76 (2) (2021) 65–66. arXiv:https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/wea.3862, doi:https://doi.org/10.1002/wea.3862. URL https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/wea.3862 9

[36] A. Quarteroni, G. Rozza, et al., Reduced order methods for modeling and computational reduction, Vol. 9, Springer, 2014. 9

[37] L. Cordier, M. Bergmann, Proper Orthogonal Decomposition: an overview, in: Lecture series 2002-04, 2003-03 and 2008-01 on post-processing of experimental and numerical data, Von Karman Institute for Fluid Dynamics, 2008., VKI, 2008, p. 46 pages.
URL https://hal.science/hal-00417819 9

[38] F. Chinesta, P. Ladeveze, E. Cueto, A short review on model order reduction based on proper generalized decomposition, Archives of Computational Methods in Engineering 18 (4) (2011) 395–404. 9

[39] C. Prud'Homme, D. V. Rovas, K. Veroy, L. Machiels, Y. Maday, A. T. Patera, G. Turinici, Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods, J. Fluids Eng. 124 (1) (2002) 70–80. 9

[40] C. W. Rowley, T. Colonius, R. M. Murray, Model reduction for compressible flows using pod and galerkin projection, Physica D: Nonlinear Phenomena 189 (1-2) (2004) 115–129. 9

[41] C. Ghnatios, E. Hachem, A stabilized mixed formulation using the proper generalized decomposition for fluid problems, Computer methods in applied mechanics and engineering 346 (2019) 769–787.

[42] S. Deparis, G. Rozza, Reduced basis method for multi-parameter-dependent steady navier–stokes equations: applications to natural convection in a cavity, Journal of Computational Physics 228 (12) (2009) 4359–4378. 9

[43] M. Ohlberger, S. Rave, Reduced basis methods: Success, limitations and future challenges, arXiv preprint arXiv:1511.02021 (2015). 10

[44] C. Greif, K. Urban, Decay of the kolmogorov n-width for wave problems, Applied Mathematics Letters 96 (2019) 216–222.

[45] M. Bachmayr, A. Cohen, Kolmogorov widths and low-rank approximations of parametric elliptic pdes, Mathematics of Computation 86 (304) (2017) 701–724. 10

[46] V. N. Temlyakov, Nonlinear kolmogorov widths, Mathematical Notes 63 (6) (1998) 785–795. 10

[47] R. Zimmermann, B. Peherstorfer, K. Willcox, Geometric subspace updates with applications to online adaptive nonlinear model reduction, SIAM Journal on Matrix Analysis and Applications 39 (1) (2018) 234–261. 10

[48] T. Lieu, M. Lesoinne, Parameter adaptation of reduced order models for three-dimensional flutter analysis, in: 42nd AIAA Aerospace Sciences Meeting and Exhibit, 2004, p. 888. 10

[49] Y. Maday, B. Stamm, Locally adaptive greedy approximations for anisotropic parameter reduced basis spaces, SIAM Journal on Scientific Computing 35 (6) (2013) A2417–A2441. 10

[50] K. Washabaugh, D. Amsallem, M. Zahr, C. Farhat, Nonlinear model reduction for cfd problems using local reduced-order bases, in: 42nd AIAA Fluid Dynamics Conference and Exhibit, 2012, p. 2686. 10

[51] G. N. Coleman, R. D. Sandberg, A primer on direct numerical simulation of turbulence-methods, procedures and guidelines (2010). 10

[52] J. P. Boris, F. F. Grinstein, E. S. Oran, R. L. Kolbe, New insights into large eddy simulation, Fluid dynamics research 10 (4-6) (1992) 199. 10

[53] J. P. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, D. J. Mavriplis, Cfd vision 2030 study: a path to revolutionary computational aerosciences, Tech. rep. (2014). 10

[54] P. Huang, J. Bardina, T. Coakley, Turbulence modeling validation, testing, and development, NASA technical memorandum 110446 (1997) 147. 10

[55] D. C. Wilcox, Formulation of the k-w turbulence model revisited, AIAA Journal 46 (11) (2008) 2823–2838. arXiv:https://doi.org/10.2514/1.36541, doi:10.2514/1.36541.
URL https://doi.org/10.2514/1.36541

[56] S. R. Allmaras, F. T. Johnson, Modifications and clarifications for the implementation of the spalart-allmaras turbulence model, in: Seventh international conference on computational fluid dynamics (ICCFD7), Vol. 1902, Big Island, HI, Springer, 2012. 10

[57] J. Fröhlich, D. Von Terzi, Hybrid les/rans methods for the simulation of turbulent flows, Progress in Aerospace Sciences 44 (5) (2008) 349–377. 10

[58] A. Afzal, Z. Ansari, A. R. Faizabadi, M. Ramis, Parallelization strategies for computational fluid dynamics software: state of the art review, Archives of Computational Methods in Engineering 24 (2) (2017) 337–363. 10

[59] A. C. Crespo, J. M. Dominguez, A. Barreiro, M. Gómez-Gesteira, B. D. Rogers, Gpus, a new tool of acceleration in cfd: efficiency and reliability on smoothed particle hydrodynamics methods, PloS one 6 (6) (2011) e20685. 10

[60] F. Alauzet, A. Loseille, A decade of progress on anisotropic mesh adaptation for computational fluid dynamics, Computer-Aided Design 72 (2016) 13–39. 11

[61] D. C. Duro, S. E. Franklin, M. G. Dubé, A comparison of pixel-based and object-based image analysis with selected machine learning algorithms for the classification of agricultural landscapes using spot-5 hrg imagery, Remote sensing of environment 118 (2012) 259–272. 11

[62] W. Zhiqiang, L. Jun, A review of object detection based on convolutional neural network, in: 2017 36th Chinese control conference (CCC), IEEE, 2017, pp. 11104–11109. 11

[63] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, Y. Miao, Review of image classification algorithms based on convolutional neural networks, Remote Sensing 13 (22) (2021) 4712. 11

[64] Q. Geng, Z. Zhou, X. Cao, Survey of recent progress in semantic image segmentation with cnns, Science China Information Sciences 61 (2018) 1–18. 11

[65] L. Deng, Y. Liu, Deep learning in natural language processing, Springer, 2018. 11

[66] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, K. Shaalan, Speech recognition using deep neural networks: A systematic review, IEEE access 7 (2019) 19143–19165. 11

[67] L. Gondara, Medical image denoising using convolutional denoising autoencoders, in: 2016 IEEE 16th international conference on data mining workshops (ICDMW), IEEE, 2016, pp. 241–246. 11

[68] G. Caldarini, S. Jaf, K. McGarry, A literature survey of recent advances in chatbots, Information 13 (1) (2022) 41. 12

[69] M. Cornia, M. Stefanini, L. Baraldi, R. Cucchiara, Meshed-memory transformer for image captioning, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10578–10587. 12

[70] Q. Tan, N. Liu, X. Hu, Deep representation learning for social network analysis, Frontiers in big Data 2 (2019) 2. 12

[71] Q. Zhang, J. Lu, Y. Jin, Artificial intelligence in recommender systems, Complex & Intelligent Systems 7 (2021) 439–457. 12

[72] C. A. Bailer-Jones, D. J. MacKay, P. J. Withers, A recurrent neural network for modelling dynamical systems, network: computation in neural systems 9 (4) (1998) 531. 13

[73] M. W. Gardner, S. Dorling, Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences, Atmospheric environment 32 (14-15) (1998) 2627–2636. 13

[74] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, et al., Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence, Tech. rep., USDOE Office of Science (SC), Washington, DC (United States) (2019). 13

[75] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561 (2017). 13

[76] E. Kharazmi, Z. Zhang, G. E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint arXiv:1912.00873 (2019). 13

[77] E. Samaniego, C. Anitescu, S. Goswami, V. M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Computer Methods in Applied Mechanics and Engineering 362 (2020) 112790. 13

[78] X. Jia, J. Willard, A. Karpatne, J. S. Read, J. A. Zwart, M. Steinbach, V. Kumar, Physics-guided machine learning for scientific discovery: An application in simulating lake temperature profiles, ACM/IMS Transactions on Data Science 2 (3) (2021) 1–26. 13

[79] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, J. Liang, Convolutional neural networks for medical image analysis: Full training or fine tuning?, IEEE transactions on medical imaging 35 (5) (2016) 1299–1312. 13

[80] J. Lu, F. Gao, Model migration with inclusive similarity for development of a new process model, Industrial & engineering chemistry research 47 (23) (2008) 9508–9516. 13

[81] X. Jia, J. Zwart, J. Sadler, A. Appling, S. Oliver, S. Markstrom, J. Willard, S. Xu, M. Steinbach, J. Read, et al., Physics-guided recurrent graph model

for predicting flow and temperature in river networks, in: Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), SIAM, 2021, pp. 612–620. 13

[82] A. Daw, R. Q. Thomas, C. C. Carey, J. S. Read, A. P. Appling, A. Karpatne, Physics-guided architecture (pga) of neural networks for quantifying uncertainty in lake temperature modeling, in: Proceedings of the 2020 siam international conference on data mining, SIAM, 2020, pp. 532–540. 13

[83] N. Muralidhar, J. Bu, Z. Cao, L. He, N. Ramakrishnan, D. Tafti, A. Karpatne, Phynet: Physics guided neural networks for particle drag force prediction in assembly, in: Proceedings of the 2020 SIAM International Conference on Data Mining, SIAM, 2020, pp. 559–567. 13

[84] J. Sun, Z. Niu, K. A. Innanen, J. Li, D. O. Trad, A theory-guided deep-learning formulation and optimization of seismic waveform inversion, Geophysics 85 (2) (2020) R87–R99. 14

[85] R. Wang, R. Walters, R. Yu, Incorporating symmetry into deep dynamics models for improved generalization, arXiv preprint arXiv:2002.03061 (2020). 14

[86] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics 807 (2016) 155–166. 14

[87] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778. 14

[88] P. Rajendra, V. Brahmajirao, Modeling of dynamical systems through deep learning, Biophysical Reviews 12 (6) (2020) 1311–1320. 14

[89] B. S. R. Kunduri, M. Maimaiti, J. B. H. Baker, J. M. Ruohoniemi, B. J. Anderson, S. K. Vines, A deep learning-based approach for modeling the dynamics of ampere birkeland currents, Journal of Geophysical Research: Space Physics 125 (8) (2020) e2020JA027908. doi:https://doi.org/10.1029/2020JA027908. 14

[90] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, E. Holtham, Reversible architectures for arbitrarily deep residual neural networks, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and

Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018, p. 8. 14

[91] S. Greydanus, M. Dzamba, J. Yosinski, Hamiltonian neural networks, Advances in neural information processing systems 32 (2019). 14

[92] P. Toth, D. J. Rezende, A. Jaegle, S. Racanière, A. Botev, I. Higgins, Hamiltonian generative networks, arXiv preprint arXiv:1909.13789 (2019). 14

[93] C. Q. Casas, R. Arcucci, P. Wu, C. Pain, Y.-K. Guo, A reduced order deep data assimilation model, Physica D: Nonlinear Phenomena 412 (2020) 132615. 14

[94] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook (2023) 353–374. 15

[95] Q. Wang, J. S. Hesthaven, D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, Journal of Computational Physics 384 (2019) 289–307. doi:https://doi.org/10.1016/j.jcp.2019.01.031.
URL https://www.sciencedirect.com/science/article/pii/S0021999119300828 15

[96] Z. Wang, D. Xiao, F. Fang, R. Govindan, C. C. Pain, Y. Guo, Model identification of reduced order fluid dynamics systems using deep learning, International Journal for Numerical Methods in Fluids 86 (4) (2018) 255–268. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.4416, doi:https://doi.org/10.1002/fld.4416.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.4416 15

[97] R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, Physics of Fluids 33 (3) (2021). 15

[98] K. Lee, K. T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, Journal of Computational Physics 404 (2020) 108973. doi:https://doi.org/10.1016/j.jcp.2019.108973.
URL https://www.sciencedirect.com/science/article/pii/S0021999119306783 15

[99] Y. Kim, Y. Choi, D. Widemann, T. Zohdi, A fast and accurate physics-informed neural network reduced order model with shallow masked

autoencoder, Journal of Computational Physics 451 (2022) 110841. doi:https://doi.org/10.1016/j.jcp.2021.110841.
URL https://www.sciencedirect.com/science/article/pii/S0021999121007361 15

[100] T. Daniel, F. Casenave, N. Akkari, D. Ryckelynck, Model order reduction assisted by deep neural networks (rom-net), Advanced Modeling and Simulation in Engineering Sciences 7 (2020) 1–27. 15

[101] N. Thuerey, K. Weißenow, L. Prantl, X. Hu, Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows, AIAA Journal 58 (1) (2020) 25–36. 15

[102] A. Mohan, D. Daniel, M. Chertkov, D. Livescu, Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence, arXiv preprint arXiv:1903.00033 (2019).

[103] R. Wang, K. Kashinath, M. Mustafa, A. Albert, R. Yu, Towards physics-informed deep learning for turbulent flow prediction, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 1457–1466. 15

[104] B. D. Tracey, K. Duraisamy, J. J. Alonso, A machine learning strategy to assist turbulence model development, in: 53rd AIAA aerospace sciences meeting, 2015, p. 1287. 16

[105] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics 807 (2016) 155–166.

[106] A. Beck, D. Flad, C.-D. Munz, Deep neural networks for data-driven les closure models, Journal of Computational Physics 398 (2019) 108910. 16

[107] H. Kim, J. Kim, S. Won, C. Lee, Unsupervised deep learning for super-resolution reconstruction of turbulence, Journal of Fluid Mechanics 910 (2021) A29. 16

[108] B. Liu, J. Tang, H. Huang, X.-Y. Lu, Deep learning methods for super-resolution reconstruction of turbulent flows, Physics of Fluids 32 (2) (2020). 16

Chapter 2

Deep learning model to assist multiphysics conjugate problems

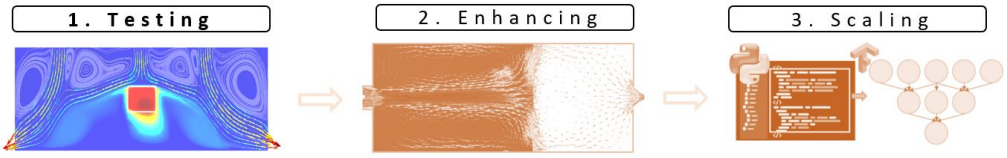


Figure 2.1: Introductory Figure for Chapter Two

Abstract *The availability of accurate and efficient numerical simulation tools has become of utmost importance for the design and optimization phases of existing industrial processes. The latter requires the computation of multiple physical fields governed by coupled systems of partial differential equations and tends to require large computational resources. Recently, the coupling of machine learning techniques with numerical simulation tools has allowed lifting part of this computational burden, by replacing parts of the resolution process with trained neural networks, which execution cost is far less than their traditional counterparts. The work of this chapter falls under the fist stage of all the three thesis stages, as highlighted in Figure 2.1, where the initial coupling of a deep learning model with CFD is tested. Precisely, an auto-encoder convolutional neural network is suggested to reduce the resolution cost of the forced cooling of a hot workpiece in a confined space by modeling the scalar transport equation coupled to the Navier–Stokes equations. Although the proposed model was trained on a relatively limited amount of data, it was able to generalize accurately for different cooling setups with different inlet locations, thus leading to a reliable deep learning-assisted numerical solver.*

Abstract *L'importance des outils de simulation numérique précis et performants a considérablement augmenté, notamment dans le contexte de la gestion des phases*

de conception et d'optimisation des processus industriels existants, qui nécessitent un calcul intensif impliquant de multiples champs physiques régis par des systèmes couplés d'équations aux dérivées partielles. Récemment, le couplage des techniques de machine learning avec les outils de simulation numérique a permis de réduire une partie de cette charge de calcul, en remplaçant certaines étapes du processus de résolution par des réseaux neuronaux entraînés, dont le coût d'exécution est bien inférieur à celui de leurs homologues traditionnels. Le travail de ce chapitre relève de la première étape des trois étapes de la thèse, comme indiqué dans la Figure 2.1, où le couplage initial d'un modèle deep learning avec la CFD est testé. Plus précisément, un réseau de neurones convolutionnels auto-encodeur est suggéré pour réduire le coût de résolution du refroidissement forcé d'une pièce chaude dans un espace confiné en modélisant l'équation de transport scalaire couplée aux équations de Navier-Stokes. Bien que le modèle proposé ait été entraîné sur une quantité relativement limitée de données, il a été en mesure de généraliser de manière précise pour différentes configurations de refroidissement avec différentes positions d'entrée, conduisant ainsi à un solveur numérique fiable assisté par le deep learning.

Contents

2.1 Introduction

Current industries are continuously aiming to satisfy the needs of the demanding market with high-quality products in the shortest feasible delay. This objective can only be achieved by rapidly and accurately forecasting the results of every single stage in the production process, thus establishing a precise idea of the final product from the early design phases. Prior simulation of problems involving fluid flows is accomplished using numerical analysis methods for computational fluid dynamics [1]. These methods serve numerous fields such as weather simulation, aerospace, aerodynamics, and many others.

Accurate modeling of many real-life phenomenons requires the coupling of multiple physics. Turbulent flow problems past a certain obstacle, for example, can be modeled by the nonlinear Navier–Stokes equations coupled with a single-equation turbulence model, [2], or a two-equation model, [3–5]. Other problems, that are involved with the cooling and heating of a workpiece using natural or forced convection, require the coupling with a heat transfer equation [6, 7].

Unfortunately, multiphysics problems governed by the coupling of the Navier–Stokes equations with a scalar transport equation, whether it is a weak or a strong coupling, are computationally expensive to solve, thus limiting the practicability of modeling complex industrial applications. Moreover, the partial differential equations in coupled problems have, as input parameters, fields set up in huge ambient spaces such as the velocity field convecting the scalar in transport equations. The high dimensionality of these input spaces makes extrapolations or interpolation quite difficult. However, the recent increased availability of computational capacities and data resources has brought the capabilities of deep learning to the front of the stage. In the past decade, the potential of machine learning has been demonstrated in multiple domains, such as natural language processing [8], machine translation [9], speech recognition [10], or computer vision [11]. The recent years also witnessed applications to computational mechanics for the simulation of physical problems. For example, In [12], Raissi *et al.* introduced a regularization mechanism that informs the deep learning model about the governing equations of the physical problem. In [13], the authors suggested using a deep classifier to adapt the reduced-order model to an input tensor parametrizing an anisothermal elastoplastic problem in structural mechanics. Neural networks that operate directly on graph-structured data, [14–16], have also been exploited to assist in various physical problems. In [17], the authors suggested incorporating neural networks to infer problem-specific coefficients for the estimation of spatial derivatives required for the solution of various PDEs modeling physical phenomena such as shock formations, solitary waves on a river bore, and flame fronts. Another method where a deep learning model is used to reduce the computational cost of a physical simulation is suggested in [18]. The authors proposed assisting the in-plane-out-of-plane PGD solver by introducing a model re-

sponsible for inferring the out-of-plane function and tested their methodology for a 3D heat conduction problem in a plate.

In the present work, an auto-encoder architecture [19, 20] is employed to model the scalar transport equation in a coupled system. This system, consisting of the Navier–Stokes equations along with the heat energy equation, is required to simulate the cooling process of a work object using forced convection. The required data to train the model is obtained using an industrial-level computational fluid dynamics solver [21]. The latter exploits the immersed volume method (IVM) [22] equipped with interface capturing and anisotropic mesh adaptation method [23], to overcome the difficulties encountered in specifying the specific solid-fluid interface. The resulting equations are resolved using a continuous finite element method along with the variational multiscale approach (VMS) [24–26]. The mathematical formulation of IVM, in the context of finite element VMS methods, is detailed in the following papers [27, 28].

The deep learning model architecture used can accurately model the scalar energy equation and thus infer the required future temperature field at the next time step. The model is trained to predict the field with a larger time step than that of the solver, thus allowing to span the whole time domain in much fewer computational steps. Moreover, the trained model reveals precise interpolation ability for both, time and cooling inlets positions, thus permitting us to exploit its potential for numerous flow setups by moderately training it on a few snapshots, scattered across the required time domain, obtained from discrete setups. The model also manages to span independently the whole time domain, *i.e.* relying solely on its prediction and without referring back to the traditional solver, and returns reliable results with much less computational time.

The success of this model paves the way for assisting the simulation of numerous industrial problems where coupling with a scalar transport equation exists. Moreover, other types of transport equations can be examined in future work with the same methodology, thus broadening the potential of this technique. The paper commences by stating the governing equations and the methods used to resolve them numerically in section 2.2. Details of the problem setup are provided in section 2.3. Section 2.4 details the sets used to train and test the model. The model architecture and its training are respectively presented in sections 2.6 and 2.7. Finally, the performance of the model on multiple flow setups and its ability to span independently the required time domain are briefed in 2.8. This contribution is concluded finally in section 2.11.

2.2 Governing Equations

The Navier–Stokes momentum and continuity equations are usually accompanied by a third scalar transport equation to simulate different physical applications. For example, the turbulent flow past an object requires supporting these equations with a turbulent model such as the Spallart–Allmaras model [29]. In case of heat transfer problems, the equations are accompanied by a heat equation to govern the change of temperature in the domain. The general form of the governing equations is shown in (2.1), where the third equation is a comprehensive form of the scalar transport equation that can be used to compute different fields based on the problem setup.

$$\begin{aligned} \rho(\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}) &= \nabla \cdot (-p\mathbf{I} + 2\mu\boldsymbol{\varepsilon}(\mathbf{u})) + \boldsymbol{\psi}, \\ \nabla \cdot \mathbf{u} &= 0, \\ \alpha \partial_t F + \beta \mathbf{u} \cdot \nabla F &= \nabla \cdot (\gamma \nabla F) + \chi. \end{aligned} \tag{2.1}$$

In the above system, $\mathbf{u}$ and p are respectively the velocity and pressure fields, while F is the coupled scalar field required for the test case simulation. In the case of heat transfer, F would represent the temperature distribution in the domain, while for turbulent flows, F would represent the Spallart–Allmaras scalar, $\tilde{\nu}$. ρ and μ are the fluid's density and dynamic viscosity respectively, while α, β, and γ are coefficients specific to the coupled model. Finally, $\boldsymbol{\varepsilon}(\mathbf{u}) = (\nabla\mathbf{u} + \nabla\mathbf{u}^T)/2$ is the rate of deformation tensor, while $\boldsymbol{\psi}$ and χ are the source terms.

For problems concerned with an interaction between a fluid and a solid phase, several coefficients should be predefined to govern the exchange at the interface. The investigation of the value of the coefficients is an exhaustive task and requires previous research. To overcome this difficulty, the immersed volume method is suggested [7]. Using IVM, the problem is defined using a single fluid domain but with different properties for the solid and the fluid materials. Unifying the domain eliminates the need of specifying coefficients that may vary with shape, size, or position. Thus, the interaction between both phases is implicitly derived due to different properties across the elements. Sections 2.2.1, 2.2.2, and 2.2.3 introduce the main building ingredients required for the successful implementation of this method. Furthermore, the resulting variational formulation of the problem is prone to instability and numerical oscillations due to the presence of large convective fields. The variational multi-scale method is introduced in section 2.2.4 to stabilize the discretized form of the problem, resulting in accurate solution fields compared to the standard Galerkin formulation.

2.2.1 Level-set method

The efficiency of the IVM relies on multiple ingredients. The interface between the solid and the fluid must be accurately defined to enable the initialization of the varying properties between the solid object and the fluid surrounding it. This interface is specified using a smoothly varying signed distance function, hereafter denoted ϕ. Conventionally, ϕ attains a positive value in the fluid domain Ω_f, a negative value within the solid domain Ω_s, and is equal to zero on the interface $\Gamma_{interface} = \Omega_s \cap \Omega_f$, as summarized in (2.2). The following framework, used to specify the solid phase and define the interface by a zero-level set distance function, is known as the level set method [30].

$$\phi(x,y) = \begin{cases} -d\left((x,y), \Gamma_{\text{interface}}\right) & \text{if } (x,y) \in \Omega_s, \\ 0 & \text{if } (x,y) \in \Gamma_{\text{interface}}, \\ d\left((x,y), \Gamma_{\text{interface}}\right) & \text{if } (x,y) \in \Omega_f \end{cases} \tag{2.2}$$

where $d\left((x,y), \Gamma_{\text{interface}}\right)$ is the nearest distance from a point in the domain, (x,y), to the fluid-solid interface, $\Gamma_{\text{interface}}$. Thus, the interface between the solid phase and the fluid phase, $\Gamma_{\text{interface}}$, will be identified by the zero-level set of ϕ as seen in equation (2.2). In the case of advection, the distance function is cast into a scalar transport equation. This equation accounts for any variation in the shape, size, displacement, etc., of the solid. The solution of such an equation requires numerical attention to ensure feasibility for long-time simulations and avoid the vanishing of the function with time. For more details regarding the level set method, the reader is invited to read [30–33]. It should be noted that in our test case, the solid phase is fixed in position and vary neither in size nor in shape. Thus, the level set function does not evolve as the solution proceeds.

2.2.2 Mixing laws

After defining the level set function that enables the specification of different phases location, the distribution of properties along the domain can be initialized with the aid of appropriate mixing laws. Almost all physical properties are defined using the mixing law introduced in (2.3):

$$p(x,y) = p_f H_\epsilon\left(\phi(x,y)\right) + p_s\left(1 - H_\epsilon\left(\phi(x,y)\right)\right), \tag{2.3}$$

where p is any physical property that affects the resulting simulation such as the density, viscosity, or specific heat. p_f and p_s are the values of this property for both the fluid phase and the solid phase, respectively. H_ϵ is the smoothed Heaviside function [34], which is used to obtain improved continuity at the fluid-solid interface by introducing a virtual thickness to the interface. The smoothed Heaviside function

is parameterized by the interface thickness ϵ, which depends on the mesh size at the interface and is used to specify the virtual thickness of the interface. The smoothed Heaviside function is defined as shown in equation (2.4):

$$H_\epsilon(\phi) = \begin{cases} 1 & \text{if } \phi > \epsilon, \\ \frac{1}{2}\left(1 + \frac{\phi}{\epsilon} + \frac{1}{\pi} sin(\frac{\pi\phi}{\epsilon})\right) & \text{if } |\phi| \leq \epsilon, \\ 0 & \text{if } \phi < -\epsilon. \end{cases} \tag{2.4}$$

It should be noted that references [35, 36] suggest using a harmonic mixing law for the thermal conductivity λ, rather than the general law defined in (2.3). The harmonic law is defined in equation (2.5) and is used to ensure continuity of the heat flux in addition to enhanced numerical accuracy:

$$\frac{1}{\lambda(x,y)} = \frac{1}{\lambda_f} H_\epsilon\left(\phi(x,y)\right) + \frac{1}{\lambda_s}\left(1 - H_\epsilon\left(\phi(x,y)\right)\right) \tag{2.5}$$

2.2.3 Anisotropic mesh adaptation

Simulations of processes with multiple phases, such as flow past a solid object, usually require a highly refined mesh to capture all the physics near the interface, leading to increased computational requirements. The computation time of such simulations can be drastically reduced using anisotropic mesh adaptation while maintaining the accuracy and reliability of the simulation, as shown in [37, 38]. The anisotropic adaptation reduces the number of nodes required by focusing on regions with large variations in the physical or geometrical fields. The metric used to perform the adaptation may depend on multiple variables where a method based on the intersection of all variables' metrics is used to compute it. For example, in the case of forced convection, the normalized fields of the velocity components, velocity magnitude, the level set, and the temperature, are used to compute the adaptation metric. Moreover, anisotropic mesh adaptation avoids numerical oscillations by aligning the element edges with the interface. Also, it reduces the virtual thickness of the interface due to the refinement of the mesh size along it. The adaptation method used in this paper follows the same method used in [39]. The reader is advised to refer to [40] for more details regarding the method summarized below.

To compute the required adaptation metric, the procedure starts by estimating an error associated with each edge. The error indicator function e^{ij}, associated to an edge $x^{ij} = x^j - x^i$ (where node j lies in the patch $\Gamma(i)$ of the nodes sharing a single edge with the node i), is defined using the exact interpolation error shown in (2.6):

$$e^{ij} = \left| g^{ij} \cdot x^{ij} \right|, \tag{2.6}$$

where g^{ij} is the change in the gradient along edge x^{ij} of a P1 finite element Lagrange approximation function u_h, *i.e.* $g^{ij} = g^j - g^i = \nabla g \cdot x^{ij}$. However, the gradient is not known at the nodes, and thus an estimated error function is defined using a recovered gradient G^i, computed using a length distribution tensor X^i, as shown in equation (2.7):

$$e^{ij} = G^{ij} \cdot x^{ij}, \tag{2.7}$$

where the recovered gradient at node i, G^i, is defined as:

$$G^i = (X^i)^{-1} \sum_{j \in \Gamma(i)} U^{ij} x^{ij}$$

while the length distribution tensor X^i, is equal to:

$$X^i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} x^{ij} \otimes x^{ij}$$

After computing the estimated error, a stretching factor relative to every edge is defined:

$$s^{ij} = \frac{e_{ij}}{e(N)},$$

where $e(N)$ is the total mesh estimated interpolation error. Finally, the required metric $\tilde{M}^i$, is computed as shown in (2.8):

$$\tilde{M}^i = \left(\tilde{X}^i\right)^{-1}, \tag{2.8}$$

where,

$$\tilde{X}^i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} s^{ij} \otimes s^{ij}.$$

2.2.4 Variational multi-scale approach

The system of governing equations (2.1) can then be resolved using the finite element method, to obtain accurate solutions eligible to be used as ground truth for the training of a deep learning model. To obtain the weak form of (2.1), each equation is multiplied by its specific weighting function, and integration by parts is performed, leading to the following weak form:

$$(\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}), \boldsymbol{w}) + (2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}), \boldsymbol{\varepsilon}(\boldsymbol{w})) - (p, \nabla \cdot \boldsymbol{w}) + (\nabla \cdot \boldsymbol{u}, q) = (\boldsymbol{\psi}, \boldsymbol{w}),$$
$$((\alpha \partial_t F + \beta \boldsymbol{u} \cdot \nabla F), s) + (\gamma \nabla F, \nabla s) = (\chi, s),$$

where $\boldsymbol{w}$, q, and s are respectively the test functions for velocity, pressure, and the required scalar field F. Yet, the Galerkin discrete formulation of the above weak form may fail if the flow is advection-dominated, or if the space discretization of the problem variables does not satisfy the LBB criteria (also known as the Babuska-Brezz condition or the inf-sup condition) [41]. To ensure the convergence of the above system to a unique solution not polluted by artificial oscillations, the variational multiscale method (abridged to VMS) will be utilized [24].

The VMS method starts by decomposing the required fields and their weighting functions, *i.e.* the velocity, pressure, and the scalar variable governed by the transport equation, into two scales: (i) a resolvable coarse-scale, and (ii) an unresolved fine-scale. Replacing the decomposed fields into the variational form results in two sub-problems specific for every scale. For the fine-scale Navier–Stokes problem, a separation technique is proposed to facilitate the resolution of the unknown fine fields [42, 43]. The continuity equation is replaced by a pressure Poisson equation allowing to approximate the fine scale pressure as a product of a stabilization parameter and the residual of the continuity equation. The fine-scale velocity field is similarly expressed from the momentum equation, with the aid of a bubble function, as a product of another stabilization parameter, and the residual of the coarse-scale momentum equation. The assumptions required to reach the above forms of the fine-scale fields are justified in [44–46]. The definition of the stabilization parameters appearing in the continuity and momentum equation can be found in [47, 48]. A similar approach is applied for the convection-diffusion-reaction transport equation, [49, 50], and the definition of its corresponding stabilization parameters is stated in [51, 52].

For the coarse-scale problem, the fine-scale fields are replaced by their corresponding expressions obtained from solving the fine-scale problem. This eliminates the appearance of the fine scales but preserves their effects on the coarse-scale problem as shown in (2.9):

$$
\begin{aligned}
&(\rho(\partial_t \boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u})\ ,\ \boldsymbol{w}) + (2\mu\boldsymbol{\varepsilon}(\boldsymbol{u})\ ,\ \boldsymbol{\varepsilon}(\boldsymbol{w})) - (p\ ,\ \nabla\cdot\boldsymbol{w}) + (\nabla\cdot\boldsymbol{u}\ ,\ q) = (\boldsymbol{\psi}\ ,\ \boldsymbol{w}) \\
&\qquad + \sum_{K\in\mathcal{T}_h} [(\tau_1\mathcal{R}_M\ ,\ \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{w})_K + (\tau_1\mathcal{R}_M\ ,\ \nabla q)_K + (\tau_2\mathcal{R}_C\ ,\ \nabla\cdot\boldsymbol{w})_K] \\
&\big((\alpha\partial_t F + \beta\boldsymbol{u}\cdot\nabla F)\ ,\ s\big) + (\gamma\nabla F\ ,\ \nabla s) = (\chi\ ,\ s) \\
&\quad + \sum_{K\in\mathcal{T}_h} [(\tau_3\mathcal{R}_F\ ,\ \boldsymbol{u}\cdot\nabla s)_K + (\tau_4\mathcal{R}_F\ ,\ (\boldsymbol{u}\cdot\nabla F/||\nabla F||^2)\nabla F\cdot\nabla s)_K]
\end{aligned}
\tag{2.9}
$$

where $\mathcal{R}_M$, $\mathcal{R}_C$, and $\mathcal{R}_F$ are respectively the approximate residuals of the momentum, continuity, and scalar transport equation. The diffusion term is neglected in the residuals as can be seen in (2.10):

$$
\begin{aligned}
-\mathcal{R}_M &= \rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u}) + \nabla p - \psi, \\
-\mathcal{R}_C &= \nabla \cdot \boldsymbol{u}, \\
-\mathcal{R}_F &= \alpha \partial_t F + \beta \boldsymbol{u} \cdot \nabla F - \chi.
\end{aligned} \tag{2.10}
$$

It can be seen that the resulting discrete stabilized variational form of the problem, (2.9), includes additional integrals over the sum of element interiors compared to the standard Galerkin formulation. These additional terms enrich the coarse-scale solution of the fields with fine-scale characteristics. Moreover, adding these terms relieves the restrictions on the discrete fields spaces, enhances its accuracy, and stabilizes the problem by including dissipative small scales contributions. For more extensive details regarding the VMS method, the reader is referred to [53, 54]. The resulting set of equations is solved sequentially using an industrial-level in-house VMS solver, the results of which were previously validated [27, 28]. The solver computes the velocity and pressure fields before computing the additional scalar governed by the scalar transport equation. It should be noted that all linear systems are preconditioned with a block Jacobi method supplemented by an incomplete LU factorization, and solved with the GMRES algorithm.

2.3 Problem Definition

The problem detailed in this paper concerns the forced cooling of a hot workpiece by a cold fluid flow, placed in a rectangular cavity. Such a problem requires the coupling of Navier–Stokes equations with a heat energy equation responsible for the evolution of temperature across the domain. Thus, the problem is governed by a set of equations similar to (2.1), but with a scalar transport equation specific to heat transfer:

$$
\rho c_p(\partial_t T + \mathbf{u} \cdot \nabla T) = \nabla \cdot (\lambda \nabla T) + \chi,
$$

where ρ, c_p, and λ are respectively the fluid density, specific heat, and thermal conductivity. This equation is a scalar transport equation similar to the one defined in (2.1) with $\alpha = \beta = \rho c_p$ and $\gamma = \lambda$. However, the fluid flow and its properties are assumed to be invariant for the variations in the temperature field, meaning that the coupling in this problem is weak, and the interaction between the computed fields is unidirectional.

A schematic of the problem setup is shown in figure 2.2. The setup consists of a $0.2H \times 0.4H$ hot rectangular object placed at the center of a closed rectangular cavity with a width of $4H$ and a height of H. The upper wall of the rectangular cavity contains three similar inlets, $0.2H$ in width, required for the blowing of the cold fluid. The distance of the side inlets to the origin of the cartesian coordinate

system coinciding with the center of mass of the object, d_{in1} and d_{in2}, is varied throughout the different data sets generated to enable us to train the model for different setups and test its generalization capacity on new ones as will be detailed later. Finally, the sidewalls of the rectangular cavity are equipped with outlets, having the same dimensions as the inlets, to allow the escape of the hot fluid. The outlets are positioned at the bottom of the sidewalls as is depicted in the problem setup schematic.

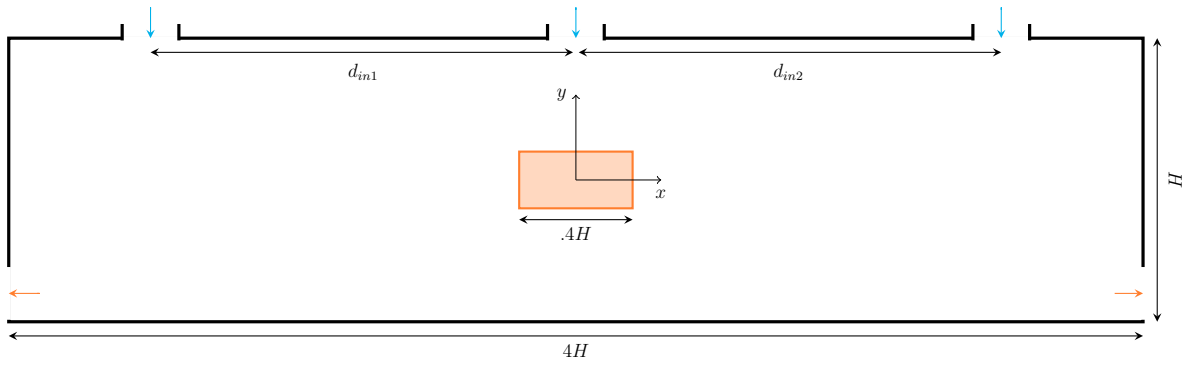


Figure 2.2: ***Setup schematic*** *required for cooling a hot object with fluid flow. The cold fluid inlets are represented with blue arrows while the hot fluid outlets with red ones. This specific setup shown has inlets placed symmetrically on both sides with* $d_{in1} = d_{in2} = 1.5H$.

To define the flow problem, the magnitude of the velocity at the inlets is set to $V_{in} = 1\text{m/s}$ with a fluid density of $\rho_f = 1\text{kg/m}^3$ and dynamic viscosity of $\mu_f = 0.001\text{Pa·s}$, thus resulting in a flow with Reynolds number, $Re = \frac{\rho_f V_{in} l}{\mu_f}$, equal to 200. The outlets are designated with a zero-pressure condition. A no-slip boundary condition is specified on the remaining boundaries. As detailed in section 2.2, the immersed volume method is utilized to simulate the flow around the immersed object. Thus, the solid phase is defined as a fluid with relatively high density and viscosity, $\rho_s = 100\text{kg/m}^3$ and $\mu_s = 1000\text{Pa·s}$. The very large magnitude of the ratio between the solid viscosity and that of the fluid, $\frac{\mu_s}{\mu_f} = 10^6$, ensures the satisfaction of the no-slip condition at the solid-fluid interface along with a zero velocity in the solid domain. Also, due to zero velocity in the solid domain, the convective term in the energy equation diminishes, resulting in a pure conduction equation whenever the radiation effect is neglected (*i.e.* $\chi = 0$). The source term in the NS equations is also set to zero, thus neglecting any buoyancy forces or stress on the fluid.

Concerning the heat equation, the hot temperature of the solid is initialized to $T_h = 150°\text{C}$, whereas the cold temperature of the fluid at the inlet, T_c, is set to 10°C. Isothermal condition is enforced on all the walls with $T_w = T_c = 10°\text{C}$. The heat exchange at the interface is implicitly regulated by specifying different values for the

thermal properties of the composite fluid. For instance, the thermal conductivity λ_f and specific heat $c_{p,f}$, for the fluid are set to 0.5W/m·K and 1000J/kg·K respectively, thus resulting in a Prandtl number, $Pr = \frac{c_{p,f}\mu_f}{\lambda_f}$, of 2, while those of the solid are set to: $\lambda_s = 15$W/m·K and $c_{p,s} = 300$J/kg·K.

An unstructured triangular mesh of 15 000 elements, accompanied with anisotropic mesh adaptation as detailed in section 2.2.3, is used to discretize the computational domain. Figure 2.3 shows the developed mesh at a specific time step after the flow has stabilized. A visualization of the computed fields that will be fed to the model, *i.e.* the temperature and velocity components, is also shown in figure 2.4.

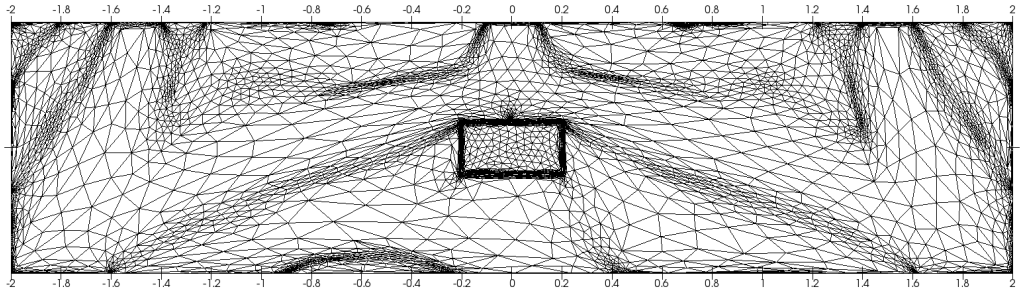


Figure 2.3: ***Adapted mesh*** *at a certain time step used to discretize the computational domain.*

2.4 Generated Datasets

The main objective of this paper is to assist a numerical solver by directly inferring the temperature field using a deep learning network modeling the scalar transport equation. Thus, the model should be trained to infer the temperature field for different cooling setups. To attain this objective, the fields are computed for multiple inlet positions with the same flow conditions. The dataset is assembled with the temperature, at a certain time step, along with the velocity components as input features and the temperature field, at a future time step, as the desired target. Figure 2.5 shows a representative sketch of the model with its corresponding inputs and inferred output.

