National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Assessment of Hybrid Computational Fluid Dynamics-Deep Learning Solvers for Unsteady Problems

Diploma Thesis

Antonis Tzanetakis

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Aknowledgements

Foremost, I extend my deepest gratitude to Professor Kyriakos C. Giannakoglou for his mentorship. His expert guidance, patient support, and generosity throughout the full course of this endeavor have been invaluable. Observing his unwavering dedication to educating and extracting the utmost potential from each student has genuinely been inspiring to me. I am truly honored to have worked under his supervision.

I would also like to thank researchers Dr. Varvara Asouti and Dr. Marina Kontou, part of the PCOpt team of NTUA, whose expertise, eager assistance and joyous approach to problem-solving have supported and enriched my thesis experience.

I am also grateful to my friends, especially to Nikos, for our shared journey and mutual encouragement through university life which has been both challenging and joyful.

Lastly, to my family—my parents, brother, and cat-brother—for their love and support.

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Assessment of Hybrid Computational Fluid Dynamics-Deep Learning Solvers for Unsteady Problems

Diploma Thesis

Antonis Tzanetakis

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Abstract

This thesis delves into the realm of Computational Fluid Dynamics (CFD), or Computational Mechanics in general, assisted by Deep Learning (DL) techniques. It programs and evaluates two hybrid CFD-DL approaches which can accelerate the solution of unsteady problems. The discretization error, which is heavily dependent on the grid step size, is an important source of error in numerical simulations and is what these methods aim to address. These hybrid solvers run on a coarse grid but produce solutions corresponding to much finer grids by utilizing Artificial Neural Networks (ANNs) trained on high resolution data.

The first technique replaces the conventional reconstruction step of the Finite Volumes Method (FVM) with an ANN-driven process. Namely, the ANN is trained to produce space and time dependent coefficients of a stencil, that when combined with the local field values on a coarse grid, reconstruct the field at the cell faces to attain results corresponding to solutions of much finer grids. In the second technique, the numerical solver operates on a coarse grid to generate low-resolution solutions, which are corrected at each time step by an ANN to achieve high-resolution results. Both methods are implemented to solve two 1D unsteady problems: the 1D advection equation and the 1D linear acoustics equation.

Key findings include the scalability and robustness of these hybrid approaches in varied conditions and equation parameters. This is supported by extensive testing and a study of the behavior of the models for different hyperparameter values. This thesis combines current research in the Parallel CFD & Optimization Unit (PCOpt) of NTUA with ideas from two recent papers in the literature and underscores the potential of integrating data-driven techniques in numerical models.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης

Αξιολόγηση Υβριδικών Επιλυτών Υπολογιστικής Ρευστοδυναμικής-Βαθιάς Μάθησης για Χρονικά Μη-Μόνιμα προβλήματα

Διπλωματική Εργασία

Αντώνης Τζανετάκης

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Περίληψη

Η διπλωματική εργασία διερευνά τρόπους υποβοήθησης της Υπολογιστικής Ρευστοδυναμικής (ΥΡΔ) από τεχνικές Βαθιάς Μάθησης (ΒΜ). Προγραμματίζονται και αξιολογούνται δύο υβριδικές προσεγγίσεις ΥΡΔ-ΒΜ, που επιταχύνουν την επίλυση χρονικά μη-μόνιμων προβλημάτων. Οι μέθοδοι διαχειρίζονται το σφάλμα διακριτοποίησης, που καθοριζεται από το βήμα του πλέγματος, και είναι σημαντική πηγή σφάλματος στις αριθμητικές προσομοιώσεις. Χρησιμοποιώντας Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) εκπαιδευμένα σε δεδομένα υψηλής ανάλυσης, οι υβριδικοί επιλύτες τρέχουν σε αραιό πλέγμα, παράγοντας λύσεις που αντιστοιχούν σε πολύ πυκνότερα πλέγματα.

Η πρώτη τεχνική αντικαθιστά το κλασικό βήμα ανακατασκευής της μεθόδου των Πεπερασμένων Όγκων με μια διαδικασία που βασίζεται σε ΤΝΔ. Συγκεκριμένα, το ΤΝΔ εκπαιδεύεται να παράγει χωροχρονικά μεταβαλλόμενους συντελεστές διακριτοποίησης, που συνδυάζονται με τις κομβικές τιμές του πεδίου σε ένα αραιό πλέγμα. Έτσι, ανακατασκευάζει το πεδίο στα όρια των κελιών δίνοντας αποτελέσματα ως σε πολύ πυκνότερα πλέγματα. Στη δεύτερη τεχνική, η αριθμητική επίλυση γίνεται σε ένα αραιό πλέγμα παράγοντας λύσεις χαμηλής ανάλυσης, οι οποίες διορθώνονται σε κάθε χρονικό βήμα από ένα ΤΝΔ αναπαράγοντας αποτελέσματα αρκετά υψηλότερης ανάλυσης. Και οι δύο μέθοδοι εφαρμόζονται σε δύο 1Δ χρονικά μη-μόνιμα προβλήματα: την εξίσωση μεταφοράς και την εξίσωση γραμμικής ακουστικής.

Πιστοποιείται η σωστή εκπαίδευση και λειτουργία αυτών των υβριδικών μοντέλων σε ποικίλες συνθήκες και παραμέτρους των επιλυόμενων ΜΔΕ με δοκιμές για διαφορετικές τιμές των υπερπαραμέτρων. Η εργασία συνδυάζει τρέχουσα έρευνα στη Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης του ΕΜΠ με ιδέες από δύο πρόσφατα άρθρα της βιβλιογραφίας.

Acronyms

ΕΜΠ	Εθνικό Μετσόβιο Πολυτεχνείο
ΕΘΣ	Εργαστήριο Θερμικών Στροβιλομηχανών
ΜΠΥΡ&Β	Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης
ΥΡΔ	Υπολογιστική Ρευστοδυναμική
ΒΜ	Βαθιά Μάθηση
ΤΝΔ	Τεχνητά Νευρωνικά Δίκτυα
ΠΟ	Πεπερασμένοι Όγκοι
ΜΔΕ	Μερικές Διαφορικές Εξισώσεις

NTUA	National Technical University of Athens
PCopt	Parallel CFD & Optimization unit
CFD	Computational Fluid Dynamics
BC	Boundary Conditions
IC	Initial Conditions
CFL condition	Courant-Friedrichs-Lewy condition
CR	Coarsening Ratio
Hi-Fi	High-Fidelity
Lo-Fi	Low-Fidelity
Hi-Res	High-Resolution
Low-Res	Low-Resolution
TVD	Total Variation Diminishing

ODE	Ordinary Differential Equations
PDE	Partial Differential Equations
1D, 2D, 3D	1,2,3-Dimensional
N-S	Navier Stokes
FVM	Finite Volume Method
FV	Finite Volumes
ML	Machine Learning
DL	Deep Learning
NNs	Neural Networks
DNNs	Deep Neural Networks
PCA	Principal Component Analysis
POD	Proper Orthogonal Decomposition
SVD	Singular Value Decomposition
PINNs	Physics Informed Neural Networks
UAT	Universal Approximation Theorem
CNNs	Convolutional Neural Networks
FC	Fully Connected
ReLU	Rectified Linear Unit
LReLU	Leaky Rectified Linear Unit
ELU	Exponential Linear Unit
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm

SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
MAE	Mean Absolute Error
EA	Evolutionary Algorithms
PSO	Particle Swarm Optimization
AD	Automatic Differentiation
TF 2.x	TensorFlow version 2.x
LI	Learned Interpolation
LC	Learned Corrections
CORR	Corrections
MAEA	Metamodel-Assisted Evolutionary Algorithms
ILSVRC 2015	ImageNet Large Scale Visual Recognition Challenge 2015

Contents

Chapter 1

Introduction

Over the past decades, the field of Computational Fluid Dynamics (CFD) has prospered, and made significant contributions to technological advancements. The establishment and prosperity of the field have been primarily driven by two pivotal factors: advancements in computational power and the refinement of numerical algorithms [52]. The growth in computational capabilities, often refered to as Moore's Law, has played a pivotal role in the evolution of CFD. The increased processing power available has enabled more complex and accurate simulations, allowing for finer discretization of fluid dynamics equations and the handling of more intricate fluid flow phenomena. Alongside hardware advancements, the development and refinement of numerical algorithms have been crucial. Improved numerical methods for solving the Navier-Stokes equations, turbulence modeling techniques, and optimization algorithms have all contributed to more efficient and accurate CFD simulations. In recent years, the same pattern has appeared for ML: the exponential increase in available computational power has coincided with a surge in data availability and the advancement of statistical techniques, giving rise to data-driven engineering.