The fields are computed using a CFD solver that discretizes the domain into an unstructured irregular triangular mesh. However, since the model used will be based on 2D convolutional layers, the computed fields must be interpolated to a structured encoding mesh. Thus, two encoding meshes are suggested, a $161 - by - 641$ mesh for the input field and a $176 - by - 656$ mesh for the output fields. The size of the output mesh is implicitly specified by the model architecture since no interpolation or cropping layers are utilized to enforce a certain size, thus preserving the information

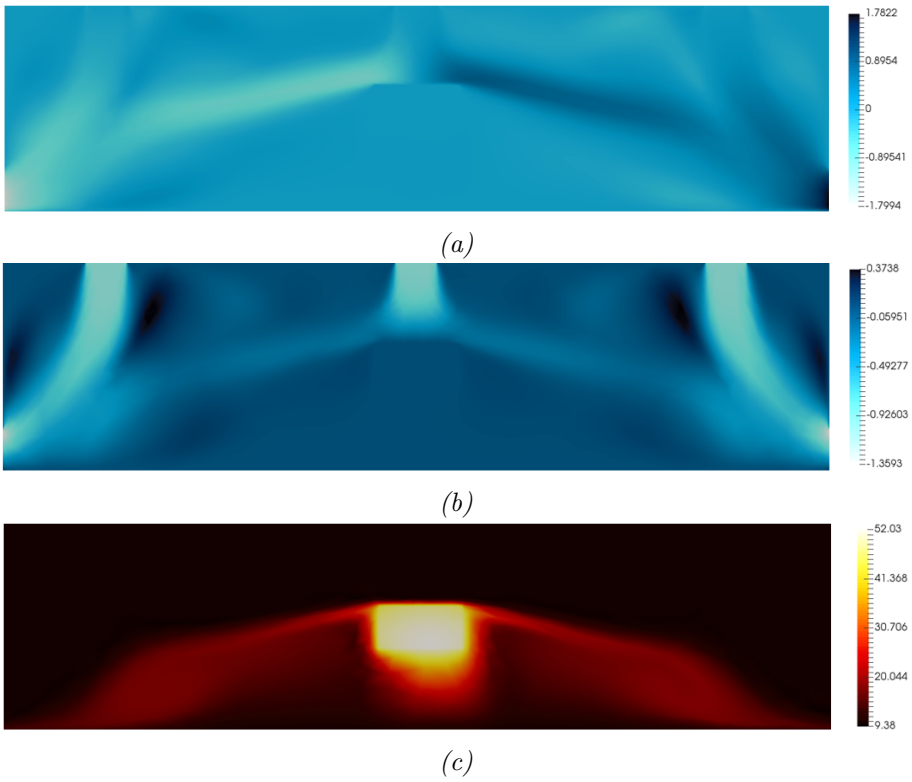


Figure 2.4: ***Simulated fields*** *obtained at a developed time step after the flow has stabilized. The fields shown are the (a) Velocity x-component, (b) Velocity y-component, and (c) Temperature.*

encoded along the feature maps boundaries. The computed fields will be transported from the unstructured triangular mesh, using linear Lagrange interpolation elements, to the required structured representation.

The first dataset, denoted as **SymVar**, is obtained by computing the fields for 18 different symmetrical inlet positions. The inlet positions are obtained by varying the distance from the origin to the center of the side inlets, d_{in1} and d_{in2}, between $0.1H$ and $1.8H$ with a step of $0.1H$. The data is sampled after the flow has stabilized. The solver's time step, Δt_{solver}, is equal to 0.1 sec. However, the model will be trained to infer the fields with 30 times larger time step, meaning that, given the fields at t_1, the model infers the temperature field at $t = t_1 + 30\Delta t$. The choice of a larger model time step, compared to that of the original solver, is motivated by reducing the number of operations required to span the whole time domain. Yet, the size of the increase depends on the scalar field being inferred and should avoid discarding variations necessary for the inferring of the future time step. Moreover, increasing the incremental step will lead to a decrease in the data available for training, thus

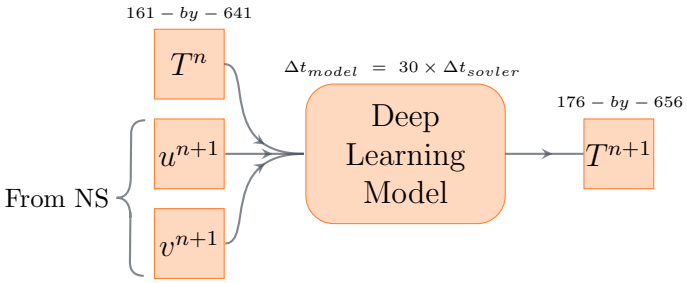


Figure 2.5: ***Model sketch***

decreasing the model inferring quality. Thus, the size of the incremental step of the model is a parameter that its choice is affected by the transport equation being modeled and the required model accuracy. 329 snapshots of fields are gathered from every simulation between $t_i = 50$ s. and $t_f = 217$ s., leading to a total of 5922 snapshots. It should be noted that for the training dataset, the fields are sampled once every 5 time steps only, allowing us to test the ability of the model to infer the in-between fields.

70% of the **SymVar** dataset is reserved for training, while the remaining portion is used to test and validate the model. Each sample of the dataset contains, as features, an encoded representation of the velocity fields, $\mathbf{u}^{n+1}$, and the temperature field T^n. The temperature field at the next desired step, T^{n+1}, is provided for every sample as a label. The notation $n+1$ denotes that the fields are computed at $t = t_n + 30\Delta t$. Other datasets are obtained and used only to test the performance of the model. Dataset **Sym1.05** contains samples from a CFD simulation where the inlets are symmetrically placed at $d_{in1} = d_{in2} = 1.05H$ from the origin. Dataset **UnSym** is obtained by placing the inlets at different distances from the origin, *i.e.* $d_{in1} = 1.3H$ and $d_{in2} = 1H$. The third dataset, **SameSide**, is collected by placing both inlets on the same side with $d_{in1} = 0.8H$ and $d_{in2} = 1.6H$. Figure 2.6 summarizes the location of the inlets for the different datasets used.

2.5 Introducing the deep learning model components

The following section aims at providing a brief introduction for the main ingredients constituting the model architecture. The core building block of any convolutional neural network is the convolutional layer. Its main role is to automatically extract higher abstract features from the input data into feature maps. This layer essentially comprises a convolution operation with a specific padding and a stride, followed by a non-linear activation. The following section will elaborate on each of the following elements.

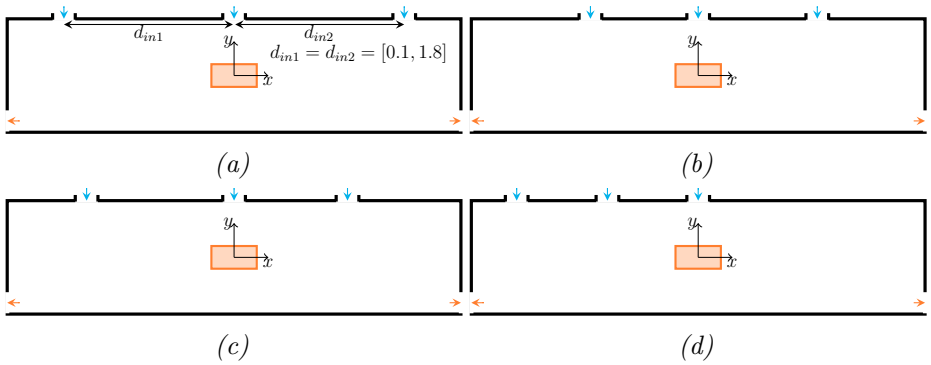


Figure 2.6: ***Gathered datasets*** *used for training and testing the model generalization capacity from different cooling setups: (a)* ***SymVar*** *Dataset, (b)* ***Sym1.05*** *Dataset, (c)* ***UnSym*** *Dataset and (d)* ***SameSide*** *Dataset. The setups shown in 5b, 5c and 5d are never seen during training.*

Convolution Operation

The input data of a convolutional layer is usually encoded on a structured grid, similar to pixels in an image, and could be described using a three dimensional tensor with a shape of $h \times w \times c$ with h, w and c respectively the tensors height, width and number of channels. Each input pixel in this data is in high correlation with its neighbouring pixels and essential information for extracting higher level abstract features is embedded in the pixel neighborhood. Thus, to avoid any loss of structural information by flattening an image into a vector and operating on it using an MLP, where the notion of locality is irrelevant, convolutional neural networks were invented [55, 56].

The constituting operation of such a layer is the convolution operation. In this operation a set of learnable filters, also known as kernels, are applied to the input data as shown in Figure 2.7a. Each kernel is usually a square shaped tensor, $k^2 \times c$, where k is known as the kernel size and c is the number of channels in the input data. Each filter convolves across the input data in a systematic manner to extract a feature map. The size of the kernel defines the receptive field of the output pixel in the extracted feature map. Usually small kernels with k less than five are preferred to avoid increasing the number of weights quadratically. At each position where the filter overlaps with the tensor, the $k^2 \times c$ local components of the tensor are flattened into a vector, and a dot product is computed between these components and the kernel weights. The resulting scalar output at every location is then stacked in a 2D tensor, while respecting the location of its receptive field, to attain the required feature map. The multiple feature maps obtained from each kernel convolution are finally concatenated in a single 3D tensor to obtain the output of the convolution layer as shown in Figure 2.7b. Finally, it should be noted that every pixel in the

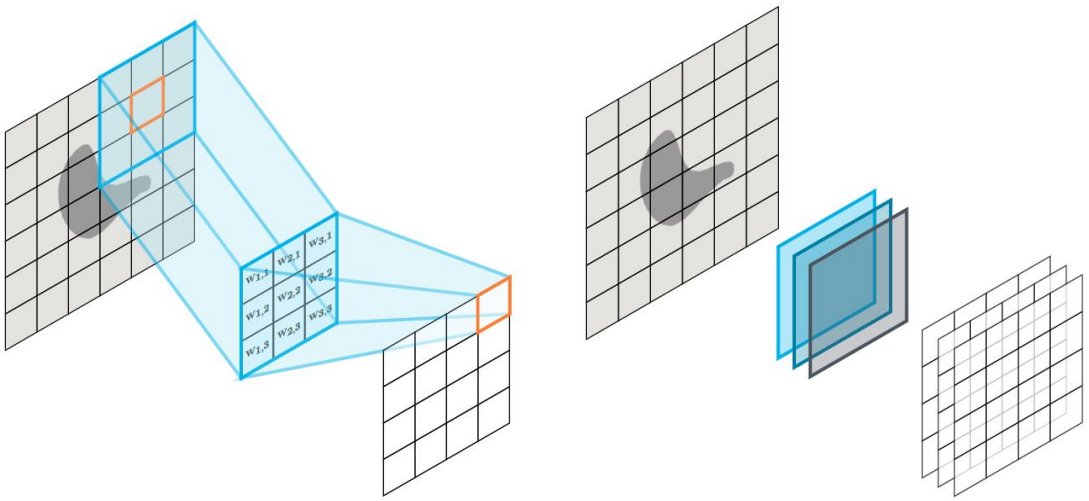


Figure 2.7: Convolution operation on a $6 \times 6 \times 1$ single channel input image. In (a) a symmetric filter with $k = 3$ is used to convolve the input image. In (b) the feature maps of three different filters are concatenated to obtain the output of the convolution layer.

output feature map could be visualized as a neuron as shown in Figure 2.8. The input of this neuron are the local tensor components while its weights are the kernel parameters.

Padding and Stride

A kernel in a convolutional layer may not be able to convolve over all the pixels available in the input image. This problem occurs near the boundaries where the size of the kernel exceeds the remaining available pixels in the image. Since the kernel outsizes the image at these boundary pixels, the information in these pixels is discarded or only considered few times compared to pixels away from the boundaries. [57–59] are few examples of works that aimed at assessing the importance of padding and its impact on the final model performance. In this manuscript, the effect of different padding schemes on the model performance is evaluated in Section 2.10.

Briefly speaking, padding is a term relevant to convolutional neural networks, and is simply surrounding the borders of the input image or hidden feature maps by an extra layer of pixels, thus extending its area. This prevents the fast shrinking of the input image in network with deep architectures and most importantly preserves information from border pixels. Various schemes of padding are excessively utilized such as same-padding, reflect-padding or symmetric-padding. In addition to these, plenty of other less utilized schemes, suited for specific applications, can be found

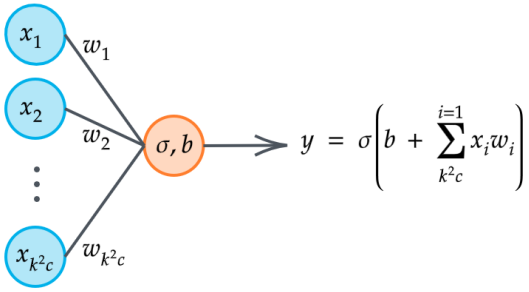


Figure 2.8: A neuron depicting a pixel, y, in the output feature map. The input of this neuron are the local tensor components x_i and the weights are the kernel parameters, w_i. The number of input pixels and trainable parameters depend on the kernel size, k, and input number of channels, c. σ and b are respectively the non linear activation function and the bias.

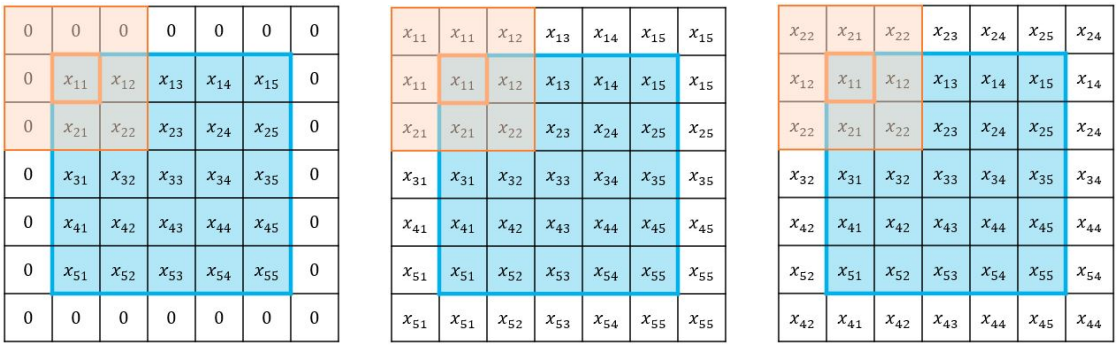


Figure 2.9: Examples of different padding schemes that preserve the size of the input tensor for a single-stride convolution with a kernel size of three. The padding schemes are: (a) same-, (b) symmetric-, and (c) reflect- padding.

by investigating the literature [60, 61]. Figure 2.9 illustrates visually the various padding schemes mentioned.

The size of the padding, along with the kernel size and convolution stride, determine the size of the output tensor. The stride, s, is a parameter used to tune the compression of the convolution operation and the overlapping of receptive fields. Thus, it specifies the rate of sliding of the kernel, *i.e.* the number of pixels by which the kernel slides after each operation, as shown in Figure 2.10. To maintain the same size of the input image in the output feature map, a single-stride convolution operation is usually accompanied with a padding of p layers on each side computed using equation 2.11,

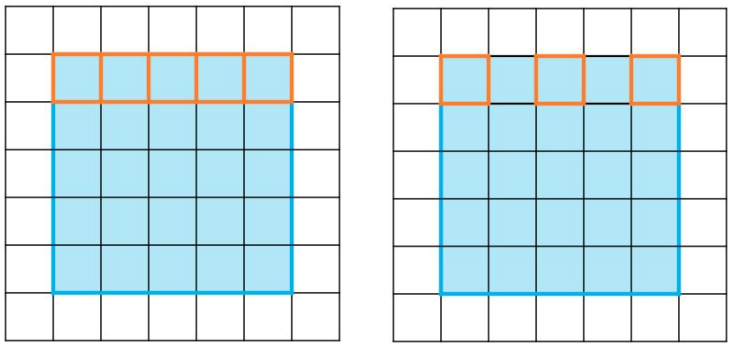


Figure 2.10: Examples showing the central of the convolving kernel on the first row of the input tensor for two different stride values: (a) $s = 1$ and (b) $s = 2$.

$$2p = k - 1 \tag{2.11}$$

Where k is the size of the kernel and $2p$ is the total number of padding layers along both sides of the image. Usually, odd-sized kernels are used to ensure the same number of padding layers is added on each side of the image and thus preserving the spatial dimensionality of the image. If the size of the kernel is even, a common practice is to pad $\lfloor k-1/2 \rfloor$ on one side and $\lceil k-1/2 \rceil$ on the other side. In general, the shape of the output tensor, $h_{out} \times w_{out} \times c_{out}$, resulting from a convolution operation with a pad p, stride s, and d_l kernels on an input tensor of shape, $h_{in} \times w_{in} \times c_{in}$, is computed as shown in Equation 2.12,

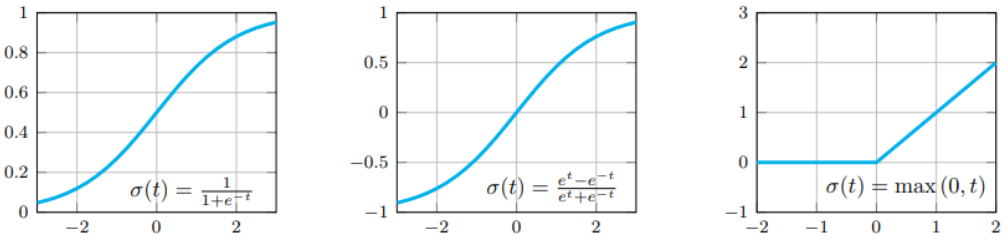
$$\begin{aligned} h_{out} &= \frac{h_{in} - k + 2p}{s} + 1, \\ w_{out} &= \frac{w_{in} - k + 2p}{s} + 1, \\ c_{out} &= d_l. \end{aligned} \tag{2.12}$$

Non-linear activation

The model's approximation capacity is intrinsically tied to its depth and the count of trainable parameters. Nevertheless, employing multiple hidden layers without applying a non-linear activation function to the output of the weighted sum is essentially equivalent to utilizing a single linear layer. Figure 2.11 illustrates this concept using a basic MLP with only one hidden linear layer. The visualization demonstrates that the entire model could be reduced to a single linear layer owing to the lack of nonlinearity.

(a) *(b)*

Figure 2.11: Two equivalent MLPs with similar model output due to absence of nonlinearity. Both models have $\mathbf{x} \in \mathbb{R}^{3\times1}$, and $y \in \mathbb{R}$ as their inputs and output, respectively. In (a), $\mathbf{h} \in \mathbb{R}^{2\times1}$ is the hidden layer activation vector, with $\mathbf{W}^{(1)} \in \mathbb{R}^{2\times3}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{1\times2}$ as the hidden and the output layer weights, respectively. In (b) both layers of the model in (a) could be replaced with a single layer having $\mathbf{W}' \in \mathbb{R}^{1\times3}$ as its weight matrix.



(a) *(b)* *(c)*

Figure 2.12: Examples of popular activation functions: (a) Sigmoid function, (b) Hyperbolic tangent function, and (c) ReLU function.

To benefit from the deep neural networks' approximation capacity when employing multiple layers of hidden neurons, it is essential to introduce a nonlinear activation function after each layer. Commonly chosen nonlinear functions include the sigmoid activation, hyperbolic tangent (tanh), and rectified linear unit (ReLU). Figure 2.12 illustrates the behavior of these functions concerning their input values. By incorporating these nonlinear activation functions, deep neural networks can maintain their expressive power and avoid collapsing into a single linear layer.

Deconvolution Layers

Convolutional neural networks typically aim to extract hierarchical features from input images to perform various tasks, including classification, object detection, or scalar prediction. To reduce the computational load and emphasize their approximation capacity for relevant information, multiple downsampling layers are incorporated into the architecture. These downsampling layers typically take the form of

pooling layers, most commonly max pooling or average pooling [62], or strided convolutions (convolution layers with a stride larger than one). However, certain models are employed for other tasks that require reconstructing the dimensions of the input data or generating an image from a lower-dimensional input. An example of such models includes convolutional autoencoders [63]. These architectures typically consist of two main parts: an encoder and a decoder. The encoder is responsible for extracting a compressed representation of the input using various downsampling layers, while the decoder reconstructs the input dimensions by upsampling the latent features.

Two common types of upsampling layers are typically used to increase the size of the encoded features and reconstruct the input dimensions: the upsample layer and the transpose convolution layer. The upsample layer is a straightforward method that expands the spatial dimensions of its input by replicating the values of the original feature map. While this method is simple and computationally efficient, it lacks any form of learning due to its reliance on deterministic nearest-neighbor interpolation. On the other hand, the Transpose convolution layer, also known as deconvolution or fractionally strided convolution, is a more complex upsampling method. It is a learnable layer that interprets the coarse input data to fill the upscaled data with more meaningful information rather than simply replicating its values.

More precisely, the transposed convolution layer perfroms upsampling and convolution at the same time. Similar to a convolutional layer, the number of kernels d_l, the size of the kernel k, the pad p and the stride s, are all hyperparameters that will impact the size of the output. The number of channels in the output tensor is similarly imposed by the number of kernels, with each channel being the convolution result between one kernel and the input. For each kernel, the tranposed convolution initially upsamples the input tensor by inserting $s-1$ zeros between every two neighboring pixels. To recover an output shape scaled by s, "same padding" is performed on the upsampled input following Equation 2.11. Finally, the upscaled output is obtained by performing a single-stride convolution on the obtained modified input. The following procedure is visualized in Figure 2.13.

Batch Normalization

Although, deep neural networks are advantageous over shallow models due to their significantly increased approximation capacity, however, this advantage comes with the trade-off of longer and more challenging training. One major source of this challenge arises from the dynamic variations in the distribution of internal hidden layer activations during the optimization process, a phenomenon known as 'internal covariate shift.' Specifically, during optimization, weight updates for a given layer are computed under the assumption that other parameters remain constant. However,

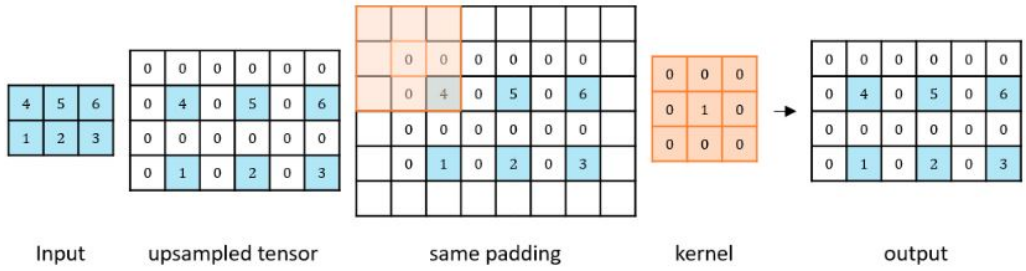


Figure 2.13: Different steps of the transposed convolution on a single channel input tensor with a kernel of size $k = 3$. The stride is set to 2, $s = 2$, and same padding is utilized to preserve the expected output dimension. The kernel weights are here specified as shown in figure although in practice they are trainable parameters.

due to the simultaneous updates of various parameters and the resulting variation in the distribution of hidden layer activations, the optimization algorithm is hindered as it chases a continuously moving target.

To address this challenge, batch normalization, has been suggested [64]. This technique standardizes the activations of a layer using its mini-batch statistics, thus, reducing the dramatic fluctuations in their distributions across the different update steps. Other than reducing the internal covariate shift, employing batch normalization also serves in smoothing the landscape of the associated optimization problem, thus, further contributing in accelerating convergence and reducing training times [65]. Finally, Batch Normalization introduces some regularization to the network due to the noise in the computed mini-batch statistics, thus also serving in reducing the generalization error of the model.

In essence, batch normalization computes the mean and variance for each activation within a hidden layer across the mini-batch. These statistics are then utilized to standardize the layer's activations. Moreover, to preserve the network's representation power, additional trainable parameters, β and γ, are introduced. These parameters automatically shift and scale the standardized layer activations, as depicted in Equation 2.13.

$$\mathbf{h}^{(l)}_{norm} = \frac{\gamma^{(l)} \times (\mathbf{h}^{(l)} - \mu^{(l)})}{\sqrt{{\sigma^{(l)}}^2 + \epsilon}} + \beta^{(l)}, \tag{2.13}$$

where $\mathbf{h}^{(l)}$, $\mu^{(l)}$, and ${\sigma^{(l)}}^2$ are the l^{th} layer activations, computed mean and variance, respectively. ϵ is a small constant for numerical stability, while $\gamma^{(l)}$ and $\beta^{(l)}$ are the introduced learnable parameters.

Finally, a moving average of the computed mean and standard deviation, $\mu^{(l)}_{mov}$

and $\sigma_{mov}^{(l)}$, are also computed across the various update steps, as shown in Equation 2.14, to allow consistent standardization during inference.

$$\begin{aligned} \mu_{mov}^{(l)} &= \mu_{mov}^{(l)} \times m + \mu^{(l)}(1-m), \\ \sigma_{mov}^{(l)} &= \sigma_{mov}^{(l)} \times m + \sigma^{(l)}(1-m), \end{aligned} \tag{2.14}$$

where m is a momentum hyperparameter while $\mu^{(l)}$ and $\sigma^{(l)}$ are the mini-batch specific statistics.

2.6 Model Architecture

The task of the considered neural network is to model the scalar transport equation, shown in (2.1). Thus, instead of solving both the NS equations and the transport equation using traditional numerical methods, a deep learning regression model will be responsible for the resolution of the scalar field F. The architecture of the model used to accomplish the above-mentioned task is an auto-encoder-like end-to-end network [19], presented in figure 2.14. While auto-encoders are designed to generate a compressed representation of their input [66–69], in the suggested model, the output of the model is a distinct field. Similar architectures have been previously employed to infer other physical fields for fluid flow problems [70, 71]. The model consists mainly of three components: encoder, bottleneck, and decoder. The encoder gathers the information from the input fields and compresses them to a reduced dimension representation denoted as the bottleneck (also known as the latent space). The spatial reduction, in the encoder, is handled using downsampling convolutional layers with a stride of two. These layers are preceded by a single-stride convolutional layer and followed by batch normalization [64], and ReLU non-linear activation [72]. This pattern of layers is repeated $N = 4$ times as seen in the upper branch of figure 2.14. The encoder is followed by a decoder responsible for inferring the required scalar field from the compressed latent space. The building block of the decoder consists of a single-stride convolutional layer, followed by a deconvolutional layer responsible for the upsampling of the input to a larger dimensional space, in addition to a non-linear activation function. This pattern of layers is repeated the same number of times as its contrast downsampling pattern in the encoder. The decoder is represented by the bottom branch in figure 2.14.

It should be mentioned that the physical fields, fed to the model, are priorly normalized. This standard preprocessing operation unifies the scale of all the fields and thus ensures a faster training process as is detailed in appendix 2.9. Moreover, the convolutional layers present throughout the model are preceded by a padding layer following the reflect scheme. Using such a scheme preserves the physical statistical distribution of the feature maps without introducing artifacts near the boundaries

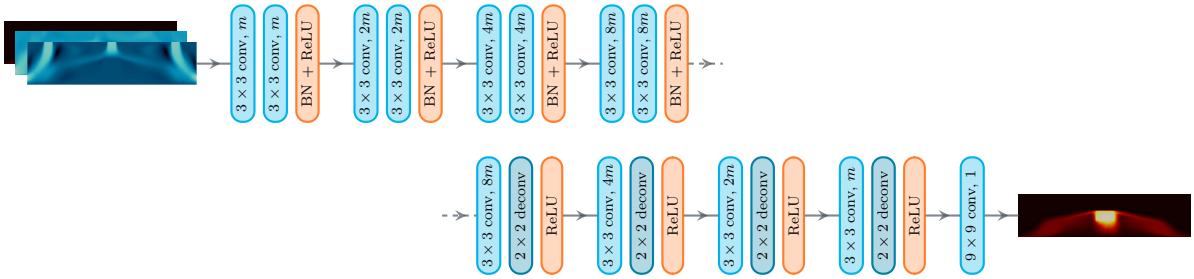


Figure 2.14: ***Deep learning model architecture*** *used to infer the required temperature given temperature, at previous time step, and velocity components as feature inputs.*

[58]. Appendix 2.10 supports our choice of reflect padding by comparing the performance of the model with different schemes. The resolution of the discretization in the encoding meshes is specified while keeping in mind the available memory to store the data and train the model. Finer structured meshes would allow the model to execute superior predictions but at the cost of larger memory requirements and longer training times. The aspect ratio of the encoding mesh is chosen similar to that of the computational domain, *i.e.* $\frac{n_y}{n_x} \approx \frac{\mathrm{H}}{4H}$, where n_x and n_y are the number of nodes in the x and y direction respectively. This choice of aspect ratio is made to ensure the feature maps are not biased toward a spatial dimension despite what consequences would have resulted from such bias. Moreover, the input encoding mesh dimensions are selected while assuring the downsampling convolutional layers present in the encoder do not violate equation (2.15):

$$h_i = s \times (h_o - 1) + k - 2p, \tag{2.15}$$

where h_i and h_o are respectively the input and output dimension size of the image and s, k and p are respectively the kernel stride, size and padding of the considered layer. Violating equation (2.15) for downsampling convolutional layers leads to uneven padding operations, which may deteriorate the performance of the model [58]. It should be noted that the next possible encoding mesh size respecting equation (2.15) and maintaining aspect ratio close to 1/4, will possess 21,588 additional pixels which require around 20% supplementary storage and memory space for every field.

2.7 Training

The inferring quality of the deep learning model described in setion 2.6 depends on the trainable set of weights and biases. These parameters define the model and its ability to infer the required scalar fields. After randomly initializing them with

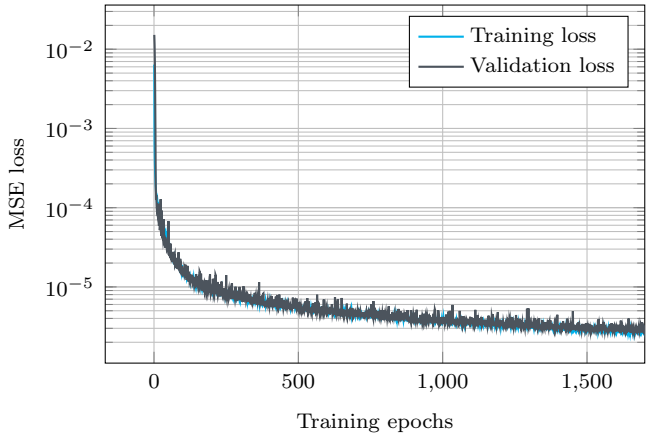


Figure 2.15: ***Training curve*** *showing the evolution of the MSE loss, for both the training and validation dataset, over the number of epochs.*

Gaussian distribution, the optimal set of parameters is obtained by minimizing a certain loss function quantifying the error between the inferred fields and the reference ones. In our model, 2 937 025 parameters are optimized, using Adam optimizer [73], with a learning rate equal to 10^{-5}. The optimizer iteratively minimizes the mean squared error (MSE), computed between the reference fields and the predicted ones and defined as in (2.16):

$$\mathcal{L} = \frac{1}{N_{\text{samples}} \times n_x \times n_y} \sum_{i=1}^{N_{\text{samples}}} \sum_{r=1}^{n_y} \sum_{c=1}^{n_x} \left(F_{r,c}^{i} - \tilde{F}_{r,c}^{i} \right)^2 \tag{2.16}$$

where N_{samples} is the number of samples for every optimization step, n_x and n_y are respectively the number of nodes in the x and y direction for the output encoding mesh, *i.e.* $n_x = 176$ and $n_y = 656$, $F_{r,c}^{i}$ is the true scalar value at a specific location for a sample i and $\tilde{F}_{r,c}^{i}$ is the value inferred by the model. The training is performed using the TensorFlow API [74], on an Nvidia Tesla V100 GPU. The optimization is performed using mini-batches of size 32 for a total of 1651 epochs, requiring approximately 21 hours. The training is terminated using the early stopping criterion after a satisfying accuracy is attained and before the performance on the validation dataset starts to deteriorate, as can be seen in figure 2.15.

2.8 Results and discussion

Although the temperature fields are inferred on the whole computational domain, the main interest of any industrial practitioner will be biased toward the temperature

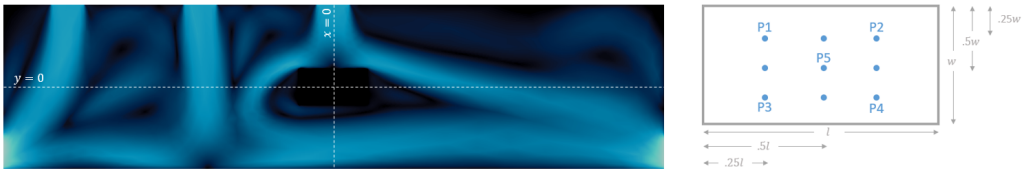


Figure 2.16: ***Location of sampling probes****. On left, probes that capture the variation of the field along a specific line. On right, probes inside working object to capture variation over time.*

fields within the object being cooled. To focus more on this region, temperature probes are installed at various positions within the object as is demonstrated in figure 2.16. These probes allow plotting the evolution of the temperature field over the simulation time. Plots of the variation of the temperature field over predefined lines in the computational domain will also be obtained. Moreover, the plot of the temperature field in the region of interest, and the computation of a normalized error, will also allow the evaluation of the model on various data sets with different flow characteristics.

First, the input fields from a simulation with symmetrical inlets, placed at $d_{in1} = d_{in2} = 1.5H$, will be provided for the model to assess its interpolation ability over time. This setup is partially included in the **SymVar** dataset used to train the model where only 12% of the total 1971 time steps are previously seen during training and no snapshot was provided at a step exceeding 2300. This dataset will be referred to as **Sym1.5** in the figures and tables used to assess the model performance. The model extrapolation ability over time was tested for 200 extra time steps. Figure 2.17 shows the plots of temperature at various probes where almost exact similar fields are inferred by the model over the whole range of provided time steps. If the model is required to predict for further time intervals, the time domain of the provided training dataset should be expanded to avoid divergence in model inferrings'.

The generalization capacity of the model to completely unseen inlet setups is also evaluated. Three data sets were created for this purpose: **Sym1.05, UnSym**, and **SameSide**. The first dataset, **Sym1.05**, with symmetrical inlets, placed at $d_{in1} = d_{in2} = 1.05H$, has similar flow characteristics as the dataset used to train the model but with different positions of inlets, not seen before by the model. The remaining two data sets, **UnSym** and **SameSide**, have both their inlets placed at an unsymmetrical location with respect to the origin. However, the **SameSide** dataset, with both inlets placed on the left of the object, shows a very large change in the characteristics of the flow compared to the other sets. Figure 2.18 shows the flow fields resulting from all three different setups.

Figure 2.19 shows the line plots of temperature across specific x and y values shown in figure 2.16. The line plots are obtained at the same single time step for all

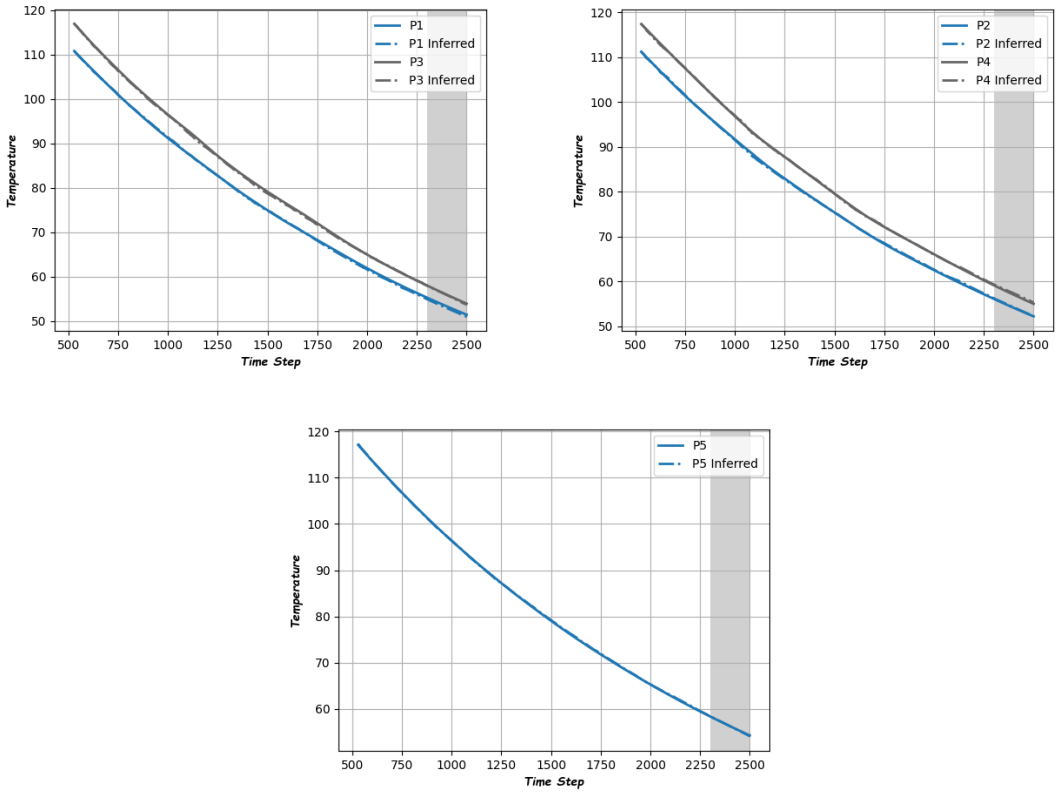


Figure 2.17: ***Plot of computed and inferred field at various probe locations for setup with symmetrical inlets at*** $d_{in1} = d_{in2} = 1.5$***.*** *The predicted fields are plotted with dashed lines, while the computed ones are plotted with solid lines. The shaded region, shown after time step* 2300*, represents the region where extrapolation over time occurs.*

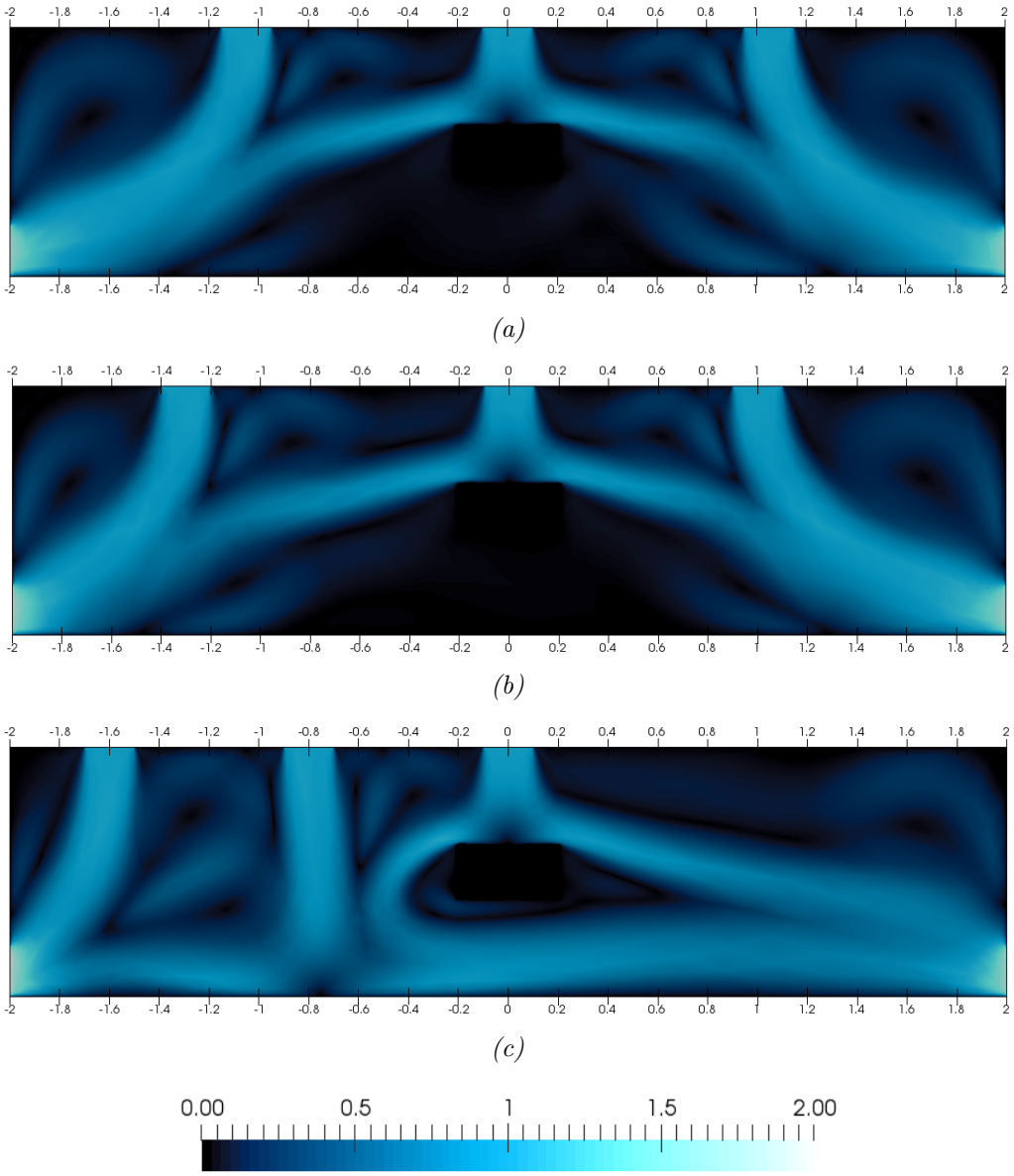


Figure 2.18: ***Velocity magnitude*** *from the different flow setups used to test the model generalization capacity. Each setup is for a specific dataset: (a)* ***Sym1.05*** *dataset, (b)* ***UnSym*** *dataset, and (c)* ***SameSide*** *dataset.*

sets. For the first three data sets, **Sym1.5**, **Sym1.05**, and **UnSym**, the model was able to accurately infer the variation of temperature over the whole computation domain. This can be seen by the overlapping plots of the inferred field and the label field obtained from a CFD simulation. For the **SameSide** dataset, although a larger deviation between the fields can be observed, mainly in the fluid domain where the flow characteristics drastically change, the model is still able to infer the general trend of the variation of the field. To quantify the quality of the inferred fields over the different inlet setups, the l_2 normalized error, defined in equation (2.17), is computed for each sample:

$$error = \frac{||T^n - \tilde{F}^n||}{||T^n||}, \tag{2.17}$$

where $||\cdot||$ is the l_2 norm of the physical field. T^n is the true scalar field at time step n, and $\tilde{F}^n$ is the model inferred one.

The model performance is approximately similar over the **Sym1.5, Sym1.05**, and **UnSym** sets with an average error ranging from 1.27×10^{-2} to 1.54×10^{-2}. The error increases for the **SameSide** set, as expected due to a large change in flow characteristics, with a maximum not exceeding 1.35×10^{-1}. All results are summarized in table 2.1 along with the time step that corresponds to the maximum error in each setup. Moreover, the error in the domain of the object is also computed neglecting the surrounding domain, and shown in the last row of Table 2.1. This error is an order of magnitude lower than its counterpart due to fewer changes in the object domain with the variation in the positions of the inlets. The error fields maps, computed using the absolute error normalized with the true temperature in Kelvin, are shown in figure 2.23. Moreover, a visual comparison between the inferred fields and their computed counterpart in the region of interest, *i.e.* the domain of the object, is shown in figure 2.22. The obtained fields are almost identical except for the **SameSide** set where deviation from the true field occurs but maintains the same range of temperature values.

Finally, the trained network, modeling the scalar transport equation, is incorporated in the solution loop as shown in figure 2.20. Initially, the CFD solver is utilized to resolve both governing equations, *i.e.* the Navier-Stokes and the transport equations, until all flows are well established. Starting $t = 50$ sec, the trained deep learning model can be used to infer the scalar field. The temperature, T^n, along with the velocity fields obtained from resolving the Navier-Stokes equation, u^{n+1} and v^{n+1}, are interpolated into the required encoding mesh of the model. The model will infer the temperature field T^{n+1}, at the next model time step, $t^{n+1} = t^n + 30\Delta t_{solver}$. The inferred temperature field is recycled by encoding it and feeding it back to the model for multiple times f. After multiple predictions, the traditional transport solver is utilized to redirect the solution before proceeding again with the model.

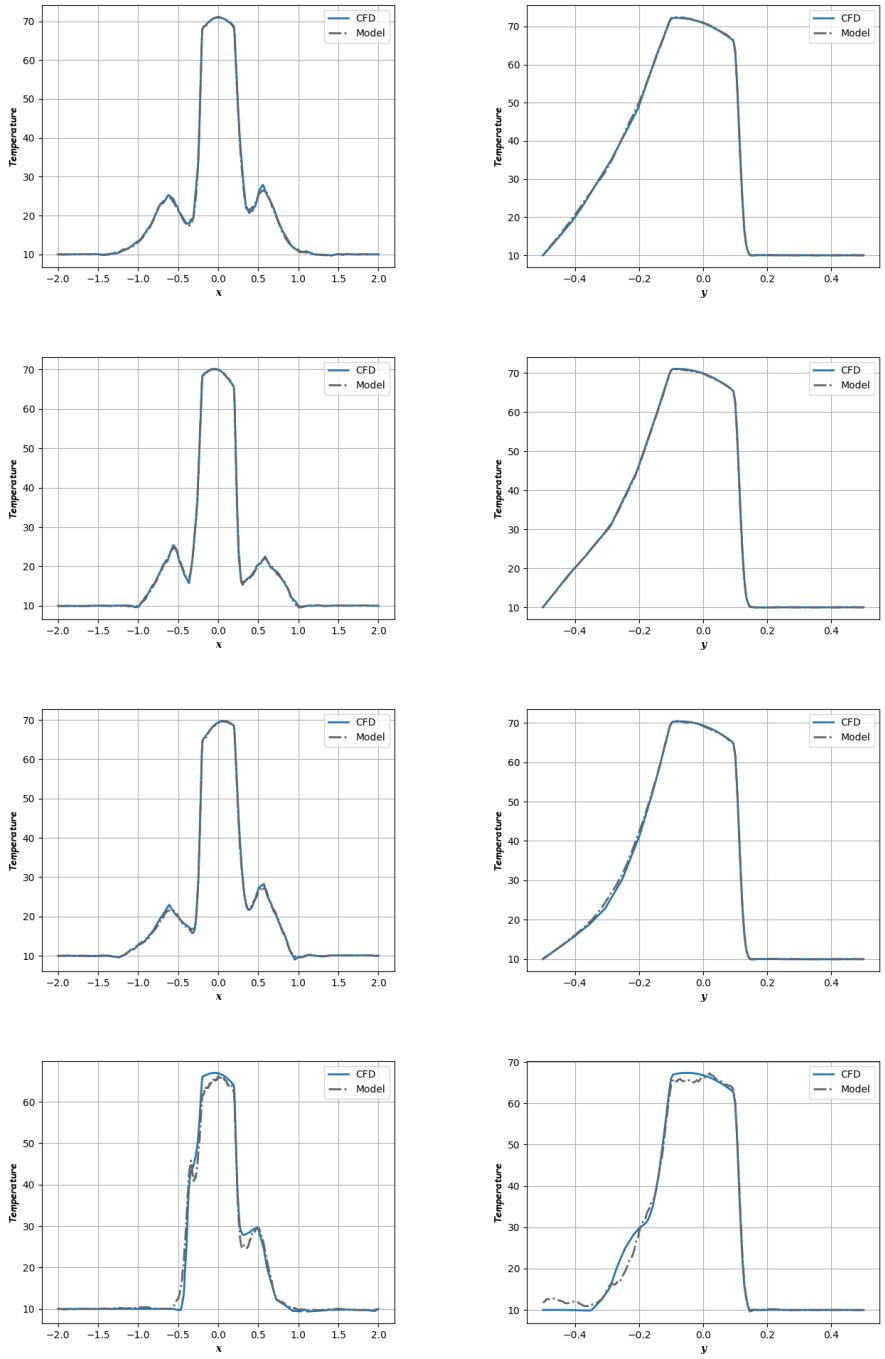


Figure 2.19: ***Line plot of temperature variation over a certain direction.*** *The temperature field, both computed and inferred, at the same time step,* $TS = 1750$*, is plotted along both lines,* $x = 0$ *and* $y = 0$*, for different inlet setups. Every row of plots corresponds to a separate dataset. The order of sets is:* ***Sym1.5, Sym1.05, UnSym****, and* ***SameSide****.*

Dataset	Sym1.5	Sym1.05	UnSym	SameSide
Average	1.27×10^{-2}	1.54×10^{-2}	1.33×10^{-2}	1.09×10^{-1}
Maximum	1.92×10^{-2}	2.35×10^{-2}	1.66×10^{-2}	1.35×10^{-1}
Time Step	1575	567	1745	637
Focused Average	2.6×10^{-3}	3.09×10^{-3}	3.58×10^{-3}	3.23×10^{-2}

Table 2.1: ***Summary of obtained error.*** *The average of the l_2 normalized error is computed across all samples. The time step corresponds to the step at which the maximum error is found. The focused average is computed only over the object domain.*

The model predicted field, T^{n+f}, is interpolated back to the unstructured mesh to enable the finite element transport equation solver to operate on. The input required by the traditional solver to resolve the field at the next time step is identical to that of the model. The solver inferred field, T^{n+f+1}, can then be refed to the deep learning model to advance in the solution. This sequence is repeated until the whole time domain is spanned. The number of times the model is used consecutively f, affects the quality of the solution. Large f values allow advancing with larger time steps throughout the required time interval but may lead to less accurate solutions.

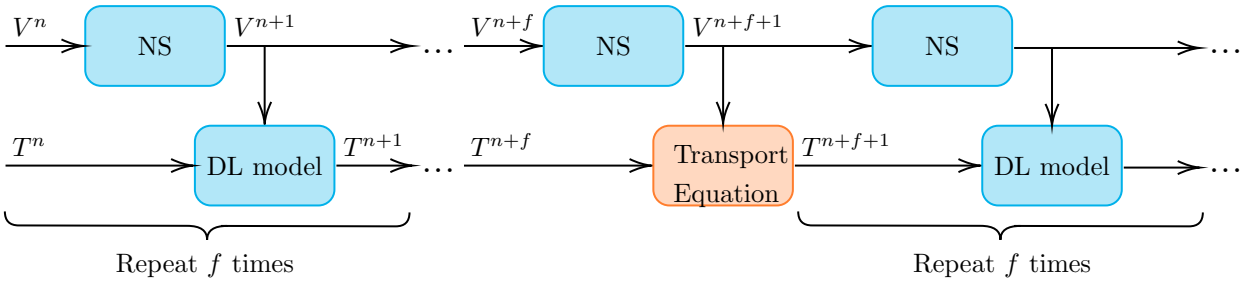
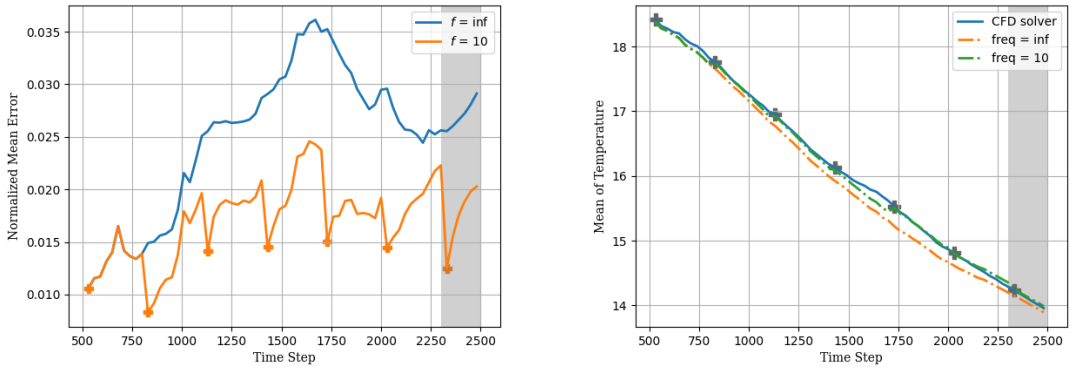


Figure 2.20: ***Implementing the deep learning model in the solution loop.***

The total time required for the industrial-level CFD solver to resolve both governing equations for $t \in [50, 250]$ sec is 491.92 seconds. Around 22% of this computational time is required for only resolving the scalar temperature field T. To explore the potential of our model, the value of f is set to infinity, meaning that the temperature field across the whole time domain will be resolved by the deep learning model without referring back to the scalar finite element solver. Knowing that $\Delta t_{\text{model}} = 30 \times \Delta t_{\text{solver}}$, the model resolves the temperature field across the whole computation domain in 8.71s., *i.e.* around 12 times faster than the finite element scalar equation solver. At last, the l_2 normalized error for the predicted fields on the **Sym1.5** dataset, with f set to infinity, maintains an average of 2.49×10^{-2} and does

not exceed the 3.61×10^{-2} level as can be seen in figure 2.21a. The mean temperature field over the whole spatial domain is computed and compared to that obtained with the CFD solver as shown in figure 2.21b. Moreover, the results obtained with f set to 10 are also shown in both figures, 2.21a and 2.21b.



(a) *(b)*

Figure 2.21: ***Implementation results for different values of*** *f with (a) showing the plot of error evolution and (b) the plot of the average temperature evolution. In both figures, the cross sign is shown for the prediction obtained after the traditional solver is used.*

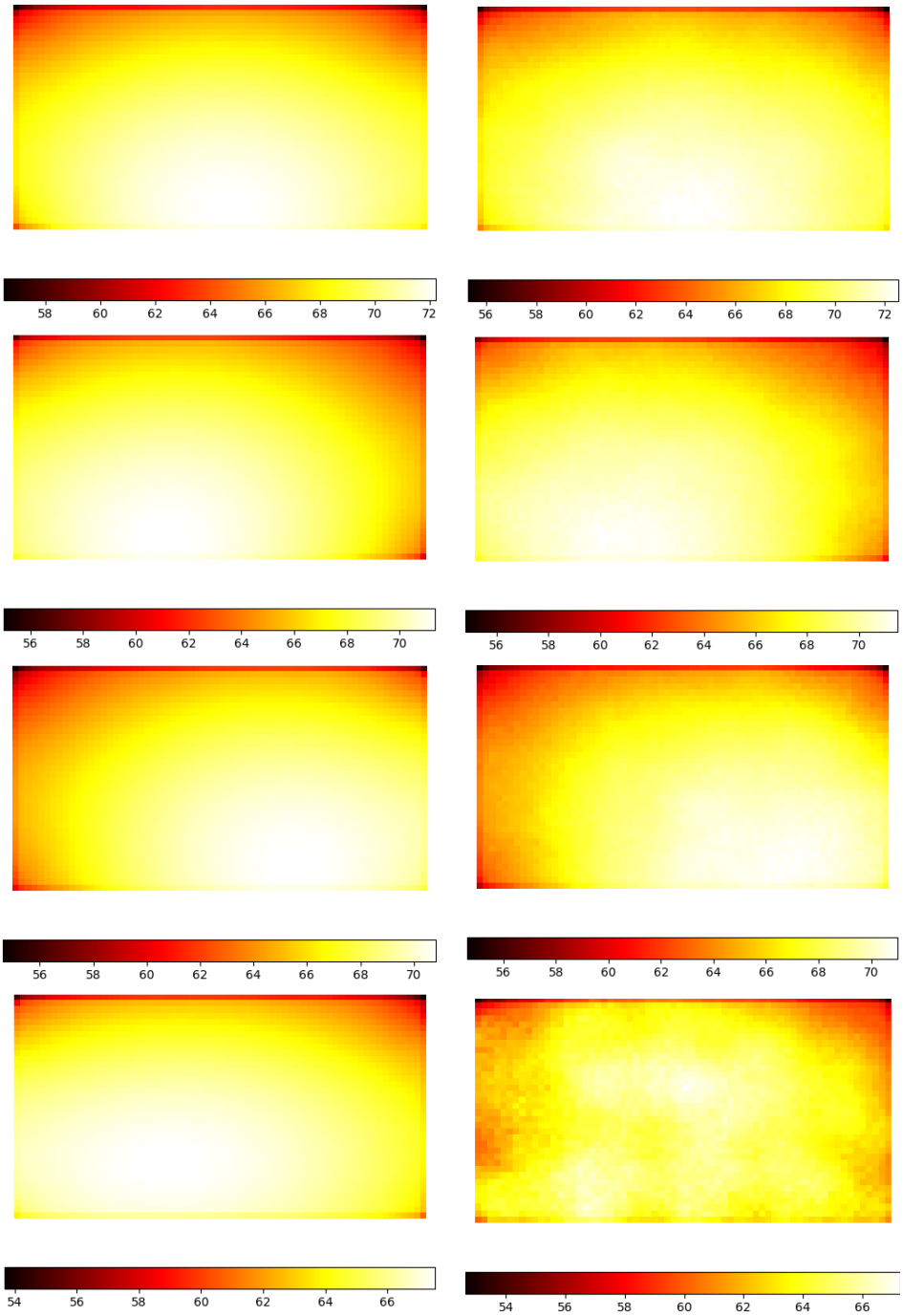


Figure 2.22: ***Temperature field in object domain****. The temperature field, computed on left and inferred on right, at the same time step, $TS = 1750$, is plotted in the region of interest, for different inlet setups. Every row of plots corresponds to a separate dataset. The order of sets is:* ***Sym1.5, Sym1.05, UnSym****, and* ***SameSide****.*

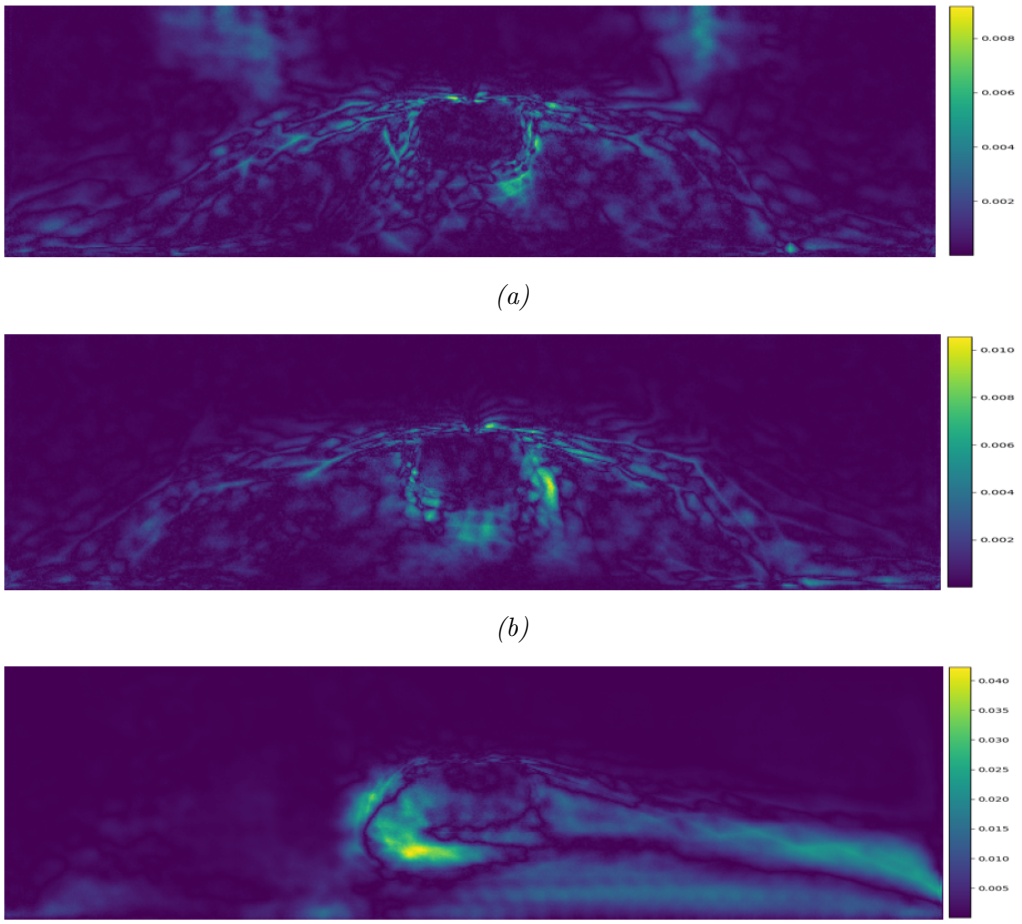


Figure 2.23: ***Error field maps*** *obtained for the different inlet setups of the (a)* ***Sym1.05*** *dataset, (b)* ***UnSym*** *dataset and the (c)* ***SameSide*** *dataset.*

2.9 Exploring performance for different scaling methods

For deep learning approaches using gradient descent to optimize their parameters, it is suggested to maintain the input features at a similar scale. This practice is motivated by a faster convergence of the optimizer toward the requested minimum [75]. Moreover, scaling the target fields avoids encountering the exploding gradient problem. Multiple methods exist to scale the fields such as the min-max scaler, the standard scaler, the robust scaler, etc. The most common methods used are the min-max scaler, also known as field normalization, and the standard scaler (*i.e.* field standardization). The choice of the suitable scaling method depends on multiple factors and may vary between fields. For example, different resources suggest normalizing in case the field distribution is not Gaussian, and others suggest stan-

dardizing if outliers exist. In the present work, the temperature field is normalized, given its non-Gaussian distribution, and the performance of the model is evaluated for both scaling methods available for the velocity field. Moreover, the model is also trained with raw data as a reference for both scaling methods. Equation (2.18) is used to normalize a field, while equation (2.19) is used to standardize it:

$$X' = \frac{X - min}{max - min}, \tag{2.18}$$

$$X' = \frac{X - \mu}{\sigma}, \tag{2.19}$$

where min, max, μ, and σ are respectively the minimum, maximum, mean, and standard deviation of the field X, obtained based on the training dataset.

The plot of the loss curve over the validation dataset will be used to compare among the different scaling methods. Figure 2.24 shows the evolution of the validation loss for three similar models trained with fields scaled using different methods for 600 epochs. The inferred temperature field is restored to its original scale before computing the loss to allow the comparison between the different models.

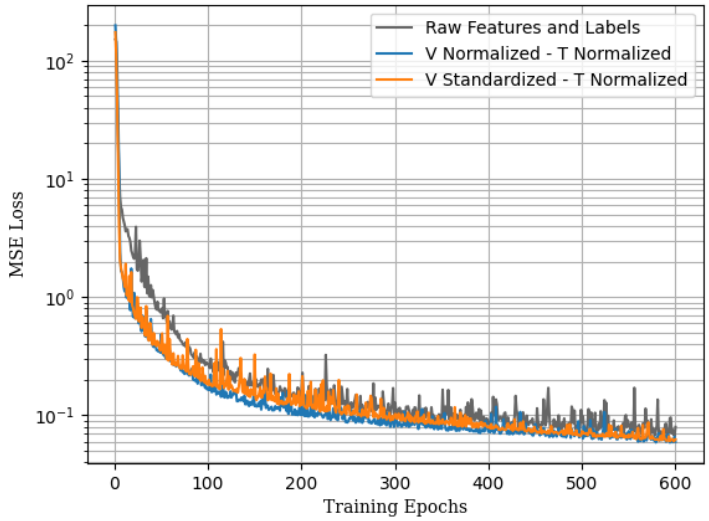


Figure 2.24: Validation loss for different scaling methods.

By inspecting the above figure, the advantages of scaling fields before training are obvious since both models with scaled data encounter a faster decrease in the loss than the one with the raw data. Approximately similar performance is observed for both scaling methods, however, we chose to normalize both fields rather than

standardizing velocity and normalizing temperature since slightly lower losses are obtained in some training epoch intervals. Using scaled data, the loss at epoch 600 decreases to 5.96×10^{-2} for normalizing both fields, and 6.25×10^{-2} for standardizing velocity and normalizing temperature, whereas, for raw data, a higher loss, 7.97×10^{-2}, is obtained.

2.10 Exploring performance for different padding methods

To choose the appropriate padding scheme, the model was trained with different padding methods. The investigation was limited to only three schemes and the validation loss, obtained after training is terminated using the early stopping criteria, is used to compare between the different schemes. Figure 2.25 shows the loss obtained during training over the validation dataset for the three padding schemes: same, reflect, and symmetric padding. Although the performance is slightly affected by the scheme used and all three curves look similar, the model trained with reflect padding scheme returned the lowest mean squared error with a value of 2.6420×10^{-6}, compared to 2.78×10^{-6} and 2.88×10^{-6} for same padding and symmetric padding, respectively.

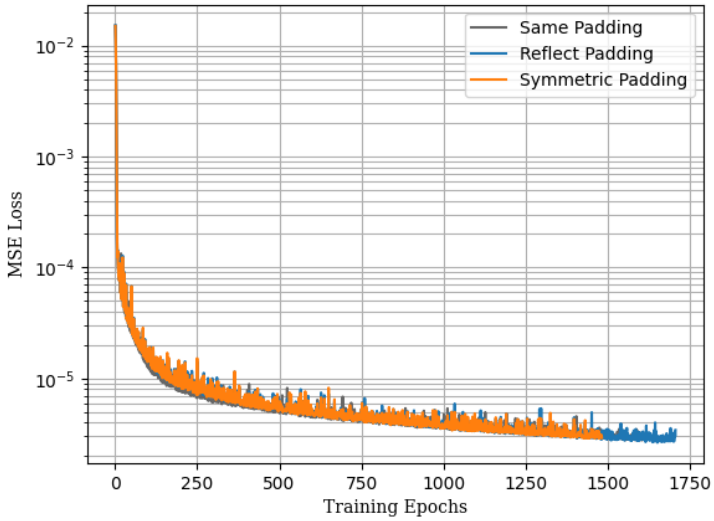


Figure 2.25: Validation loss for different padding methods.

Table 2.2 summarizes the obtained losses over both the training and the validation dataset. Based on the shown losses, the model trained with reflect padding and obtained at epoch 1650 will be used to predict the temperature field for different setups as detailed in section 2.8. The slightly larger training loss obtained for the

same and reflect padding, compared to that of validation, can be accounted for the stochastic nature of training along with the fact that the training loss is computed half an epoch earlier than the validation loss.

Padding	Same	Symmetric	Reflect
Training	3.02e-6	2.87e-6	2.69e-6
Validation	2.78e-6	2.88e-6	2.64e-6
Epoch	1553	1427	1651

Table 2.2: Summary of obtained losses for different padding schemes over both the training and the validation dataset

2.11 Conclusion

In this work, a deep learning model, based on a convolutional network with an autoencoder architecture, is employed for inferring a certain scalar field. The model was trained for predicting the temperature field in a conjugate heat transfer problem and thus models the scalar transport equation in the coupled system of equations. Although the model was trained on samples obtained from a few different symmetrical cooling setups, it was able to accurately infer the fields for new inlet positions, whether symmetrical or not, with a maximum error of 1.35×10^{-1}. Also, the ability of the model to interpolate in the time-domain, permitted training on multiple cooling setups, with a few snapshots for each setup, distributed on this domain. Moreover, the model manages to span the whole time domain independently, *i.e.* without referring back to the scalar transport equation solver, 12 times faster than the industrial level CFD solver, and returns reliable predictions for the evolution of the temperature field with an average l_2 normalized error equals to 2.49×10^{-2}. These results motivate us to exploit the capacity of deep learning models, with similar architecture, for assisting in the simulation of numerous number of physical problems governed by a similar set of equations where the Navier–Stokes equations are coupled with a scalar transport equation responsible for the computation of a certain physical field.

Bibliography

[1] J. Blazek, Computational fluid dynamics: principles and applications, Butterworth-Heinemann, 2015. 32

[2] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, in: 30th aerospace sciences meeting and exhibit, 1992, p. 439. 32

[3] P. Huang, J. Bardina, T. Coakley, Turbulence modeling validation, testing, and development, NASA technical memorandum 110446 (1997) 147. 32

[4] W. Jones, B. E. Launder, The prediction of laminarization with a two-equation model of turbulence, International journal of heat and mass transfer 15 (2) (1972) 301–314.

[5] D. C. Wilcox, Formulation of the k-w turbulence model revisited, AIAA Journal 46 (11) (2008) 2823–2838. arXiv:https://doi.org/10.2514/1.36541, doi:10.2514/1.36541.
URL https://doi.org/10.2514/1.36541 32

[6] E. Hachem, G. Jannoun, J. Veysset, M. Henri, R. Pierrot, I. Poitrault, E. Massoni, T. Coupez, Modeling of heat transfer and turbulent flows inside industrial furnaces, Simulation Modelling Practice and Theory 30 (2013) 35–53. doi:https://doi.org/10.1016/j.simpat.2012.07.013.
URL https://www.sciencedirect.com/science/article/pii/S1569190X12001219 32

[7] E. Hachem, T. Kloczko, H. Digonnet, T. Coupez, Stabilized finite element solution to handle complex heat and fluid flows in industrial furnaces using the immersed volume method, International Journal for Numerical Methods in Fluids 68 (1) (2012) 99–121. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.2498, doi:https://doi.org/10.1002/fld.2498.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2498 32, 34

[8] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing (2018). arXiv:1708.02709. 32

[9] S. Yang, Y. Wang, X. Chu, A survey of deep learning techniques for neural machine translation (2020). arXiv:2002.07526. 32

[10] L. Deng, G. Hinton, B. Kingsbury, New types of deep neural network learning for speech recognition and related applications: an overview, in: 2013 IEEE

International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 8599–8603. doi:10.1109/ICASSP.2013.6639344. 32

[11] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: A brief review, Computational intelligence and neuroscience 2018 (2018). 32

[12] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707. doi:https://doi.org/10.1016/j.jcp.2018.10.045.
URL https://www.sciencedirect.com/science/article/pii/S0021999118307125 32

[13] T. Daniel, F. Casenave, N. Akkari, D. Ryckelynck, Model order reduction assisted by deep neural networks (rom-net), Advanced Modeling and Simulation in Engineering Sciences 7 (2020) 1–27. 32

[14] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks (2018). arXiv:1806.01261. 32

[15] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P. W. Battaglia, Learning mesh-based simulation with graph networks (2021). arXiv:2010.03409.

[16] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. W. Battaglia, Learning to simulate complex physics with graph networks (2020). arXiv:2002.09405. 32

[17] Y. Bar-Sinai, S. Hoyer, J. Hickey, M. P. Brenner, Learning data-driven discretizations for partial differential equations, Proceedings of the National Academy of Sciences 116 (31) (2019) 15344–15349. doi:10.1073/pnas.1814058116.
URL http://dx.doi.org/10.1073/pnas.1814058116 32

[18] C. Ghnatios, G. El Haber, J.-L. Duval, M. Ziane, F. Chinesta, Artificial intelligence based space reduction of structural models, ESAFORM 2021 (04 2021). doi:10.25518/esaform21.2004. 32

[19] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by error propagation, Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985). 33, 51

[20] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, arXiv preprint arXiv:1409.3215 (2014). 33

[21] Y. Mesri, H. Digonnet, T. Coupez, Advanced parallel computing in material forming with cimlib, European Journal of Computational Mechanics 18 (7-8) (2009) 669–694. arXiv:https://doi.org/10.3166/ejcm.18.669-694, doi:10.3166/ejcm.18.669-694.
URL https://doi.org/10.3166/ejcm.18.669-694 33

[22] J. Kim, D. Kim, H. Choi, An immersed-boundary finite-volume method for simulations of flow in complex geometries, Journal of computational physics 171 (1) (2001) 132–150. 33

[23] C. Gruau, T. Coupez, 3d tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric, Computer Methods in Applied Mechanics and Engineering 194 (48-49) (2005) 4951–4976. 33

[24] T. J. Hughes, G. R. Feijóo, L. Mazzei, J.-B. Quincy, The variational multiscale method—a paradigm for computational mechanics, Computer methods in applied mechanics and engineering 166 (1-2) (1998) 3–24. 33, 38

[25] R. Codina, Stabilization of incompressibility and convection through orthogonal sub-scales in finite element methods, Computer Methods in Applied Mechanics and Engineering 190 (13) (2000) 1579–1599. doi:https://doi.org/10.1016/S0045-7825(00)00254-1.
URL https://www.sciencedirect.com/science/article/pii/S0045782500002541

[26] Y. Bazilevs, V. Calo, J. Cottrell, T. Hughes, A. Reali, G. Scovazzi, Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows, Computer Methods in Applied Mechanics and Engineering 197 (1) (2007) 173–201. doi:https://doi.org/10.1016/j.cma.2007.07.016.
URL https://www.sciencedirect.com/science/article/pii/S0045782507003027 33

[27] E. Hachem, H. Digonnet, E. Massoni, T. Coupez, Immersed volume method for solving natural convection, conduction and radiation of a hat-shaped disk inside a 3d enclosure, International Journal of numerical methods for heat & fluid flow (2012). 33, 39

[28] E. Hachem, S. Feghali, R. Codina, T. Coupez, Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation, International Journal for Numerical Methods in Engineering 94 (9) (2013) 805–825. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.4481, doi:https://doi.org/10.1002/nme.4481.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.4481 33, 39

[29] S. R. Allmaras, F. T. Johnson, Modifications and clarifications for the implementation of the spalart-allmaras turbulence model, in: Seventh international conference on computational fluid dynamics (ICCFD7), Vol. 1902, Big Island, HI, 2012. 34

[30] S. Osher, J. A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations, Journal of computational physics 79 (1) (1988) 12–49. 35

[31] S. Osher, R. Fedkiw, Level set methods and dynamic implicit surfaces, Vol. 153, Springer Science & Business Media, 2006.

[32] J. A. Sethian, Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science, Vol. 3, Cambridge university press, 1999.

[33] D. Enright, R. Fedkiw, J. Ferziger, I. Mitchell, A hybrid particle level set method for improved interface capturing, Journal of Computational physics 183 (1) (2002) 83–116. 35

[34] E. W. Weisstein, Heaviside step function, https://mathworld. wolfram. com/ (2002). 35

[35] S. V. Patankar, Numerical heat transfer and fluid flow, CRC press, 2018. 36

[36] S. V. Patankar, A numerical method for conduction in composite materials, flow in irregular geometries and conjugate heat transfer, in: International Heat Transfer Conference Digital Library, Begel House Inc., 1978. 36

[37] L. Formaggia, S. Perotto, Anisotropic error estimates for elliptic problems, Numerische Mathematik 94 (1) (2003) 67–92. 36

[38] J. Hoffman, C. Johnson, Adaptive finite element methods for incompressible fluid flow, in: Error estimation and adaptive discretization methods in computational fluid dynamics, Springer, 2003, pp. 97–157. 36

[39] C. Gruau, T. Coupez, 3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric, Computer Methods in Applied Mechanics and Engineering 194 (48-49) (2005) Pages 4951–4976. doi:10.1016/j.cma.2004.11.020.
URL https://hal-mines-paristech.archives-ouvertes.fr/hal-00517639 36

[40] T. Coupez, Metric construction by length distribution tensor and edge based error for anisotropic adaptive meshing, Journal of computational physics 230 (7) (2011) 2391–2405. 36

[41] D. Boffi, F. Brezzi, M. Fortin, et al., Mixed finite element methods and applications, Vol. 44, Springer, 2013. 38

[42] L. P. Franca, A. Nesliturk, On a two-level finite element method for the incompressible navier–stokes equations, International Journal for Numerical Methods in Engineering 52 (4) (2001) 433–453. 38

[43] A. I. Nesliturk, Approximating the incompressible Navier-Stokes equations using a two-level finite element method, University of Colorado at Denver, 1999. 38

[44] F. Brezzi, L. Franca, T. Hughes, A. Russo, $b = \alpha g$, Computer Methods in Applied Mechanics and Engineering 145 (3) (1997) 329–339. doi:https://doi.org/10.1016/S0045-7825(96)01221-2.
URL https://www.sciencedirect.com/science/article/pii/S0045782596012212 38

[45] F. Brezzi, M. Fortin, Mixed and hybrid finite element methods, Vol. 15, Springer Science & Business Media, 2012.

[46] T. Dubois, F. Jauberteau, R. Temam, Dynamic multilevel methods and the numerical simulation of turbulence, Cambridge University Press, 1999. 38

[47] R. Codina, Stabilized finite element approximation of transient incompressible flows using orthogonal subscales, Computer Methods in Applied Mechanics and Engineering 191 (39) (2002) 4295–4321. doi:https://doi.org/10.1016/S0045-7825(02)00337-7.
URL https://www.sciencedirect.com/science/article/pii/S0045782502003377 38

[48] E. Hachem, S. Feghali, R. Codina, T. Coupez, Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation, International Journal for Numerical Methods in Engineering 94 (9) (2013) 805–825. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.

4481, doi:https://doi.org/10.1002/nme.4481.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.4481 38

[49] R. Codina, Comparison of some finite element methods for solving the diffusion-convection-reaction equation, Computer Methods in Applied Mechanics and Engineering 156 (1) (1998) 185–210. doi:https://doi.org/10.1016/S0045-7825(97)00206-5.
URL https://www.sciencedirect.com/science/article/pii/S0045782597002065 38

[50] S. Badia, R. Codina, Analysis of a stabilized finite element approximation of the transient convection-diffusion equation using an ale framework, SIAM Journal on Numerical Analysis 44 (5) (2006) 2159–2197. arXiv:https://doi.org/10.1137/050643532, doi:10.1137/050643532.
URL https://doi.org/10.1137/050643532 38

[51] A. N. Brooks, T. J. Hughes, Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations, Computer Methods in Applied Mechanics and Engineering 32 (1) (1982) 199–259. doi:https://doi.org/10.1016/0045-7825(82)90071-8.
URL https://www.sciencedirect.com/science/article/pii/0045782582900718 38

[52] A. C. Galeão, E. G. Dutra do Carmo, A consistent approximate upwind petrov-galerkin method for convection-dominated problems, Computer Methods in Applied Mechanics and Engineering 68 (1) (1988) 83–95. doi:https://doi.org/10.1016/0045-7825(88)90108-9.
URL https://www.sciencedirect.com/science/article/pii/0045782588901089 38

[53] E. Hachem, B. Rivaux, T. Kloczko, H. Digonnet, T. Coupez, Stabilized finite element method for incompressible flows with high reynolds number, Journal of computational physics 229 (23) (2010) 8643–8665. 39

[54] E. Hachem, T. Kloczko, H. Digonnet, T. Coupez, Stabilized finite element solution to handle complex heat and fluid flows in industrial furnaces using the immersed volume method, International Journal for Numerical Methods in Fluids 68 (1) (2012) 99–121. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.2498, doi:https://doi.org/10.1002/fld.2498.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2498 39

[55] K. Fukushima, Neocognitron: A hierarchical neural network capable of visual pattern recognition, Neural networks 1 (2) (1988) 119–130. 44

[56] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks 3361 (10) (1995) 1995. 44

[57] A. Wiranata, S. A. Wibowo, R. Patmasari, R. Rahmania, R. Mayasari, Investigation of padding schemes for faster r-cnn on vehicle detection, in: 2018 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), IEEE, 2018, pp. 208–212. 45

[58] B. Alsallakh, N. Kokhlikyan, V. Miglani, J. Yuan, O. Reblitz-Richardson, Mind the pad – {cnn}s can develop blind spots, in: International Conference on Learning Representations, 2021.
URL https://openreview.net/forum?id=m1CD7tPubNy 52

[59] M. Dwarampudi, N. V. S. Reddy, Effects of padding on lstms and cnns (2019). arXiv:1903.07288. 45

[60] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, B. Catanzaro, Image inpainting for irregular holes using partial convolutions (2018). arXiv:1804.07723. 46

[61] A.-D. Nguyen, S. Choi, W. Kim, S. Ahn, J. Kim, S. Lee, Distribution padding in convolutional neural networks, in: 2019 IEEE International Conference on Image Processing (ICIP), 2019, pp. 4275–4279. doi:10.1109/ICIP.2019.8803537. 46

[62] J. J. Weng, N. Ahuja, T. S. Huang, Learning recognition and segmentation of 3-d objects from 2-d images, in: 1993 (4th) International Conference on Computer Vision, IEEE, 1993, pp. 121–128. 49

[63] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook (2023) 353–374. 49

[64] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015). arXiv:1502.03167. 50, 51

[65] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, How does batch normalization help optimization?, Advances in neural information processing systems 31 (2018). 50

[66] C. Zhou, R. C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 665–674. 51

[67] L. Gondara, Medical image denoising using convolutional denoising autoencoders, in: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), IEEE, 2016, pp. 241–246.

[68] Q. Meng, D. Catchpoole, D. Skillicom, P. J. Kennedy, Relational autoencoder for feature extraction, 2017 International Joint Conference on Neural Networks (IJCNN) (2017). doi:10.1109/ijcnn.2017.7965877.
URL http://dx.doi.org/10.1109/IJCNN.2017.7965877

[69] C. Doersch, Tutorial on variational autoencoders, arXiv preprint arXiv:1606.05908 (2016). 51

[70] K. Fukami, Y. Nabae, K. Kawai, K. Fukagata, Synthetic turbulent inflow generator using machine learning, Phys. Rev. Fluids 4 (2019) 064603. doi:10.1103/PhysRevFluids.4.064603.
URL https://link.aps.org/doi/10.1103/PhysRevFluids.4.064603 51

[71] A. Mohan, D. Daniel, M. Chertkov, D. Livescu, Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence (2019). arXiv:1903.00033. 51

[72] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: G. Gordon, D. Dunson, M. Dudík (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Vol. 15 of Proceedings of Machine Learning Research, PMLR, Fort Lauderdale, FL, USA, 2011, pp. 315–323.
URL https://proceedings.mlr.press/v15/glorot11a.html 51

[73] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980. 53

[74] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems (2016). arXiv:1603.04467. 53

[75] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, L. Shao, Normalization techniques in training dnns: Methodology, analysis and application (2020). arXiv:2009.12836. 62

Chapter 3

Deep learning model for Two-Fluid Flows

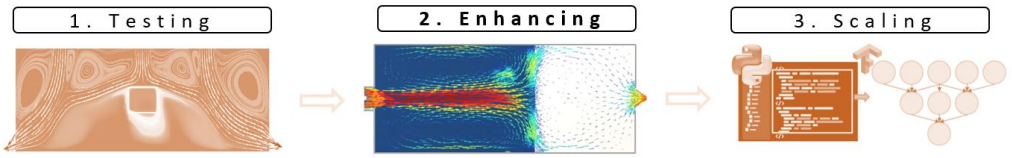


Figure 3.1: Organization of current thesis

Abstract *Various industries rely on numerical tools to simulate multiphase flows due to the wide occurrence of this phenomenon in nature, manufacturing processes, or the human body. However, the significant computation burden required for such simulations directs the research interest toward incorporating data-based approaches in the solution loop. Although, these approaches returned significant results in various domains, incorporating them in the computational fluid dynamics field is wrangled by their casting aside of the already known governing constitutional laws along with the natural incompatibility of various models with unstructured irregular discretization spaces. This work falls under the second stage of the thesis organization, as shown in Figure 3.1, where the model framework introduced in the first stage is enhanced to tackle more complex problems. Precisely, it suggests a coupling framework, between a traditional finite element CFD solver and a deep learning model, for tackling multiphase fluid flows without abandoning the benefits of physics-enriched traditional solvers. The tailored model architecture, along with the coupling framework, allows tackling the required problem with a dynamically adapted unstructured irregular triangular mesh, thus dodging the limitation of traditional convolution neural networks. Moreover, the various ingredients that allowed the model to simulate the complex and computation-demanding Navier-Stokes flow equation, such as relying on a sequential validation dataset while exposing the model training to a noise inherited from the quality of its inferring, along with the proper choice of model*

inputs, are highlighted and elaborated throughout this paper. To the authors' knowledge, this work is the first of its type to introduce a data-based graph-based approach for solving multiphase flow problems with a level-set interface capturing method.

Abstract *Diverses industries s'appuient sur des outils numériques pour simuler les écoulements multiphasiques en raison de la fréquence de ce phénomène dans la nature, les processus de fabrication ou le corps humain. Cependant, la charge de calcul importante nécessaire pour de telles simulations oriente l'intérêt de la recherche vers l'intégration d'approches basées sur des données dans la boucle de résolution. Bien que ces approches aient donné des résultats significatifs dans divers domaines, leur intégration dans le domaine de la dynamique des fluides numérique est entravée par leur mise de côté des lois constitutionnelles déjà connues, ainsi que par l'incompatibilité naturelle de divers modèles avec les espaces de discrétisation irréguliers non structurés. Ce travail s'inscrit dans le cadre de la deuxième étape des trois étapes de la thèse, comme le montre la Figure 3.1, où le cadre du modèle introduit lors de la première étape est amélioré pour aborder des problèmes plus complexes. Plus précisément, il propose un cadre de couplage entre un solveur CFD à éléments finis traditionnel et un modèle deep learning pour aborder les écoulements de fluides multiphasiques sans perdre les avantages des solveurs traditionnels enrichis en physique. L'architecture du modèle, associée au couplage, permet de résoudre le problème requis avec un maillage triangulaire irrégulier non structuré adapté de manière dynamique, échappant ainsi à la limitation des réseaux neuronaux convolutionnels traditionnels. De plus, les différents éléments qui ont permis au modèle de simuler l'équation complexe des écoulements de Navier-Stokes, tels que le recours à un ensemble de données de validation séquentielle tout en exposant l'entraînement du modèle à un bruit hérité de la qualité de son inférence, ainsi que le choix approprié des entrées du modèle, sont mis en évidence et développés tout au long de ce chapitre. À la connaissance des auteurs, ce travail est le premier en son genre à introduire une approche basée sur des données et basée sur des graphes pour résoudre les problèmes d'écoulement multiphasique avec une méthode de capture d'interface, la méthode level-set.*

Contents

3.1 Introduction

Multiphase flows dominate a large number of phenomena in both nature and industry [1]. Blood in the human body, groundwater flow through porous media, and crude oil or drilling fluid flow in the petroleum industry, are a few real-life examples of simultaneous flow of materials with two or more thermodynamic phases. These problems are of great interest to the scientific community, in general, and for the computational fluid dynamics (CFD) community, in particular [2], due to the difficulty of experimental investigation. However, simulating such flows requires coupling various partial differential equations together in a computationally expensive environment with major challenges [3]. Coupling the flow and scalar transport equations is highly popular for various industrial multiphysics problems. One of the methods for solving multiphase flow problems is based on coupling the Navier-Stokes equation, responsible for resolving the flow fields, along with a scalar transport equation, the level set method, responsible for tracking the interface. Examples of other problems governed by a similar set of equations include turbulent flow problems, in which the level set equation is replaced by another scalar equation, the one-equation Spalart Allmaras turbulence model [4], or convective heat exchange problems, where the energy equation replaces the level set equation. Although considerable efforts in the scientific community have been invested throughout the past decades to develop highly advanced numerical solvers, resolving large governing systems consisting of multiple differential equations is still challenging regarding the required time for obtaining the solution. Nevertheless, a large portion of this time is monopolized for resolving the flow equations. This is mainly due to the high dimensionality of the Navier-Stokes equations, along with the presence of nonlinear terms resulting in solution instabilities requiring additional treatment, especially for high convective flows. This high computational cost renders simulating complex industrial processes impractical and thus hinders the ability of industries to respond to the highly demanding market requests. All of these drawbacks impose the scientific community to search for techniques that may aid in resolving this issue.