Statistical approaches have historically played a significant role in the field of CFD, offering valuable tools for analyzing and understanding complex fluid flow phenomena. These methods have been used to extract meaningful patterns from large datasets, enabling the simplification and interpretation of intricate fluid dynamics. Principal Component Analysis (PCA) [48] and Proper Orthogonal Decomposition (POD) [33] are prime examples, employed to reduce the dimensionality of CFD data and identify dominant flow features [27, 33].

In recent years, ML algorithms [5, 15], have been increasingly adopted to predict flow behaviors and enhance simulation efficiency [7]. Data-driven methods have been successfully used in turbulence modeling [30, 25], optimization of aerodynamic designs [8, 24], and real-time flow prediction [44], marking a shift towards more

efficient and accurate CFD simulations by leveraging the vast quantities of data generated in simulations and experiments. Deep Learning (DL), a subset of ML, has emerged as a particularly successful technique in this domain due to its flexibility and proven effectiveness across various disciplines. Its ability to handle large volumes of data and learn complex patterns makes it particularly suited for CFD, where simulations naturally generate extensive structured time- series data.

In the realm of DL applied to CFD, methodologies have evolved into three main categories: end-to-end DL solvers, physics-informed DL solvers, and hybrid CFD-DL solvers. End-to-end DL solvers attempt to learn fluid dynamics directly from data, bypassing the explicit use of governing equations like the Navier-Stokes (N-S) equations. However, this approach has shown limitations, including the requirement for extensive training data, poor generalization, and lack of interpretability [43, 41]. These limitations partly stem from the inherent complexity of fluid flow phenomena modeled by the N-S equations, for which analytical solutions are often unattainable. Physics-informed DL solvers softly incorporate known physical laws, such as those encapsulated in the N-S equations, to guide the learning process of NNs. This approach tends to require less data and offers better interpretability than end-to-end models. Hybrid CFD-DL solvers combine traditional CFD methods with DL techniques, leveraging the strengths of both to improve accuracy and computational efficiency. Such hybrid models benefit from the robustness of conventional CFD in capturing the fundamental physics while utilizing DL for reducing the cost of complex, high-dimensional problems in fluid dynamics. Next, some of the most important methods of both physics informed and hybrid solvers will be presented to provide the context of the technique used in the current thesis.

The fundemental physics-informed DL solvers, introduced by [45], are called Physics-Informed Neural Networks (PINNs). PINNs integrate Partial Differential Equations (PDEs) into the loss function of NNs, utilizing automatic differentiation to calculate the partial derivatives of the PDE. The core concept of PINNs lies in aligning the training process, conducted through gradient-based optimizers, with the minimization of a loss function that encompasses a balance between data (which in the context of the aforementioned paper is initial and boundary conditions) and PDE residuals. This formulation results in a combination of supervised and unsupervised loss terms. PINNs are characterized by their meshless nature, ease of implementation, and scalability, with a minimal data requirement typically consisting of sampled boundary and initial conditions. However, a notable limitation of this technique is the necessity to re-train the NN for each new set of initial or boundary conditions, which can impede generalization.

In CFD, the primary source of error is typically discretization error, which is heavily dependent on the grid step size [1]. Essentially, the finer the grid, the lower the discretization error, but this translates to increased computational cost. This issue is further intensified in 2D and 3D simulations, where computational requirements increase exponentially with finer grid resolutions. Additionally, the non-linearity

of many CFD problems and the need for iterative methods in large-scale simulations exacerbate the computational load. In practical engineering applications, such as shape optimization, the frequent need of calling these computationally intensive PDE solvers multiple times intensifies the problem of using expensive fine grid simulations. Furthermore, the resolution of the computational grid also plays a pivotal role in the solver's ability to accurately capture the dynamics governed by PDE, particularly in representing complex features like shocks. For instance, when operating on a fine grid, a typical numerical solver is generally capable of resolving all degrees of freedom inherent in the PDE, thereby capturing sharp features effectively. In contrast, solving the same PDE on a coarser grid often leads to a loss of detail and smearing of features, primarily due to the increased discretization error associated with larger grid steps as previously discussed. Interestingly, if the solution obtained from a fine grid is resampled to match the resolution of a coarse grid, it often retains most of its sharp features. This observation suggests that the limitations in achieving high accuracy on a coarse grid are not necessarily rooted in the spatial resolution itself, but rather in the numerical errors introduced by the solver at this coarser scale.

Hybrid approaches mostly exploit this very fact. By leveraging NNs, these hybrid methods effectively compensate in some way for the numerical inaccuracies, bringing the coarse-grid solutions closer to the fidelity of fine-grid results. This strategy presents a significant advantage, as it allows for computationally efficient simulations on coarser grids while still maintaining a high level of accuracy typically associated with fine-grid computations.

A well received approach involves adopting a predictor-corrector idea. In this methodology, the numerical solver operates on a coarse grid to generate a low-resolution solution, which is then refined or "corrected" by a NN to align with coarsened high-resolution solutions. This approach is rooted in the understanding that, while analytical expressions for numerical errors in most CFD problems are elusive, these errors often exhibit regular, predictable patterns. As demonstrated by [54], such patterns render numerical errors feasible learning objectives for NNs.

This concept has been implemented in various forms, such as the one in [42], where the NN correction is applied as an additive adjustment to the low-resolution solution in an offline way. In this model, the numerical solver is treated as a black box. Another more robust implementation is found in the "Solver-in-the-loop" paper by [54], where the low-resolution solution itself directly feeds into the NN, with the NN output being the corrected solution field.

A critical aspect of these approaches, and generally of methods which integrate the NN within unsteady numerical solvers, is their consideration of the solver's response to corrections. This is particularly crucial as numerical errors can accumulate over time. In an inference scenario, continuously inputting NN-adjusted snapshots back into the solver can lead to solutions that progressively deviate from the training data distribution. This divergence can potentially destabilize the solver, as the

errors compound with each successive iteration. This challenge, and strategies to address it, are also discussed in greater detail in subsequent chapters of the thesis.

Another novel approach involves generating space and time-dependent coefficients for discretized PDEs [2]. While the data-driven coefficients generated for PDE in this approach are highly tailored, enhancing the model's adaptability and precision, it's important to note that their specificity also limits their application exclusively to the particular equation for which they were developed. This means that these coefficients, while highly effective for their intended PDE, cannot be readily generalized or applied to different equations without undergoing a separate, equation-specific training process. However in many fields of study, including engineering, the number of important equations is relatively limited. This aspect significantly mitigates the concern of equation-specificity. For instance, in fields like fluid dynamics, thermodynamics, or electromagnetism, key phenomena are often described by a core set of well-established equations. Therefore, developing tailored coefficients for these specific equations can be highly beneficial and applicable across a wide range of problems within these disciplines.

This method capitalizes on the Finite Volume Methods (FVM) framework, specifically focusing on the reconstruction phase. Traditional FVM relies on assuming a polynomial representation of the quantity across each cell. For example, first-order upwind methods assume a constant value of the variables of the PDE across the cell (first-order reconstruction), while second-order methods assume a linear variation (second-order reconstruction). The innovative approach proposed in the paper [2] replaces this conventional reconstruction step with a NN-driven process. Here, the NN is trained to produce coefficients that, when combined with local field values, effectively transfer these values to the cell faces.

The purpose of this diploma thesis is to extend the ideas and methodologies developed by the Parallel CFD & Optimization Unit (PCOpt) of NTUA, which has been actively engaging in integrating NNs to enhance CFD and optimization processes. This includes work in areas such as turbulence modeling [25], shape optimization [24, 26] and Metamodel-Assisted Evolutionary Algorithms (MAEA) for optimization [12, 49]. The thesis focuses on testing two hybrid methods that combine CFD and ML for accelerating unsteady PDE solutions, specifically the approaches by [54] and [2] as discussed above.