On the other hand, current advances and expansion in computational power, along with the increased availability of data and numerous resources for its accumulation, all suggested machine learning in general, and deep learning (DL) in particular, as a promising candidate for aiding in the solution of the above-mentioned problem. Throughout the past decade, these data approaches have succeeded in performing tasks in various fields previously considered challenging to tackle, such as machine translation [5], natural language processing [6], speech recognition [7], and computer vision [8]. The proven ability of these early data-based approaches led to their fast diffusion to the world of computational mechanics and mainly that of fluid mechanics in multiple aspects: Turbulence modeling [9], flow prediction [10], and drag and lift forecasting [11, 12] are just a few examples of where these

models turned out to be beneficial. Moreover, dynamic problems with an evolving solution of interest, such as the tackled multiphase flow problem, or problems with a final oscillating state, require more tailored models that are able to capture the solution dynamics and act as physical simulators. Various works can be found in the literature regarding this matter: [13–16] introduced a recurrent autoencoder that operates on the reduced latent dimension of flow fields to infer its evolution in time. The authors in [17] focused on modeling the energy equation in a multi-physics environment using an encoder-decoder architecture to aid in simulating the forced convective cooling of a workpiece. In [18], the ability of convolutional neural networks (CNNs) was compared to that of recurrent ones for predicting surface waves governed by Saint-Venant equations, while in [19], they focused on LSTM based approach for predicting the changes of pressure fields in an eulerian setup. Moreover, Lino *et al.* [20], explored the ability of U-Net architecture to simulate surface waves. Finally, the use of DL models as simulators was not only limited to the computational fluid dynamics community but extended to simulate dynamics of a variety of systems such as traffic forecasting [21, 22], molecular dynamics [23–25], biological dynamics [26], etc.

The appealing characteristics of CNNs regarding their maturity for performing various tasks, along with their interesting properties of translational invariance and locality in inferring latent features, attracted most of the research effort in the last few years [27, 28]. However, various drawbacks can be observed for such models that hinder their applicability to solving complex CFD problems. Mainly, these models are constrained to an image-like rectangular-grid input/output data, which imposes an additional computation overhead required for encoding the physical fields living on unstructured irregular meshes into an appropriate space to be able to operate on. Moreover, this encoding step imposes using a homogeneous resolution along the whole computational domain, which is usually accompanied by a loss of physics in high-resolution regions in the original mesh. However, recent research efforts were able to bring into light DL models that can mitigate the topological limitations of CNNs by directly performing convolutions on graph-structured data [29]. Two methods can be distinguished for performing convolution operations on graphs: spectral convolutions [30, 31] and spatial convolutions [32–34]. Although spectral convolutions are limited to a fixed graph structure due to the global Fourier transform operation on the graph, thus rendering training on data with variable or evolving domain discretizations impossible, various works employed it for CFD simulations [35]. Regarding spatial convolutions, the graph networks framework, introduced by Battaglia *et al.* [34], is of particular interest since it extends and generalizes to various graph approaches. The framework suggests computing the required features of the output graph by simply proceeding from the edge, then to the node, and finally to the global level using multiple update and aggregation

functions. This framework gained large popularity in various fields [36–38] and also contributed to the CFD workflow in various aspects: [39] proposed an encoder-decoder graph model architecture to predict the steady-state laminar flow field past random 2D obstacles. [40, 41] proposed a particle-based graph neural network that models the dynamics of different substances with different material properties, [42] predicted the time evolution by learning the operators and potentials of the generic formulation for dissipative dynamic problems, *i.e.*, the Couette and cylinder flows. Finally, [43] proposed a mesh-based model to simulate the continuum mechanics of compressible and incompressible flows by inferring the flow fields at the next time step.

Although graph convolutional neural networks attracted many researchers in the CFD field due to their natural accordance with unstructured discretizations, most of the work relating both deep learning and multiphase flows relied on standard architectures and did not expand to graph-based methods. Dang *et al.* [44] suggested a long-short term memory convolutional neural network for inferring the water cut and the average flow velocity for oil-water flows in an industrial setup. In this work, the impedance and phase-angle readings, at a specific pipe cross-section, from a multichannel measurement system are fed to the model to allow it to infer the required flow scalars. In [45], the authors also suggested convolutional neural networks, with custom sparse layers, to tackle single- and two-phase flow problems governed by a scalar transport equation and Darcy's law for resolving, respectively, the saturation and the velocity fields. Other works, such as [46], extended previous contributions for Fourier Neural operators (FNO) on single-phase flows, [47], to tackle multiphase flow problems. The authors proposed an FNO-based model architecture to address the pressure buildup and gas saturation in the CO_2-water multiphase flow problems. Also, various works incorporated various machine learning methods for the particle-laden two-fluid flows where one of the phases is formed from dispersed particles while the other carrier fluid is continuously connected. Examples of such flows include the flow of polluted air, dust storms, aerosol sprays, or even blood flows. In [48], the authors merited from the low prediction time of convolutional neural networks to allow for real-time sediment monitoring where the output signal of a particle-laden droplet-driven triboelectric nanogenerator (PLDD-TENG) is used to predict the particle parameters. Both, Balachandar *et al.* and Seyed-Ahmadi *et al.*, highlighted in their works, [49, 50], the limitations of Fully Connected Neural Networks (FCNN) in the prediction of hydrodynamic forces on individual particles, in arrays of randomly distributed spheres, due to a large number of input parameters accompanied by the scarcity of training data. To overcome this limitation, the authors of [50] suggested a Physics-Informed Neural Network, with shared parameters and superposed pairwise particle interactions, that is able to directly predict these forces and torques. On the other hand, in [49] the authors suggested a two-step

process consisting of an accurate flow prediction step followed by a force evaluation using the predicted flows to overcome the problem of overfitting in the direct force prediction. Finally, interested readers are suggested to also consider other works where data-based approaches, operating on a structured grid or vectorized encodings, are utilized to assist in the simulation of multiphase flows [51–55].

In the present work, a message-passing graph convolution neural network is employed to simulate the dynamics of the flow equations in a strongly coupled system consisting of the Navier-Stokes equations along with the level set equation. The suggested coupling framework, along with the graph-based architecture of the model, allows capturing the evolution of the problem while maintaining the dynamic adaptation of the irregular unstructured triangular mesh. The problem under study concerns the multiphase flow of a liquid into a gaseous domain. Up to the current knowledge of the authors, the following work is the first to tackle multiphase flow problems with a Navier-Stokes convected level-set interface using a graph-based deep learning approach. The deep learning model architecture used can accurately model the Navier-Stokes equations and thus infer the required flow fields at the next time step. The model is trained to predict the field with a larger time step than that of the solver, thus allowing it to span the whole time domain in much fewer computational steps. Moreover, the trained model reveals precise interpolation ability for velocity inlet magnitudes, thus permitting us to exploit its potential for numerous flow setups by moderately training it on a few snapshots, scattered across the required time domain, obtained from discrete setups. The model also manages to span, independently, the time domain, *i.e.*, relying solely on its prediction and without referring back to the traditional solver, and returns reliable results with much less computational time.

The current work is organized as follows: the governing equations and the methods used to resolve them numerically are presented in Sec. 3.2. Details of the problem setup are then provided in Sec. 3.3. The intentions behind various chosen paths, along with the suggested coupling framework, are detailed in Sec. 3.4. Later, the model architecture and its training are respectively presented in Secs. 3.6 and 3.7. Finally, a visual and numerical evaluation of the model performance on unseen trajectories along with its ability to span independently the required time domain are briefed in Sec. 3.8. This contribution is concluded finally in Sec. 3.9.

3.2 Governing Equations

The performance of any data-based approach relies mainly on the quality of the provided dataset. Special attention is denoted to the numerical method used to generate the required datasets to guarantee our deep learning model's high caliber and reliability. Various methods can be found in the literature for simulating flows

with multiple phases. On the flow modeling level, these methods can be classified into either Eulerian [56–58], Eulerian-Lagrangian [59, 60], or particle-resolved methods [61–63]. Moreover, on the interface modeling level, two large classes can be distinguished: the implicit interface capturing class [64, 65], or the explicit interface tracking class [66]. This section introduces the main ingredients required for successfully simulating our multiphase flow using a Eulerian implicit interface capturing method and a monolithic mesh-based approach.

Thoroughly, the interface in immiscible multiphase flows, *i.e.*, the thin layer that separates both fluids, witnesses abrupt changes in thermodynamic properties, such as density and viscosity. The flow characteristics for every fluid rely mainly on these properties. Thus, an adequate interface capturing technique that specifies the interface's location, using an implicit auxiliary function, and follows its evolution in time is required, as detailed in Sec. 3.2.1. After specifying the location of the interface in the computational domain, the mixing law required to smoothly define the variation of the fluid properties across this domain is detailed in Sec. 3.2.2. Moreover, Sec. 3.2.3 introduces the gradient-based mesh adaptation method needed to reduce the computational cost of such complex simulations and accurately capture the interface and its evolution. Finally, the evolution of the interface is guided by the flow fields resolved using the Navier-Stokes equation. Thus, Sec. 3.2.4 is solely devoted to detailing the method used to resolve these equations along with a stabilization algorithm required for convective-dominated flows discretized with the classical Galerkin formulation.

3.2.1 Level-Set Method: Convective-Reactive Method for Interface Capturing and Evolution

Accurately resolving multiphase flows relies on precisely capturing the interface separating the different fluids and its evolution. An interface capturing method, known as the level set method, is utilized for this objective. This method relies on a signed distance function with a zero level set on the interface. The method uses a geometric brute force algorithm to determine the smallest distance, $d((x,y),\Gamma_i))$, to the interface, Γ_i, at every point in the domain, $(x,y) \in \Omega$. The different fluid domains, Ω_{f_1} and Ω_{f_2}, are then differentiated by signing the obtained distance:

$$\phi(x,y) = \begin{cases} -d\big((x,y),\Gamma_i\big) & \text{if } (x,y) \in \Omega_{f_1}, \\ 0 & \text{if } (x,y) \in \Gamma_i, \\ d\big((x,y),\Gamma_i\big) & \text{if } (x,y) \in \Omega_{f_2} \end{cases} \tag{3.1}$$

The evolution of the interface over time and the variation in each domain shape,

size, etc., is governed by a scalar transport equation:

$$\frac{\partial \Phi}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \Phi = 0, \tag{3.2}$$

where $\boldsymbol{u}$ is the convecting flow velocity. However, relying solely on the transport equation to model the interface evolution will diffuse the iso values near the interface and deport ϕ from being a distance function with a unity gradient. This leads to a nonphysical varying interface thickness and inappropriate distribution of properties. The level set function is reinitialized by solving a Hamilton-Jacobi problem that recovers the analytical unity value of the gradient to solve the above problem [67].

However, the above procedure is rendered inefficient due to: first, the requirement of a two-step process, *i.e.*, transporting then reinitializing the distance function at every time increment, and second, convecting the function on the whole computation domain despite its sole importance across the interface. This process is optimized by reducing the two-step procedure into a one-equation single-step evolution model by embedding the Hamilton-Jacobi equation into the transport equation. Moreover, rather than considering the whole domain, the function in the proximity of the interface is only convected by applying a filter that results in a zero-gradient at nodes lying outside the interface thickness.

It should be noted that the level set method also suffers from violating the mass conservation for every phase due to numerical dissipation [68]. The numerical solution of partial differential equations introduces additional terms in the discretized form leading to both numerical dissipation and dispersion [69]. Although this dissipation eases the convergence and the stability of the resulting solver and damps under-resolved high frequency modes, it deteriorates the solution fidelity and induces amplitude errors. Other interface capturing methods, such as the Volume of Fluid method (VoF) for example, are able to exactly, by construction, to conserve mass and overcome the above limitation since they solve directly for the advection equation of the volume fraction [70]. However, for the level-set method, this error can be reduced by using a high-resolution discretization near the interface. The resulting sharp gradient in the filtered level set function aids our gradient-based metric mesh adaptation method in focusing on the interface and better capturing it using highly stretched elements, thus reducing this conservation error.

3.2.2 Mixing Laws: Initialize the Flow Properties

The main objective of following the evolution of the interface is to accurately define, at every point in space, the physical properties that parameterize the partial differential equations governing the simulation. The signed filtered level set function, defined in the previous section, is used to implicitly annotate the different physical subdomains in our monolithic computational domain and to define the distance of

the nodes to the interface. This function, in addition to a Heaviside function, H, will be used to compute the global material properties, *e.g.*, density, viscosity, etc., based on the mixing law shown in Eq. 3.3:

$$p(x,y) = p_{f_1} H\big(\phi(x,y)\big) + p_{f_2}\Big(1 - H\big(\phi(x,y)\big)\Big), \tag{3.3}$$

where p_{f_i} is the physical property specific for each phase. The classical Heaviside function, H, causes numerical instabilities near the interface due to the abrupt changes in physical properties. To overcome this problem, a smoothed Heaviside function [71], H_ϵ, parametrized by a virtual interface thickness, ϵ, is instead utilized in correspondence with the above mixing law and defined in Eq. 3.4. Other Heaviside functions that aid in tackling the above problem can be found in the literature [72].

The virtual interface thickness, ϵ, is specified based on the discretization mesh size near the interface and allows smoothing of the transition. Adequate adaptive mesh methods aid in reducing this virtual thickness and obtaining a more accurate physical distribution of properties.

$$H_\epsilon(\phi) = \begin{cases} 1 & \text{if } \phi > \epsilon, \\ \frac{1}{2}\big(1 + \frac{\phi}{\epsilon} + \frac{1}{\pi} sin(\frac{\pi\phi}{\epsilon})\big) & \text{if } |\phi| \leq \epsilon, \\ 0 & \text{if } \phi < -\epsilon \end{cases} \tag{3.4}$$

Moreover, it should be noted that specific physical properties, such as thermal conductivity, require adapted mixing laws to avoid inaccurate results along the interface [73, 74]. However, these properties are not utilized in the multiphase flow simulations considered in this current work and thus will not be discussed.

3.2.3 Anisotropic mesh adaptation

Accurate capturing of all the physics across the multiphase flow interface requires a high-resolution mesh. Having a coarse mesh across this delicate region induces various numerical and physical problems. Other than the increase of mass conservation error encountered in level-set methods over coarse resolutions, inaccurate distribution of material properties over nonphysical wide virtual interface thickness occurs. Moreover, discontinuities in physical properties result in sharp gradient fields. These sharp gradients may induce numerical instabilities if intersecting an arbitrary discretization mesh.

To overcome the above difficulties without inducing a high computational cost, usually present in multiphase flow simulations over high-resolution computational domains, an anisotropic mesh adaptation technique, driven by gradient-based directional error estimators, is employed. The adaptor can accurately capture the evolving interface by locally refining the discretization around the zero iso value of

the level set using highly stretched and well-oriented elements in the regions with high variations. The procedure to compute the adaptation metric is briefly summarized in this section, and interested readers may refer to [75] for further details.

First, the interpolation error, E^{ij}, along a mesh edge, e^{ij}, can be computed using the gradients at the edge extremities, *i.e.*, g^i and g^j, of the approximate solution, u_h. However, a gradient recovery procedure is required since the computational domain is discretized using first-order elements, and thus no gradient is defined at the node level. To obtain the recovered gradient, G^i, at vertex i, a local optimization problem is solved and results in a gradient defined in terms of a length distribution tensor, X^i. This recovered gradient is then used to estimate the error as shown in Eq. 3.5,

$$E^{ij} = (G^j - G^i) \cdot e^{ij} \tag{3.5}$$

Then, a stretching vector, s^{ij}, based on the estimated error, is computed for every edge,

$$s^{ij} = \frac{E_{ij}}{e(N)}, \tag{3.6}$$

where $e(N)$ is the total mesh estimated interpolation error. Finally, the required metric, suitable for performing the gradient-based mesh adaptation, is defined as shown in Eq. 3.7,

$$\tilde{M}^i = \left(\frac{1}{|\Gamma(i)|} \Big(\sum_{j \in \Gamma(i)} s^{ij} \otimes s^{ij} \Big) \right)^{-1}, \tag{3.7}$$

where $\Gamma(i)$ is the set of nodes sharing a single edge with node i.

It should be noted that the framework suggested in this work for tackling multiphase flow simulation with the aid of deep learning models is able to preserve all the stated benefits of using the above adaptive mesh refinement method. This is mainly due to the modularity of our framework, where bonding with the traditional solver is maintained.

3.2.4 The Stabilized Navier-Stokes Flow Equations

To obtain the unsteady and incompressible flow fields required to convect the interface, the famous Navier-Stokes equations must be resolved. The equations are constituted of a momentum conservation equation and a continuity equation reflecting the incompressibility of the fluid. The unsteady flow simulation tackled in this work is restricted to Newtonian incompressible fluids, and thus the required velocity, $\mathbf{u}$, and pressure, p, fields are governed by the equations shown in 3.8.

$$\begin{aligned} \rho(\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}) &= \nabla \cdot (-p\mathbf{I} + 2\mu\boldsymbol{\varepsilon}(\mathbf{u})) + \boldsymbol{\psi}, \\ \nabla \cdot \mathbf{u} &= 0, \end{aligned} \tag{3.8}$$

where ρ and μ are, respectively, the global density and dynamic viscosity of the fluid. The strain rate tensor, ε, is computed using the flow gradient's symmetric part, *i.e.*, $\varepsilon(\mathbf{u}) = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)$. All external forces acting on the system are accounted for in $\boldsymbol{\psi}$. Moreover, the following equations should be accompanied by the appropriate initial and boundary conditions for well defining the problem.

However, analytical solutions to these equations are rare and limited to a few simplified problems. Thus, a finite element method is suggested to obtain the required fields accurately. To derive the variational formulation, the Green theorem is first utilized to obtain the weak formulation of the above flow equations. Next, the finite element approximation is obtained by discretizing the computational domain into K triangular simplex elements using the classic Galerkin approach. Thus, resulting in the following discrete problem,

$$\begin{aligned} (\rho(\partial_t \boldsymbol{u}_h + \boldsymbol{u}_h \cdot \nabla \boldsymbol{u}_h)\ ,\ \boldsymbol{w}_h) + (2\mu\boldsymbol{\varepsilon}(\boldsymbol{u})_h\ ,\ \boldsymbol{\varepsilon}(\boldsymbol{w})_h) - (p_h\ ,\ \nabla \cdot \boldsymbol{w}_h) &= (\boldsymbol{\psi}\ ,\ \boldsymbol{w}_h), \\ (\nabla \cdot \boldsymbol{u}_h\ ,\ q_h) &= 0, \end{aligned} \tag{3.9}$$

where $\boldsymbol{u}_h$ and p_h are the approximate finite solution of velocity and pressure, and $\boldsymbol{w}_h$, and q_h are their respective test functions.

However, the above variational formulation suffers from numerical instabilities and thus requires additional treatment. The instabilities originate either from the nonlinear convective term present in high Reynolds number flows or from the classic Galerkin space discretization that does not satisfy the Ladyzhenskaya–Babuška–Brezzi (LBB) criteria [76]. The Variational Multiscale approach (VMS) [77] is suggested in this work to tackle the above instabilities and will be briefly outlined in this section.

The VMS approach is based on enriching the standard Galerkin formulation with additional weighted residual-based stabilization terms. These terms are obtained by enriching the functional spaces with orthogonal subscales. Initially, the solution fields and their weighting functions are decomposed into a resolvable coarse-scale component and an unresolved fine-scale component. Introducing the decomposed terms into the unstable variational formulation results in two sub-problems relative to every scale. First, the fine-scale problem is simplified using various assumptions, justified in [78–80], and is used to approximate the subscale flow components with the aid of a separation technique [81, 82]. The fine scales are approximated as the product of stabilization parameters and the residuals of the coarse-scale momentum and continuity equations. Next, the derived approximations are used to substitute the fine-scale components present in the coarse-scale problem, thus eliminating their appearance but preserving their effects on the coarse fields. Finally, the obtained

coarse-scale stabilized form, enriched with fine-scale characteristics, is resolved for the required flow fields. It should be noted that various definitions can be found in the literature for the stabilization coefficients with an ongoing interest in enhancing the resulting convergence behavior [83].

3.3 Problem Definition

The multiphase flow considered in this work is the unsteady forced vertical flow of a fluid into the domain of another. Thus, a fluid substitutes another fluid, with different thermodynamical properties, in an unsteady, dynamic process. This problem is similar to the tank-filling problem, where fluid fills an empty tank from a bottom inlet. This problem is governed by a multiple set of equations consisting of the Navier stokes equations, 3.8, responsible for resolving the flow fields as mentioned in Sec. 3.2.4, along with a convective self-reinitialization level set equation, Eq. 3.1, accountable for capturing the interface and its evolution as detailed in Sec. 3.2.1.

It should be noted that the governing system of equations is a strongly coupled system leading to a bidirectional interaction between systems' partial differential equations. The convected level set function strongly affects the distribution of the thermodynamical parameters, which in their turn, are used to recompute the convecting flow fields using the Navier-Stokes equations. Thus, the resulting equations are solved sequentially using an industrial-level in-house VMS solver. The solver starts by computing the flow fields, *i.e.*, the velocity and pressure, then resolves for the next-step level-set function using the computed fields. The solver preconditions all linear systems with a block Jacobi method supplemented by an incomplete LU factorization and solves it with the generalized minimal residual (GMRES) algorithm. Finally, it is important to state that the solver's accuracy and reliability have been tested and validated in various works related to multiphase flows [3, 84, 85], thus minimizing the concerns of having a deteriorated dataset that may reduce the performance of our DL model.

The problem setup consists of a rectangular domain with a bottom inlet and a top outlet. The tank width and height are respectively H and $2H$. The $0.2H$-wide inlet is positioned at the center of the bottom wall and is required to fill the tank with fluid 1. It should be noted that the side walls of the inlet are slightly extruded by a value of $0.1H$ to enhance numerical stability. Similarly, a same-width outlet is positioned at the center of the opposing wall to allow the escape of fluid 2. All information regarding the problem setup is summarized in Figure 3.2a. Moreover, multiphase flows with a fluid filling from the bottom are usually characterized by a dome-shaped interface, as shown in Figure 3.2b. This dome's height, h, is of particular interest for comparison with experimental results or for validating the simulation accuracy and will be later utilized to evaluate the performance of our

suggested resolution method.

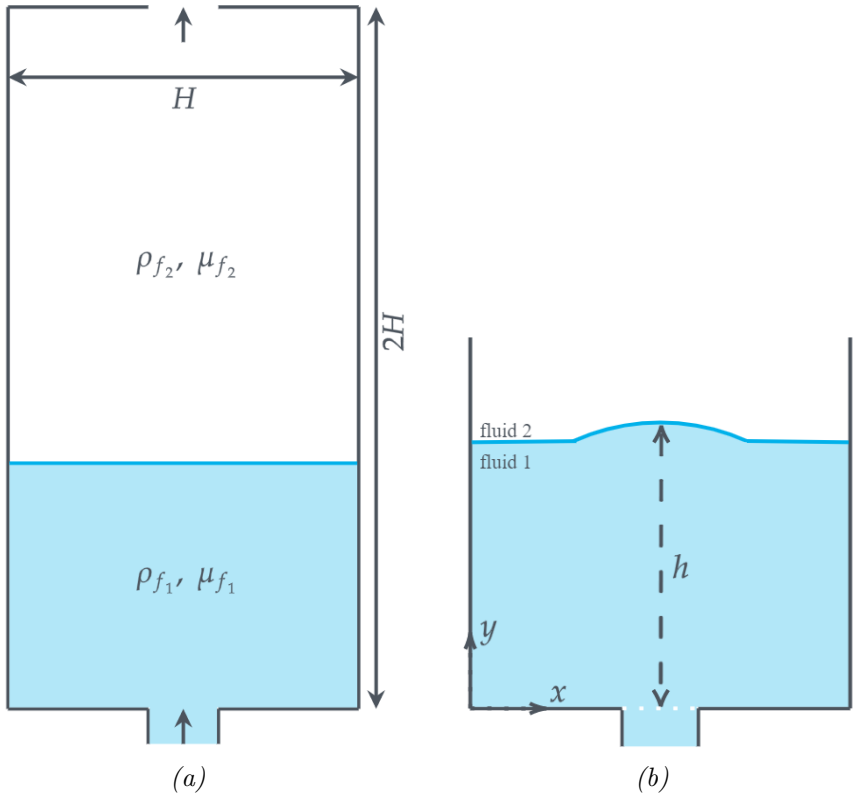


Figure 3.2: ***Problem Setup*** *of a multiphase flow problem with (a) showing fluid* 1 *with thermal properties,* ρ_{f_1} *and* μ_{f_1}*, filling a* $2H$*-by-*H *tank previously occupied by a fluid with different properties,* ρ_{f_2} *and* μ_{f_2}*. In (b) the dome height,* h*, later used to evaluate framework performance, is shown.*

To define the flow problem, the thermodynamic properties of each fluid phase are specified. The density and dynamic viscosity of the filling fluid, *i.e.*, fluid 1, are $\rho_{f_1} = 10^2$ kg/m^3 and $\mu_{f_1} = 10^{-1}$ Pa s. Similarly, the properties of the escaping fluid, *i.e.*, fluid 2, are set to $\rho_{f_2} = 10^{-1}$ kg/m^3 and $\mu_{f_2} = 10^{-4}$ Pa s. The interface separating fluids 1 and 2 is initially located by a zero level set, defined by a distance function at an offset of $0.01H$ below the bottom wall, thus allowing fluid 1 to approximately fill the extruded inlet pipe. For the flow fields, a no-slip boundary condition is specified on the top and bottom walls, whereas a symmetric condition is specified for the side walls. The outlet is designated with zero pressure while the velocity at the inlet is varied to allow us to train the model for different flow fields and test its generalization on new ones, as will be detailed later. Finally, the source

term in the momentum equation, $\boldsymbol{\psi}$, is set to zero since no buoyancy forces or stresses on the fluid are considered. The required physical fields are computed on an unstructured irregular anisotropic triangular mesh. The number of elements in this mesh is limited to $20,000$. The mesh is dynamically adapted with the convected interface, as is discussed in Sec. 3.2.3. The evolution of the discretization domain throughout the simulation, along with the various computed physical fields, are depicted in Figure 3.3 for various time steps.

3.4 Main Idea

Various multiphysics CFD simulations are governed by coupling Navier-Stokes equations to a scalar transport equation. Most of the computation burden is monopolized by the numerical solver required to resolve the flow fields governed by the Navier-Stokes equations. This is clearly depicted in Figure 3.4, showing the percentage distribution of CPU time, over both the NS solver and the scalar transport equation solver, for different problems. For certain problems, such as forced convective cooling or the turbulent flow past an object, the computational cost of the scalar transport equation is not negligible, although dominated by that of the Stokes equations. Thus, it will remain beneficial to model the scalar equation using data-based approaches as done in previous works, [9, 17]. However, with only 2% of the total computational time required to resolve the level-set convection equation in multiphase flows, it will be impractical to model it using a deep learning model.

Thus, this work intents is to aid a traditional finite element solver by directly inferring the flow fields using a deep learning network. The network is trained to model the Navier-Stokes equation and predict the governing flows for various inlet velocities. To attain this objective, the required physical fields are initially obtained for discrete values of inlet velocity magnitudes. Specifically, the amplitude of the inlet vertical velocity component is varied in the range of $u_{x,in} = [0.5, 1]$ m/s by a step of 0.05 m/s to generate multiple simulation trajectories for filling a 2-by-1 m^2 tank. The obtained trajectories, comprised of a varying number of snapshots due to the difference in the time required for filling the tank, are distributed into three separate datasets: a training dataset with 6 trajectories and around 3000 snapshots, a validation dataset, and a testing dataset with single trajectories and around 500 snapshots each. The first dataset, composed of most trajectories, is used to determine the appropriate set of trainable parameters, as detailed in Sec. 3.7. The remaining two datasets are employed to ensure the robustness of the model, and its performance on unseen data, respectively.

The choice of the input and output fields of the model is inspired by those required by the original equation. For the input fields, the sets are assembled with the velocity components, u_x^n and u_y^n, and the computed global density, ρ^n, at a certain

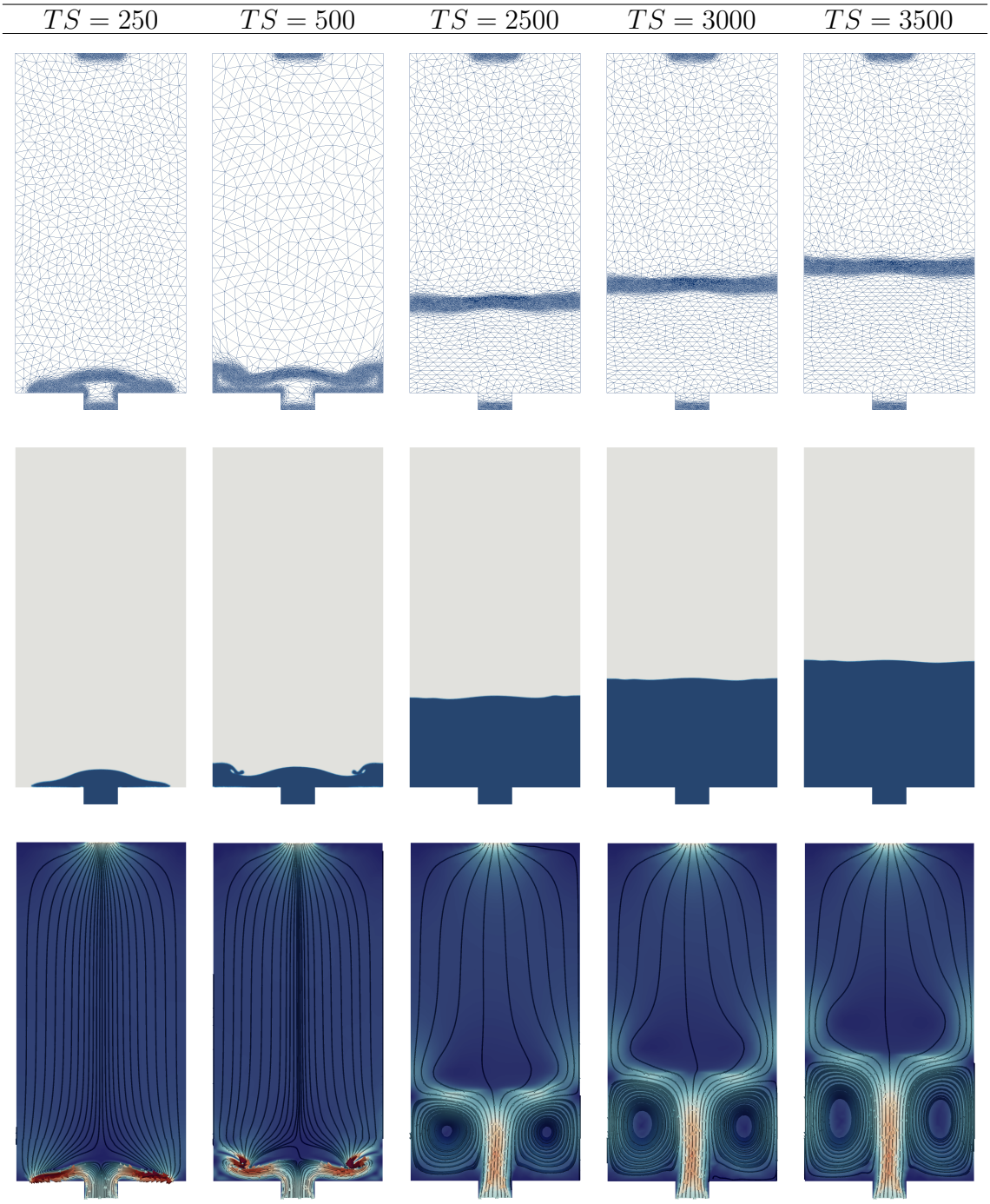
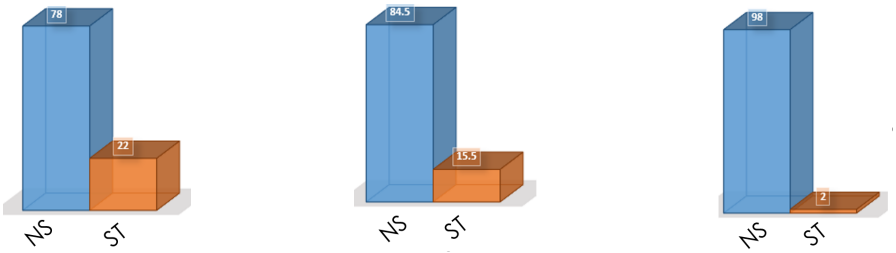


Figure 3.3: ***The evolution of the discretization mesh and the physical fields over simulation time.*** *The snapshots, from left to right, are obtained at the following time steps: 250, 500, 2500, 3000, and 3500. A simulation with an inlet velocity of* 0.55 *m/s is utilized. The first row shows the discretization mesh obtained using the gradient-based mesh adaptive method over the filtered level-set function. The second and third rows show the density and the streamlined velocity fields.*



(a) (b) (c)

Figure 3.4: Comparison of the percentage of computation time required for each solver for various multiphysics problems: (a) Convective cooling, (b) Resolving turbulent flows, and (c) Multiphase flows.

time step, $t = t^n$, along with a binary scalar required to flag the boundary nodes, b^n. The number of thermodynamic properties provided to the model input space is limited to density only, without extending to the remaining properties, since a single property only is sufficient to flag the physical subdomains in the global computational domain and thus maintain a low dimension input space. Concerning the outputs, both the next-step pressure, p^{n+1}, and acceleration, a_x^n and a_y^n, computed using a linear finite difference scheme, are used. The required next-step velocity is easily attained using a forward Euler scheme, *i.e.*, $\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t_m \times \boldsymbol{a}^n$. A sketch of the model's inputs and outputs is shown in Figure 3.5.

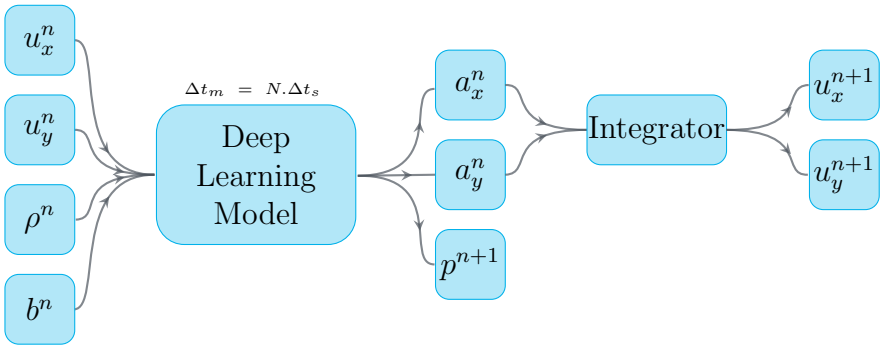
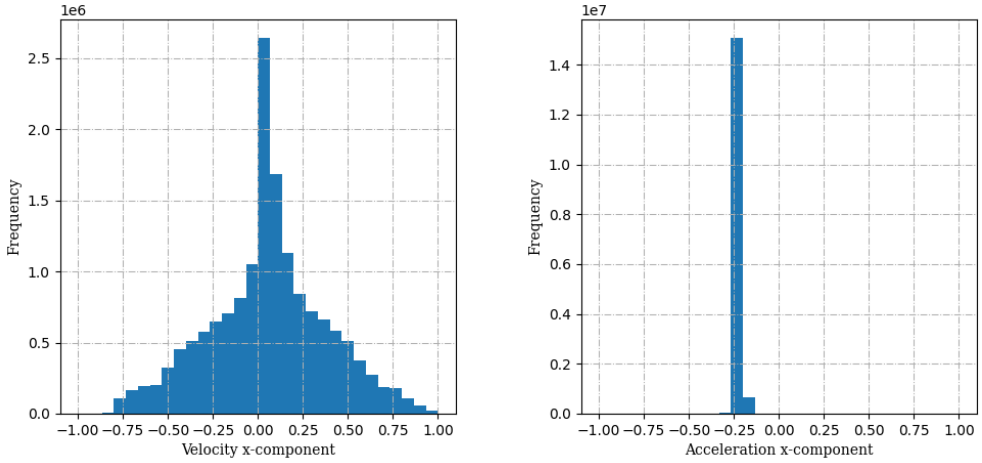


*Figure 3.5: **Model sketch** showing the various input fields at t^n, and the required fields at the next model time step, $t^{n+1} = t^n + \Delta t_m$.*

The choice of acceleration as an output field, rather than directly inferring the velocity field, is initially motivated by the model architecture with residual connections that aims at predicting the change in the input fields. Moreover, the smaller variation in the acceleration distribution compared to that of velocity, as shown in the histogram plots in Figure 3.6, facilitates the inferring objective of our data-based approach. This is highlighted later in Sec. 3.8, where the performance of models

trained with different output fields is compared.



(a) *(b)*

Figure 3.6: ***Fields distribution*** *for both (a) velocity and (b) acceleration x-components. Both fields are normalized and transformed to the same scale before obtaining their distribution to ease comparison. It should be noted that similar distributions are obtained for the y-components.*

As detailed later in Sec. 3.6, the model used is able to perform direct convolutions on graph-structured data. This feature elevates the previous requirement of having the data transported into a structured grid encoding to allow traditional convolutional neural networks to operate on them [17]. Most importantly, working directly on the original discretization domain avoids losing physical information, usually encountered if interpolating into a coarser resolution, and avoids the additional computational burden required if a finely structured grid encoding is utilized. Thus, the computed physical fields, along with the positional information, are directly encoded into a graph having an identical topology as the unstructured irregular triangular discretization used by the finite element solver. Specifically, for every time step, the physical fields are encoded as node features for the mesh vertices, whereas positional information is provided as edge features for the mesh edges. It should be noted that the model architecture utilized requires an identical graph structure for both its input and output fields. In the case of a dynamic mesh, this requirement demands interpolating the resolved physical fields at the next time step to the parent mesh at the previous time step. However, since the mesh evolves leisurely, the output mesh is almost similar to the input mesh, and thus, the error encountered due to this interpolation step is negligible.

The time step of the model is chosen to be larger than that of its parent solver,

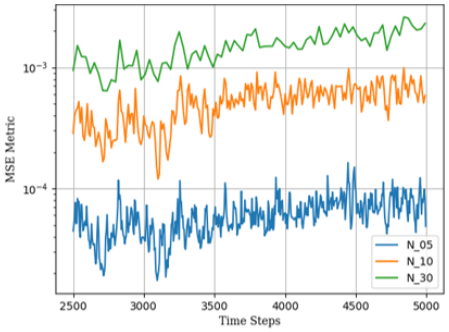


Figure 3.7: ***Plot of the a posteriori error metric over simulation time*** *for various model time step size, $N = 5$, 10 and 30. The model used for comparison is a simplified version of the main model with fewer parameters to reduce the required training time.*

i.e., $\Delta t_{model} = N \times \Delta t_{solver}$, where N is a user-defined parameter based on the tackled problem characteristics and the allowed margin of error in its application. This allows for further reduction in computational cost offered by spanning the time domain at faster rates and is rendered possible due to the lack of direct constraints of the CFL condition (Courant–Friedrichs–Lewy) on DL models. However, the choice of the step size should be made in accordance with the modeled partial differential equation and its dynamics to avoid inducing instable fluctuations and discarding variations necessary for the inference of future time steps. Very large time steps lead to deterioration in the resolution of the time-space discretization and are accompanied by depreciation of the available samples in the training dataset, leading to a reduction in model performance. Thus, the choice of N is a tradeoff between the attained reduction in computational cost and model performance, as seen in Figure 3.7.

The framework utilized to tackle the multiphase flow described in this work is depicted in Figure 3.8. Initially, the governing equations are resolved solely using the traditional solver until the initial transition phase, with abrupt variations in fields, is bypassed. After that, at $t = t^n$, the deep learning model is engaged to infer the flow fields instead of the CFD solver. The flow fields, $\mathbf{u}^n$, along with the thermodynamic properties, ρ^n, at time step t^n, are fed to the deep learning model. The model infers the required flow fields at the next time step t^{n+1}, where $t^{n+1} = t^n + \Delta t_{model}$. The predicted fields are then transmitted to the solver responsible for convecting the interface where the level set function, α^{n+1}, is computed. The updated distribution of thermodynamical properties, ρ^{n+1}, is then found. Finally, the inferred flow fields are then recycled by feeding them back to the model, along with the solver computed properties, for multiple steps specified by the control parameter f. After multiple predictions, the finite element flow solver is re-engaged to redirect the solution and

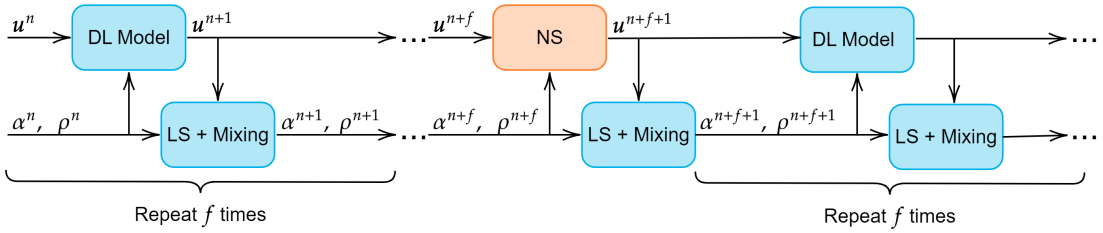


Figure 3.8: ***Coupling Framework*** *for multiphase flow problems governed by strongly coupled Navier-Stokes equation, NS, and the level-set equation, LS.*

enforce the governing constitutional laws, such as the conservation of mass and momentum. Finally, the outputted corrected flow fields, $\mathbf{u}^{n+f+1}$, are then utilized to proceed again with the deep learning model, and the same sequence is repeated until the whole time domain is spanned.

The advantage of the introduced coupling framework lies in its ability to tackle, in a modular approach, complex CFD problems requiring the resolution of various partial differential equations. The deep learning model doesn't infer all required physical fields in an end-to-end manner. Instead, it focuses its approximation capabilities on modeling only the equations requiring high computational cost, such as the Navier-Stokes equations, and allows the handling of the remaining cheap equations using their traditional well-established accurate solvers, as depicted in Figure 3.4 which compares the required portion of each solver from the total computation time for the various multiphysics problems, [17, 86], resolved using our in-house finite element library, [87]. Furthermore, as the data-based solver propagates in time, the accumulation of approximation errors might result in inaccurate outcomes. Thus, to elevate this issue, the number of data-based inferrings is controlled using a recurrent correction frequency parameter, f, that can be either predefined to a certain value based on experience or related to an error quantification metric, such as the residual of the equations. Finally, modularizing the resolution process and maintaining a live connection with the CFD solver allows performing tasks not yet fully approached with deep learning, either due to their complexity or due to the lack of research in this specific area, such as adapting and evolving the discretization mesh which is a required key parameter for interface tracking problems.

The coupling framework, incorporating our data-based approach in the solution loop, extends the previous framework introduced in [17]. First, the deep learning model employed in this work allows operating directly on the triangular discretization mesh without any prior requirement of interpolation into a structured grid encoding. Moreover, the framework is generalized to tackle strongly coupled problems where the solution of both the scalar and the Navier-Stokes equations affects

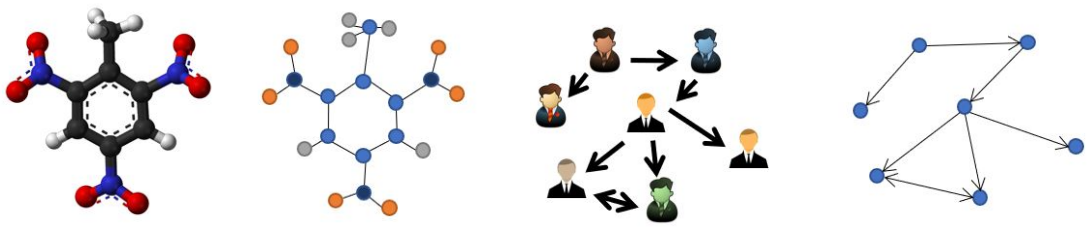
the computation of the required physical fields at the next step. This necessitated introducing additional communication bridges between solvers to allow the exchange of needed information. Finally the data-based approach is incorporated in this work to infer the flow fields rather than a single scalar field. This paves the way for our framework to tackle a broader range of simulation problems where inferring the flow fields is required, despite the remaining set of equations involved.

3.5 Introducing the deep learning model components

The following section is reserved for providing a brief overview of the main deep learning components later utilized in defining the model architecture. It primarily focuses on introducing the message-passing approach for convolution over graphical data. The distinction between the convolutional kernels in traditional CNN models and those in graph neural networks is highlighted. Lastly, the LayerNorm technique, used to expedite model training and mitigate internal covariate shifts, is discussed.

Convolutions on graphical data

The success of deep learning models across various tasks has sparked interest in their application to novel domains. However, certain applications demand the handling of data structured as graphs, where conventional architectures like multilayer perceptrons or convolutional neural networks prove unsuitable. A graph, denoted as $G = (V, E)$, comprises unstructured data with nodes represented as $V = \{v_i\}_{i=1}^{n_v}$ and edges as $E = \{e_i\}_{i=1}^{n_e}$. Graphs find wide-ranging utility in representing data such as traffic networks, social interactions, or chemical compounds. Additionally, unstructured discretization spaces employed in computational physics can be viewed as graphs, with mesh vertices as nodes and the connectivity between mesh elements as edges. Figure 3.9 shows examples of different type of data that could be represented with graphs.



(a) *(b)*

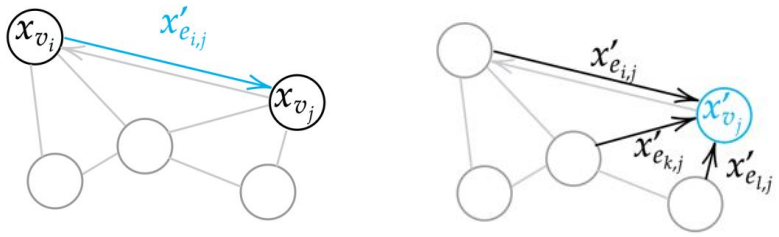
Figure 3.9: Representing data using graphs: (a) A chemical molecule converted into a graph, and in (b) a social network represented as a graph with directed edges.

Several challenges hinders the applicability of conventional deep learning architectures on graphical data. First, the unstructured nature of graphs with completely arbitrary indexing of nodes and edges constitutes the first challenge. Also, Graphs, in general, are dynamic and lack a fixed form, meaning that multiple visual representations can coexist for a single graph instance. Also, in contrast to images, where every pixel is consistently surrounded by a well-defined number of neighboring pixels, the number of connections between nodes in a graph can vary significantly. The above reasons motivated the search for new deep learning models that are adapted for directly operating on graphical data.

The initial approach for convolving over graphical data was based on spectral convolution, drawing inspiration from graph signal processing. This approach relied on the graph Fourier transform to substitute convolution with the product of the eigen-decomposed feature matrix and the convolution kernel in the Fourier Space [88]. However, its dependence on an expensive eigen-decomposition operation and the kernel's rigidity in relation to graph topology have impeded its practicality, thus giving preference to alternative methodologies. Several other methodologies have been proposed to perform convolution operations directly on graph-structured data within the spatial domain. One of the most common approaches is neural message passing, initially introduced by Gilmer *et al.* [32], and later extended into the Graph Networks Framework [34]. The convolution operation based on the message-passing approach can be decomposed into two main steps, as illustrated in Figure 3.10. In the first step, edge features are updated by incorporating information from the connected nodes using an edge kernel, f^E. Subsequently, the updated edge features are aggregated at each node and then used to update the node's features via a node kernel, f^V. In each message-passing layer, both the edge and node kernels, f^E and f^V, which are essentially two shallow multi-layer perceptrons, are applied to all edges and nodes, thus performing a single convolution operation across the entire graph. Similar to convolutional neural networks, multiple layers of message-passing are stacked to enhance the model's approximation capacity and expand the scope of communication for each neuron.

Convolution kernels

In a traditional convolutional layer applied over a structured input data, a vector or a tensor, referred to as a kernel, is employed to perform convolution across the input data. This kernel systematically traverses the input to calculate a point-wise weighted sum at each position, and subsequently applies a non-linear activation function. The following convolution operation is marked by two main features: locality and translation invariance. Locality denotes that the computed output at any specific location depends solely on the nearby data points, with no regard for distant elements. Also, translation invariance is attained through the consistent reuse of the



(a) *(b)*

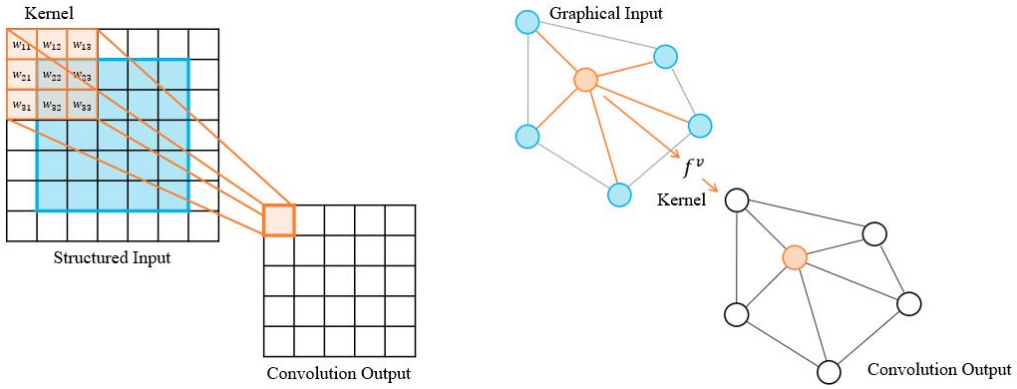
Figure 3.10: Message passing on graphs. The procedure is divided into two steps: (a) update the edge features using the features of the connected node. In (b) the updated edge features are aggregated on each node to update the node features.

same convolutional kernel across all regions within the input. In convolution over graphical data using the message-passing approach, the objective is to inherit these characteristics. However, this task is complicated by the unstructured nature of input graphs, which renders the use of structured kernels for computing non-linear weighted sums over local regions impractical. To address this challenge, two distinct multilayer perceptrons, denoted as f^E and f^V, substitute the traditional convolution kernels and are employed for convolution over all edges and nodes within the input graph, respectively. These functions are consistently applied across each element of the graph, ensuring uniformity in the update process within the same set of entities. Furthermore, this update process is constrained to the local neighborhood of each entity, thereby preserving the inherent biases stemming from the graph's underlying topology. Figure 3.11 visualizes the convolution operation for both a structured image and an unstructured graph.

LayerNorm

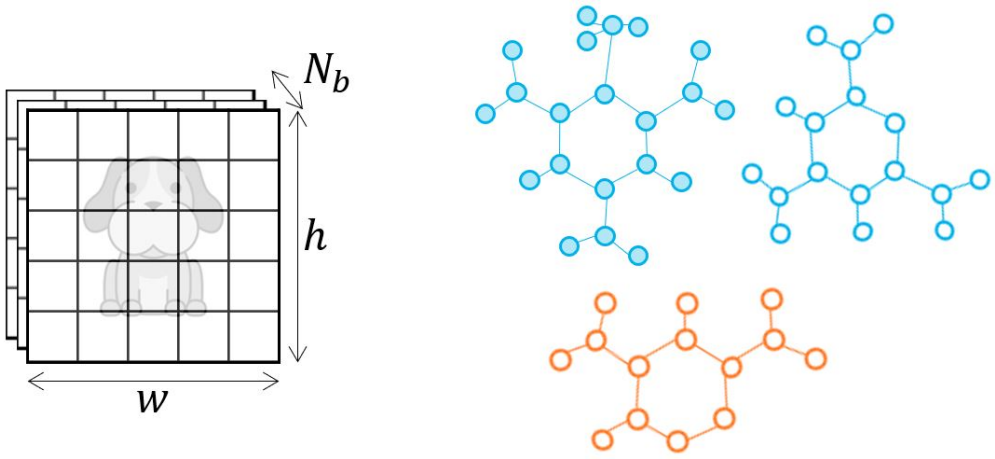
Mini-batch training is usually employed to optimize the model parameters based on a batch of input data rather than relying on a single example at every update step. However, the definition of a batch on graphical data might differ from that on images. Previously, for images, a tensor with the shape of $N_b \times h \times w \times c$ was used to represent a batch of N_b images, each with a height, width, and channel of h, w, and c, respectively. However, for unstructured graphical data with each having a different number of vertices, relying on 4D tensors to represent batches of data samples is impossible. Instead, a global graph is constructed to allow mini-batch training. A global graph is simply a large graph that concatenates individual independent graphs, as shown in Figure 3.12.

The variation in the distribution of the internal hidden activations for every hid-



(a) *(b)*

Figure 3.11: Convolution for different data structures. In both inputs, the input entities considered by the convolution operation at that position are highlighted with orange. The weights of the kernel in (a), w_{ij}, or the parameters of the convolution function f^v, are maintained for the same for all the entities. It should be noted that in (b) only the convolution over nodes is visualized for simplicity.



(a) *(b)*

Figure 3.12: Batches of data of different types: (a) A batch of N_b structured singe channel images, and in (b) A Global graph constituted of three separate graphs.

den layer is preferably maintained limited across the different update steps to allow faster convergence of the training algorithm [89]. For traditional convolutional neural networks, a batch normalization layer could be appended to the model architecture after every convolution layer to standardize the computed hidden activations. LayerNorm is a similar technique that reduces the internal covariate shift and ensures faster convergence of the optimization algorithm [90]. Contrary to batch normalization, previously discussed in 2.5, LayerNorm standardizes the activations of its input batch using each input sample independently rather than relying on the stats of each activation across the whole batch. In the context of graph neural networks, LayerNorm layers can be appended to the model architecture following every Graph Network (GN) block, *i.e.*, on the updated features after each convolution operator, f^e and f^v. Assume $\mathbf{x'}_{v_i}$ to be the updated feature vector of node i with a latent dimension of d_l. The computed stats for a LayerNorm layer, added after the update function, will rely solely on the entries of $\mathbf{x'}_{v_i}$, as shown in Eq. 3.10,

$$\begin{aligned} \mu_{v_i} &= \frac{\sum_j^{d_l} x'_{v_{i,j}}}{d_l}, \\ \sigma^2_{v_i} &= \frac{\sum_j^{d_l} (x'_{v_{i,j}} - \mu_{v_i})^2}{d_l} \end{aligned} \tag{3.10}$$

The mean and variance specific for node i, *i.e.*, μ_{v_i} and $\sigma^2_{v_i}$, are then used to standardize the entries in the node feature vector, before passing it to the next GN block, as shown in Eq. 3.11,

$$\mathbf{x'}_{v_i,norm} = \frac{\gamma_{v_i} \times (\mathbf{x'}_{v_i} - \mu_{v_i})}{\sqrt{{\sigma_{v_i}}^2 + \epsilon}} + \beta_{v_i}, \tag{3.11}$$

where ϵ is a small constant for numerical stability, while γ_{v_i} and β_{v_i} are the introduced learnable parameters to avoid constraining the layers' approximation capacity.

3.6 Model Architecture

The introduced framework consists of two main solvers: the traditional finite element solver and the data-based solver. To facilitate the coupling between both solvers and reduce the data processing steps, the latter solver possesses a graph-network-based architecture. The presence of an extensive volume of data characterized by a graphical structure, such as social networks, citation networks, molecular compounds, etc., where heterogeneous relations exist between the various entities, urged the development of data-based approaches that operate directly on such data and that maintain the relational bias originating from the nodal connections. Interest in predictions,

on such data, exists at various levels: Graph-level, Node-level, or Edge-level. For a Graph-level prediction, a property for the entire graph is required, such as predicting the chemical properties of the molecular compound or associating a label for the whole image [91, 92]. Whereas, for a node-level prediction, a property is required for each node within the graph rather than a global property for the whole graph such as the semantic class of each pixel in an image [93]. Finally, for an edge-level prediction, identifying the relations between nodes and labeling the connections is the prediction target [94]. For all three levels of prediction, a single class of models, known as Graph Neural networks, is capable of handling the required tasks using the message-passing embeddings-update procedure proposed by Gilmer *et al.*, [32], and encompassed by the Graph Network framework later suggested in [34]. This class learns to progressively transform all input graph attributes, on the various levels, until attaining the required attributes on the target level, while preserving the graphical structure of the input graph. The process can be simply described in a three-step procedure that progresses by encoding the information in the data to the various attributes of the graph, updating these attributes through a series of message-passing GN blocks, and finally predicting the required level property. Thus, this architecture allows encoding the information on the discretization mesh to a computational graph without squandering the attained topological knowledge. The details of this process, along with all the required steps for operating on graphical data, are provided throughout this section in the scope of the current task tackled throughout this paper. The main blocks of the used model architecture, along with the required input and output fields, are also depicted in Figure 3.13.

To start, the current mesh state, M^t, is encoded into a graph, $G = (V, E)$, consisting of nodes, V, and edges, E. Each node in the graph is a representative of a vertex in the parent discretization mesh. The bidirectional edges connecting graph nodes together are an incarnation of the connections in the input mesh state. The model maps the features present in the input graph, G^{in}, to the output graph, G^{out}, through a series of convolutional operations on the node, $X_V \in \mathbb{R}^{N_V \times d_V}$, and edge feature matrices, $X_E \in \mathbb{R}^{N_E \times d_E}$, with N_V and N_E the number of nodes and edges, respectively, and d_V and d_E the dimensions of the constituting feature vectors. Nodes feature vectors, $\mathbf{x}_v \in \mathbb{R}^{d_V}$, consist of the required physical fields with,

$$\mathbf{x}_v^{in} = (u_x^n, u_y^n, \rho^n, b) \in \mathbb{R}^4 \text{ and } \mathbf{x}_v^{out} = (a_x^n, a_y^n, p^{n+1}) \in \mathbb{R}^3,$$

where $\mathbf{x}_v^{in}$ and $\mathbf{x}_v^{out}$ are the node feature vectors for the input, G^{in}, and output graphs, G^{out}, respectively. On the other hand, relative positional information is provided as edge features to achieve spatial equivariance with,

$$\mathbf{x}_{e,ij}^{in} = (\mathbf{u}_{ij}, |\mathbf{u}_{ij}|) \in \mathbb{R}^3,$$

where $\mathbf{u}_{ij}$ is the relative displacement vector between nodes i and j, and $|\mathbf{u}_{ij}|$ is its norm.

After allocating the information in the mesh state, M^t, to the node and edge feature matrices, the model encodes the information into a higher dimension latent space using a separate encoder for each feature matrix, *i.e.*, En^E and En^V. The encoding is performed using 2-layer MLPs, operating on every node and edge feature vector, with an output latent space dimension of $d_l = 128$.

$$\mathbf{x}_{e_{ij}} \leftarrow En^E\left(\mathbf{x}_{e_{ij}}^{in}\right) \text{ and } \mathbf{x}_{v_i} \leftarrow En^V\left(\mathbf{x}_{v_i}^{in}\right) \text{ with } \mathbf{x}_{e_{ij}}, \mathbf{x}_{v_i} \in \mathbb{R}^{d_l}$$

The high-dimension latent space vectors are further processed using a series of 10 message-passing Graph Net blocks [34]. The trainable parameters are unshared across the message-passing blocks, maintaining a unique set for every block and increasing the approximation power of the model as a whole. The output dimension of both the updated node and edge feature vectors is kept constant and equal to that of the encoding space, *i.e.*, $d_l = 128$, and residual connections are added between consecutive block updates.

The message-passing graph convolution operation consists of two main steps: an edge features update, followed by a node features update. The updated edge features, $\mathbf{x}'_{e_{ij}}$, are obtained by passing the current features, $\mathbf{x}_{e_{ij}}$, along with the corresponding node features, $\mathbf{x}_{v_i}$ and $\mathbf{x}_{v_j}$, to an edge convolution operator, f^E. Next, the updated features of the edges relative to a node are aggregated and passed, along with current node features, $\mathbf{x}_{v_i}$, to a node convolution operator, f^V, thus obtaining the updated node features. Both edge and node convolution operators, f^E and f^V, are implemented using 2-layer MLPs with 128 hidden neurons in each layer and a ReLU nonlinear activation function followed by a LayerNorm layer.

$$\mathbf{x}'_{e_{ij}} \leftarrow f^E\left(\mathbf{x}_{e_{ij}}, \mathbf{x}_{v_i}, \mathbf{x}_{v_j}\right) + \mathbf{x}_{e_{ij}} \text{ and } \mathbf{x}'_{v_i} \leftarrow f^V\left(\mathbf{x}_{v_i}, \sum_j \mathbf{x}'_{e_{ij}}\right) + \mathbf{x}_{v_i}$$

Finally, the updated node features, obtained after multiple message passing operations, are decoded to the required physical fields using a 2-layer 128 hidden neurons MLP and a linear output layer with a dimension equal to that of the required outputs, d_o.

$$\tilde{\mathbf{x}}_{v_i}^{out} \leftarrow De^V\left(\mathbf{x}_{v_i}\right)$$

3.7 Training

The model parameters are initially defined using the Glorot scheme, [95]. However, the model performance and inferring quality solely depend on the value of these parameters. A loss function, quantifying the error between the model predictions

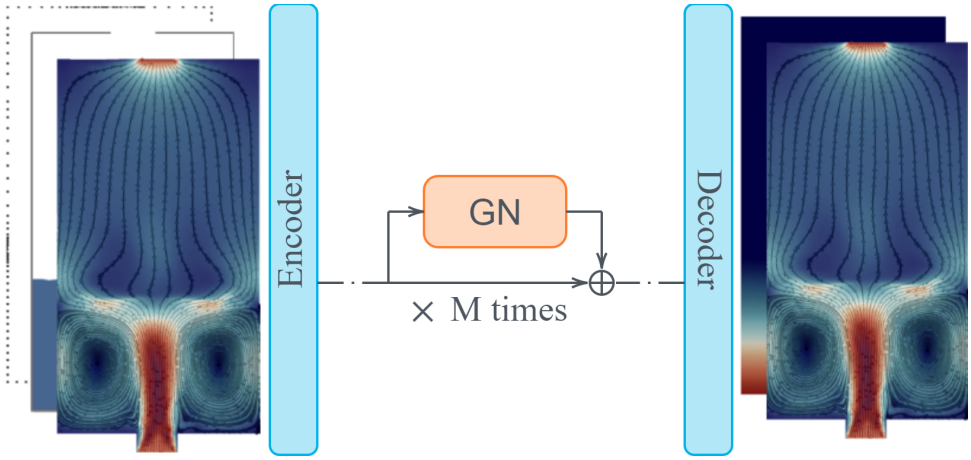


Figure 3.13: ***Model Architecture*** *with an Encoder-Processor-Decoder structure. The processor is constituted from multiple GN blocks with unshared parameters and residual connections.*

and the true computed CFD fields, is defined to determine the optimal set of parameters. Various choices of loss functions are available in the literature; however, in this work, the mean squared error, MSE, defined in Eq. 3.12, is utilized,

$$\mathcal{L} = \frac{1}{N_V \times d_o} \sum_{i=1}^{N_V} \sum_{j=1}^{d_o} (x_{v_i,j}^{out} - \tilde{x}_{v_i,j}^{out})^2 \tag{3.12}$$

where N_V is the number of nodes in the global graph, d_o is the dimension of the output node feature vector, $x_{v_i,j}^{out}$ is the j^{th} true component at node i and $\tilde{x}_{v_i,j}^{out}$ is the model inferred one.

The MSE is known for its popularity in regression models and ability to restrict model predictions of outliers, thus limiting the possibility of diverging the coupled solver due to mispredicted fields [96]. The training objective is to minimize the loss function over the training dataset by calibrating the value of the trainable parameters using a variant of the gradient descent algorithm. A subset of the training dataset, consisting of 8 randomly sampled snapshots, is assembled into a single global graph containing the disjoint subgraphs. The considered fields are priorly standardized to a mean of zero and a unity variance to avoid a feature-scale biased update mechanism. After computing the loss function based on the predicted fields and their corresponding labels, its gradient with respect to the model weights is determined using the well-known backpropagation algorithm [97]. The gradients are finally used to optimize the parameters using the Adam optimizer, [98], with a learning rate of $1e-4$. This process is repeated for a maximum of $375,000$ steps, and training is terminated using early stopping criteria based on a sequential validation

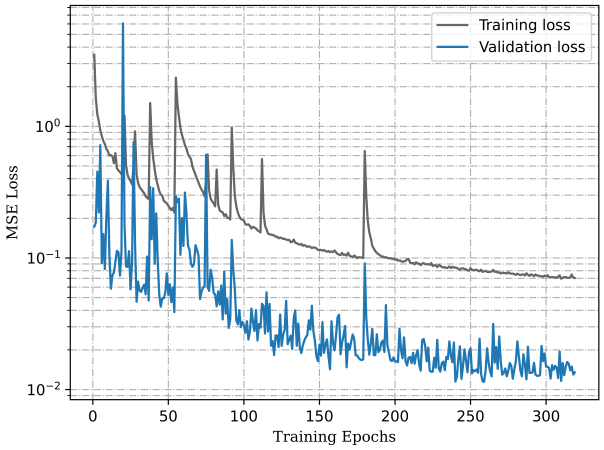


Figure 3.14: ***Training curve*** *showing the evolution of the MSE loss, for both the training and validation dataset, over the number of epochs.*

loss, as will be detailed later.

An in-house Tensorflow implementation of model architecture and training procedure is used [99]. The training of the $1,588,483$ parameters is performed on Nvidia Tesla V100 GPU for a total of 258 epochs. Figure 3.14 shows the evolution of the losses over the epochs prior to converging to an MSE of $7.91e-2$ and $1.14e-2$ for the training and validation error, respectively, without any signs of overfitting. The training of the $1,588,483$ parameters for 258 epochs over a single GPU device is accomplished within 23 hours. Although this training time may be considered as an offline cost and is of least importance compared to the online cost of inferring, it should be noted that shorter training times could be attained by releasing hardware limitation and employing distributed training, [100], or by employing various optimization and implementation methods that focuses on reducing this time and easing the exploration of the training parametric space [101].

The validation loss is the main parameter controlling the whole training process and the resulting model. Thus, particular attention is provided to the computation of this loss and its evolution through the training process. The primary purpose behind computing the validation loss is to search the hyperparameter space for the best model performance. This is also applicable in searching for the appropriate model trainable parameters, *i.e.*, its weights and biases. The early stopping criterion is employed to avoid overfitting the model's trainable parameters to the training dataset by terminating the training procedure based on the dynamics of the validation loss. If the model's performance over the validation dataset is no longer enhanced, the training is terminated, and the model with the lowest error metric is

chosen. For the following task under study, a patience window of 60 epochs is set to avoid very early stopping due to the noisy training nature. Usually, the validation loss inherits the exact nature of the training loss, meaning that it is computed using a similar metric and on an independent dataset representing the training dataset, *i.e.*, the validation dataset. However, since the main intention behind our model is to employ it as an incremental solver, the loss is evaluated over multiple validation trajectories, consisting of a sequence of snapshots rather than single-step inferrings. This guarantees, in addition to the model's ability to generalize to unseen datasets, its edge in rolling out in time over models trained with traditional validation procedures. It should be noted that to avoid lengthy training processes, the length of the trajectories is limited to only 50 steps. The evaluation is performed on a simplified problem where the thermodynamical parameters are approximated rather than computed using the CFD solver.

Another key parameter for employing the model as an incremental solver is the addition of noise during training. For the model to roll out in time, it should be able to dampen the error in its inferrings and avoid its accumulation if the predicted physical fields are required for the next-step inferring. This is rendered possible by initially exposing the model to input fields that are deteriorated by a normally distributed noise with a variance in the proximity of its single step inferring error. The model is also trained to infer deflected acceleration fields that aid in canceling the noise while computing the following step velocity fields, as suggested in [102]. To determine the amount of deflection, d, that must be annexed to the inferred normalized acceleration, $\bar{\boldsymbol{a}}^n$,while training, to have noise-free next-step velocity fields, $\boldsymbol{u}^{n+1}$, the integration update scheme is shown in Eq. 3.13,

$$\boldsymbol{u}^{n+1} = \big(\sigma_a(\bar{\boldsymbol{a}}^n + d) + \mu_a\big)\Delta t_m + \big(\sigma_u(\bar{\boldsymbol{u}}^n + \epsilon^n) + \mu_v\big), \tag{3.13}$$

where ϵ^n is the noise present in the input velocity, $\boldsymbol{u}^n$. σ and μ are, respectively, the standard deviation and the mean of their corresponding field, while Δt_m is the model's time step. Thus, to cancel the noise present in the input velocity fields, n, the model is required to infer the normalized acceleration fields, deflected by d and computed as shown in Eq. 3.14,

$$d = -\frac{\sigma_u}{\sigma_a \Delta t_m} \times n \tag{3.14}$$

To demonstrate the superiority of our model, trained with noisy inputs and deflected outputs and elected using early stopping criteria based on a sequential validation dataset, the dynamics of the flow fields are plotted and compared to those obtained with a basic model. The basic model is trained without any noise addition, and its training is terminated using the traditional early stopping criteria. Figure 3.15 shows the error plot of the inferred flows over the simulation time for

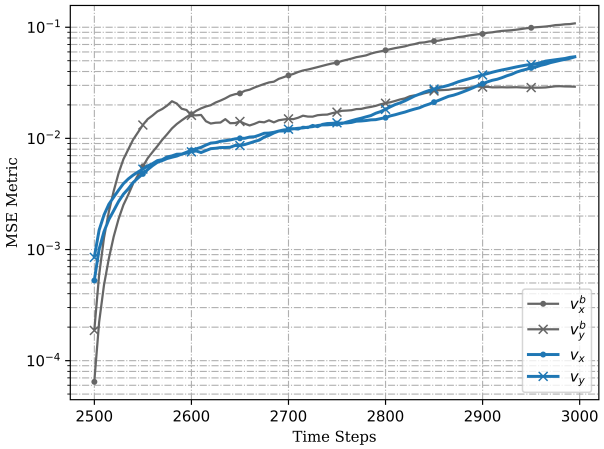


Figure 3.15: ***Plot of error for flow fields predictions*** *obtained using a basic model, shown in grey, and our suggested model, shown in blue. The recurrent correction frequency is set to infinity for both rollouts and an inlet velocity, $u_{x,in} = 0.75$ m/s, not shown during training, is used for comparison.*

both models, obtained with a recurrent correction frequency set to infinity, *i.e.*, solely relying on the trained model. Although initially, the basic model returns flow fields with a lower error metric due to the absence of any noise in the single-step training process, however, after a few steps, the accuracy of the inferrings starts to deteriorate compared to our trained model. Thus, the addition of noise and using a sequential validation dataset increases the robustness of the model against noisy inferrings and ensures longer rollout trajectories with a smaller overall error.

3.8 Results and Discussion

The framework's ability to span the time domain is initially assessed visually, as shown in Figures 3.16 and 3.17. The figures are obtained at various time steps and for multiple coupling frequencies. Although the training data is restricted to 500 snapshots from each trajectory, ranging between $TS = 2500$ and $TS = 3000$, the model's ability to extrapolate to unseen regimes is assessed by obtaining the fields at $TS = 3250$ and $TS = 3500$ shown in the last two columns of both figures, thus exceeding the training domain by almost 500 model time steps. The first column corresponds to the fields obtained at $TS = 2500$ while the last for $TS = 3500$, thus incrementing by a step of 250 between each column. The fields shown in the second row are obtained with a recurrent correction frequency of 1, meaning that the traditional solver is incorporated after every model inferring. However, for the

final row, the frequency is set to infinity, meaning that the model solely spans the whole time domain without any inference from the Navier-Stokes CFD solver.

Although visual agreement can be clearly recognized between various simulation setups, Figure 3.18 further investigates the quality of the coupling framework suggested in this work. Multiphase flows with a fluid filling from the bottom are usually characterized by a dome-shaped interface, as shown in Figure 3.2b. The dome's height, h, is of particular interest for comparison with experimental results or for validating the simulation accuracy. To evaluate the accuracy of the results incorporating a deep learning model to those obtained using the traditional finite element method, the dynamics of this metric over the simulation time are recorded and plotted for various correction frequencies, as shown in Figure 3.18. Although the value of the dome height, obtained from a framework relying solely on the deep learning model to resolve the flow equations, starts to exceed the true height, $f = 0$, after recycling the inferred fields for various steps, the margin of error is still acceptable as is shown with the grey plot labeled by $f = \text{inf}$. As expected, restricting the number of model predictions to a finite number, $f = 0$ and $f = 1$, redirects the solution fields and damps the accumulation of error, even for extended rollouts. Finally, it should be noted that the model temporal training domain is restricted to half the domain used during testing.

The l_2 normalized error is also computed, over the predicted flow velocities, to compare the different correction frequencies over a testing trajectory within the training domain time interval. The error metric, at every time step, is defined as shown in Eq. 3.15, and the results are shown in Table 3.1.

$$error = \frac{1}{2}\left(\frac{||u_x^n - \tilde{u}_x^n||_2}{||u_x^n||_2} + \frac{||u_y^n - \tilde{u}_y^n||_2}{||u_y^n||_2}\right), \tag{3.15}$$

where $||\cdot||_2$ is the l_2 norm of the physical field. u_x^n and u_y^n are respectively the true x and y components of the velocity field while $\tilde{u}_x^n$ and $\tilde{u}_y^n$ are their predicted counterparts at time step n. As is expected, as the frequency increases, the computed error metric will increase due to the less enforcing of governing laws imposed by the traditional finite element solver. The maximum average error is obtained for a simulation that relies solely on the deep learning model to span the whole time domain and is restricted to 1.29×10^{-1}. Finally, it can be noted that the maximum error for all frequencies occurs toward the final time steps due to the accumulation of errors throughout the simulation.

For investigating the attained reduction in computational cost, the time required for inferring the flow fields using the traditional finite element solver is recorded and compared to that required by the deep learning model on both a CPU and a GPU device and for various recurrent correction frequencies. Initially, the traditional solver requires around 122.09 seconds to span 500 incremental steps. Employing

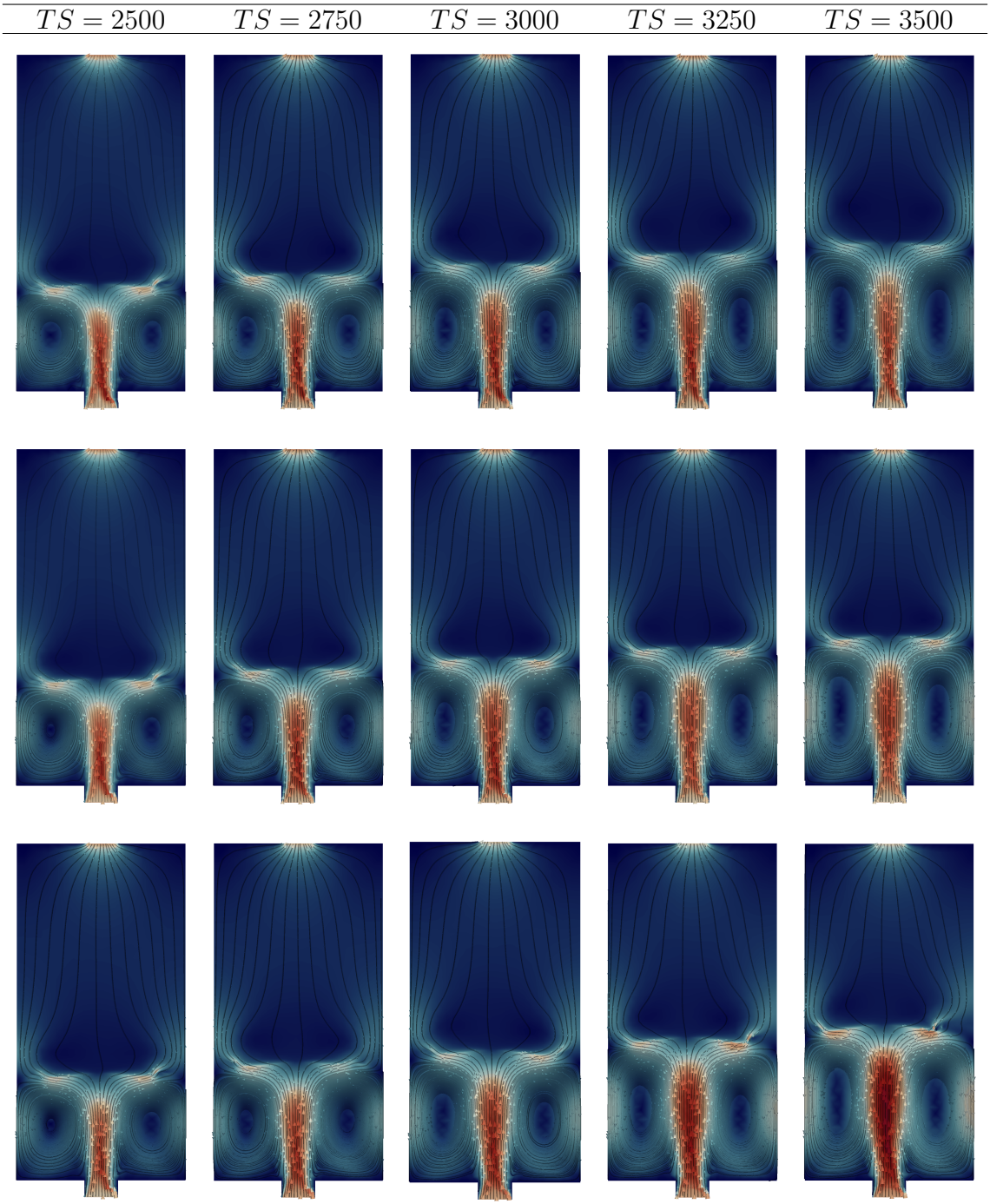


Figure 3.16: ***Visual comparison of the evolution of the velocity field for various correction frequencies.*** *The figures show the velocity fields along with their streamlines for an inlet velocity not observed by the model during training. For every frequency, various snapshots are shown, starting at* $TS = 2500$ *and incrementing by* 250 *steps for every column. Every row of figures corresponds to a separate frequency. The order of frequencies is:* 0, 1, *and* ∞, *where* $f = 0$ *corresponds to the true CFD results.*

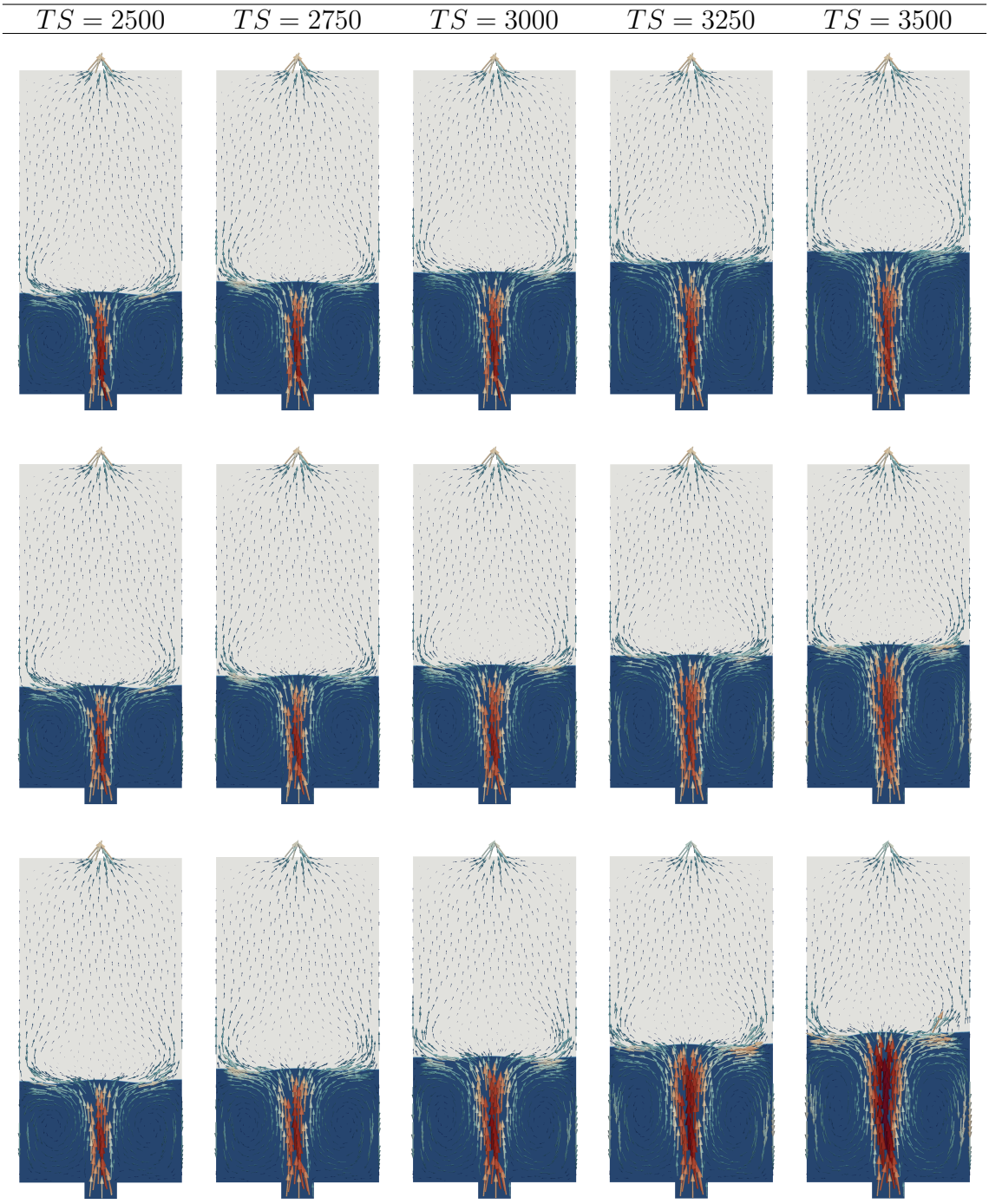


Figure 3.17: ***Visual comparison of the evolution of flow for various coupling frequencies.*** *The figures show the global density, along with the driving velocity field vectors, for an inlet velocity not observed by the model during training. For every frequency, various snapshots are shown, starting at* $TS = 2500$ *and incrementing by* 250 *steps for every column. Every row of figures corresponds to a separate correction frequency. The order of frequencies is:* 0, 1, *and* ∞, *where* $f = 0$ *corresponds to the true CFD results.*

f	1	2	∞
Average	9.44×10^{-2}	9.89×10^{-2}	1.29×10^{-1}
Maximum	1.31×10^{-1}	1.16×10^{-1}	1.56×10^{-1}
Time Step	2995	2995	3000

Table 3.1: ***Summary of obtained error.*** *The average of the l_2 normalized error is computed across all samples. The time step corresponds to the step at which the maximum error is found.*

the data-based model on the same CPU device and with the recurrent correction frequency set to 1, a reduction of around 4.9% is obtained. Increasing the recurrent frequency and relying more on the deep learning model leads to a further reduction in the computation cost. That said, with $f = 2$, the reduction increases to 6.5% and for $f \leftarrow \infty$, a reduction of 9.9% is attained. Knowing that Graph neural networks operate most efficiently on a graphical processing unit [103], an order of magnitude higher reduction in computational cost is obtained with the prediction time dropping to around 10.88 seconds, thus predicting the flow fields around 11 times faster than the traditional CFD solver. Further reduction is also possible by enhancing the current model implementation and considering various methods, such as pruning [104] and quantization [105], for reducing its inference time since the primary focus of this work was to investigate the coupling framework and its approximation capacity for multiphase flow problems without large consideration to the prediction speed.

Finally, to validate the choice of using acceleration as the model's output rather than directly inferring the required velocity fields, the performance of trained models with different output fields is compared. A model that predicts the velocity at the next time step rather than the change in the field, *i.e.*, the acceleration, is trained in a similar environment as the basic model previously described. Moreover, both models are trained without any noise addition and using traditional early stopping criteria to equate the odds between them. The physical fields used to asses the superiority of a model over another are obtained from simulations with an infinite correction frequency. This is done to limit the effect of external factors on the quality of predictions. Figure 3.19, showing the evolution of error over time for the various fields, clearly favors predicting acceleration rather than velocity.