These methods significantly accelerate simulations by enabling results comparable to those obtained on a fine grid to be achieved on a much coarser grid. This effectively means that we can perform complex fluid dynamics simulations with reduced computational resources while maintaining high accuracy in the results. By minimizing the grid size without compromising the quality of the output, these methods also offer a practical solution for conducting detailed simulations in scenarios where computational power is limited, thus making high-fidelity CFD simulations more accessible and feasible in a wider range of settings and applications (e.g. combined with ideas of [44] for flow control). They are applied to two 1D unsteady PDE prob-

lems: the 1D advection equation and the 1D linear acoustics equation, aiming to validate their efficacy and robustness. It must be noted that for the effective generalization of the NN within the solver, it's crucial that the training data encompasses all characteristic features of the flow.

Initially, the thesis involves generating training data through purely numerical solutions of the discretized PDE on a fine grid across various initial conditions and PDE parameters. A portion of the TensorFlow code developed for the purely numerical solution is reused in building the hybrid models. While running the solver, a coarsening process is applied to transfer maximum information from the fine grid results to the coarse grid. The hybrid models are then created using TensorFlow 2.x and Keras Model subclassing, integrating NNs with the previously developed numerical components. It must be aknowledged that the open-source codebasis that was available in the context of the original paper of [2] provided many insights in the details of the relevant method and was very important in successfully developing the code for this thesis. After identifying effective hyperparameters, the models are trained using the coarsened fine grid solutions, saving the NN weights upon completion. This enables the use of the hybrid solver for time integration under different initial conditions or PDE parameters (within the trained data distribution), offering near-fine grid accuracy at significantly increased computational speed. The thesis also includes a parametric study to evaluate the performance of the models under various hyperparameter settings, highlighting that the number of recurrent steps is the most critical factor. Finally, insights and implications are drawn from the results.

Chapter 2

Neural Networks

2.1 Introduction

A Neural Network (NN) is a collection of nodes, or "neurons," connected by edges, or "synapses," which transmit signals. Each neuron receives input, processes it through an activation function, and passes the output it computes to the next layer. The simplest form of a NN is the feedforward neural network, where the information moves in only one direction—from the input to the output nodes, through the hidden nodes. The mathematical representation of a single node is:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

where x_i represents the inputs to the neuron, w_i are the synaptic weights, and b is the bias. The function σ denotes an activation function that introduces non-linearity, adding to the complexity the neuron can model.

This structure allows the neuron to perform a weighted sum of its inputs, add a bias, and then apply a non-linear transformation. The ability of a neuron to adjust its weights and biases through optimization ("training") is fundamental to its capability to adapt and "learn" from a given dataset.

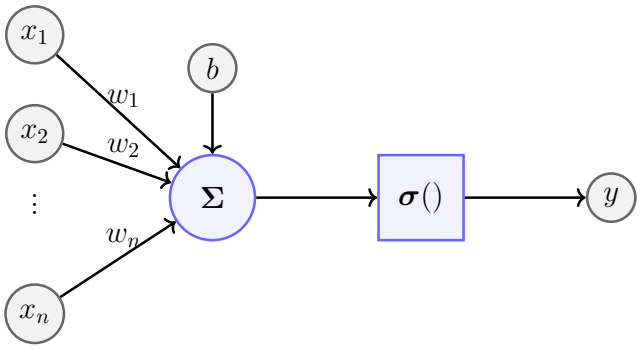


Figure 2.1: *A graphical representation of a single Neural Network node/"neuron". Here $x_1, ..., x_n$ represent the inputs to the neuron, $w_1, ..., w_n$ are the synaptic weights, and b is the bias. The function σ denotes an activation function.*

The ability of a single neuron to "learn" statistical correlations is severely limited [34]. However stacking multiple layers of neurons in a new superstructure (a network) is far more capable. Actually, the reason of the widespread adoption and success of NNs as a regression function can be summarized by the fundamental result known as the Universal Approximation Theorem (UAT) [10, 21, 20]. This theorem states that a feedforward network with a single hidden layer containing a finite number of neurons can succeed -meaning it possesses the minimum inherent complexity- in approximating continuous functions on compact subsets of R^N , under minimal assumptions on the activation function. This represents a guarantee of the expressivity that NNs possess as function approximators [15]. In theory, this means that how well the underlying "real" function is approximated is simply a matter of how representative the data is of the sample space and how well it is optimized.

From a statistical perspective, a NN can be viewed as a complex and highly flexible fitting function. It is designed to model the underlying relationships in data which represent the ground truth ("training patterns") by adjusting its parameters (weights and biases) to minimize a cost function ("loss function"). During training, pairs of input and output data are shown to the NN until the best values of the trainable parameters are found, so that it successfully maps the provided inputs to the provided outputs with minimum error. The loss function measures this error between the predicted output of the NN and the actual target values in the training data.

At its core, a NN is essentially performing a series of function mappings from the input layer to the output layer. This can be expressed as a composition of functions, where each layer f_n transforms the input into a new representation, working from the input layer to the output layer.

$$NN(x, W) = f_N \circ f_{N-1} \circ ... \circ f_1(x, W) \tag{2.2}$$

where x is the input vector and W is the matrix containing all optimizable parameters (weights and biases) of the NN, and assuming identical activation functions at every layer:

$$f_n(x, W_n) = \sigma(W_n x + b_n) \tag{2.3}$$

describes the operation within each layer, where W_n is the weight matrix for the n-th layer, b_n is the bias for the n-th layer, and σ is the activation function.

A typical graph representation can be seen in Fig.2.2.

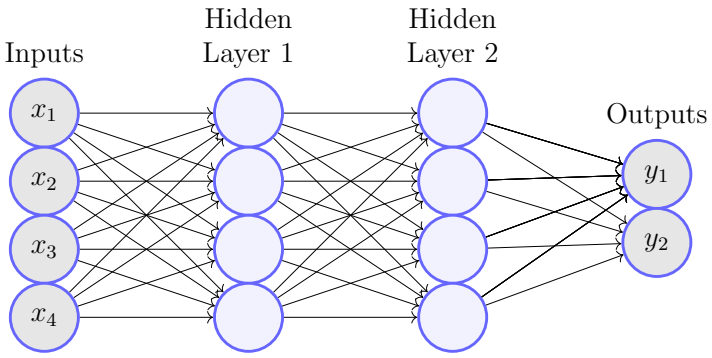


Figure 2.2: *A graphical representation of a typical NN with four inputs, two outputs and two hidden layers in between.*

NNs come with two main costs: the expense of acquiring the data and the cost of training the model, typically accomplished using backpropagation and gradient descent. However, they are highly computationally efficient during testing or runtime. In the context of CFD runs, this efficiency could become particularly valuable during optimization cycles, where the simulation model is called several times as incremental changes are made to the design variables.

They have also several limitations. A substantial amount of data is needed to successfully train a model from scratch (it can be easier to just fine-tune a pre-trained model for widespread applications). Moreover, the larger and more complex the model, the greater the volume of data required to train a useful generalizable NN. For example, in fluid dynamics, data aquisition would mean making several calls to a CFD software or collecting experimental data, both of which would cost greatly in time and resources. Plus, the higher the quality of the data, the more they would cost to aggregate. Additionally, even if these models excel at interpolation, meaning they can predict well within the range of the data they were trained on, they are problematic in extending to cases beyond the training data distribution (extrapolation) [16, 58]. Finally, despite the guarantee provided by the UAT, finding the optimal values for the parameters of a NN to achieve a meaningful or minimal representation is a challenging task [38]. Often, this compels users of this technology to increase

the complexity of the model significantly by substantially increasing the number of trainable parameters. Meaning that to ensure that the network can capture the complexity of the function it is trying to approximate, practitioners often increase the number of layers (making the network deeper) and/or the number of neurons in each layer (making the network wider) to provide the network with more "capacity" or "power" to learn and represent more intricate patterns in the data. However, it is entirely possible that the same NN -without increasing its complexity- could be theoretically sufficient [50] for the fitting task, but does not perform perfectly for a number of reasons (e.g. convergence to local minima, data issues,hyperparameters choices) [13]. This, in turn, makes the model more computationally expensive both during training and at runtime. It also must be noted that the training phase may not be a one-time event. If a data pipeline is in place, the NN may require ongoing fine-tuning or even re-training.

When dealing with physics simulations, another concern is that the phenomena described by PDEs and their spatiotemporal behavior can be exceptionally intricate (as they have more degrees of freedom than ODEs), making it even more challenging for fitting functions to effectively capture them. Multiscale phenomena, for instance (such as turbulent flow), would be a very hard problem, as fitting functions tend to focus on capturing the bigger-scale patterns in the data, whereas it is well known in physics that turbulence, which is a small scale phenomenon, significantly influences the overall solution.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), to be exclusively used in this work, were introduced by [28] as a specialized kind of NN used predominantly in processing data with a grid-like topology [15], such as images, videos and, more recently, simulations. CNNs provide a mechanism to effectively recognize spatial hierarchies in data. Convolutional networks are a type of NN where, in one or more layers, the standard matrix multiplication is replaced with convolution operations.

Generally, a convolution is a mathematical operation that combines two functions to produce a third function. In the context of CNNs, supposing there is a large matrix D of size $N \times M$ representing the input of the operation, and a small matrix F ("filter" or "kernel") of size $n \times m$, the convolution operation involves sliding the kernel accross the data matrix computing the sum of element-wise multiplications at each position. This sum forms a single value in the output ("feature map"). The convolution operation at every spatial position (x,y) of the data matrix D can be computed as:

$$(F * D)(x, y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} F(i, j) D(x + i, y + j) \tag{2.4}$$

More specifically, the detailed steps for a typical 2D convolution (depth is excluded for simplicity here) are the following:

1. **Initialization**: Position the top-left corner of the filter F at the top-left corner of the input matrix D.
2. **Compute dot products**: At each position (x, y), perform element-wise multiplication between the filter F and the overlapping sub-matrix of D, and sum these products. This summation yields a single scalar value that constitutes one element of the output feature map.
3. **Slide the filter**: Move the filter F across the input matrix D by a specified number of matrix elements (the 'stride') horizontally (or vertically once it reaches the edge of the D matrix), repeating the multiplication and summation process at each step.
4. **Form the output**: The values obtained from the summation process at each filter position, form the output of the operation, called "feature map" or "activation map".

This convolution operation is schematically presented in Fig.2.3.

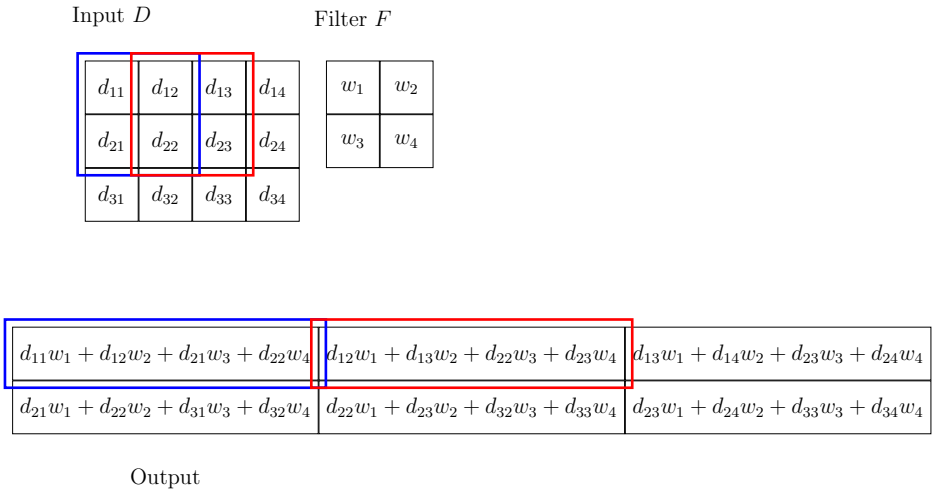


Figure 2.3: *A schematic representation of the convolution operation for a* 3×4 *input matrix* D *and a* 2×2 *filter* F*. Firstly, the dot product of the filter and the first submatrix of* D *is computed (blue box in input and output). Then the filter slides over (in this example, stride* $= 1$*) to the next submatrix and the new dot product is computed (red box in input and output). This is repeated until the whole input matrix* D *has been slided over and the output matrix has been competely filled.*

It is clear that each convolution operation examines a small region of the input matrix D at a time, so when all of these sums are aggregated in the feature map, this will allow it to distill local spatial relationships between different regions in the input.

In CFD, convolution is a common operation: To compute spatial derivatives, for example, via a finite difference scheme, a stencil of coefficients is chosen which then slides over the grid's field values, computing dot products at each position which amount to the approximated spatial deriatives at each point on the grid. This spatial derivatives snapshot would, in that case, be the "feature map".

In general, when depth is also included, filters of convolution must match the input tensor's dimensions. For instance, a 2D field snapshot of $N \times M$ points with L variables would amount to a (N, M, L) tensor. Then, the filter's dimensions would be (n, m, l), where $n \leq N, m \leq M, l = L$. This can also be generalized for a 3D field. Now, as described above, for such a filter a scalar is computed at all possible positions. When a scalar is computed at all positions, the output is an activation map with dimensions $(N', M', 1)$ where $N'\&M'$ values are based on the input tensor size, the input size and the stride (and in general $N' \leq N, M' \leq M$).

Of course in practice, many independent filters N_F -containing independent weights- are convolved with the input, which means that the input is tranformed to a tensor with a shape of (N', M', N_F). All these filters together constitute a convolutional layer. What the convolutional layer has achieved is a re-representation of the input tensor in terms of the weights of each filter. Now this (N', M', N_F) tensor, after being activated by a non-linear function (e.g. ReLU), serves as an input of the next convolutional (or fully-connected) layer and so on and so forth.

It must also be noted that because convolution produces by default a result of smaller spatial dimensions compared to the input (as $N' \leq N, M' \leq M$) padding may be needed. Padding is the process of simply extending the tensor to be convolved with some lines of cells or pixels so as to keep the dimension of every re-representation at the same size (which would mean that $N' = N, M' = M$ is forced) or to prevent it from quickly diminishing. In the context of the CFD example discussed above, with a stencil of coefficients being convolved with the field values of a grid, this is equivalent to adding ghost cells to the computational domain.

In a CNN, the hidden layers include one or more layers that perform convolutions. Otherwise, a typical CNN architecture may also include pooling and Fully Connected (FC) layers. An example of a prototype of a CNN architecture is presented in Fig.2.4

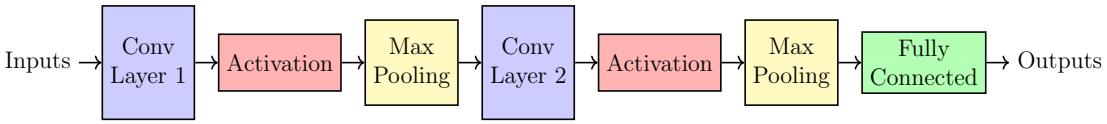


Figure 2.4: *A simple CNN architecture could include convolutional layers, activation functions, pooling layers (e.g. max pooling) and a FC layer.*

Pooling layers perform a downsampling operation of some sort but their usage has generally been declining [51], so they will not be further analyzed. Finally, FC layers are commonly used as the final (or couple of final) layer(s), as a way to flatten the activation maps to a vector. In this thesis, pooling and FC layers are not used.

One of the key advantages of CNNs is their inherent ability of translation invariance [28]. Once a feature is learned at one location, the same feature can automatically be recognized at a different location, making CNNs very efficient in recognizing patterns no matter where they appear in the field of view (or on a grid, in a CFD application). This is particularly useful in tasks like object detection and classification in images, where the location of the object is not fixed. But it is also one of the main reasons that CNNs are the most prominent choice when dealing with CFD simulations. The way that hyperbolic systems are treated -which comes down to the method of the characteristics- clearly indicates that information translation is the most dominating pattern in fluid dynamics. This means that having translational invariance built-in from the start would provide huge gains in saving costs of the NN having to learn it purely from data by itself. It is noted that it is also possible to attain rotation or any transformation invariance for a CNN, but this comes with the cost of significantly altering (and increasing) the training data accordingly.

2.3 Neural Network setup

When setting up a NN, various elements need to be carefully considered to ensure optimal performance. Below some of these critical components are briefly presented: activation functions, weight initialization, and regularization.

Activation Functions

Activation functions are an essential component of NNs as they are the reason the NN can learn complex behavior. If they were not present, multiple layers of the NN would just collapse to a single linear transformation, which has very limited capacity to model statistical correlations. This can be seen in the following formula:

$$y = W_N W_{N-1} ... W_1 x = W_{tot} x \tag{2.5}$$

However, if an activation function is included at each layer (here identical activations are presumed at each layer), a more complex structure can be built:

$$y = \sigma_N(W_N\sigma_{N-1}(...W_2\sigma_1(W_1x)) \tag{2.6}$$

The choice of activation function has a big influence on the NN's training speed and performance. At the beginnings of the AI field, the activation function that was used was a step function [46]. This was an attempt to mimic the way the human brain was thought to function at the time. However its shortcomings of being non-differentiable and always saturated (which means the neuron's response becomes insensitive to changes in its input) has now rendered it obsolete. Nowadays, most common activation functions include sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit (ReLU), Leaky ReLU (LReLU) and Exponential Linear Unit (ELU) functions. The behaviour of their output y for an input z, can be observed in Fig.2.5. Then, they are further analyzed below.