3.9 Conclusion

In current contribution, a coupling framework between a traditional finite element CFD solver and a deep learning model was presented. This coupling framework was

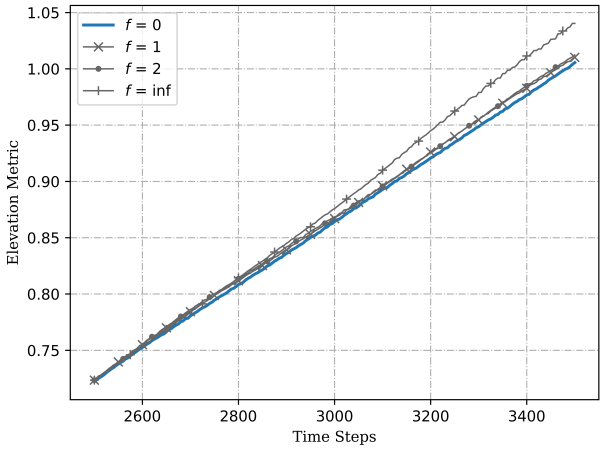
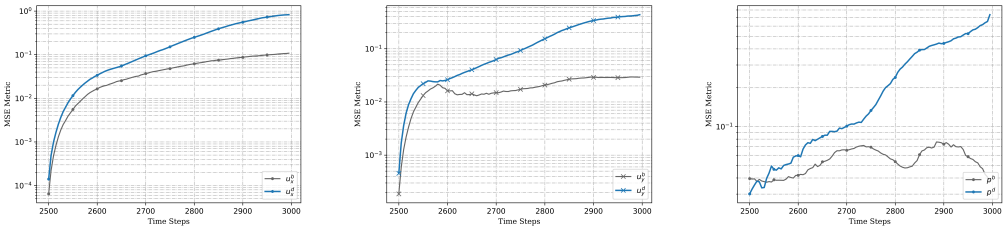


Figure 3.18: ***Interface metric evolution*** *for various coupling frequencies. The plot shown in blue and labeled with $f = 0$ represents the results obtained using solely CFD finite element solver. The remaining plots, shown in grey, represent results obtained using the introduced coupling framework for various frequencies.*



(a) *(b)* *(c)*

Figure 3.19: ***Roll-out performance comparison*** *for models trained with either acceleration (whose results shown in grey), or velocity (shown in blue) as their output field. The comparison is done for the obtained fields at the next time step: (a) u_x, (b) u_y, (c) p.*

employed to resolve the strongly coupled equations required for tackling multiphase flow problems with a level-set interface capturing method. Since most of the computational cost for resolving such a problem is monopolized by the flow equations' finite element solver, *i.e.*, around 98 % of the computational time is required for resolving the flow fields compared to only 2 % for resolving the level-set scalar field, the deep learning model was chosen to predict the evolution of the flow fields in time. The task of the model was facilitated by the proper choice of model targets, an appropriate training process, and the use of a sequential validation dataset to terminate training. Moreover, the graph-based model architecture, along with the suggested coupling framework, allowed for the dynamic adaptation of the unstructured irregular triangular mesh with the interface evolution. Various visual and numerical tests were used to evaluate the model's performance on new and unseen simulation trajectories. Finally, incorporating the finite element solver in the simulation trajectory damped the accumulated error resulting from the data approach inferring by reimposing the well-established physical laws. This is made evident by the computed error metric for different recurrent correction frequencies, where the l_2 normalized error for a frequency set to $f = 1$ averages at 9.44×10^{-2} while for $f = \inf$, the average is around 1.29×10^{-1}.

Although significant potential can be seen for the proposed method, mainly for finite values of recurrent correction frequencies, various ameliorations can be addressed to enhance its performance for infinite correction frequencies. Relying on a greedy procedure for the choice of training data samples or additionally constraining the model predictions using the already known physics are expected to enhance current performance and partly relieve the CFD solver from imposing the required physical constraints. Moreover, although the current framework is introduced for 2D multiphase flow problems, other flow setups with different characteristics or even new applications could be exploited with slight modifications. This abstraction is inherited from the constituting pillars' robustness. That is, both employed methods, the finite element technique, and the deep learning approach, have acquired reliability with the successful implementation of the former, over a wide range of applications [106], and the universality of the latter, secured by the universal approximation theorem [107, 108]. Also, various physical problems are governed by a similar set of equations, where the flow equations are coupled to a scalar transport equation and thus are foreseen as a fertile space for investigating the applicability of the current framework. Finally, the modern results attained by deep learning models over three-dimensional data, [109, 110], pave the path for extending the current framework, in future work, for simulating 3D multiphase flow problems where the impact of simulation time is of great importance.

Bibliography

[1] A. Prosperetti, G. Tryggvason, Computational methods for multiphase flow, Cambridge university press, 2009. 78

[2] J. Blazek, Computational fluid dynamics: principles and applications, Butterworth-Heinemann, 2015. 78

[3] M. Khalloufi, Y. Mesri, R. Valette, E. Massoni, E. Hachem, High fidelity anisotropic adaptive variational multiscale method for multiphase flows with surface tension, Computer Methods in Applied Mechanics and Engineering 307 (2016) 44–67. 78, 87

[4] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, in: 30th aerospace sciences meeting and exhibit, American Institute of Aeronautics and Astronautics, 1992, p. 439. 78

[5] S. Yang, Y. Wang, X. Chu, A survey of deep learning techniques for neural machine translation (2020). arXiv:2002.07526.
URL https://arxiv.org/abs/2002.07526 78

[6] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing (2018). arXiv:1708.02709. 78

[7] L. Deng, G. Hinton, B. Kingsbury, New types of deep neural network learning for speech recognition and related applications: an overview, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 8599–8603. doi:10.1109/ICASSP.2013.6639344. 78

[8] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: A brief review, Computational intelligence and neuroscience 2018 (2018). 78

[9] A. Patil, J. Viquerat, A. Larcher, G. El Haber, E. Hachem, Robust deep learning for emulating turbulent viscosities, Physics of Fluids 33 (10) (2021) 105118. doi:10.1063/5.0064458.
URL https://doi.org/10.1063/5.0064458 78, 89

[10] J. Chen, J. Viquerat, E. Hachem, U-net architectures for fast prediction of incompressible laminar flows, arXiv preprint arXiv:1910.13532 (2019). 78

[11] J. Viquerat, E. Hachem, A supervised neural network for drag prediction of arbitrary 2d shapes in laminar flows at low reynolds number, Computers & Fluids 210 (2020) 104645. doi:https:

//doi.org/10.1016/j.compfluid.2020.104645.
URL https://www.sciencedirect.com/science/article/pii/S0045793020302164 78

[12] Y. Zhang, W. J. Sung, D. N. Mavris, Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient. arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2018-1903, doi:10.2514/6.2018-1903.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2018-1903 78

[13] F. J. Gonzalez, M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, ArXiv abs/1808.01346 (2018). 79

[14] Deep Convolutional Recurrent Autoencoders for Flow Field Prediction, Vol. Volume 8: CFD and FSI of International Conference on Offshore Mechanics and Arctic Engineering. doi:10.1115/OMAE2020-18556.
URL https://doi.org/10.1115/OMAE2020-18556

[15] I. K. Deo, R. Jaiman, Predicting waves in fluids with deep neural network, Physics of Fluids 34 (6) (2022) 067108. doi:10.1063/5.0086926.
URL https://doi.org/10.1063/5.0086926

[16] R. Gupta, R. Jaiman, A hybrid partitioned deep learning methodology for moving interface and fluid–structure interaction, Computers & Fluids 233 (2022) 105239. doi:https://doi.org/10.1016/j.compfluid.2021.105239.
URL https://www.sciencedirect.com/science/article/pii/S0045793021003479 79

[17] G. El Haber, J. Viquerat, A. Larcher, D. Ryckelynck, J. Alves, A. Patil, E. Hachem, Deep learning model to assist multiphysics conjugate problems, Physics of Fluids 34 (1) (2022) 015131. 79, 89, 92, 94

[18] S. Fotiadis, E. Pignatelli, M. L. Valencia, C. Cantwell, A. Storkey, A. A. Bharath, Comparing recurrent and convolutional neural networks for predicting wave propagation (2020). doi:10.48550/ARXIV.2002.08981.
URL https://arxiv.org/abs/2002.08981 79

[19] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, in: Computer graphics forum, Vol. 38, Wiley Online Library, 2019, pp. 71–82. 79

[20] M. Lino, C. Cantwell, S. Fotiadis, E. Pignatelli, A. Bharath, Simulating surface wave dynamics with convolutional networks, arXiv preprint arXiv:2012.00718 (2020). 79

[21] J. Zhao, T. Zhu, R. Zhao, P. Zhao, Layerwise recurrent autoencoder for real-world traffic flow forecasting, in: Intelligence Science and Big Data Engineering. Big Data and Machine Learning, Springer International Publishing, 2019, pp. 78–88. 79

[22] S. Sun, H. Wu, L. Xiang, City-wide traffic flow forecasting using a deep convolutional neural network, Sensors 20 (2) (2020). doi:10.3390/s20020421.
URL https://www.mdpi.com/1424-8220/20/2/421 79

[23] K. Endo, K. Tomobe, K. Yasuoka, Multi-step time series generator for molecular dynamics, Proceedings of the AAAI Conference on Artificial Intelligence 32 (1) (Apr. 2018). doi:10.1609/aaai.v32i1.11863.
URL https://ojs.aaai.org/index.php/AAAI/article/view/11863 79

[24] J. C. S. Kadupitiya, G. C. Fox, V. Jadhao, Solving newton's equations of motion with large timesteps using recurrent neural networks based operators, Machine Learning: Science and Technology 3 (2) (2022) 025002. doi:10.1088/2632-2153/ac5f60.
URL https://doi.org/10.1088/2632-2153/ac5f60

[25] J. Kadupitiya, F. Sun, G. Fox, V. Jadhao, Machine learning surrogates for molecular dynamics simulations of soft materials, Journal of Computational Science 42 (2020) 101107. doi:10.1016/j.jocs.2020.101107. 79

[26] A. Demirkaya, T. Imbiriba, K. Lockwood, S. Rampersad, E. Alhajjar, G. Guidoboni, Z. Danziger, D. Erdoğmuş, Cubature kalman filter based training of hybrid differential equation recurrent neural network physiological dynamic models, in: 2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), 2021, pp. 763–766. doi:10.1109/EMBC46164.2021.9631038. 79

[27] N. Aloysius, M. Geetha, A review on deep convolutional neural networks, in: 2017 International Conference on Communication and Signal Processing (ICCSP), 2017, pp. 0588–0592. doi:10.1109/ICCSP.2017.8286426. 79

[28] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, T. Chen, Recent advances in convolutional neural networks, Pattern Recognition 77 (2018) 354–377. doi:https://doi.org/10.1016/j.patcog.2017.10.013.
URL https://www.sciencedirect.com/science/article/pii/S0031320317304120 79

[29] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1) (2009) 61–80. doi:10.1109/TNN.2008.2005605. 79

[30] J. Bruna, W. Zaremba, A. Szlam, Y. Lecun, Spectral networks and locally connected networks on graphs, in: International Conference on Learning Representations (ICLR2014), CBLS, April 2014, 2014. 79

[31] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016). 79

[32] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: Proceedings of the 34th International Conference on Machine Learning, Vol. 70 of Proceedings of Machine Learning Research, PMLR, 2017, pp. 1263–1272. 79, 96, 100

[33] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017.
URL https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[34] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, arXiv (2018).
URL https://arxiv.org/pdf/1806.01261.pdf 79, 96, 100, 101

[35] F. De Avila Belbute-Peres, T. Economon, Z. Kolter, Combining differentiable PDE solvers and graph neural networks for fluid flow prediction, in: H. D. III, A. Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, Vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 2402–2411.
URL https://proceedings.mlr.press/v119/de-avila-belbute-peres20a.html 79

[36] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, P. Velickovic, ETA Prediction with Graph Neural Networks in Google Maps, Association for Computing Machinery, New

York, NY, USA, 2021, p. 3767–3776.
URL https://doi.org/10.1145/3459637.3481916 80

[37] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, P. Battaglia, Graph networks as learnable physics engines for inference and control, in: International Conference on Machine Learning, PMLR, 2018, pp. 4470–4479.

[38] H. Shao, T. Kugelstadt, T. Hädrich, W. Pałubicki, J. Bender, S. Pirk, D. L. Michels, Accurately solving physical systems with graph learning, arXiv preprint arXiv:2006.03897 (2020). 80

[39] J. Chen, E. Hachem, J. Viquerat, Graph neural networks for laminar flow prediction around random two-dimensional shapes, Physics of Fluids 33 (12) (2021) 123607. arXiv:https://doi.org/10.1063/5.0064108, doi:10.1063/5.0064108.
URL https://doi.org/10.1063/5.0064108 80

[40] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. W. Battaglia, Learning to simulate complex physics with graph networks, in: International Conference on Machine Learning, 2020. 80

[41] C. Yang, W. Gao, D. Wu, C. Wang, Learning to simulate unseen physical systems with graph neural networks, arXiv preprint arXiv:2201.11976 (2022). 80

[42] Q. Hernandez, A. Badias, F. Chinesta, E. Cueto, Thermodynamics-informed graph neural networks, IEEE Transactions on Artificial Intelligence (2022) 1–1doi:10.1109/TAI.2022.3179681. 80

[43] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P. W. Battaglia, Learning mesh-based simulation with graph networks, in: International Conference on Learning Representations, 2021. 80

[44] W. Dang, Z. Gao, L. Hou, D. Lv, S. Qiu, G. Chen, A novel deep learning framework for industrial multiphase flow characterization, IEEE Transactions on Industrial Informatics 15 (11) (2019) 5954–5962. doi:10.1109/TII.2019.2908211. 80

[45] Y. Wang, G. Lin, Efficient deep learning techniques for multiphase flow simulation in heterogeneous porousc media, Journal of Computational Physics 401 (2020) 108968. doi:https://doi.org/10.1016/j.jcp.2019.108968. 80

[46] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, S. M. Benson, U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow, Advances in Water Resources 163 (2022) 104180. doi:https://doi.org/10.1016/j.advwatres.2022.104180. 80

[47] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations (2020). doi:10.48550/ARXIV.2010.08895. 80

[48] L. Yang, Y. Wang, Z. Zhao, Y. Guo, S. Chen, W. Zhang, X. Guo, Particle-laden droplet-driven triboelectric nanogenerator for real-time sediment monitoring using a deep learning method, ACS applied materials & interfaces 12 (34) (2020) 38192–38201. 80

[49] S. Balachandar, W. Moore, G. Akiki, K. Liu, Toward particle-resolved accuracy in euler–lagrange simulations of multiphase flow using machine learning and pairwise interaction extended point-particle (piep) approximation, Theoretical and Computational Fluid Dynamics 34 (4) (2020) 401–428. 80

[50] A. Seyed-Ahmadi, A. Wachs, Physics-inspired architecture for neural network modeling of forces and torques in particle-laden flows, Computers & Fluids 238 (2022) 105379. doi:https://doi.org/10.1016/j.compfluid.2022.105379. 80

[51] Y. Jiang, J. Kolehmainen, Y. Gu, Y. G. Kevrekidis, A. Ozel, S. Sundaresan, Neural-network-based filtered drag model for gas-particle flows, Powder Technology 346 (2019) 403–413. doi:https://doi.org/10.1016/j.powtec.2018.11.092. 81

[52] H. Wang, M. Zhang, Y. Yang, Machine learning for multiphase flowrate estimation with time series sensing data, Measurement: Sensors 10-12 (2020) 100025. doi:https://doi.org/10.1016/j.measen.2020.100025.

[53] R. Xu, D. Zhang, M. Rong, N. Wang, Weak form theory-guided neural network (tgnn-wf) for deep learning of subsurface single- and two-phase flow, Journal of Computational Physics 436 (2021) 110318. doi:https://doi.org/10.1016/j.jcp.2021.110318.

[54] H. Zheng, Z. Huang, G. Lin, A physics-constrained neural network for multiphase flows, Physics of Fluids (2022). arXiv:https://doi.org/10.1063/5.0111275, doi:10.1063/5.0111275.

[55] M. Shirzadi, T. Fukasawa, K. Fukui, T. Ishigami, Application of deep learning neural networks for the analysis of fluid-particle dynamics in fibrous filters,

Chemical Engineering Journal 455 (2023) 140775. doi:https://doi.org/10.1016/j.cej.2022.140775. 81

[56] D. A. Drew, Mathematical modeling of two-phase flow, Annual Review of Fluid Mechanics 15 (1) (1983) 261–291. doi:10.1146/annurev.fl.15.010183.001401.
URL https://doi.org/10.1146/annurev.fl.15.010183.001401 82

[57] D. A. Drew, S. L. Passman, Theory of multicomponent fluids, Vol. 135, Springer Science & Business Media, 2006.

[58] H. Enwald, E. Peirano, A.-E. Almstedt, Eulerian two-phase flow theory applied to fluidization, International Journal of Multiphase Flow 22 (1996) 21–66. 82

[59] S. Subramaniam, Lagrangian–eulerian methods for multiphase flows, Progress in Energy and Combustion Science 39 (2) (2013) 215–245. doi:https://doi.org/10.1016/j.pecs.2012.10.003.
URL https://www.sciencedirect.com/science/article/pii/S0360128512000603 82

[60] H. Braess, P. Wriggers, Arbitrary lagrangian eulerian finite element analysis of free surface flow, Computer methods in applied mechanics and engineering 190 (1-2) (2000) 95–109. 82

[61] J. Monaghan, A. Kocharyan, Sph simulation of multi-phase flow, Computer Physics Communications 87 (1) (1995) 225–235, particle Simulation Methods. doi:https://doi.org/10.1016/0010-4655(94)00174-Z.
URL https://www.sciencedirect.com/science/article/pii/001046559400174Z 82

[62] S. Tenneti, R. Garg, S. Subramaniam, Drag law for monodisperse gas–solid systems using particle-resolved direct numerical simulation of flow past fixed assemblies of spheres, International journal of multiphase flow 37 (9) (2011) 1072–1092.

[63] A. M. Vijayan, D. A. Levin, Kinetic modeling of fractal aggregate mobility, Physics of Fluids 34 (4) (2022) 043315. 82

[64] D. J. Benson, Volume of fluid interface reconstruction methods for multi-material problems, Applied Mechanics Reviews 55 (2) (2002) 151–165. arXiv:https://asmedigitalcollection.asme.org/appliedmechanicsreviews/article-pdf/55/2/151/5439062/151_1.pdf, doi:10.1115/1.1448524.
URL https://doi.org/10.1115/1.1448524 82

[65] S. Osher, R. P. Fedkiw, Level set methods: An overview and some recent results, Journal of Computational Physics 169 (2) (2001) 463–502. doi:https://doi.org/10.1006/jcph.2000.6636.
URL https://www.sciencedirect.com/science/article/pii/S0021999100966361 82

[66] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.-J. Jan, A front-tracking method for the computations of multiphase flow, Journal of Computational Physics 169 (2) (2001) 708–759. doi:https://doi.org/10.1006/jcph.2001.6726.
URL https://www.sciencedirect.com/science/article/pii/S0021999101967269 82

[67] M. Sussman, P. Smereka, S. Osher, A level set approach for computing solutions to incompressible two-phase flow, Journal of Computational Physics 114 (1) (1994) 146 – 159. 83

[68] D. Enright, R. Fedkiw, J. Ferziger, I. Mitchell, A hybrid particle level set method for improved interface capturing, Journal of Computational physics 183 (1) (2002) 83–116. 83

[69] F. S. Schranner, J. A. Domaradzki, S. Hickel, N. A. Adams, Assessing the numerical dissipation rate and viscosity in numerical simulations of fluid flows, Computers & Fluids 114 (2015) 84–97. 83

[70] C. Bilger, M. Aboukhedr, K. Vogiatzaki, R. Cant, Evaluation of two-phase flow solvers using level set and volume of fluid methods, Journal of Computational Physics 345 (2017) 665–686. 83

[71] E. W. Weisstein, Heaviside step function, https://mathworld. wolfram. com/ (2002). 84

[72] K. Yokoi, A density-scaled continuum surface force model within a balanced force formulation, Journal of Computational Physics 278 (2014) 221–228. 84

[73] S. V. Patankar, Numerical heat transfer and fluid flow, CRC press, 2018. 84

[74] S. V. Patankar, A numerical method for conduction in composite materials, flow in irregular geometries and conjugate heat transfer, in: International Heat Transfer Conference Digital Library, Begel House Inc., 1978. 84

[75] S. El Aouad, A. Larcher, E. Hachem, Anisotropic adaptive body-fitted meshes for cfd, Computer Methods in Applied Mechanics and Engineering 400 (2022) 115562. doi:https://doi.org/10.1016/j.cma.2022.115562. 85

[76] D. Boffi, F. Brezzi, M. Fortin, et al., Mixed finite element methods and applications, Vol. 44, Springer, 2013. 86

[77] T. J. Hughes, Multiscale phenomena: Green's functions, the dirichlet-to-neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods, Computer Methods in Applied Mechanics and Engineering 127 (1) (1995) 387–401. doi:https://doi.org/10.1016/0045-7825(95)00844-9. 86

[78] F. Brezzi, L. Franca, T. Hughes, A. Russo, $b = \alpha g$, Computer Methods in Applied Mechanics and Engineering 145 (3) (1997) 329–339. doi:https://doi.org/10.1016/S0045-7825(96)01221-2.
URL https://www.sciencedirect.com/science/article/pii/S0045782596012212 86

[79] F. Brezzi, M. Fortin, Mixed and hybrid finite element methods, Vol. 15, Springer Science & Business Media, 2012.

[80] T. Dubois, F. Jauberteau, R. Temam, Dynamic multilevel methods and the numerical simulation of turbulence, Cambridge University Press, 1999. 86

[81] L. P. Franca, A. Nesliturk, On a two-level finite element method for the incompressible navier–stokes equations, International Journal for Numerical Methods in Engineering 52 (4) (2001) 433–453. 86

[82] A. I. Nesliturk, Approximating the incompressible Navier-Stokes equations using a two-level finite element method, University of Colorado at Denver, 1999. 86

[83] R. Codina, Stabilized finite element approximation of transient incompressible flows using orthogonal subscales, Computer Methods in Applied Mechanics and Engineering 191 (39) (2002) 4295–4321. doi:https://doi.org/10.1016/S0045-7825(02)00337-7.
URL https://www.sciencedirect.com/science/article/pii/S0045782502003377 87

[84] R. Valette, A. Pereira, S. Riber, L. Sardo, A. Larcher, E. Hachem, Viscoplastic dam-breaks, Journal of Non-Newtonian Fluid Mechanics 287 (2021) 104447. doi:https://doi.org/10.1016/j.jnnfm.2020.104447.
URL https://www.sciencedirect.com/science/article/pii/S0377025720301919 87

[85] E. Hachem, M. Khalloufi, J. Bruchon, R. Valette, Y. Mesri, Unified adaptive variational multiscale method for two phase compressible–incompressible

flows, Computer Methods in Applied Mechanics and Engineering 308 (2016) 238–255. doi:https://doi.org/10.1016/j.cma.2016.05.022.
URL https://www.sciencedirect.com/science/article/pii/S0045782516304236 87

[86] A. Patil, J. Viquerat, G. E. Haber, E. Hachem, Robust deep learning for emulating turbulent viscosities (2021). arXiv:2107.11235. 94

[87] Y. Mesri, H. Digonnet, T. Coupez, Advanced parallel computing in material forming with cimlib, European Journal of Computational Mechanics 18 (7-8) (2009) 669–694. arXiv:https://doi.org/10.3166/ejcm.18.669-694, doi:10.3166/ejcm.18.669-694.
URL https://doi.org/10.3166/ejcm.18.669-694 94

[88] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Advances in neural information processing systems 29 (2016). 96

[89] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR, 2015, p. 448–456. 99

[90] J. L. Ba, J. R. Kiros, G. E. Hinton, Layer normalization, arXiv preprint arXiv:1607.06450 (2016). 99

[91] A. M. Schweidtmann, J. G. Rittig, A. Konig, M. Grohe, A. Mitsos, M. Dahmen, Graph neural networks for prediction of fuel ignition quality, Energy & fuels 34 (9) (2020) 11395–11407. 100

[92] J. Long, et al., A graph neural network for superpixel image classification, in: Journal of Physics: Conference Series, Vol. 1871, IOP Publishing, 2021, p. 012071. 100

[93] X. Qi, R. Liao, J. Jia, S. Fidler, R. Urtasun, 3d graph neural networks for rgbd semantic segmentation, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5199–5208. 100

[94] J. Kim, T. Kim, S. Kim, C. D. Yoo, Edge-labeling graph neural network for few-shot learning, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 11–20. 100

[95] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Y. W. Teh, M. Titterington (Eds.), Proceedings of the

Thirteenth International Conference on Artificial Intelligence and Statistics, Vol. 9 of Proceedings of Machine Learning Research, PMLR, Chia Laguna Resort, Sardinia, Italy, 2010, pp. 249–256.
URL https://proceedings.mlr.press/v9/glorot10a.html 101

[96] V. Cherkassky, Y. Ma, Comparison of loss functions for linear regression, in: 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541), Vol. 1, IEEE, 2004, pp. 395–400. 102

[97] R. Hecht-Nielsen, Theory of the backpropagation neural network, in: Neural networks for perception, Elsevier, 1992, pp. 65–93. 102

[98] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.
URL http://arxiv.org/abs/1412.6980 102

[99] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/ 103

[100] C.-C. Chen, C.-L. Yang, H.-Y. Cheng, Efficient and robust parallel dnn training through model parallelism on multi-gpu platform, arXiv preprint arXiv:1809.02839 (2018). 103

[101] K. S. Chahal, M. S. Grover, K. Dey, R. R. Shah, A hitchhiker's guide on distributed training of deep neural networks, Journal of Parallel and Distributed Computing 137 (2020) 65–76. doi:https://doi.org/10.1016/j.jpdc.2019.10.004.
URL https://www.sciencedirect.com/science/article/pii/S0743731518308712 103

[102] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. W. Battaglia, Learning to simulate complex physics with graph networks, in: International Conference on Machine Learning, 2020. 104

[103] Z. Zhang, J. Leng, L. Ma, Y. Miao, C. Li, M. Guo, Architectural implications of graph neural networks, IEEE Computer architecture letters 19 (1) (2020) 59–62. 109

[104] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, arXiv preprint arXiv:1611.06440 (2016). 109

[105] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 2704–2713. 109

[106] M. G. Larson, F. Bengzon, The finite element method: theory, implementation, and applications, Vol. 10, Springer Science & Business Media, 2013. 111

[107] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (4) (1989) 303–314. 111

[108] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366. 111

[109] E. Trivizakis, G. C. Manikis, K. Nikiforaki, K. Drevelegas, M. Constantinides, A. Drevelegas, K. Marias, Extending 2-d convolutional neural networks to 3-d for advancing deep learning cancer classification with application to mri liver tumor differentiation, IEEE Journal of Biomedical and Health Informatics 23 (3) (2019) 923–930. doi:10.1109/JBHI.2018.2886276. 111

[110] M. Hillier, F. Wellmann, B. Brodaric, E. de Kemp, E. Schetselaar, Three-dimensional structural geological modeling using graph neural networks, Mathematical Geosciences 53 (8) (2021) 1725–1749. 111

Chapter 4

Tackling the Curse of Dimensionality in DL Models

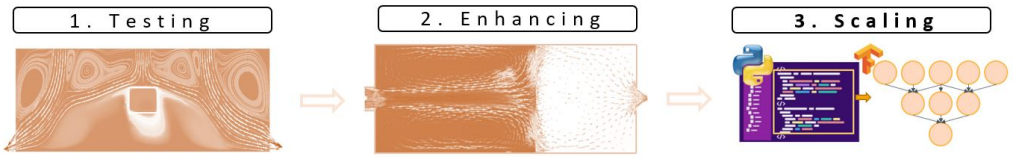


Figure 4.1: Organization of current thesis

Abstract *Throughout the last decade, multiple disciplines and industries have tended toward incorporating and relying on data-based approaches, including the CFD community, where various efforts were exerted to either infer a physical field, accelerate a traditional solver, or enhance an obtained solution. The mastery of these approaches in multiple tasks, previously rendered impossible or challenging to perform, along with their contribution to simplifying or accelerating various problem solutions, directed the current research toward investigating the maximum potential of these models on more complex tasks. All of the above fueled the current trend of relying on large models with millions of parameters and their training on massive and complex datasets. However, the attained boost in approximation capacity powered by the more significant number of trainable parameters is associated with various caveats. One of these significant drawbacks is the additional pressure on the current hardware infrastructure due to the immense growth in the required memory footprint. This issue squanders the ambitions of various research groups in further exploring the potential of deep learning models and necessitates joining efforts to handle it prior to exploring more complicated problems. Thus, this work falls under the third and final stage of the following manuscript ogranization, as shown in figure 4.1, where it aims at complementing the previous suggested frameworks with advanced implementation techniques to tackle memory problem and allow future scal-*

ing to higher dimensional problems. Precisely, it focuses on exploring an assembly of techniques that address this issue while avoiding mutating the model's architecture or deteriorating its accuracy. The result of inheriting these techniques, in the context of Computational Fluid Dynamics problems, is highlighted through the training of a Graph Neural Network for inferring the Navier Stokes Flow Fields in a multiphase flow problem with dynamically adapted anisotropic triangular mesh. Finally, an 78 % reduction in the habitually required memory, accompanied by eight times shorter training step times, is made possible by incorporating various techniques simultaneously, as will be detailed.

Abstract *Tout au long de la dernière décennie, de nombreuses disciplines et industries ont montré une tendance à intégrer et à dépendre de méthodes basées sur les données, y compris la communauté de la CFD, où des efforts considérables ont été déployés pour soit inférer un champ physique, soit accélérer un solveur traditionnel, soit améliorer une solution obtenue. La maîtrise de ces approches dans de multiples tâches, auparavant jugées impossibles ou difficiles à réaliser, ainsi que leur contribution à la simplification ou à l'accélération de diverses solutions de problèmes, ont orienté la recherche actuelle vers l'exploration du potentiel maximal de ces modèles sur des tâches plus complexes. Tous les éléments ci-dessus ont renforcé la tendance actuelle qui consiste à s'appuyer sur de vastes modèles dotés de millions de paramètres et à les entraîner sur des ensembles de données massifs et complexes. Néanmoins, l'augmentation de la capacité d'approximation obtenue grâce à un nombre plus important de paramètres ajustables est assortie de plusieurs inconvénients. L'un de ces inconvénients majeurs réside dans la pression supplémentaire exercée sur l'infrastructure matérielle actuelle en raison de l'immense croissance de la taille mémoire requise.Cette problématique compromet les ambitions de divers groupes de recherche en ce qui concerne l'exploration plus approfondie du potentiel des modèles de deep learning et nécessite une collaboration pour la résoudre avant d'aborder des problèmes plus complexes. Ainsi, ce travail s'inscrit dans la troisième et dernière étape de l'organisation du manuscrit suivante, comme le montre la figure 4.1, où il vise à compléter les cadres suggérés précédemment avec des techniques d'implémentation avancées pour résoudre le problème de la mémoire et permettre une extension future vers des problèmes de dimensions supérieures. Précisément, il se concentre sur l'exploration d'un ensemble de techniques visant à résoudre cette problématique tout en évitant de modifier l'architecture du modèle ou de compromettre sa précision. Les résultats de l'application de ces techniques, dans le contexte des problèmes de dynamique des fluides numérique, sont mis en évidence par l'entraînement d'un réseau neuronal graphique pour l'inférence des champs d'écoulement de Navier-Stokes dans un problème d'écoulement multiphasique avec un maillage triangulaire anisotrope adaptatif dynamique. Finalement, une*

réduction de 78 % de la mémoire habituellement requise, accompagnée de temps de formation huit fois plus courts, est rendue possible en incorporant simultanément diverses techniques, comme cela sera détaillé.

Contents

4.1 Introduction

In the modern era, the industrial landscape heavily depends on numerical simulation tools to expedite operations. Many of these processes encompass fluid flows coupled with other physical phenomena. Examples of such multiphysics problems include the turbulent flow past an obstacle [1], the cooling or heating of a workpiece using either natural or forced convection [2, 3], fluid-structure interaction [4], multiphase flows [5], and a wide range of other captivating scenarios. Unfortunately, solving multiphysics problems, particularly with multiphase flows, remains challenging, despite the scientific community's continuous efforts to forge cutting-edge numerical solvers. This challenge arises from the substantial computational burden entailed for resolving large governing systems, compromising a multitude of partial differential equations, along with the high dimensionality and nonlinearity of Navier-Stokes equations [5]. Conversely, the demonstrated potential of data-based methodologies in diverse domains like computer vision [6], machine translation [7], speech recognition [8], and natural language processing [9], has positioned them as promising candidates for tackling the aforementioned challenge. Consequently, the fluid mechanics community swiftly integrated these approaches with a wide range of applications while embracing a multitude of architectures. Initially, Deep learning models based on multilayer perceptrons were first introduced; Raissi et Al. [10] employed them for predicting the flow fields at specific locations within the domain while informing the loss function of the governing equations. Similarly, in [11], a model is used to predict problem-specific coefficients for estimating spatial derivatives in shock formation phenomena. Furthermore, in [12], the authors utilized an MLP to predict the out-of-plane 1D function for resolving heat problems using Proper Generalized Decomposition (PGD). Subsequently, models based on convolutional neural networks gained growing interest owing to their remarkable traits in extracting spatial features, relying on local connectivity, and sharing parameters across the input space. These models were employed in diverse applications such as forecasting drag and lift forces [13, 14], predicting the flow [15], or inferring scalar field in multiphysics problems [16, 17]. Finally, models that operate directly on graphical data started to gain popularity due to their ability to operate directly on unstructured discretization mesh, thus preserving the topological information [18, 19]. For instance, in the work conducted by Chen et al. [20], an encoder-decoder Graph Neural Network was employed to predict the steady-state laminar flow field past random 2D obstacles. Similarly, Belbute-Peres [21] utilized these models for ameliorating the solution of a low-resolution Navier-Stokes solver.

Inspecting the trend of architectures utilized in the CFD community within the last years indicates a preference toward relying on larger and more complex models. Various factors encouraged this expansion in the size of DL models. The increase in data availability, supported by advancements in data-collection technologies, open

data initiatives, and data augmentation techniques, motivated researchers to investigate larger datasets and extract more complex patterns and representations. The model's capacity to extract abstract information from datasets strongly depends on its number of trainable parameters and architecture [22]. Model scaling techniques, such as depth scaling [23, 24], width scaling [25], resolution scaling [26], or even compound scaling [27, 28], are known to lead to lower error at the expense of the number of floating point operations. These results, coupled with the growing demand for attaining higher model performance and accuracy in various fields, drove researchers to compete in designing complex architectures with a large number of parameters. Examples of model architectures that have been known for their state-of-the-art results on various benchmarks include: ResNet-50 for image classification, segmentation and object detection with over 23 million parameters [29], VGG-19 (Visual Geometry Group-19) also used in the field of computer vision and relies on more than 138 million parameters [23], BERT (Bidirectional Encoder Representations from Transformers) for natural language processing tasks with around 345 million parameters [30], and finally GPT-3 (Generative Pre-trained Transformer 3), also used for natural language processing but with 175 billion parameters [31]. The following set of examples strongly highlights the current trend of increasing model sizes and the number of parameters in deep learning research.

The current hype of large complex deep learning models trained on massive datasets imposes various challenges, mainly concerning computational resources. Starting from the memory required for loading and preprocessing the extensive input data, moving to the required resources for the model's trainable parameters, and finally to the memory necessary for storing intermediate values, later used for performing backpropagation [32], all of the following subjects the training process to the well-known out-of-memory (OOM) error [33]. Initial attempts to mitigate such an error might be at the expense of the model's accuracy. To alleviate the OOM error, practitioners might lean toward pruning the model architecture, reducing the model's depth or width, or using shared parameters across model layers [34, 35]. However, such actions compromise the model's capacity to learn complex representations and might weaken its performance [36]. Others might rely on cutting off the number of samples in a batch to reduce the number of intermediate variables and, thus, the peak memory required for a single update step. However, this results in a noisy gradient estimate, thus threatening the stability and accuracy of the updates and questioning their efficiency [37, 38]. Influenced by reducing the training samples in a batch, Hu et Al. suggested a novel training methodology [39], specific for physics-informed neural networks with high-dimensional partial differential equations, in which the gradient updates are computed based on a random subset of the residual gradients across the dimensions. Although this approach returned interesting results on various equations, however, it remains directly bound

to residual-based models for high-dimensional partial differential equations. Finally, limited memory resources also hinder tuning the model parameters and exploring its hyper-parametric space, thus leading to suboptimal model parameter settings and inferior accuracy [40].

Various methods can be found in the literature to overcome memory limitations and enable the training of complex models with large datasets. Distributed training is one of the common approaches that rely on expanding the hardware infrastructure to compensate for the additional memory requirements and reduce the required training time [41], mainly for GPUs with limited per-unit storage capacities. This approach distributes the training process across multiple devices or machines. Various techniques are utilized for employing distributed training: Model parallelism [42, 43], data parallelism [44], or even a combination of both [45]. For model parallelism, the model is initially partitioned into submodels, also known as chunks, that are then distributed across the available devices. The computed outputs are then combined to obtain the required model output. Although this technique eases large model deployability, however, it is naturally suited for architectures with distinct parts and may require significant modifications for models with high sequential or temporal dependencies across layers [46]. On the other hand, data parallelism relies on splitting the data batch into multiple non-overlapping subsets to be concurrently processed on separate devices. The computed gradients from the various subsets are further reduced into a single set before optimizing the model's parameters. Although both techniques offer multiple benefits depending on the specific use case, data parallelism is more prevalent in deep learning applications with extensive training data sets due to its ease of implementation and universality concerning model architectures [47]. Although relying on distributed training offers a convenient and easy technique to overcome the OOM error without risking the model's accuracy or performance since no modifications are performed on the model architecture, however, it necessitates the availability of substantial computational infrastructure compromised of multiple accelerated processing units accompanied by high capacity storage units with large memory bandwidth to avoid bottlenecks and ease the transfer of data between. Such infrastructure requires a substantial amount of funds, and thus its expense might stall various researchers and organizations, with limited resources, from further exploring new deep learning model architectures [48].

On the other hand, carefully scanning the literature reveals other methods employed in different fields and for various applications that can reduce the memory footprint and allow for training larger deep learning models without endangering the model's accuracy or expanding the hardware infrastructure. Various works suggested training the model with reduced precision by binarizing the weights [49], activations [50], or all the tensors, including the gradients [51], or by quantizing the parameters to different bit counts [52, 53]. Of particular interest in this area is the

work of Micikevicius et al. [54], which relied on a floating point half-precision for all tensors and arithmetics with no hyperparameters tuning. Their work resulted in a reduction in the memory footprint for various deep learning architectures without accuracy deterioration. On another side, statically defining model's computational graph before execution allows for various optimizations that can also be employed to enhance performance and reduce the memory footprint. This is possible due to the complete graph analysis and the required peak memory prediction that opens the door for further optimizations, such as constant folding, pruning redundant graph nodes, etc [55]. Also, mitigating dynamic execution and relying instead on static one enables more efficient use of hardware resources by eliminating unnecessary memory allocation and avoiding fragmentation [56, 57]. Finally, the above set of optimizations can be further extended to computation-specific optimizations on certain subclusters in the computational graph with the aid of JIT compilers [58]. The latter set of optimizations is distributed between both platform-agnostic optimizations and target-specific optimizations, offering a further reduction in computational cost and memory footprint [59].

In the following work, we've addressed the problem of limited resources encountered while training a deep learning model with a specific focus on CFD applications with mesh adaptation. Precisely, the graph-structured deep learning model, suggested in [60] for inferring the Navier-Stokes flow fields in a two-fluid flow problem with dynamically evolving anisotropic discretization space, is implemented and retrained in different environments in an attempt to reduce its required memory footprint. The tested approaches focus on lightening the memory requirement without expanding the hardware infrastructure and while maintaining the same model architecture, thus avoiding incurring additional hardware costs or endangering the model's performance. The approach is based on adopting the mixed precision training methodology, previously suggested in [54], and employed for various model architectures based on convolutional neural networks or on recurrent ones for a graph-structured flow predicting model. The possible reductions are further enforced by incorporating multiple static computational graph optimizations, thus joining forces of numerous techniques simultaneously rather than relying on a single exclusive method alone. This allowed for retraining the same number of model parameters eight times faster with an 80 % lower peak memory, compared to the initial default implementation, on the exact same hardware. The current work is organized as follows: Sec. 4.2 starts by showcasing the reason behind memory build-up in neural network training and highlights its scaling with the input dimensions for a simple MLP. Next, in Sec 4.3, three techniques for counteracting this buildup are discussed, and their effect on both the memory and training step time is shown for the same MLP. The introduced methods will then be employed, in Sec. 4.4, to a graph-structured deep learning model used for inferring the Navier-Stokes flow fields

in a two-fluid flow problem with mesh adaptation. This contribution is concluded finally in Sec. 4.5.

4.2 The curse of input dimensionality in Deep Learning models

Deep learning models are simply non-linear functions parametrized by a set of trainable parameters and are commonly used to approximate a particular target function. The value of their parameters is usually determined by training them on a large dataset. However, during this training, memory buildup occurs due to relying on the backpropagation algorithm for computing the gradients required by the optimizer. Section 4.2.1 expands the various points for a single update step and elaborates on the reason for memory buildup in the backpropagation algorithm, while the next section, Section 4.2.2, demonstrates this memory buildup for the common multilayer perceptron and illustrates its scaling with the dimension of the input.

4.2.1 Training a Deep Learning model using Backpropagation

A deep learning model used to approximate a target function, $F^*(\mathbf{x}) = \mathbf{y}$, is simply a nonlinear function, $F(\mathbf{x};\theta) = \tilde{\mathbf{y}}$, that maps an input, $\mathbf{x}$, into an approximated output, $\tilde{\mathbf{y}}$, and that is parametrized by a set of trainable parameters, θ, also known as the model's weights and biases. The values possessed by this set of parameters are initially randomly chosen; however, the output accuracy of the model solely depends on their values. Thus, they are continuously updated to converge the model outputs toward the desired targets until the difference between the predictions and desired targets, quantized using a loss function, is minimized. The non-convex loss function, due to the model's non-linearity, imposes the use of iterative gradient-based optimizers. Thus, to update the parameters, the gradient of the loss function with respect to the parameters is required and is computed using the backpropagation algorithm [32].