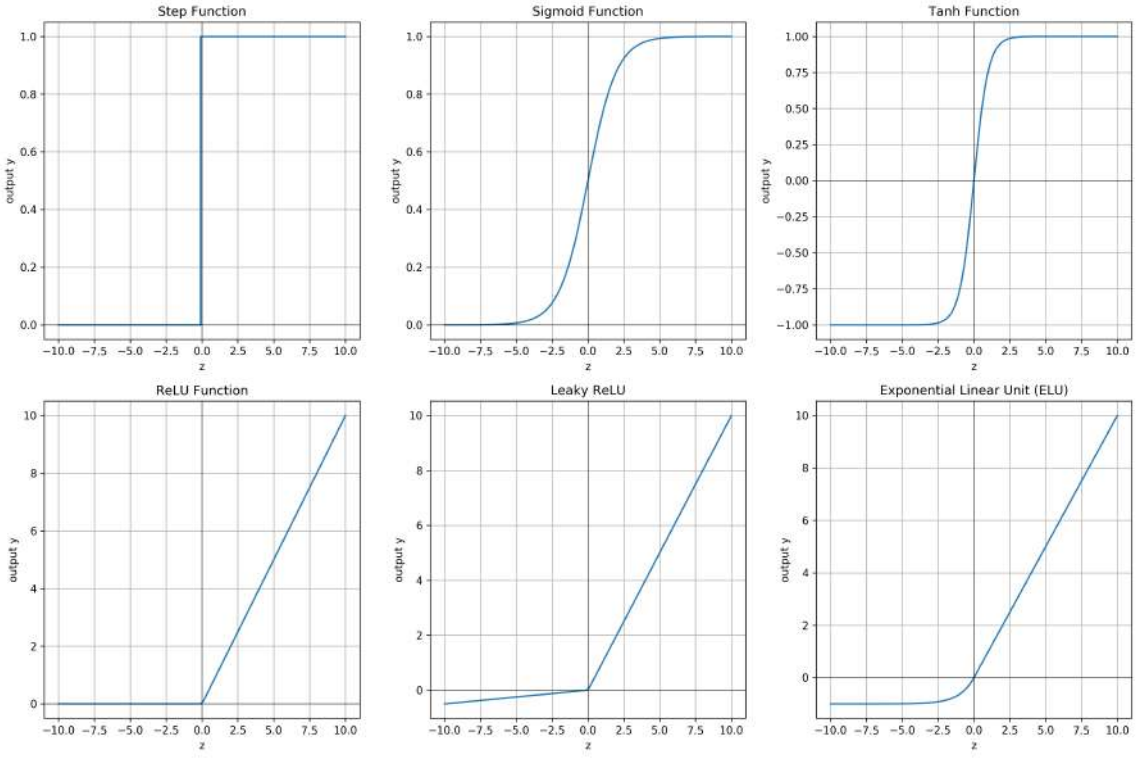


Figure 2.5: *A graphical representation of common activation functions.*

Sigmoid activation function and the hyperbolic tangent one are given by

$$sigmoid(z) = \frac{1}{1+e^{-z}}, \quad tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.7}$$

Sigmoid was one of the first moderately successful activation functions [47]. Sigmoid and hyperbolic tangent activation functions both suffer from some disadvantages. Firstly, if the neuron is saturated, meaning that $z = -10$ or $z = 10$ (see Fig.2.5), the gradients are "killed" during backprogation. The core issue is that the sigmoid function squashes its inputs to a range between 0 and 1, and its derivative is maximally 0.25 (at the origin). This means that if the neuron is saturated, when the input z is relatively small or big, the gradients during backpropagation will be very small. Meaning that when these small gradients are multiplied through an increasing number of layers, this leads to an exponential decrease of the gradient magnitude in the layers near the start. This makes it difficult for the network to learn, as the weights in the earlier layers receive very small updates, leading to slow convergence.

Next, ReLU and its most common variants LReLU and ELU are presented, though there are other ones like Gaussian Error LU (GELU) [18] which has shown promising results in CFD-DL combination settings [26].

In [14], which introduced ReLU, it was demonstrated that deep NNs with ReLU activation functions can achieve better performance than those with traditional sigmoid or tanh activations, especially the deeper the architectures get.

$$ReLU(z) = \max(0, z) \tag{2.8}$$

LReLU [35] was proposed to mitigate the "dying ReLU" problem. This problem occurs when neurons activated by ReLU are continuously fed negative inputs which results in them constantly outputing zeros (see Fig.2.5). This means that they do not contribute to the learning process anymore. Leaky ReLU allows a small, non-zero gradient (regulated by the constant a in Eq.2.9) to pass through when the input is negative, which allows backpropagation to continue updating the weights of these neurons.

$$\text{LReLU}(z) = \begin{cases} x & \text{if } z > 0 \\ \alpha x & \text{if } z \geq 0 \end{cases} \tag{2.9}$$

ELU [9] was proposed to preserve the benefits of LReLU while also providing real differentiability everywhere (whereas ReLU and LReLU are differentiable everywhere except at $z = 0$), where, in practice, a value for the gradient is arbitrarily chosen (0 or 1 usually).

$$\text{ELU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(\exp(z) - 1) & \text{if } z \geq 0 \end{cases} \tag{2.10}$$

where again, constant a regulates how small or big the gradient is for negative input values $z \leq 0$.

ReLU and its variants are generally preferred for hidden layers in deep networks due to their computational efficiency and also due to their proven ability to mitigate the vanishing gradient problem as previously discussed. However, a significant advantage of sigmoid and tanh functions over LU variants are that they are C^∞ functions. This means that their derivatives of higher order are also existent and computable. This is extremely important in certain cases where higher order derivatives are needed, for instance in a PINN [45] where derivatives of a PDE of an arbitrary order may be needed or for using a second-order optimization method like L-BFGS [32, 36]. Hence in these cases the hyperbolic tangent function may still be the best alternative. In the present thesis, ReLU activation function was chosen, mainly because it still dominates the CNN literature and also due to its reduced computational cost (e.g. in contrast to ELU).

Weight Initialization

In a NN, weights define the strength and direction of influence between neurons. The adjustment of these weights through training is what enables the network to learn from data. The training process involves many iterations of updating the weights to minimize the difference between the actual output of the network and the desired output, a process guided by backpropagation (see sec.2.5) and optimization techniques like Stochastic Gradient Descent (SGD). More details for the training procedure of a NN can be found in sec.2.4.

Weight initialization is the process of assigning initial values to these weights and is critical to the NN's training behavior. Starting with arbitrary or random values might lead to problems: excessively large weights can cause neurons to become saturated, leading to issues such as exploding gradients, where changes in weights blow up. In constrast, starting with very small weights might result in the vanishing gradients problem, where changes in weights are so small that the optimization process of the NN becomes exceedingly slow or even stops [3]. Different initialization strategies aim to balance these concerns by setting the weights to values that are neither too large nor too small and to preserve a reasonable and diverse norm of neuron outputs, and hence, a reasonable norm of gradients during training.

Xavier initialization [13] is designed to address the issue of initializing the weights in a Deep NN (DNN) in such a way that the variance of the inputs is maintained through each layer. The key idea is to keep the scale of the gradients roughly the same in all layers, preventing the gradient problems that were described previously. The Xavier initialization sets a layer's weights W to values randomly drawn from a Gaussian distribution with zero mean and a variance of $2/(N_{in}+Nout)$, where N_{in} is the number of neurons feeding into the layer and N_{out} is the number of neurons the layer feeds into. This choice is based on the assumption that the activation function is symmetric (e.g. tanh). However, even for non-symmetric activation functions this initialization method provides a good starting point which has been shown to work quite well, and has even become the default choice for weight initialization in Keras.

He initialization's [17] main idea is to address the problems associated with training DNNs that use ReLU -or one of its variants- as activation functions. It sets the initial weights of the network layers to random values drawn from a Gaussian distribution with mean 0 and variance $1/N_{in}$, where N_{in} is the number of input units in the weight tensor.

As one can see in the variance term, Xavier initialization considers both the number of input and output units, while He initialization only considers the number of input units. This is due to ReLU activations not outputing values in a symmetric way around zero, and because when passing through a ReLU activation, the variance is halved (as all negative values are sent to zero).

The significance of weight initialization becomes more relevant as the NNs get deeper. Then, the issues of vanishing or exploding gradients across layers are magnified as the gradients pass through more layers during the backward pass (meaning they are multiplied with inappropriately big or small values of weights or resulting activations more times). The choice of initialization method often depends on the activation function used in the network. For instance, He initialization is generally preferred for networks with ReLU activation functions, as it accounts for the non-linear nature of ReLU. The overarching goal of these methods is to ensure a stable and steady flow of gradients through the network so as to foster a more effective learning process. In the present thesis, both methods gave similar results for both Xavier and He initialization, so Xavier initialization was chosen.

Regularization

Regularization techniques are essential in the construction and optimization of NN models to prevent overfitting (and hence improve generalization), and maintain model simplicity. Overfitting occurs when a model learns patterns specific to the training data, reducing its accuracy on new data that it has not explicitly encountered while optimizing but are inside the training data distribution.

Generally, there are arbitrarily many functions that fit any arbitrary dataset. Regularizers are a way to kind of circumvent this problem by adjusting the NN to produce solutions that exist in a certain part of the solution manifold. This helps the NN choose between all these infinitely many alternatives, that all cupture the data statistics, to be a meaningful one based on the problem at hand. Practically, regularization introduces additional constraints into the model to penalize overly complex solutions. It can also be viewed as a way of trading off training loss and generalization loss on a test set. Next, some common regularization techniques for NNs are presented.

L_1 regularization [53] introduces a penalty to the loss function equal to the absolute value of the weights (a L_1 norm of the weights vector) which promotes sparsity, and L_2 regularization adds a penalty equal to the square of the magnitude of the weights (a L_2 norm of the weights vector), effectively promoting small and diffused

values, with consistent norms to one another. A combination of both regularization techniques, called Elastic Net [60], is also possible. Mathematically these can be described in the following way:

$$L_{reg} = L + \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} w_i^2 \tag{2.11}$$

where:

- L_{reg}: Regularized loss function.
- L: Original loss function.
- λ_1: Control knob of the strength of the L_1 regularization term.
- λ_2: Control knob the of strength of the L_2 regularization term.
- $|w_i|$: Absolute value of the i^{th} NN weight.
- w_i^2: Square of the i^{th} NN weight.
- n: Number of weights.

Given the regularization parameters λ_1 and λ_2, the regularization method applied is:

$$\text{Regularization method} = \begin{cases} \text{L2 regularization} & \text{if } \lambda_1 = 0 \text{ and } \lambda_2 \neq 0, \\ \text{L1 regularization} & \text{if } \lambda_2 = 0 \text{ and } \lambda_1 \neq 0, \\ \text{Elastic Net} & \text{if } \lambda_1 \neq 0 \text{ and } \lambda_2 \neq 0. \end{cases} \tag{2.12}$$

Dropout [19] is a computationally inexpensive way to regularize large neural networks. During training, a proportion of neurons is randomly set to zero within each update cycle. This prevents units from co-adapting too much and forces the network to learn more robust features.

Early stopping [39] involves halting the training process when the performance on a validation set starts to deteriorate. This simple approach assumes that as the model begins to overfit the training data, its performance on the validation set will begin to decline. There have been papers that challenge this approach as, in many problems, the performance on the validation set might firstly get worse but then get better again [40].

In the present thesis, regularization is treated with extreme caution. In this work there are included techniques that even though they serve a different primary purpose, also have a regularizing effect, in the sense that they too enforce solutions that have certain significant characteristics. This includes the layer that enforces polynomial accuracy (see 3.3.2) and the recursion or stockpiling of multiple steps

in the loss function (see sec.3.3.1). In the current thesis, using L_1 and L_2 regularization was tested and led to worse results during inference time. Hence, from the traditional regularizers, only early stopping was used.

2.4 Training a Neural Network

2.4.1 Training algorithm

Algorithmically, training a NN consists of the following steps:

1. **Acquire data:** Collect and preprocess the data suitable for the neural network. It should be expected that NNs perform well on test data that lie inside the training data distribution.
2. **Initialize the architecture and parameters of the NN:** Define the structure of the neural network and initialize parameters (weights and biases).
3. **Forward pass:**
   - Input data is passed through the network layer by layer. At each layer, the input undergoes a linear transformation: $z = Wx + b$.
   - The result is, then, passed through an activation function σ: $y = \sigma(z)$. This introduces non-linearity, allowing the network to learn complex patterns.
   - This process continues until the output layer is reached, producing the network's prediction.
   - The loss (error) is calculated using a loss function (e.g., Mean Squared Error (MSE) or Mean Absolute Error (MAE)). The loss quantifies how close the network's prediction is to the target values associated with the training set.
4. **Backward pass:** Compute the gradient of the loss function with respect to each weight using the chain rule. This process, called error backpropagation, is performed, during which information travels backwards from the output layer to the input layer.
5. **Update weights:** Use an optimization algorithm (e.g., Stochastic Gradient Descent or Adam) to adjust the weights in the direction that minimizes the loss, which is the direction of the gradient of the Loss with respect to the weights.

The forward and backward propagation steps are repeated for many iterations or epochs over the training set. An epoch is completed when all data points have been used once in training.

2.4.2 Hyperparameters tuning

Hyperparameters in NNs are the parameters whose values are set before the learning process begins. Unlike model optimizable parameters, which are learned during training, hyperparameters are predefined and govern the overall configuration and performance of a neural network. NN hyperparameters include the number of layers -or more generally, the number of optimizable parameters in the NN-, the batch size, the size of the learning rate, the number of epochs and the strength of regularization terms in the Loss function. The choice of hyperparameters affects how quickly a model learns, its overall performance, and its ability to generalize from training data to unseen data. Hence, selecting the right set of hyperparameters can potentially be the difference between a mediocre and a state-of-the-art model.

Hyperparameter tuning is a critical aspect of designing and training machine learning models, especially in cases combining CFD and DL, where models can be incredibly complex and sensitive to the settings (see sec.5.5). The typical methods for hyperparameter optimization include grid search, random search, Bayesian optimization, and meta-heuristic algorithms (Evolutionary Algorithms (EA), Particle Swarm Optimization (PSO)) [4].

Traditionally, grid and random search algorithms have been the primary methods for hyperparameter optimization. Grid search systematically works through multiple combinations of parameter values, evaluating and comparing the models' performances. It is represented as an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. However, grid search suffers from the curse of dimensionality: as the number of hyperparameters increases, the number of evaluations needed grows exponentially. This makes it computationally infeasible for large datasets or models with many hyperparameters. Random search, on the other hand, randomly samples the space of possible hyperparameters. For certain types of problems and hyperparameter spaces, it can find good solutions faster than grid search [4].

Next, Bayesian optimization uses past evaluation results to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function and then uses that model to select the most promising hyperparameters to evaluate in the true objective function. This approach is particularly powerful when the budget for evaluations is limited or when each evaluation is time-consuming.

Finally, meta-heuristic algorithms such as EA [57] and PSO [55] are designed to explore large and complex search spaces and can often find good solutions quickly for a wide range of problems.

Each method has its trade-offs in terms of computational cost, convergence speed, and suitability for the dimensionality and nature of the hyperparameter space. One might also explore hybrid approaches [11], which combine the strengths of different methods. In this thesis, random search is the technique employed for identifying

effective hyperparameters for the NNs.

2.5 Automatic Differentiation and AD&DL frameworks

Automatic Differentiation (AD) [56], is a technique to evaluate the (exact) derivative of a function specified by a computer program. AD exploits the fact that every operation, no matter how complex, is composed of elementary arithmetic operations and elementary functions, which are almost always differentiable [22]. By suitably applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically and efficiently. This capability is fundamental in DL, where optimization algorithms require the computation of gradients with respect to a great number of design variables. There are two modes of AD, forward [56] and reverse [31], each being computationally efficient in different scenarios: reverse mode is more efficient when the dimensionality of the design variables is bigger than the dimensionality of the output and conversely, forward mode is cheaper in the opposite cases. In NNs, reverse-mode AD (backpropagation [47]) is preferred because the number of design variables (weights and biases) typically far exceeds the number of output variables.