However, the memory footprint attains a peak value at every update step during parameter optimization and significantly scales with the dimension of the model input. To clarify the reason behind this memory peak value, the process of an update step will be demonstrated over a feedforward network, also known as a multi-layer perceptron (MLP). Although this demonstration is performed over an MLP, the update step is similar to all sequential models. Also, MLPs are seen as the constitutional base unit of various DL models and are of essential importance in the DL field due to their numerous offspring architectures, such as graph convolutional networks or recurrent neural networks. Thus, a single training step consists of

forward propagation, followed by a backward propagation, also known as Backprop, and finally, parameters update as shown in Figure 4.2. In the forward pass, an input $\mathbf{x}$ is propagated through the model layers, $f^i(\mathbf{x^i}; \theta^\mathbf{i})$, until an output $\tilde{\mathbf{y}}$ followed by a scalar cost $J(\theta)$ is computed. Next, the gradient of this cost with respect to the model parameters is computed by flowing the information backwardly using the backpropagation algorithm. Then, the optimizer uses the computed gradients to update the current parameters state. Finally, to ensure more stable updates and reduce the attained training noise, the error from a batch of examples can be used similarly to perform the update rather than using a single example.

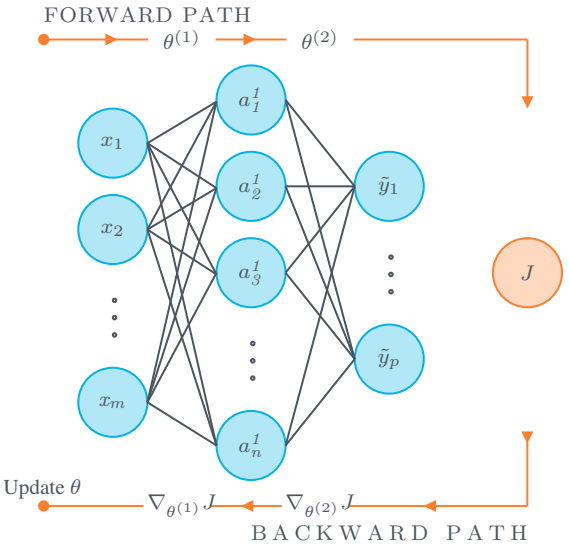


Figure 4.2: ***Training Step*** *consisting of a forward path for computing the scalar loss, J, a backward path for computing the gradients of the loss with respect to the trainable parameters, $\nabla_{\theta^{(i)}} J$, and an update step for optimizing these parameters.*

The mentioned peak in memory occurs specifically during the backward propagation step. The Backprop is not specific to MLPs and can compute the derivative for any function. Moreover, although it is mostly employed to compute the gradient of the cost function with respect to the parameters, it can also be used for computing other gradients. To describe backpropagation accurately, we will refer to the involved computations using graphs where each node in the graph represents a variable, *e.g.*, scalar, vector, matrix, etc., and each edge represents an operation, *e.g.*, addition, multiplication, ReLU, etc. In reality, the backpropagation algorithm computes the derivatives of a function relying on the chain rule of calculus. It performs a Jacobian-gradient product for each operation in the graph as shown, for example, in Eq. 4.1 for a computational graph depicting $z = f(g(\boldsymbol{x})) = f(\boldsymbol{y})$ with the input, $\boldsymbol{x}$, being a vector and in Eq. 4.2 for the input being a tensor.

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{T} \nabla_{\boldsymbol{y}} z \tag{4.1}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ and $z \in \mathbb{R}^m, \mathbb{R}^n$ and $\mathbb{R}$ respectively.

$$\nabla_{\boldsymbol{X}} z = \sum \left(\nabla_{\boldsymbol{X}} Y_j\right) \frac{\partial z}{\partial Y_j}, \tag{4.2}$$

where $z = f(g(\mathbf{X})) = f(\mathbf{Y})$ with $\mathbf{X}$ and $\mathbf{Y}$ representing tensors of any arbitrary dimension and Y_j a specific entry of the tensor.

Generally, the Backprop computes the gradient of the output node with respect to any node in the computational graph relying on an efficient implementation of the chain rule of calculus. Various computational subexpressions are numerously repeated throughout the computation of the gradients at each node using the naive implementation of the chain rule rendering the process inefficient for complex graphs. The Backprop elevates this problem at the expense of the memory footprint by storing the value of these common subexpressions avoiding unnecessary recomputations. This limits the number of operations at the node level to a single gradient computation per edge where only the gradient of every child node for the parent is required. Thus, the Backprop algorithm computational graph is a replica of the forward graph, as shown in Figure 4.3, having the same number of nodes and edges but traversed in the opposite direction where each edge imposes the computation of the gradient of the child node with respect to its parent and each node becomes its gradient with respect to the output.

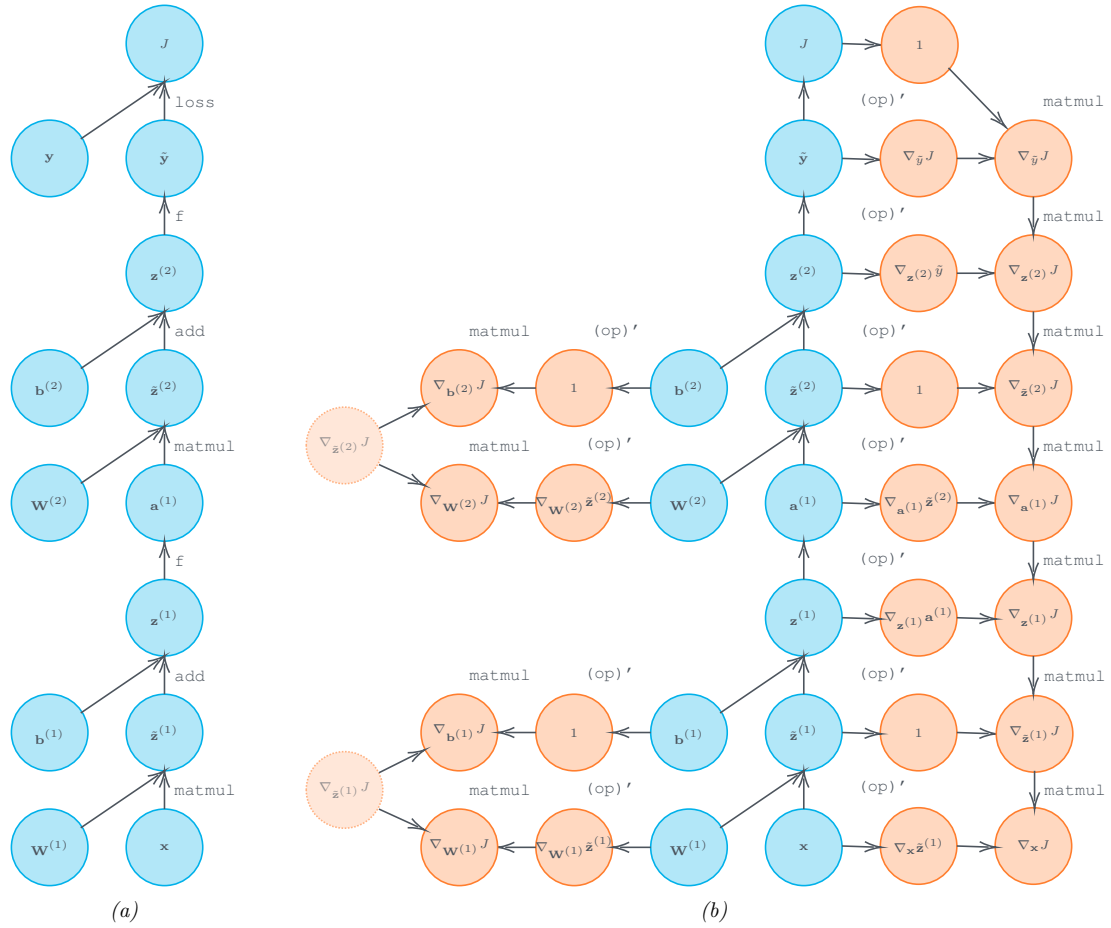


Figure 4.3: ***Computational Graphs*** *for both forward and backward path of an MLP with a single hidden layer.* $\mathbf{W}^{(i)}, \mathbf{b}^{(i)}, \tilde{\mathbf{z}}^{(i)}, \mathbf{z}^{(i)}$, *and* $\mathbf{a}^{(i)}$ *are respectively the weight tensor, bias vector, weighted sum, activation and the non-linear activation specific for the* i^th *layer. The graph operations are considered to include a non-linear operation,* f*, along to a loss computing operation, loss. In (a), the input,* $\boldsymbol{x}$*, propagates from bottom to top until a scalar loss,* J*, is computed. In (b), the gradient of the loss with respect to every nodal variable, shown in orange, is computed by propagating backwardly from top to bottom. Operations denoted by* $(op)'$ *compute the gradient of the child node with respect to its parent,* $\nabla_{\mathbf{Ch}}\mathbf{Pa}$*. Faded nodes are duplicated nodes for the ease of representation. Note that the following graphs are simplified schematic representations of the true graphs that might differ with various implementation methods.*

4.2.2 Memory overhead accompanied in a training step

In a training step, the value of the computed parameters, such as the hidden layer activations, is maintained in memory until the gradients are calculated. The value

of these parameters is required by the backpropagation algorithm for efficiently propagating the backward computational graph. For example, on a single training sample, the gradient of the scalar loss with respect to the i^{th} layer weights, $\nabla_{\mathbf{W}^{(\mathbf{i})}} J$, is equal to the product of the propagated gradient at the related activation child node, $\nabla_{\tilde{\mathbf{z}}^{(\mathbf{i})}} J$, with the i^{th-1} layer activations, *i.e.* $\nabla_{\mathbf{a}^{(\mathbf{i})}} J \times \mathbf{a}^{(\mathbf{i-1})^T}$, as shown in Figure 4.3b. Thus, these activations, along with other previously computed variables during the forward path, are kept stored in the memory until their related gradients are computed throughout the backward path. This leads to memory accumulation, with a peak value attained at the backward path, as shown in Figure 4.4, where the evolution of the memory footprint through a single training step for an MLP is plotted.

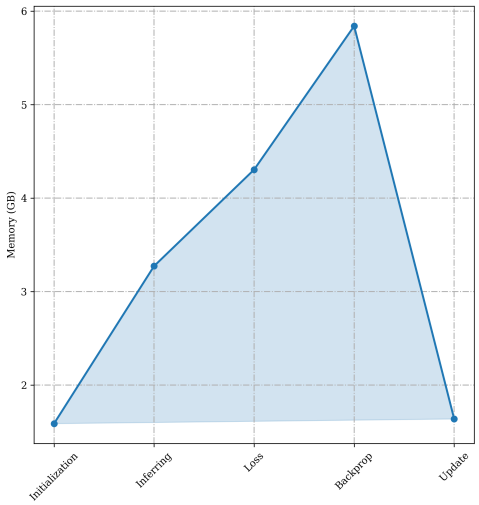


Figure 4.4: ***Memory footprint*** *for a single training step of an MLP. The network is trained on an input batch consisting of* $1M$ *samples. The network consists of* 2 *ReLU-activated hidden layers followed by a linear output layer. The dimension of all layers, including the input and output layers, is set to* 128*.*

Moreover, the magnitude of the attained peak memory is directly related to the dimension of the hidden layers along with the number of input samples. An input batch, consisting of N samples with $\mathbb{R}^m$ feature vector, will result in hidden-layer activation tensors in $\mathbb{R}^{N \times n_i}$, where n_i is the dimension of the i^{th} hidden layer. Thus, input tensors with a larger number of samples, N, will demand larger computational resources as shown in Figure 4.5 where the required peak memory for a single step training of an MLP increases proportionally with the number of samples in the input tensor.

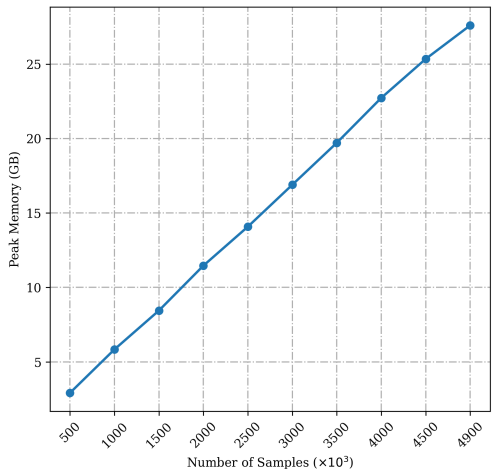


Figure 4.5: ***Variation of the attained peak memory with the number of input samples****. The network consists of* 2 *ReLU-activated hidden layers followed by a linear output layer. The dimensions of all layers, including the input and output layers, are set to* 128*.*

4.3 Review of implementation methods to counteract memory buildup

Various methods exist in the literature to lower the effect of the memory buildup occurring during the training of deep learning models. Some methods turn to alter the model's architecture, while others opt for incorporating larger hardware infrastructure. In this section, we will direct our focus toward methods that offer a reduction in the required memory footprint without demanding architectural modifications or an expansion in hardware. In particular, Section 4.3.1, provides details for training a model with reduced parameter precision without endangering the model's accuracy. While sections 4.3.2 and 4.3.3 offer a reduction in memory by optimizing the computational graph at various levels.

4.3.1 Mixed precision

Deep learning models are usually trained with a single floating point precision. This requires each model parameter, activation, or gradient to occupy 32 bits of memory. Reducing the required storage bits will induce a large reduction in the required memory during training, especially for complex models trained with a very large number of samples, along with possible speedups on modern accelerators. Micikevicius et al. explored in their work, [54], using mixed precision, with both 16- and 32-bit

floats, for training various convolutional and recurrent network architectures. The suggested methodology is applicable without changes to the model's architecture or its hyperparameters and offers relaxation for both the memory and the compute resources on most hardware.

Although the reduction in memory is guaranteed regardless of the hardware architecture, using mixed precision for specific processing units may offer no improvement in performance. Improvement in performance using mixed precision stems from either decreased memory bandwidth or reduction in arithmetic time with half-precision floats. However, Central Processing Units (CPUs) are inadequate to perform reduced precision math, and mixed precision may result in slower training. Also, some NVIDIA Graphical processing units, with compute capability lower than 7.0 that are not equipped with Tensor Cores, [61], limit their speedups to memory bandwidth savings only. Finally, the speedup on GPUs with compute capabilities of at least 8.0 is also limited to memory bandwidth savings since Tensorfloat-32, [62], automatically uses accelerated lower precision arithmetics for some operations.

However, due to the narrower dynamic range of the half-precision float point format compared to the single-precision counterpart, a loss of information may occur during model training, leading to a deterioration in the resulting model accuracy. To overcome this problem, three techniques have been suggested in [54]. First, a master copy of the model trainable parameters is maintained in the FP32 format. The intuition behind this act is backed by the fact that the weight updates may underflow in FP16 formats, *i.e.* the product of the weight gradient and the learning rate may attain values smaller than 2^{-24} during training and thus are zero-valued in the reduced precision format. Also, an update that is 2^{11} times smaller than its corresponding weight value also becomes zero due to the binary point alignment within the addition operation. Thus, maintaining a single precision copy of the weights to accumulate the updates will bypass the above-mentioned problems with a negligible 50% increase in weight memory requirement since memory consumption during training is mostly dominated by storing activations. The second technique suggested to avoid deterioration of accuracy is loss scaling. The intention behind scaling the loss is also due to the small values attained by the gradients throughout the backward pass, which might fall below the representable range of FP16, leading to the same underflow problem. Multiplying the computed loss by a scale factor before the backpropagation path shifts the computed gradients with the same scale and helps them occupy a larger portion of the FP16 representable range. The computed gradients are then unscaled before updating the FP32 master copy of the weights. Finally, care should be granted while specifying the scaling factor to avoid computing gradients with a value larger than the maximum limit of FP16, *i.e.* $65,504$, leading to an overflow. Finally, the third technique suggests carrying out certain arithmetic operations in FP32 while maintaining the memory read/write

to FP16 tensors. Examples of such operations include the large tensor reductions occurring in a softmax layer or in batch-normalization layers or the accumulation of partial products for the vector dot-products operations.

As mentioned earlier, the reduction in the memory footprint accompanied by implementing mixed precision training methodology is directly dependent on the complexity of the model and its deepness. Precisely, mixed precision stores the computed activations of the model layers, the trainable parameters, and the gradients in half-precision tensors and maintains a single-precision copy of the model parameters. This results in halving the memory required for storing the activations while increasing by half the memory required for storing the parameters. However, since the number of activations typically dominates the number of parameters, the effect of mixed precision on memory is focused on the number of activations present in the model architecture. Furthermore, certain implementations also require casting the model prediction to a single-precision tensor to ensure numerical stability, [63], thus further limiting the attained reductions in memory footprint. To showcase the effect of the number of activations on the required peak memory for training the model, the number of hidden layers in the Multilayer Perceptron model introduced in Section 4.2.2 is varied, and the required memory is registered for both, full precision and mixed precision training. The effect of casting the predictions to single precision is also considered, and the results are shown in Figure 4.6.

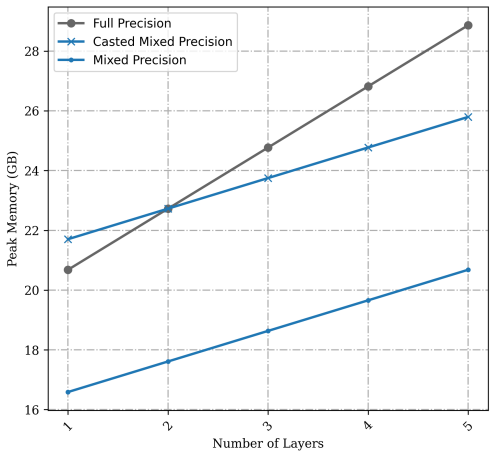


Figure 4.6: ***Variation of the attained peak memory with the number of hidden layers for reduced precision implementations.*** *The dimensions of all layers, including the input and output layers, are set to* 128.

4.3.2 Migrating eager execution and performing graph optimizations

Various platforms exist for building and training deep learning models. Nowadays, both TensorFlow and PyTorch have attained popularity within the research community due to various factors such as their large community, detailed documentation, and coding flexibility. To broaden their spectrum of users, both libraries rely on a simple Pythonic approach as their default option for building a model, *i.e.* Eager execution with TensorFlow and similarly Dynamic computational graphs with PyTorch. Although this approach is easier to employ and more intuitive, it comes at the cost of efficiency and deployability with operations being executed one-by-one in Python with small space for performing optimizations and code acceleration. Migrating these approaches to faster and more efficient execution methods is possible by considering the Graph execution in TensorFlow or Static computation graphs within PyTorch. Both these methods compile a static computational graph prior to execution, thus opening the door for potential acceleration opportunities and optimizing possibilities. The remainder of this section will rely on the terminology used for the TensorFlow Graph execution method, although similar concepts exist across the various deep learning platforms. The computational graph is held in a Dataflow, device-independent TensorFlow Graph with nodes representing operations and edges representing data flowing between the operations. To ease transforming the python-eager code into TensorFlow graph operations, TensorFlow has developed the Autograph library to handle these transformations for plenty of operations, including control flow ops, *i.e.* loops, and conditionals, that are usually easier to understand in Python format. Figure 4.7 shows the obtained TensorFlow execution graph for the previously defined MLP model prediction with five hidden layers.

The main advantage of graph execution lies in the broader range of possible optimizations, such as compiler-level transformations, optimizations in structure, and enhanced resource distribution. In TensorFlow, all of the following optimizations are handled by Grappler. Grappler's main role is to optimize the Python-level user-defined graph before converting it to a TensorFlow execution graph by running various sub-optimizers for multiple iterations. The sub-optimizers perform various operations, including graph pruning, function inlining, constant folding, arithmetic simplification, etc. For interested readers, a comprehensive list of available optimizers and further details can be found in TensorFlow documentation [63]. Finally, to showcase the attained reduction in peak memory obtained from migrating to optimized static graph execution, the predefined MLP model in previous sections is used and tested for various numbers of hidden layers. The obtained results are summarized in Figure 4.8.

However, due to the static nature of the computational graph, various points require more attention. First, since static programming requires declaring the whole

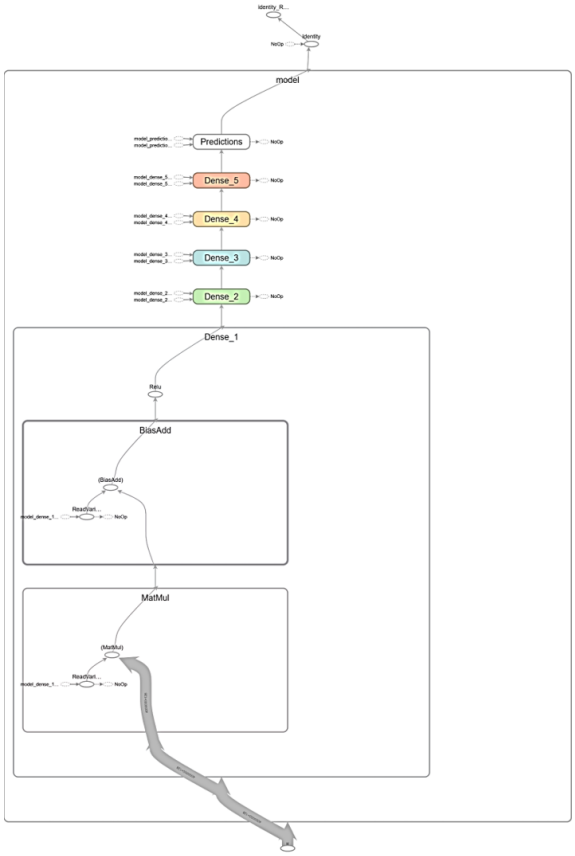


Figure 4.7: ***TensorFlow Computational Graph*** *for an MLP with five hidden layers. The dimension of all layer features is set to* 128*. The nodes represent operations, whereas the edges represent the flow of data. The flow of data occurs from bottom to top with an input batch of* $4M$ *samples.*

computational graph prior to executing, this step is costly and might result in an overall slower model if employed for functions not frequently utilized. Also, since the computational graph is statically defined, variation within the input samples regarding either the type or size, such as that occurring for data obtained from CFD simulation with dynamically adapted discretization space, requires additional effort to handle it. Moreover, since calling operations are delayed until the whole graph is compiled, debugging is foreseen as a harder task in static programming compared to dynamic one. Finally, drifting from the native Python environment to more specific static graph operations may induce implementation difficulties for unseasoned programmers.

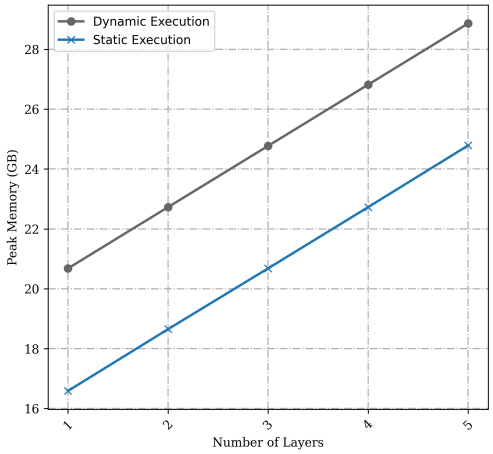


Figure 4.8: ***Variation of the attained peak memory with the number of hidden layers for various execution methods.*** *The dimension of all layers, including the input and output layers, is set to* 128.

4.3.3 Accelerated linear algebra

Multiple optimizers are employed in the static execution of a Tensorflow Graph as detailed in Section 4.3.2. However, these optimizations are only driven by pattern-matching and thus are limited to previously defined kernels. Further enhancement in performance and reduction in memory bandwidth can be attained by introducing the Accelerated Linear Algebra compiler, also known as XLA. The XLA is a domain-specific compiler for linear algebra that emits code for multiple backends and extends the available set of optimizations by introducing computation-specialized optimizations. These optimizations are distributed between both platform-agnostic optimizations and target-specific optimizations.

Briefly, the XLA compiles clusters of the Tensorflow Graph into other strongly-typed intermediate representations allowing for further optimizations before emitting backend-specific machine code. The cluster of operations to be compiled can be either explicitly specified or automatically defined. Multiple compiled clusters can exist in a Tensorflow Graph thus closing the gap in execution speed, through Just-In-Time compilation, between compilers and interpreters The choice of clusters is motivated to include the largest possible number of short-lived operations accompanied with plenty of kernel launches, *i.e.*, in the context of deep learning, the whole training step is suggested to be compiled. These clusters can be identified with the aid of a profiler or through Autoclustering After identifying the clusters of computational nodes in the Tensorflow Graph, the Tensorflow runtime translates the obtained

subgraph into an XLA High-level operations intermediate representation (HLO IR). The HLO is a functional language for modeling linear algebra computations that consists of around 50 operations compared to Tensorflow which compromises around 1500 operations. It is a strongly typed language with specified data type, shape, and layout, thus facilitating the optimization process. Multiple target-independent passes for analysis and optimization are executed on the obtained HLO graph, including common-subexpression elimination, fusion, buffer analysis, etc. Note that kernel fusion is seen as one of the most impactful optimizations for GPU, where various operations are fused into a single computation, thus reducing the number of GPU kernels to be launched and reducing the memory bandwidth by avoiding materialization of intermediate results into the memory. Next, the XLA backend, operating on the optimized HLO graph and invoking either the LLVM backend or the CUDA library, will be responsible for the target-dependent optimization and analysis along with the target-specific code generation. The whole process of XLA along with its different paths, are all summarized in Figure 4.9.

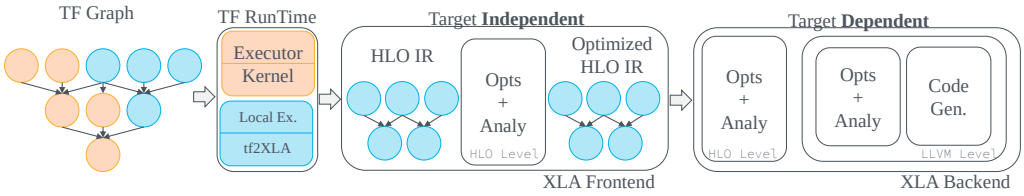
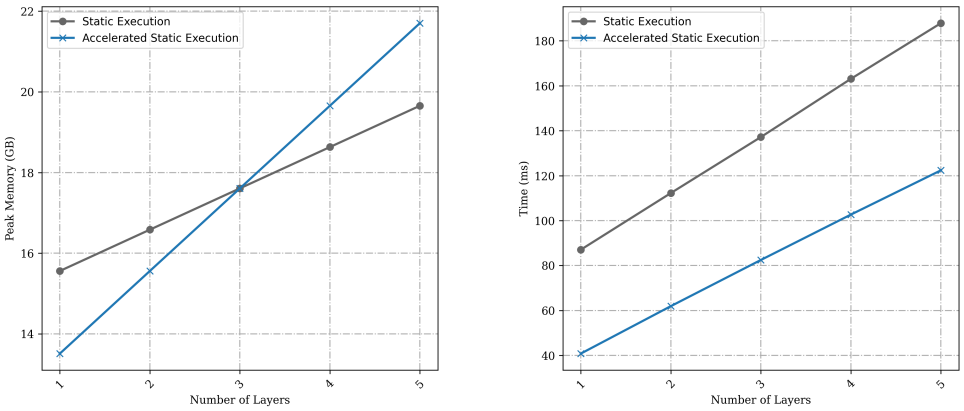


Figure 4.9: ***Schematic of the Accelerated Linear Algebra compiler*** *showing both the frontend and the backend optimizations.*

Finally, although implementing XLA is accompanied by various advantages, including enhancing performance and reducing memory footprint and bandwidth, however, various limitations may hinder its employment. Mainly, a large number of operations in Tensorflow, around 60%, are currently uncompilable with XLA. This may stem from either their inherited incompatibility, such as I/O ops, or due to the novelty of XLA and lack of development resources. Moreover, since XLA relies on strongly typed intermediate representation, dynamically evolving input shape or size will require recompilation and thus introduce an unexpected latency. Finally, since the obtained optimization is both model and hardware-dependent, deterioration in performance or additional memory overhead could be expected in certain circumstances.

The resulting performance and memory footprint optimizations attained from implementing XLA depend on the model under study. For the model utilized throughout the previous sections and for all tests, an enhancement in performance

is always visualized. However, the reduction in the memory footprint with XLA is only attained if accompanied by mixed precision. Also, since XLA performs optimizations on High-Level-Operations intermediate representations derived from TensorFlow graphs, static execution is also required. To showcase the optimizations obtained by implementing XLA, the required time and peak memory for performing a single training step are recorded for multiple model architectures, implemented with a mixed precision policy as described in Section 4.3.1, and for a various number of samples. Figures 4.10a and 4.10b, showing respectively the obtained peak memory and required training-step time for both accelerated and non-accelerated implementations, highlight the attained variation in memory optimization depending on model architecture. The acceleration in performance is maintained across various model architectures but comes at the cost of the memory footprint for models with a large number of layers.



(a) *(b)*

Figure 4.10: ***Variation of the (a) attained peak memory and (b) single training-step time with Model architecture for both accelerated and non-accelerated implementations.*** *The dimension of all layers, including the input and output layers, is set to* 128*. The number of samples used for updating parameters is* $N = 4M$*.*

4.4 Application on Graph Neural Network for CFD simulation

To test the utility of the methods suggested in Section 4.3 in the field of computational dynamics, a deep learning model with a graph-based architecture employed for predicting the Navier-Stokes flow fields on an unstructured irregular dynamic

discretization space and for multiphase flow problems is introduced in Section 4.4.1. All the necessary details for successfully embracing these techniques with this specific model are highlighted in sections 4.4.2, 4.4.3 and 4.4.4 along with the attained reduction in the computational cost.

4.4.1 Model details and architecture

To investigate the applicability and the resulting reductions of the above-stated implementation methods, a deep learning model that operates directly on graph-structured data [36, 64, 65], and employed for the inferring of the flow fields in a multiphase flow problem setup is utilized. The model architecture and the details of the problem have been previously introduced in [60] where the authors have trained a full precision model in a dynamic environment. This training environment will later be referred to as "Reference implementation" to allow for comparing the required peak memory and training time with those obtained in the new suggested training methodologies. For a fair comparison between the different training processes and the resulting model performance, all details are maintained the same with the exact same distribution of datasets across all performed trainings. Any variation from the reference environment will be explicitly specified for every suggested implementation.

The deep learning model is employed to assist in the solution of an unsteady multiphase flow problem. The problem under study involves a fluid forced to flow into another fluid's domain with differing thermodynamical properties. The domain is similar in shape to an air-filled tank with a fluid inlet at the bottom and a pressure relief outlet at the top, and the problem setup is shown in Figure 4.11. This problem is governed by the Navier-Stokes equations strongly coupled to a convective self-reinitializing level-set equation [66]. The location of the interface, and thus the distribution of the thermodynamical properties, is determined by convecting the level-set using the computed Navier-Stokes flow fields. Finally, these required physical fields are computed on a dynamically adapted unstructured anisotropic triangular discretization mesh that ensures the fine capturing of the interface and all the physics around it [67], and the set of equations are sequentially solved using an industrial-level in-house finite element VMS solver [5, 68]. Figure 4.12 shows the evolution of the discretization mesh, along with the distribution of the physical properties and the flow fields over multiple time steps, for one of the obtained solution trajectories later used for model training.

To reduce the computational cost of such a problem, a deep learning model, faster than the traditional computationally demanding Navier Stokes Finite element solver, is suggested to infer the next time step flow fields. The model input is a graph with the same topological characteristics as the solver's adapted discretization space. The computed physical fields, i.e., the velocity and the thermodynamical properties, along with a binary scalar highlighting the boundary nodes, are informed to the

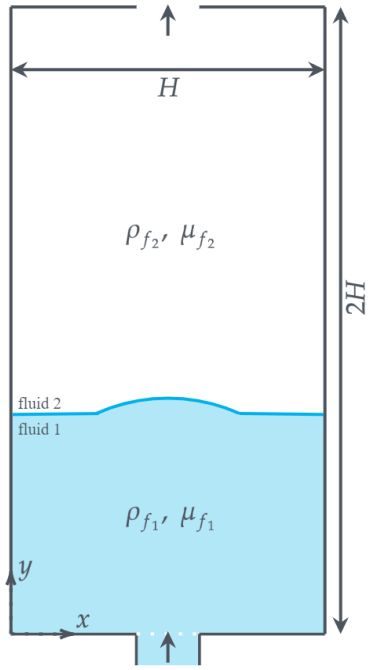


Figure 4.11: ***Problem Setup*** *of a multiphase flow problem with showing fluid* 1 *with thermal properties,* ρ_{f_1} *and* μ_{f_1}*, filling a* $2H$*-by-*H *tank previously occupied by a fluid with different properties,* ρ_{f_2} *and* μ_{f_2}*.*

model as node features, whereas spatial information is provided as edge features. The model initially encodes the nodes and edges features into a higher dimensional space using an encoder. The encoded features are then updated using ten message-passing blocks [19, 69], and the required acceleration fields are finally predicted as node features using the decoder. A schematic of the model architecture, along with the input and output fields, is shown in Figure 4.13. Each message passing step consists of an edge update function followed by another node update function. All update functions, along with the encoder and decoder, are two hidden layers MLPs with latent and output layers dimensions of 128, except for the decoder, whose output layer dimension is dictated by the size of the predicted field.

The above model architecture compromises around 1 588 483 trainable parameters. These parameters are simultaneously updated for multiple training steps to enhance the model inferring quality. To update these parameters, a mean squared error loss function, quantizing the error between the predicted and CFD fields, is computed for an input global batch graph consisting of multiple disjoint sample snapshots graphs. The gradient of this function with respect to the trainable param-

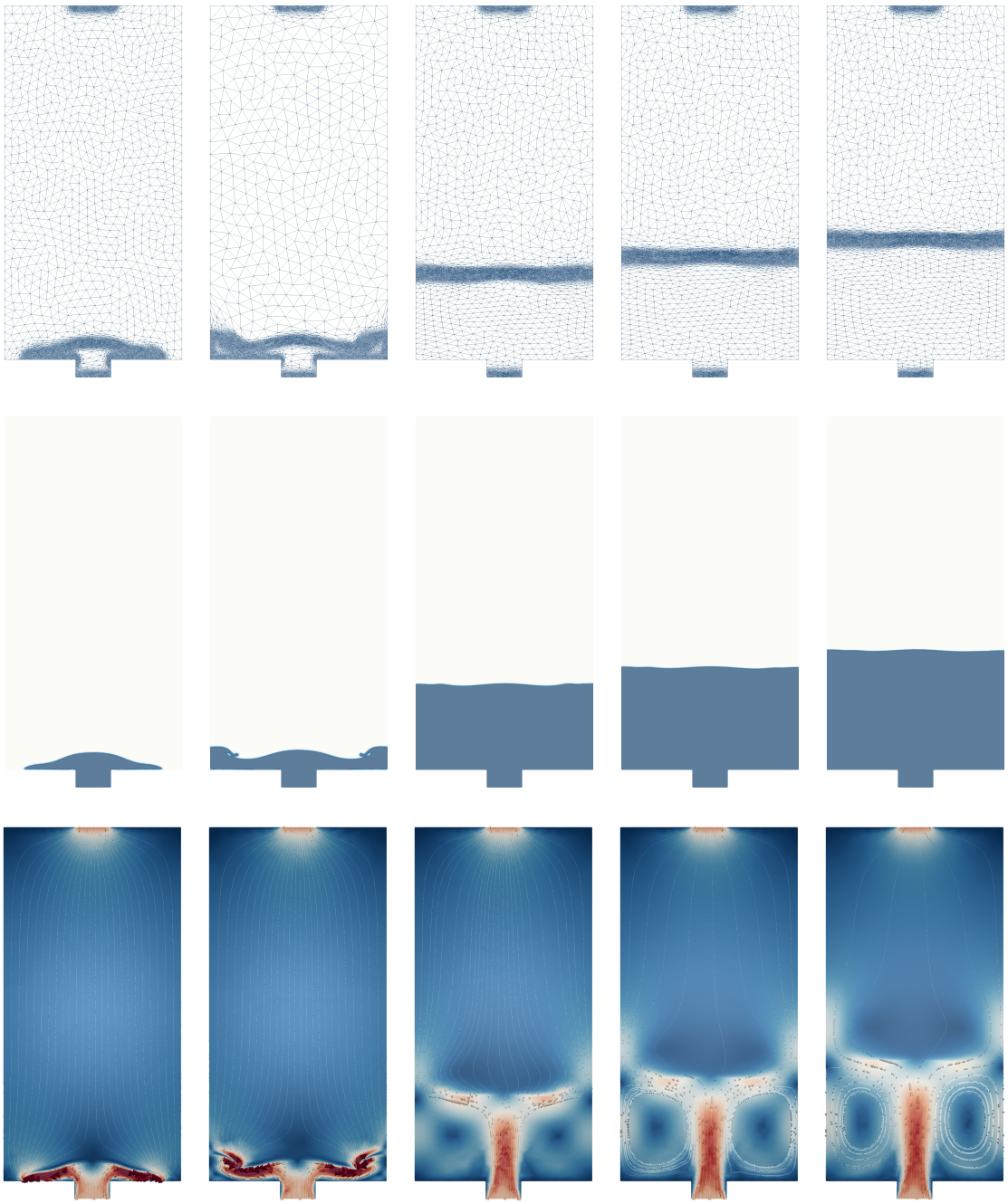


Figure 4.12: ***The evolution of the discretization mesh and the physical fields over simulation time.*** *The snapshots, from left to right, are obtained at the following time steps: 250, 500, 2500, 3000, and 3500. A simulation with an inlet velocity of* 0.5 *m/s is utilized. The first row shows the discretization mesh obtained using the gradient-based mesh adaptive method over the filtered level-set function. The second and third rows show the density and the streamlined velocity fields.*

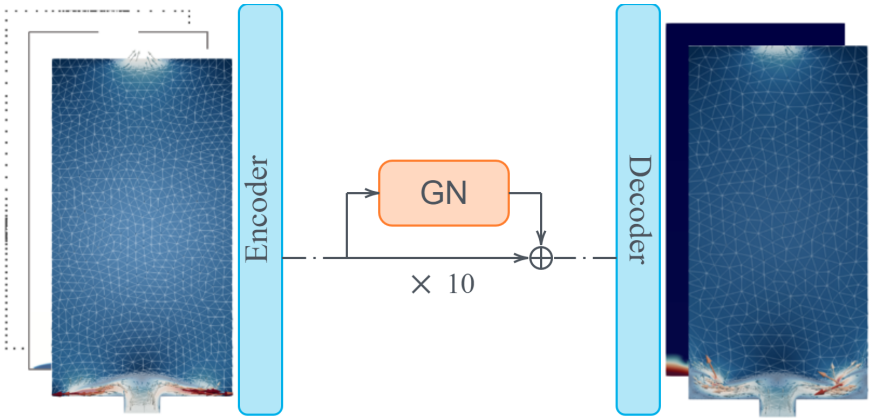
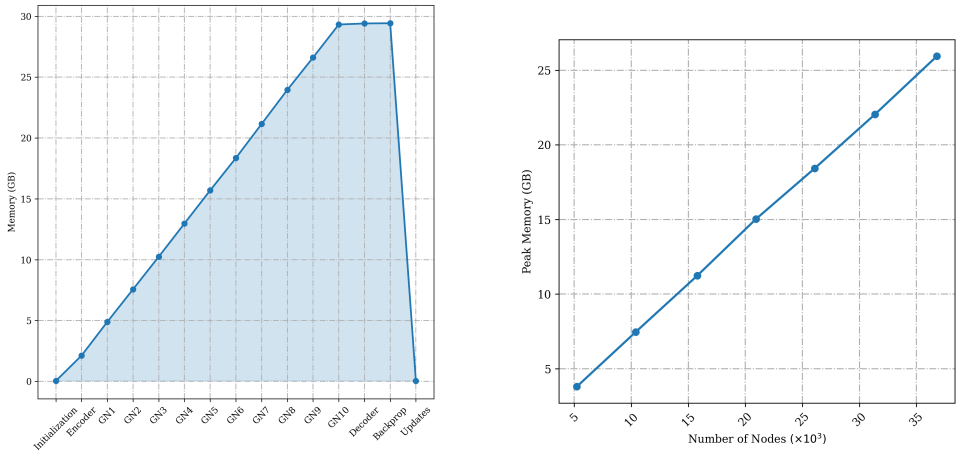


Figure 4.13: ***Model Architecture*** *with an Encoder-Processor-Decoder structure. The processor is constituted from multiple GN blocks with unshared parameters and residual connections.*

eters is determined using the backpropagation algorithm [70]. Lastly, these gradients are used to optimize the parameters using the Adam optimizer [71]. The memory footprint at every stage of a single training step, obtained using the reference implementation and for a global input graph consisting of eight disjoint subgraphs, is shown in Figure 4.14a. Also, Figure 4.14b highlights the peak memory variation with the size of the input global batch graph. Finally, it should be noted that all training implementations are performed on an NVIDIA Tesla V100 GPU.

4.4.2 Mixed precision

In this section, the training with mixed precision, following the suggested methodology in [54] with a single precision model output to keep with TensorFlow's blog recommendations, is tested for the Graph Neural Network model introduced in Section 4.4.1. To start, the model trainable parameters, corresponding to 1.6 million parameters, are stored in a half-precision floating point format. A master copy of the parameters, with a single precision format, is also maintained to accumulate the obtained gradient updates and avoid neglecting minor parameter updates. The current model architecture consisting of an encoder, 10 Graph-Network blocks, and a decoder, all based on MLPs with two hidden layers of 128 neurons and a latent output dimension of 128, results for every additional node or single direction edge in around 4608 activations. Considering the smallest global input graph utilized with 5230 nodes and 15 423 edges, we will obtain around 95.2 million activations compared to only 1.6 million parameters. Thus, the reduction in memory obtained from storing these activations in a half-precision format will dominate the additional



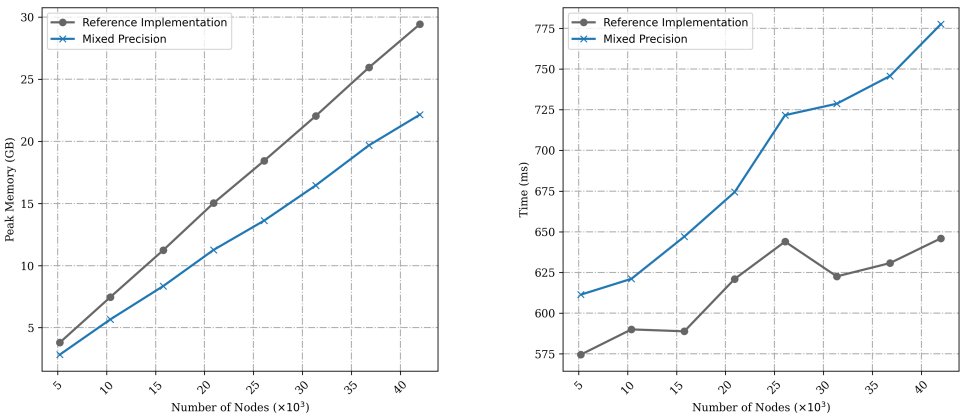
(a) *(b)*

Figure 4.14: ***Memory footprint*** *attained using the reference training methodology for the CFD deep learning model. (a) Shows the evolution of this footprint through a single training step with a global input graph consisting of eight disjoint subgraphs, while in (b), the value of the peak required memory is shown for different global input graph sizes.*

memory overhead necessary for maintaining an FP32 copy of the weights and biases, thus ensuring an overall lower peak memory. Finally, the computed loss is adaptively scaled before computing the gradients to avoid any numerical instability. The impact of mixed precision on both the memory and the time required for a single update is shown in Figures 4.15a and 4.15a, respectively.