In recent years, a range of frameworks have emerged that significantly reduce the computational demands and programming complexity associated with AD and DL. Core to this development are engines like Tapenade, TensorFlow, PyTorch and JAX. Many of these feature integration of specialized tools and auxiliary libraries, such as Keras for TensorFlow, which specifically focus on facilitating DL techniques. In this thesis, reverse AD (backpropagation) was employed for the backward pass (see section 2.3) through the hybrid models, which encompass both the NN and the differentiable numerical model components. This crucial process was handled using TensorFlow 2.x in conjunction with Keras.

Chapter 3

Hybrid Solvers

3.1 Introduction

In this thesis, the term "model" refers to any method or process that takes the input $\rho(t)$ and generates the output $\rho(t + \Delta t)$. The term "solver" refers to the model along with a loop iterating over all desired time steps. The discretization indices used throughout this chapter are: $i \in [1, M]$ for space and $n \in [1, N]$ for time. The 1D advection equation, which is the first case examined in this work (see Ch.4), serves as an example for the following sections.

A numerical model is a discretized version of the ODE/PDE. A typical numerical solver for the 1D advection equation using a FVM second-order scheme and no limiters, with a 3-point stencil is presented in Fig.3.1. In this model, spatial derivatives are computed based on the field snapshot values and the pre-defined Taylor-expansion derived coefficients. Fluxes are then calculated using these spatial derivatives and the field snapshot at the next time step is obtained through time integration. The same process can be implemented for any PDE. This model iterated over all desired time steps would constitute a complete numerical solver.

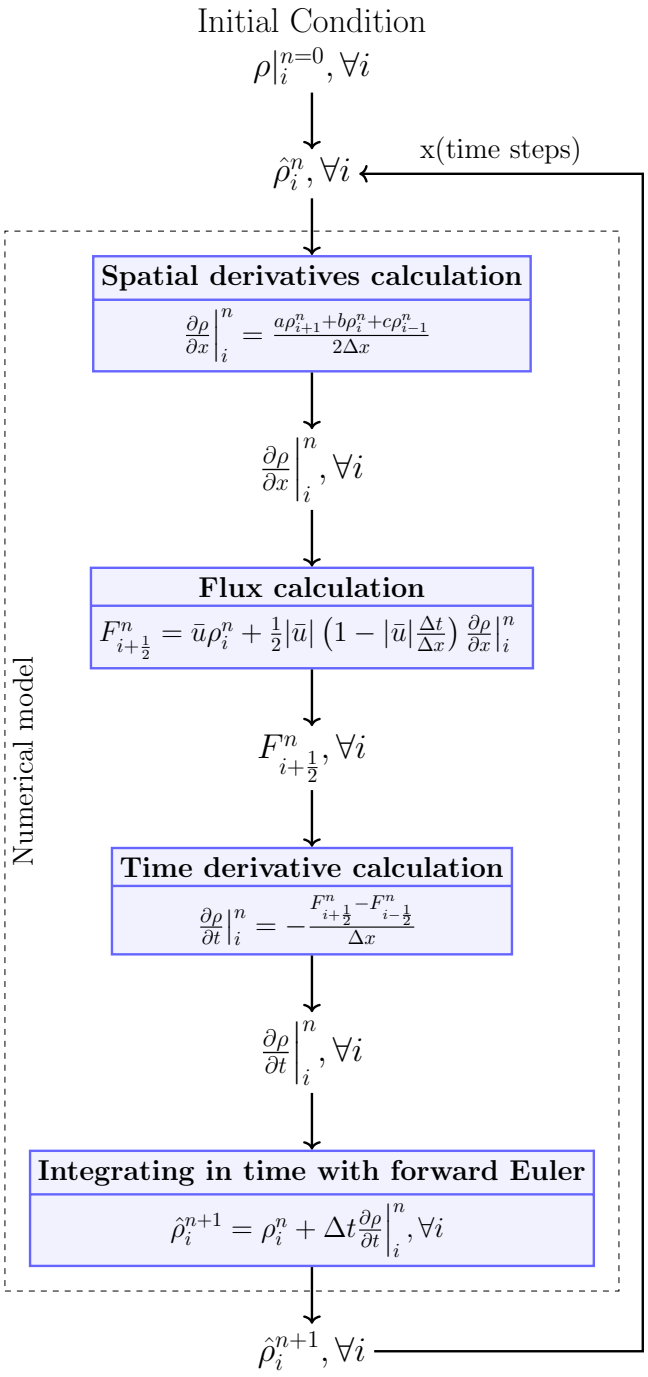


Figure 3.1: *Flowchart of a typical numerical solver for the 1D advection equation (see Ch.4) using a second-orde, not limited, FVM scheme and no limiters, with a 3-point stencil. First, spatial derivatives are computed based on the field snapshot values and the pre-defined Taylor-expansion derived coefficients $\{a,b,c\}$. Then, fluxes are computed, and finally, the field snapshot at the next time step is obtained through time integration using an explicit first-order Euler scheme. The same process can be implemented for any PDE.*

The approach examined in this thesis is the combination of a numerical model with NNs aiming to speed up the numerical solver. This chapter presents two specific hybrid models. In the first one, as initially proposed by [2], a CNN is employed to produce space and time dependent nodal coefficients, which are fed into the numerical model instead of using the traditional Taylor expansion derived coefficients. In the second hybrid model (see also [54]), a CNN provides space and time dependent corrections to the prediction of the numerical model in an online manner.

The essential parameters for the two hybrid methods are defined in Tab.3.1.

Parameter	**Description**
K	# different initial conditions
CR	coarsening ratio $= 2^{\beta}, \beta \in \mathbb{N}^* \backslash \beta < \lambda_2$
λ_1	factor that defines # grid points
λ_2	exponent that defines # grid points
M_f	# points on the fine grid $= \lambda_1 \times 2^{\lambda_2}$
M_c	# points on the coarse grid $= \frac{M_f}{CR}$
N_f	# time steps for the fine grid
N_c	# time steps for the coarse grid
S	stencil size

Table 3.1: *Important parameters and their description. Subscript f is used to denote parameters associated with the fine grid and subscript c to denote parameters associated with the coarse grid. β can be any natural number. The formula providing the number of points on the fine grid is defined in a way that facilitates experimenting with various coarsening ratios.*

3.2 Hybrid models presentation

The hybrid methods (see also [2, 54]) consist of the following basic stages:

Stage 1: Integrate the discretized PDE in time, on the fine grid (M_f points), for one initial condition (e.g. a square waves of some height). This is done for as many time-steps N_f as necessary until all characteristics of the flow have manifested (e.g. including both transient and periodic steady state). While the solver is running, coarsen in time (via down-sampling) to N_c time instances, and in space (via averaging) to M_c points and save these coarsened fields into storage. Repeat the same process for K different initial conditions (e.g. K square waves of different heights). These K sets of solutions are indepedent and can, in principal, be parellized. At the end, the training dataset is $K \times N_c \times M_c$ in size.

Stage 2: Train the CNN on the previously coarsened data. The output of the CNN are nodal coefficients (for the LI method) and field corrections (for the LC method).

Stage 3: A new initial condition of high resolution, which has not been seen by the model during the training phase, (e.g. a square wave of different height) is coarsened and fed into the hybrid solver. The CNN produces either nodal coefficients (in the LI method) or field corrections (in the LC method) and the numerical parts of the solver use these to predict the solution at the next time steps.

Each of the stages are described in detail in the following subsections. For simplicity's sake, the procedure is analyzed for the case of the 1D advection equation with constant values to the parameters of Tab.3.1. Specifically, $K = 30$ different initial conditions are used (square waves with varying heights and widths), the number of fine time steps are $N_f = 1536$, of coarse time steps $N_c = 192$, of fine grid points are $M_f = 384$ and of coarse grid points are $M_c = 48$. Also, a 3-point stencil is chosen (meaning that $S = 3$), the coarsening ratio is $CR = 8$.

3.2.1 Stage 1: Run the numerical solver, coarsen and collect the data

The numerical solver of the discretized problem of 1D advection (see Fig.3.1) runs on a $M_f = 384$-points grid ("fine grid") for $N_f = 1536$ time steps (which correspond to 2-periods) for all 30 initial conditions (can be done in parallel). While this solver runs, only every $CR = 8$-th snapshot is averaged in groups of $CR = 8$ and, then, stored. This corresponds to temporally coarsening (via downsampling) the 1536 time steps to 192 ones and spatially coarsening the 384 to 48 grid points (according to $CR = 8$). At the end of stage 1, a $K \times N_c \times M_c$ dataset is obtained, whose dimensions correspond to different initial conditions, time steps and spatial points respectively. In the case of 1D advection, a $30 \times 192 \times 48$ dataset is obtained. Stage 1 is presented in Fig.3.2.