Relying solely on mixed precision offers, on average, around a 25% reduction in computational cost with a deterioration in training step time of 12.24%. The obtained deterioration is due to the incompatibility of the utilized tensor sizes with the hardware requirements for half-precision. Although simple modifications could be performed to avoid the occurring increase in time, the authors preferred to maintain the same model specs, without any modifications, due to their intent of reporting the effect of incorporating these methods directly as ready-to-go tools along with the objective of reducing the peak memory required without a primary focus on the required time.

Finally, to investigate the effect of the employed method on both the training process and the resulting model's accuracy, the obtained validation loss at every epoch is compared with the reference full-precision training process validation loss, as shown in Figure 4.16. The obtained plot ensures that using mixed precision for our current model architecture and application does not deteriorate the training



(a) *(b)*

Figure 4.15: ***Variation of the (a) attained peak memory and (b) single training-step time with the number of Nodes in the global input graph for both full-precision and mixed-precision implementations.***

process and results in a final validation error of 1.09×10^{-2} compared to a slightly higher error of 1.14×10^{-2} in the reference training.

4.4.3 Static execution

In this section, the computational graph of the whole update step, including both the forward and the backward passes, is initially predefined prior to execution. Migrating the dynamic execution approach and relying on a static one allows for various optimizations, as previously specified in Section 4.3.2. The effect of these optimizations on both the required peak memory and the update step time is compared with the initial dynamic training process and is shown in Figures 4.17a and 4.17b for various global input graph sizes.

Particular attention shall be denoted to the definition of the static computational graph due to the effect of mesh adaptation on the shape of the input graph. The nature of the tackled application involving two-fluid flows with a dynamically evolving interface requires a fine-adapted discretization space that evolves simultaneously with the interface to capture all the physics. This requirement induces a variation in the shape of the training data at every update step due to the correlation of the discretization mesh with the topology of the input graph. Thus, to avoid retracing the computational graph for every new input shape, the graph is initially traced for specific input argument data types but with no concrete definition of shapes. Although this approach truncates the number of possible optimizations due to relaxing

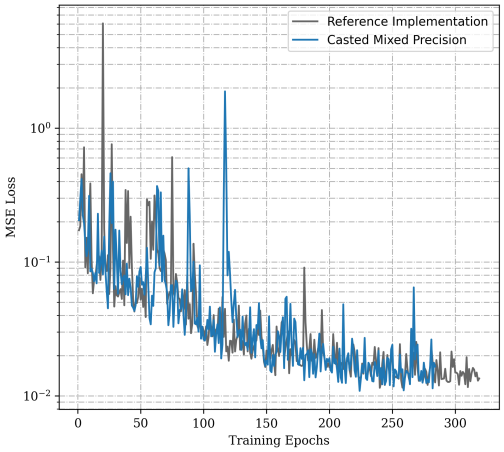


Figure 4.16: ***Validation loss*** *obtained from training two models with different precision policies. The training environment is similar for both models with a batch input graph of around* 40 000 *nodes.*

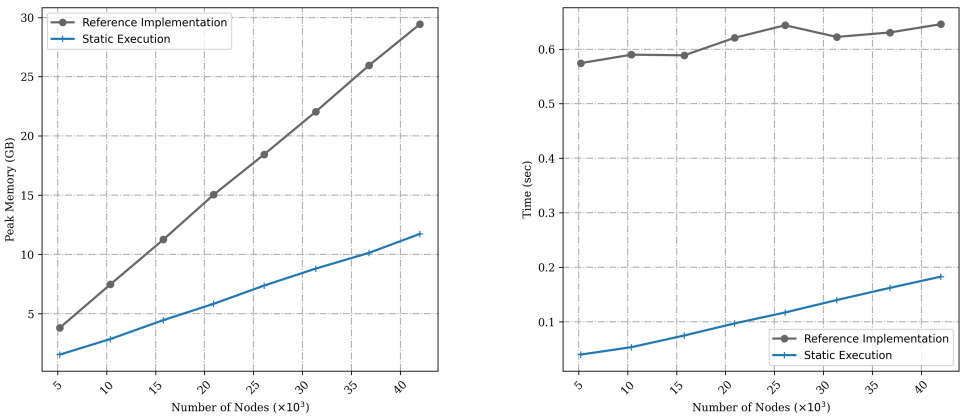
the input arguments data structure, however, it avoids the computational overhead accompanied by retracing at every single step.

Finally, combining static execution with mixed precision employed in Section 4.4.2 ensures further reduction in the required peak memory, as shown in Figures 4.18a and 4.18b. Due to incorporating mixed precision, the hardware-related increase in the update time step, compared to solely relying on static execution, could be tolerated in light of the attained reduction in the required peak memory.

Similarly, the obtained validation loss at every epoch from both approaches is compared with that obtained during the basic training approach. The similarity of the training curves shown in Figures 4.19a and 4.19b ensures the safety of employing these methods without endangering the returned model's accuracy.

4.4.4 Accelerated linear algebra

In Section 4.4.3, immigrating dynamic execution and relying on static one, where the whole computational graph is previously defined, opens the door for multiple optimizations and leads to a significant reduction in computational cost, as shown in Figures 4.17a, 4.17b, 4.18a, and 4.18b. However, all performed optimizations were restricted to general graph-level optimizations with no specific optimizations tailored for the current model computational graph. In this section, the accelerated linear algebra compiler will be utilized on a predefined cluster of nodes of the current model's graph to expand the set of possible optimizations and obtain further reduc-

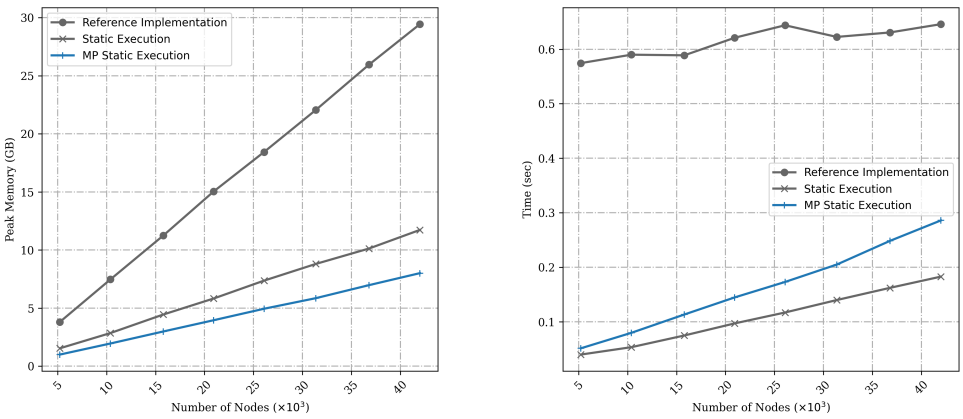


(a) *(b)*

Figure 4.17: ***Variation of the (a) attained peak memory and (b) single training-step time with the number of Nodes in the global input graph for both static execution and the reference dynamic execution.***

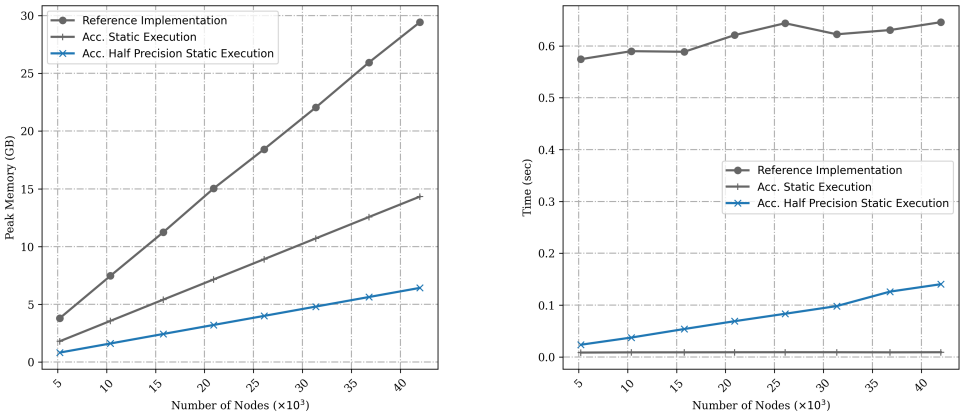
tions. As previously detailed in Section 4.3.3, this augmentation in incorporated optimizations is possible due to operating on domain-specific strongly typed intermediate representations along with the additional target-dependent optimizations. The effect of embracing these computation-specific, architecture-independent, and dependent optimizations on the resulting peak memory and computational cost of the current Graph Neural Network model is shown in Figures 4.20a and 4.20b.

Investigating Figure 4.20a highlights an average reduction in the maximum required memory of approximately 52 % between the reference training process and the current statically accelerated training. This reduction in memory is accompanied by a negligible training step time averaging around 8.89 ms, as shown in Figure 4.20b. Moreover, to attain an additional reduction in memory, the precision of model parameters is reduced, as detailed in Section 4.3.1, and the required training memory and time are plotted in Figures 4.21a and 4.21b, respectively. This results in an additional 26.47 % reduction in memory, thus a total of around 78.38 % relative to the reference case, and an average step time of around 78.93 ms, larger than that obtained if relying on a full precision accelerated training alone, but approximately eight times faster than the reference training process.



(a) *(b)*

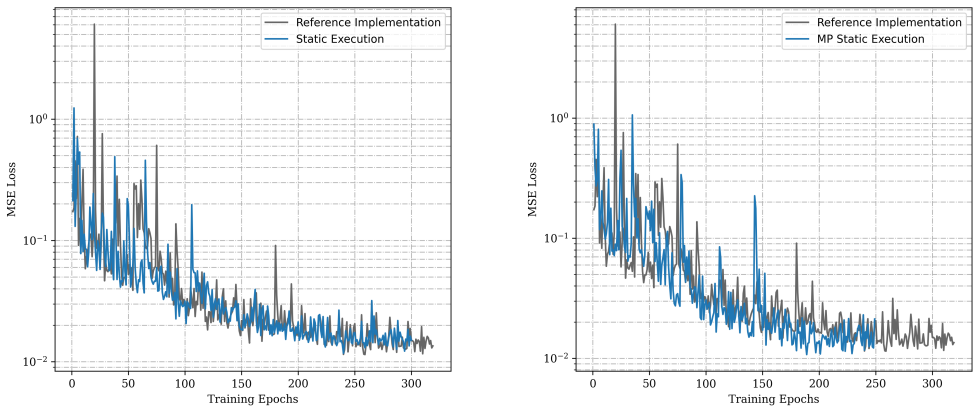
Figure 4.18: ***Variation of the (a) attained peak memory and (b) single training-step time*** *with the number of Nodes in the global input graph for both the static execution and the reference dynamic executions.*



(a) *(b)*

Figure 4.21: ***Variation of the (a) attained peak memory and (b) single training-step time*** *with the number of Nodes in the global input graph for mixed precision implementation coupled to accelerated linear algebra compilation.*

Finally, it should be noted that cooperating XLA compilers in the training process requires a compilation of an optimized strongly typed intermediate representation. Variations in the shape of the global input graph, originating from the dynamically adapted discretization space, impose the requirement of recompiling



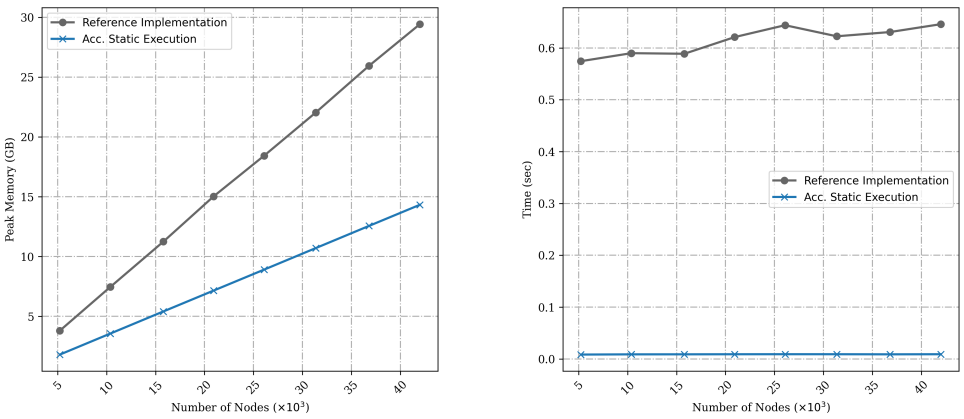
(a) *(b)*

Figure 4.19: ***Validation loss*** *obtained from training the same model architecture with various implementation techniques: (a) Comparing static execution with basic implementation and (b) Comparing mixed precision static execution with basic implementation. The training environment is similar for both models with a batch input graph of around* 40 000 *nodes.*

for every encountered new input shape, thus drastically slowing the training process. To avoid this demanding recompilation, a static input graph is required. This static graph can be attained by either efficiently padding every resulting global input graph by a subgraph to homogenize the shapes across the training steps or by hinging to a discretization space with a constant number of nodes and edges across all timesteps. However, this work will be satisfied with the currently obtained results regarding the effect of XLA on memory and time, with no further investigation of the suggested methods of homogenizing the input graph shape.

4.5 Conclusion

The current contribution presented various methods for tackling the out-of-memory problem without endangering the model's capacity or requiring more extensive hardware infrastructure. The reason behind the memory buildup and its scaling with the dimension of the input was explained in depth, and different methods employed previously in other domains to address this issue were revised. The inheritance of these techniques to the CFD field was tested using a graph-structured model to predict the flow fields in a two-fluid flow problem. The choice of the application was motivated, first, by the dynamically evolving discretization mesh, thus allowing to generalize of the conclusions to a large set of complex problems in the CFD and not



(a) *(b)*

Figure 4.20: ***Variation of the (a) attained peak memory and (b) single training-step time*** *with the number of Nodes in the global input graph for both the reference implementation and the linear algebra accelerated implementation.*

being restricted to simple problems with fixed discretization space, and second, by the novelty of the utilized model architecture that allows operating directly on the unstructured irregular anisotropic mesh thus preserving all topological information and avoiding deteriorating the physical fields by the interpolation noise common in encoding parent fields to different discretization spaces. Initially, the effect of using mixed precision for training a graph neural network inferring the Navier-Stokes flow fields was examined for its impact on both the computational cost and the model accuracy. This resulted in a 25 % reduction in the required peak memory without any deterioration in the model accuracy. Simultaneously incorporating static execution with the reduced precision training, and opening the door for the various computational graph optimizations, allows for an additional 48 % reduction in the required memory and around four times speedup in updating the parameters. This reduction and speedup are further augmented by incorporating the accelerated linear algebra compiler that allows for further domain and architecture-specific optimizations, thus finally resulting in a total of around 78 % memory reduction and 8x faster update steps compared to the initial reference training implementation. Embracing the proposed implementations and harnessing the resulting reductions in computational time and cost allows researchers to further explore the potential of large deep-learning models on more complex data sets, attain more accurate models by allowing for larger batch sizes, and finally, finetune the model's parameters by facilitating the exploration of the hyperparametric space. The significant benefits that can be witnessed for the proposed methodologies encourage exploring other dimen-

sions that might also allow training performant models without incurring additional hardware infrastructure. Relieving the model from the need for an extensive dataset by previously informing it about the governing physics or allowing for faster information propagation without requiring a large number of message-passing blocks are all a few examples of the additional methods foreseen as fertile space for investigation.

Bibliography

[1] G. Guiza, A. Larcher, A. Goetz, L. Billon, P. Meliga, E. Hachem, Anisotropic boundary layer mesh generation for reliable 3d unsteady rans simulations, Finite Elements in Analysis and Design 170 (2020) 103345. doi:https://doi.org/10.1016/j.finel.2019.103345. 129

[2] E. Hachem, G. Jannoun, J. Veysset, M. Henri, R. Pierrot, I. Poitrault, E. Massoni, T. Coupez, Modeling of heat transfer and turbulent flows inside industrial furnaces, Simulation Modelling Practice and Theory 30 (2013) 35–53. doi:https://doi.org/10.1016/j.simpat.2012.07.013.
URL https://www.sciencedirect.com/science/article/pii/S1569190X12001219 129

[3] E. Hachem, T. Kloczko, H. Digonnet, T. Coupez, Stabilized finite element solution to handle complex heat and fluid flows in industrial furnaces using the immersed volume method, International Journal for Numerical Methods in Fluids 68 (1) (2012) 99–121. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.2498, doi:https://doi.org/10.1002/fld.2498.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2498 129

[4] E. Hachem, S. Feghali, R. Codina, T. Coupez, Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation, International Journal for Numerical Methods in Engineering 94 (9) (2013) 805–825. doi:https://doi.org/10.1002/nme.4481. 129

[5] M. Khalloufi, Y. Mesri, R. Valette, E. Massoni, E. Hachem, High fidelity anisotropic adaptive variational multiscale method for multiphase flows with surface tension, Computer Methods in Applied Mechanics and Engineering 307 (2016) 44–67. 129, 146

[6] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: A brief review, Computational intelligence and neuroscience 2018 (2018). 129

[7] S. Yang, Y. Wang, X. Chu, A survey of deep learning techniques for neural machine translation (2020). arXiv:2002.07526.
URL https://arxiv.org/abs/2002.07526 129

[8] L. Deng, G. Hinton, B. Kingsbury, New types of deep neural network learning for speech recognition and related applications: an overview, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 8599–8603. doi:10.1109/ICASSP.2013.6639344. 129

[9] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing (2018). arXiv:1708.02709. 129

[10] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707. doi:https://doi.org/10.1016/j.jcp.2018.10.045.
URL https://www.sciencedirect.com/science/article/pii/S0021999118307125 129

[11] Y. Bar-Sinai, S. Hoyer, J. Hickey, M. P. Brenner, Learning data-driven discretizations for partial differential equations, Proceedings of the National Academy of Sciences 116 (31) (2019) 15344–15349. doi:10.1073/pnas.1814058116.
URL http://dx.doi.org/10.1073/pnas.1814058116 129

[12] C. Ghnatios, G. El Haber, J.-L. Duval, M. Ziane, F. Chinesta, Artificial intelligence based space reduction of structural models, ESAFORM 2021 (04 2021). doi:10.25518/esaform21.2004. 129

[13] J. Viquerat, E. Hachem, A supervised neural network for drag prediction of arbitrary 2d shapes in laminar flows at low reynolds number, Computers & Fluids 210 (2020) 104645. doi:https://doi.org/10.1016/j.compfluid.2020.104645.
URL https://www.sciencedirect.com/science/article/pii/S0045793020302164 129

[14] Y. Zhang, W. J. Sung, D. N. Mavris, Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient. arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2018-1903, doi:10.2514/6.2018-1903.
URL https://arc.aiaa.org/doi/abs/10.2514/6.2018-1903 129

[15] J. Chen, J. Viquerat, E. Hachem, U-net architectures for fast prediction of incompressible laminar flows, arXiv preprint arXiv:1910.13532 (2019). 129

[16] A. Patil, J. Viquerat, A. Larcher, G. El Haber, E. Hachem, Robust deep learning for emulating turbulent viscosities, Physics of Fluids 33 (10) (2021) 105118. doi:10.1063/5.0064458.
URL https://doi.org/10.1063/5.0064458 129

[17] G. El Haber, J. Viquerat, A. Larcher, D. Ryckelynck, J. Alves, A. Patil, E. Hachem, Deep learning model to assist multiphysics conjugate problems, Physics of Fluids 34 (1) (2022) 015131. 129

[18] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1) (2009) 61–80. doi:10.1109/TNN.2008.2005605. 129

[19] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, arXiv (2018).
URL https://arxiv.org/pdf/1806.01261.pdf 129, 147

[20] J. Chen, E. Hachem, J. Viquerat, Graph neural networks for laminar flow prediction around random two-dimensional shapes, Physics of Fluids 33 (12) (2021) 123607. arXiv:https://doi.org/10.1063/5.0064108, doi:10.1063/5.0064108.
URL https://doi.org/10.1063/5.0064108 129

[21] F. De Avila Belbute-Peres, T. Economon, Z. Kolter, Combining differentiable PDE solvers and graph neural networks for fluid flow prediction, in: H. D. III, A. Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, Vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 2402–2411.
URL https://proceedings.mlr.press/v119/de-avila-belbute-peres20a.html 129

[22] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, J. Sohl-Dickstein, On the expressive power of deep neural networks, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, Vol. 70 of Proceedings of Machine Learning Research, PMLR, 2017, pp. 2847–2854.
URL https://proceedings.mlr.press/v70/raghu17a.html 130

[23] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR abs/1409.1556 (2014). 130

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9. doi:10.1109/CVPR.2015.7298594. 130

[25] S. Zagoruyko, N. Komodakis, Wide residual networks, in: E. R. H. Richard C. Wilson, W. A. P. Smith (Eds.), Proceedings of the British Machine Vision Conference (BMVC), BMVA Press, 2016, pp. 87.1–87.12. doi:10.5244/C.30.

87.
URL https://dx.doi.org/10.5244/C.30.87 130

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861 (2017). 130

[27] M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International conference on machine learning, PMLR, 2019, pp. 6105–6114. 130

[28] P. Dollar, M. Singh, R. Girshick, Fast and accurate model scaling, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 924–932. doi:10.1109/CVPR46437.2021.00098. 130

[29] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778. doi:10.1109/CVPR.2016.90. 130

[30] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 4171–4186. doi:10.18653/v1/n19-1423.
URL https://doi.org/10.18653/v1/n19-1423 130

[31] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.
URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf 130

[32] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, nature 323 (6088) (1986) 533–536. 130, 133

[33] M. J. Islam, G. Nguyen, R. Pan, H. Rajan, A comprehensive study on deep learning bug characteristics, in: Proceedings of the 2019 27th ACM

Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019, Association for Computing Machinery, New York, NY, USA, 2019, p. 510–520. doi:10.1145/3338906.3338955.
URL https://doi.org/10.1145/3338906.3338955 130

[34] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient transfer learning, CoRR abs/1611.06440 (2016). arXiv:1611.06440.
URL http://arxiv.org/abs/1611.06440 130

[35] F. E. Fernandes Junior, L. G. Nonato, C. M. Ranieri, J. Ueyama, Memory-based pruning of deep neural networks for iot devices applied to flood detection, Sensors 21 (22) (2021). doi:10.3390/s21227506.
URL https://www.mdpi.com/1424-8220/21/22/7506 130

[36] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. Battaglia, Learning to simulate complex physics with graph networks, in: H. D. III, A. Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, Vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 8459–8468.
URL https://proceedings.mlr.press/v119/sanchez-gonzalez20a.html 130, 146

[37] P. M. Radiuk, Impact of training set batch size on the performance of convolutional neural networks for diverse datasets, Information Technology and Management Science 20 (1) (2017) 20–24. 130

[38] S. L. Smith, P.-J. Kindermans, Q. V. Le, Don't decay the learning rate, increase the batch size, in: International Conference on Learning Representations, 2018.
URL https://openreview.net/forum?id=B1Yy1BxCZ 130

[39] Z. Hu, K. Shukla, G. E. Karniadakis, K. Kawaguchi, Tackling the curse of dimensionality with physics-informed neural networks, arXiv preprint arXiv:2307.12306 (2023). 130

[40] M. Zolnouri, D. Lakhmiri, C. Tribes, E. Sari, S. L. Digabel, Efficient training under limited resources, arXiv preprint arXiv:2301.09264 (2023). 131

[41] N. A. Kako, et al., Ddls: Distributed deep learning systems: A review, Turkish Journal of Computer and Mathematics Education (TURCOMAT) 12 (10) (2021) 7395–7407. 131

[42] A. L. Gaunt, M. Johnson, M. Riechert, D. Tarlow, R. Tomioka, D. Vytiniotis, S. Webster, Ampnet: Asynchronous model-parallel training for dynamic neural networks, ArXiv abs/1705.09786 (2017). 131

[43] L. Guan, W. Yin, D. Li, X. Lu, Xpipe: Efficient pipeline model parallelism for multi-gpu dnn training, arXiv preprint arXiv:1911.04610 (2019). 131

[44] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, P. Dubey, Distributed deep learning using synchronous stochastic gradient descent, arXiv preprint arXiv:1602.06709 (2016). 131

[45] F. Lai, A. Kadav, E. Kruus, Splitbrain: Hybrid data and model parallel deep learning, ArXiv abs/2112.15317 (2021). 131

[46] Y. Park, S. Ahn, E. Lim, Y. Choi, Y. Woo, W. Choi, Deep learning model parallelism, Electronics and Telecommunications Trends 33 (4) (2018) 1–13. 131

[47] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, G. E. Dahl, Measuring the effects of data parallelism on neural network training, Journal of Machine Learning Research 20 (112) (2019) 1–49. 131

[48] E. Strubell, A. Ganesh, A. McCallum, Energy and policy considerations for modern deep learning research, Proceedings of the AAAI Conference on Artificial Intelligence 34 (09) (2020) 13693–13696. doi:10.1609/aaai.v34i09.7123. URL https://ojs.aaai.org/index.php/AAAI/article/view/7123 131

[49] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, Advances in neural information processing systems 28 (2015). 131

[50] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, Advances in neural information processing systems 29 (2016). 131

[51] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV, Springer, 2016, pp. 525–542. 131

[52] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, arXiv preprint arXiv:1606.06160 (2016). 131

[53] Y. Guo, A survey on methods and theories of quantized neural networks, arXiv preprint arXiv:1808.04752 (2018). 131

[54] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al., Mixed precision training, arXiv preprint arXiv:1710.03740 (2017). 132, 138, 139, 149

[55] R. M. Larsen, T. Shpeisman, Tensorflow graph optimizations (2019). 132

[56] T. Skotiniotis, J.-e. M. Chang, Estimating internal memory fragmentation for java programs, The Journal of systems and software 64 (3) (2002) 235–246. 132

[57] M. Rezaei, K. M. Kavi, Abt and sbt revisited: Efficient memory management techniques for object oriented and web-based applications, Scientia Iranica. Transaction D, Computer science & engineering, electrical engineering 23 (3) (2016) 1217. 132

[58] J. Aycock, A brief history of just-in-time, ACM Computing Surveys (CSUR) 35 (2) (2003) 97–113. 132

[59] C. Leary, T. Wang, Xla: Tensorflow, compiled, TensorFlow Dev Summit 1 (2) (2017). 132

[60] G. El Haber, J. Viquerat, A. Larcher, J. Alves, F. Costes, E. Perchat, E. Hachem, Deep learning model for two-fluid flows, Physics of Fluids 35 (2) (02 2023). doi:10.1063/5.0134421.
URL https://doi.org/10.1063/5.0134421 132, 146

[61] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, J. S. Vetter, Nvidia tensor core programmability, performance & precision, in: 2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW), IEEE, 2018, pp. 522–531. 139

[62] R. Krashinsky, O. Giroux, S. Jones, N. Stam, S. Ramaswamy, Nvidia ampere architecture in-depth, NVIDIA blog: https://devblogs. nvidia. com/nvidia-ampere-architecture-in-depth (2020). 139

[63] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/ 140, 141

[64] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P. W. Battaglia, Learning mesh-based simulation with graph networks, in: International Conference on Learning Representations, 2021. 146

[65] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. W. Battaglia, Learning to simulate complex physics with graph networks, in: International Conference on Machine Learning, 2020. 146

[66] C. Bahbah, M. Khalloufi, A. Larcher, Y. Mesri, T. Coupez, R. Valette, E. Hachem, Conservative and adaptive level-set method for the simulation of two-fluid flows, Computers & Fluids 191 (2019) 104223. 146

[67] S. El Aouad, A. Larcher, E. Hachem, Anisotropic adaptive body-fitted meshes for cfd, Computer Methods in Applied Mechanics and Engineering 400 (2022) 115562. doi:https://doi.org/10.1016/j.cma.2022.115562. 146

[68] E. Hachem, M. Khalloufi, J. Bruchon, R. Valette, Y. Mesri, Unified adaptive variational multiscale method for two phase compressible–incompressible flows, Computer Methods in Applied Mechanics and Engineering 308 (2016) 238–255. doi:https://doi.org/10.1016/j.cma.2016.05.022.
URL https://www.sciencedirect.com/science/article/pii/S0045782516304236 146

[69] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: Proceedings of the 34th International Conference on Machine Learning, Vol. 70 of Proceedings of Machine Learning Research, PMLR, 2017, pp. 1263–1272. 147

[70] R. Hecht-Nielsen, Theory of the backpropagation neural network, in: Neural networks for perception, Elsevier, 1992, pp. 65–93. 149

[71] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.
URL http://arxiv.org/abs/1412.6980 149

Chapter 5

Conclusions

Abstract *This final chapter concludes our manuscript by providing a brief summary of all our proposed methods for employing deep learning models for predicting fluid flows and multiphysics modeling. Additionally, this chapter sheds light on various available future paths for incorporating deep learning models along with CFD to further advance the numerical transformation in the industry.*

Abstract *Ce dernier chapitre conclut notre manuscrit en fournissant un bref résumé de toutes nos méthodes proposées pour l'utilisation de modèles d'apprentissage en profondeur pour prédire les écoulements de fluides et modéliser la multiphysique. De plus, ce chapitre éclaire diverses voies futures disponibles pour intégrer les modèles d'apprentissage en profondeur avec la CFD afin de faire progresser davantage la transformation numérique dans l'industrie.*

Contents

5.1 Summary

The manuscript begins by tracing the historical interest of researchers in understanding fluid flows and their impact on various real-world phenomena, leading to extensive research in computational fluid mechanics. Subsequently, it highlights the breakthroughs in artificial intelligence, particularly deep learning, which have evolved in parallel with computational fluid mechanics and demonstrated their effectiveness across diverse fields. Next, diverse strategies for adopting these models to address various challenges in the realm of CFD are discussed. Finally, this chapter concludes by emphasizing our interest in further exploring the potential applications of the emerging field of scientific machine learning, with a primary focus on accelerating multiphysics CFD simulations.

Chapter Two provides a comprehensive overview of the initial proposed approach for integrating deep learning methodologies into multiphysics conjugate problems. The chapter highlights the significance of multiphysics conjugate problems, emphasizing their extensive range of applications across various industrial sectors. Subsequently, it outlines the governing equations necessary for simulating forced convective cooling of a work object, a prime example of a multiphysics problem involving fluid flow and heat exchange. Additionally, it specifies all the essential computational fluid dynamics (CFD) tools required to ensure accurate simulation of this process. Next, an encoder-decoder convolutional neural network architecture is suggested for reducing the computational burden by handling part of the resolution process, precisely the energy equation. This neural network is trained to directly predict the scalar temperature field at the subsequent time step, and its robustness is thoroughly evaluated using data generated from various cooling configurations. This data is encoded on a structured grid. The motivation behind the chosen architectural design, the selection of preprocessing operations, and the intricacies of the training process are comprehensively detailed. Lastly, the chapter proposes a coupling framework that integrates the data-based model with the traditional finite element solver. This integration allowed for a significant acceleration in spanning the time domain of the solution, achieving a 12-fold improvement over the conventional scalar transport solver. Furthermore, the chapter assesses the model's performance, interpolation capabilities, and generalization ability across different flow setups, including those exhibiting significant differences in flow characteristics compared to the training dataset.

In Chapter Three, the preliminary approach suggested to address multiphysics problems in the previous chapter is expanded to encompass a larger set of problems, such as the multiphase flow problems with dynamically evolving interfaces. First, in the cooling of the workpiece problem, the flow fields computed at the next time step were assumed independent of the resolved temperature fields, thereby limiting the coupling to a single direction. However, this assumption is invalid for

many problems, including the two-fluid flow problem governed by a strongly coupled system of equations. This system of equations consists of the Navier-Stokes equations, responsible for resolving the flow fields, and the level-set equation required for capturing the evolution of the interface, thus requiring adapting the coupling framework to include additional communication bridges. Moreover, the computational cost of solving the scalar transport equation in multiphase flow problems is significantly lower than that of solving the energy equation in the previous scenario. This undermines the beneficial effect of modeling the scalar transport equation using a deep learning model and necessitates providing an alternative solution to alleviate the computational burden. Finally, accurately capturing a progressively changing interface, separating both fluids, implies employing a dynamically evolving unstructured irregular anisotropic discretization space for resolving the physical fields. This raises concerns regarding the ongoing compatibility of convolutional neural networks. Thus, Chapter Three suggests a coupling framework between the traditional finite element CFD solver and a graph convolutional deep learning model that allows tackling the required problem directly on the dynamically adapted triangular mesh. Different methodologies were proposed for successfully training the model on predicting the computationally demanding Navier-Stokes flow fields rather than the simple scalar field. Finally, the model's performance and ability to independently span the required time domain were proved on new unseen trajectories.

Finally, in the last part of the manuscript, the out-of-memory problem encountered while training deep learning models is addressed. The reason behind the accumulation of the memory during training and its scaling with the dimension of the problem under study is clarified with the aid of a multilayer perceptron. Also, different methodologies for counteracting this problem while avoiding the expansion of hardware infrastructure or endangering the model capacity are detailed. Finally, these techniques were inherited into the CFD field by employing them to train the graph-structured deep learning model introduced in the previous chapter. This adoption reduced both the required memory and training time, thus setting the ground for further exploring the potential of larger deep-learning models on more complex datasets or enhancing the accuracy of current models by facilitating the exploration of hyperparameter space.

5.2 Future Works

The topics presented in this manuscript give rise to a number of possible further developments, lying at the intersection of deep learning, computational fluid dynamics and enrichment. We close this manuscript with a short discussion of these topics.

5.2.1 Expanding to problems with larger discretization space

The methodologies suggested in chapter 4 offered a large reduction in both training time and required memory. This does not only facilitates the fine tuning process for attaining better accuracy, but also opened the door for tackling more complex CFD problems with much larger discretization space, such as three-dimensional industrial problems. A promising continuation of this work suggests the utilization of the proposed methodologies to train a deep learning model for a three-dimensional multiphysics problem, where even slight reductions in computational costs could have a significant impact.

5.2.2 Explore the advantages of incorporating Attention mechanism in model architecture

The modularity of the coupling approach, proposed initially in Chapter 2, facilitates adapting the framework for new scenarios. This is made evident by observing the proposed adjustments on the initial framework of Chapter 2 for addressing the novel challenges encountered in multiphase flows detailed in Chapter 3. Another advantage of such modularity is the ability to optimize for better accuracy by easily migrating to novel deep-learning architectures. One of many prospective suggestions is to investigate the benefits of complementing the architecture with spatial,[1–3], and temporal attention mechanisms [4, 5]. Such mechanisms proved advantageous over other sequential deep learning models in various aspects, such as their ability of parallelism, shorter training times, and minimization of the vanishing gradient problem [6]. Thus, the numerous traits of such architectures and their achievements in various fields firmly impose them on the head of the list of future investigations [7].

5.2.3 Incorporate prior physical knowledge in various stages

Finally, the diverse coupling approaches proposed in this manuscript avoid total reliance on data by incorporating prior knowledge through their connectivity with the traditional finite element CFD solver. This connection serves to redirect the model's solution and prevent the accumulation of errors. However, several unexplored avenues remain for further harnessing this prior knowledge to achieve even better results. This suggests potential directions for future work. One of these directions involves enriching the optimization process of the deep learning model with prior knowledge through physics-driven regularization [8–10]. Another direction suggests constraining the model's predictions to intermediate fields which can subsequently be used to explicitly construct the final required field, ensuring adherence to specific conservation laws and physical principles [11, 12]. By integrating

these techniques with the training methodologies outlined in Chapter 3, we anticipate the development of a deep learning model that respects geometry and aligns more closely with physical laws. This enhancement should significantly improve the generalization capabilities of the trained models and potentially reduce the required size of the training dataset.

Bibliography

[1] X. Zhu, D. Cheng, Z. Zhang, S. Lin, J. Dai, An empirical study of spatial attention mechanisms in deep networks, in: Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 6688–6697. 171

[2] J. Park, S. Woo, J.-Y. Lee, I. S. Kweon, Bam: Bottleneck attention module, arXiv preprint arXiv:1807.06514 (2018).

[3] S. Woo, J. Park, J.-Y. Lee, I. S. Kweon, Cbam: Convolutional block attention module, in: Proceedings of the European conference on computer vision (ECCV), 2018, pp. 3–19. 171

[4] C. Tan, Z. Gao, L. Wu, Y. Xu, J. Xia, S. Li, S. Z. Li, Temporal attention unit: Towards efficient spatiotemporal predictive learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 18770–18782. 171

[5] J. Fan, K. Zhang, Y. Huang, Y. Zhu, B. Chen, Parallel spatio-temporal attention-based tcn for multivariate time series prediction, Neural Computing and Applications 35 (18) (2023) 13109–13118. 171

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017). 171

[7] Z. Niu, G. Zhong, H. Yu, A review on the attention mechanism of deep learning, Neurocomputing 452 (2021) 48–62. 171

[8] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics 378 (2019) 686–707. 171

[9] L. D. McClenny, U. M. Braga-Neto, Self-adaptive physics-informed neural networks, Journal of Computational Physics 474 (2023) 111722. doi:https://doi.org/10.1016/j.jcp.2022.111722.

[10] P. Sharma, W. T. Chung, B. Akoush, M. Ihme, A review of physics-informed machine learning in fluid mechanics, Energies 16 (5) (2023) 2343. 171

[11] Q. Hernández, A. Badías, F. Chinesta, E. Cueto, Thermodynamics-informed graph neural networks, arXiv preprint arXiv:2203.01874 (2022). 171

[12] N. Wandel, M. Weinmann, R. Klein, Teaching the incompressible navier–stokes equations to fast neural surrogate models in three dimensions, Physics of Fluids 33 (4) (2021). 171

RÉSUMÉ

L'utilisation d'outils numériques efficaces et précis pour la simulation de l'écoulement des fluides est devenue indispensable dans de nombreuses industries. Cependant, la simulation des problèmes multiphysiques nécessite la résolution d'un vaste système d'équations tels que les équations Navier-Stokes couplées à d'autres équations aux dérivées partielles, exigeant beaucoup de ressources ainsi qu'un temps de calcul considérable. Récemment, le couplage des techniques deep learning avec les outils de simulation numérique a dominé la recherche et a donné des résultats encourageants dans divers domaines. Cette thèse est dédiée à explorer davantage le résultat de la convergence de ces deux disciplines, en particulier la réduction des coûts du calcul des simulations CFD multiphysiques. Ainsi, dans la première partie, un réseau de neurones, autoencoders, est intégré dans le cadre de résolution d'un problème multiphysique concernant le refroidissement d'une pièce par convection forcée. Le modèle est développé pour prédire la température du champ afin d'éviter la résolution de l'équation de transport scalaire par l'analyse par éléments finis. Bien que les paramètres du modèle aient été calibrés à l'aide d'une quantité relativement limitée de données, il a été possible de le généraliser avec précision pour différents systèmes de refroidissement avec différentes entrées non traitées lors du processus d'entraînement, cela a permis l'accélération du processus de résolution. Dans la deuxième partie, le couplage précédent a été amélioré afin de traiter un problème d'écoulement de deux fluides avec une interface évolutive. Pour maintenir la précision près de l'interface, un modèle d'apprentissage profond graphique, qui fonctionne directement sur le maillage triangulaire irrégulier en évolution dynamique, est proposé. Ce modèle est développé pour prédire les champs d'écoulement de Navier-Stokes plutôt que le champ scalaire de la fonction level-set, pour garantir la plus grande réduction des coûts de calcul. Tous les éléments qui ont facilité ce couplage sont mis en relief, et la précision du couplage a été établie sur de nouvelles trajectoires de simulation. Dans la dernière partie, le problème de dimensionnalité en machine learning a été résolu. Les causes de ce problème sont clairement mises en évidence, et les différentes méthodologies pour y faire face sont détaillées. Enfin, différentes techniques ont été proposées pour pouvoir résoudre ce problème sans avoir recours à la modification de la forme du modèle. Ces techniques ont été également introduites dans le modèle deep-learning de l'écoulement de deux fluides et ont permis de réduire à la fois l'empreinte mémoire requise ainsi que le temps d'entraînement nécessaire.

MOTS CLÉS

Réseau neuronal convolutif, Réseaux neuronaux graphiques, Dynamique des fluides numérique, Méthode des éléments finis, Ecoulements multiphasiques

ABSTRACT

The availability of accurate and efficient numerical tools for simulating various fluid flow phenomena has become of great importance across a multitude of industries. However, simulating multiphysics problems requires resolving an extensive system of governing equations incorporating the Navier-Stokes flow equations coupled with other partial differential equations. Such a task is resource-demanding and requires considerable computational time. Recently, coupling deep learning techniques with numerical simulation tools has dominated the research landscape and returned promising results in various domains. This thesis is dedicated to further exploring the potential resulting from the convergence of these two disciplines, particularly in the context of reducing computational costs of multiphysics CFD simulations. Thus, in the first part, an auto-encoder convolutional neural network is incorporated into the solution framework of a multiphysics problem concerning the forced convective cooling of a workpiece. The model is trained to predict the temperature field to escape resolving the scalar transport equation using the traditional finite element methods. Although the model parameters were calibrated using a relatively limited amount of data, it was able to generalize accurately for different cooling setups with different inlet locations, not seen during the training process, and provide adequate speedup to the resolution process. In the second part, the previous coupling framework is upgraded to tackle a two-fluid flow problem with an evolving interface. To maintain accuracy near the interface, a graph-based deep learning model that operates directly on the dynamically evolving unstructured irregular triangular mesh is suggested. This model is trained to predict the Navier-Stokes flow fields rather than the level-set scalar field to ensure attaining the most significant reduction in computational cost. All the ingredients that facilitated this coupling are highlighted, and the coupling framework accuracy is established on unseen simulation trajectories. Finally, in the last part, the problem of dimensionality during the training of deep learning models is addressed. The roots of this problem are clearly highlighted, and the various methodologies for coping with it are detailed. Lastly, an assembly of techniques is suggested to tackle this problem without requiring any mutation to the model architecture. These techniques were further employed to train the deep learning model on the two-fluid flow problem and allowed for achieving a reduction in both the required memory footprint and the training time.

KEYWORDS

Convolutional Neural Networks, Graph Neural Networks, Computational Fluid Dynamics, Finite Element Analysis, Multiphysics flows