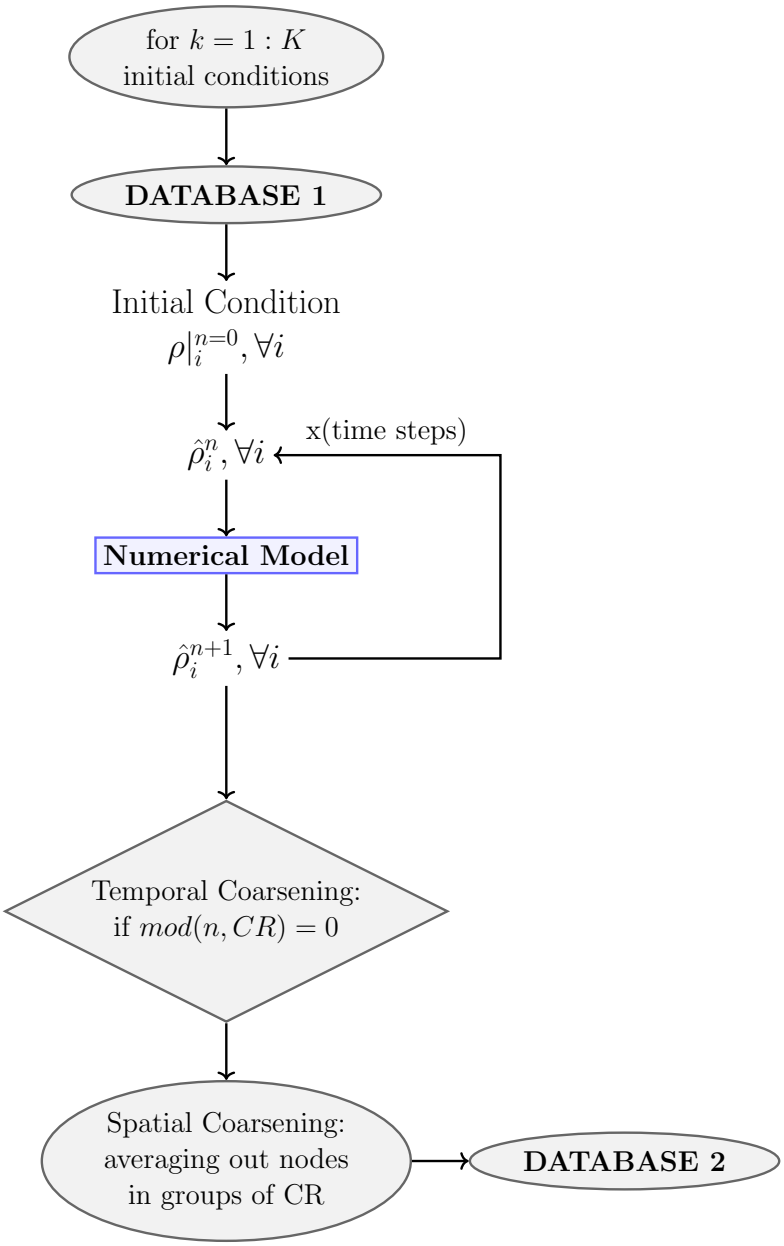


Figure 3.2: *Flowchart of Stage 1: running the numerical solver (fine grid), spatiotemporally coarsening and collecting data. For the 1D advection case (where $CR = 8$), every 8-th time step, the snapshot is spatially coarsened and saved into storage. Spatial coarsening is done via averaging out every group of $CR = 8$ nodes into a single one. The (if) node of the flowchart practically executes temporal coarsening through downsampling.*

These coarsened fields in storage constitute the data that the CNN is going to be trained on. This means that the quality of the training data depends on how well the coarsening step maps the high-resolution fields on the coarse grid. In the square waves case, because of the simple geometry involved, this is relatively easy

to perform. For more details see sec.4.3.

3.2.2 Stage 2: Training the CNN

This is the stage where the two methods of LI and LC start to diverge. Each one is presented in the following two subsections. In the training dataset there are K different initial conditions and N_c available coarse time steps. Meaning that in general there are $K \times N_c$ available (coarsened) snapshots or 30×192 for the discussed case, meaning 5760 available snapshots. If n is the counter over the $N_c = 192$ time steps, then the snapshots from $n = 0$ to $n = N_c - 1 = 191$ (which amount to $K \times (N_c - 1)$ snapshots in general and $30 \times 191 = 5730$ in the specific case) constitute the training input data. And the -displaced in time- snapshots from $n = 1$ to $n = N_c = 192$ ($K \times (N_c - 1)$ or $30 \times 191 = 5730$ snapshots) constitute the training output data. Recall that both hybrid models operate on the coarse grid.

LI method

For all the snapshots in the training input in parallel:

Step 1: The weights of the NN are initialized in an appropriate manner (Xavier initialization).

Step 2: Firstly, the coarsened field snapshots $\rho_i^n, \forall i$ are fed into the hybrid model (see Fig.3.3). These are processed by the CNN which produces nodal coefficients for the $M_c = 48$-points (coarse) grid. This means that in the current case, ($S = 3$-point stencil), 48 different groups of 3 coefficients are produced for each snapshot. As discussed, this is done in parallel for all available snapshots in the training input, meaning that, at the end, $(K \times (N_c - 1)) \times M_c \times S$ or in the specific case $(30 \times 191) \times 48 \times 3 = 5730 \times 48 \times 3$ coefficients are produced.

Step 3: These coefficients are fed into the numerical part of the solver (on the coarse grid), which is the FVM with a second-order scheme with a 3-point stencil (no limiter).

Step 2a: The coefficients are used in conjuction with the field values (see Step 1) to compute the spatial derivatives at every grid point, as follows

$$\left.\frac{\partial \rho}{\partial x}\right|_i^n = \left.\frac{a_i \rho_{i+1} + b_i \rho_i + c_i \rho_{i-1}}{2\Delta x}\right|_i^n, \forall i \tag{3.1}$$

Step 2b: The spatial derivatives are used to compute fluxes at the

faces

$$F^n_{i+\frac{1}{2}} = \bar{u}\rho^n_i + \frac{1}{2}|\bar{u}|\left(1 - |\bar{u}|\frac{\Delta t}{\Delta x}\right)\frac{\partial \rho}{\partial x}\Big|^n_i, \forall i \tag{3.2}$$

Step 2c: The fluxes are used to update time derivatives of ρ and, using forward Euler, the field snapshot at the next time step is

$$\hat{\rho}^{n+1}_i = \rho^n_i + \Delta t(-\frac{F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}}{\Delta x}), \forall i \tag{3.3}$$

These are the field snapshots estimated by the hybrid model.

Step 4: The hybrid model predictions $\hat{\rho}^{n+1}_i, \forall i$ for all training data, are a dataset of size $(K \times (N_c - 1)) \times M_c$ or $(30 \times 191) \times 48 = 5730 \times 48$ in the discussed case. Using a Loss function, these are compared with the coarsened high fidelity (Hi-Fi) snapshots that are available in the training output data $\rho^{n+1}_{\forall i}$ (of the same size).

Step 5: The weights of the CNN inside the hybrid model are updated, using Minibatch Gradient Descent, to minimize the Loss function. The weight update for the hybrid model utilizes the chain rule :

$$W^{new} = W^{old} - lr\frac{\partial L}{\partial W}\Big|^{old} = W^{old} - lr\frac{\partial L}{\partial \phi_4}\frac{\partial \phi_4}{\partial \phi_3}\frac{\partial \phi_3}{\partial \phi_2}\frac{\partial \phi_2}{\partial \phi_1}\frac{\partial \phi_1}{\partial NN}\frac{\partial NN}{\partial W}\Big|^{old}, \tag{3.4}$$

where L stands for the Loss function, W is the matrix of all the weights of the NN, $\phi_{1,2,3,4}$ are defined in Fig.3.5. Note that both the CNN and the numerical model part of the hybrid model are Automatically Differentiated (reverse mode). This is only possible for differentiable numerical parts, which is why traditional limiters are excluded from hybrid models.

This whole procedure (for the LI method) is summarized in Fig.3.3

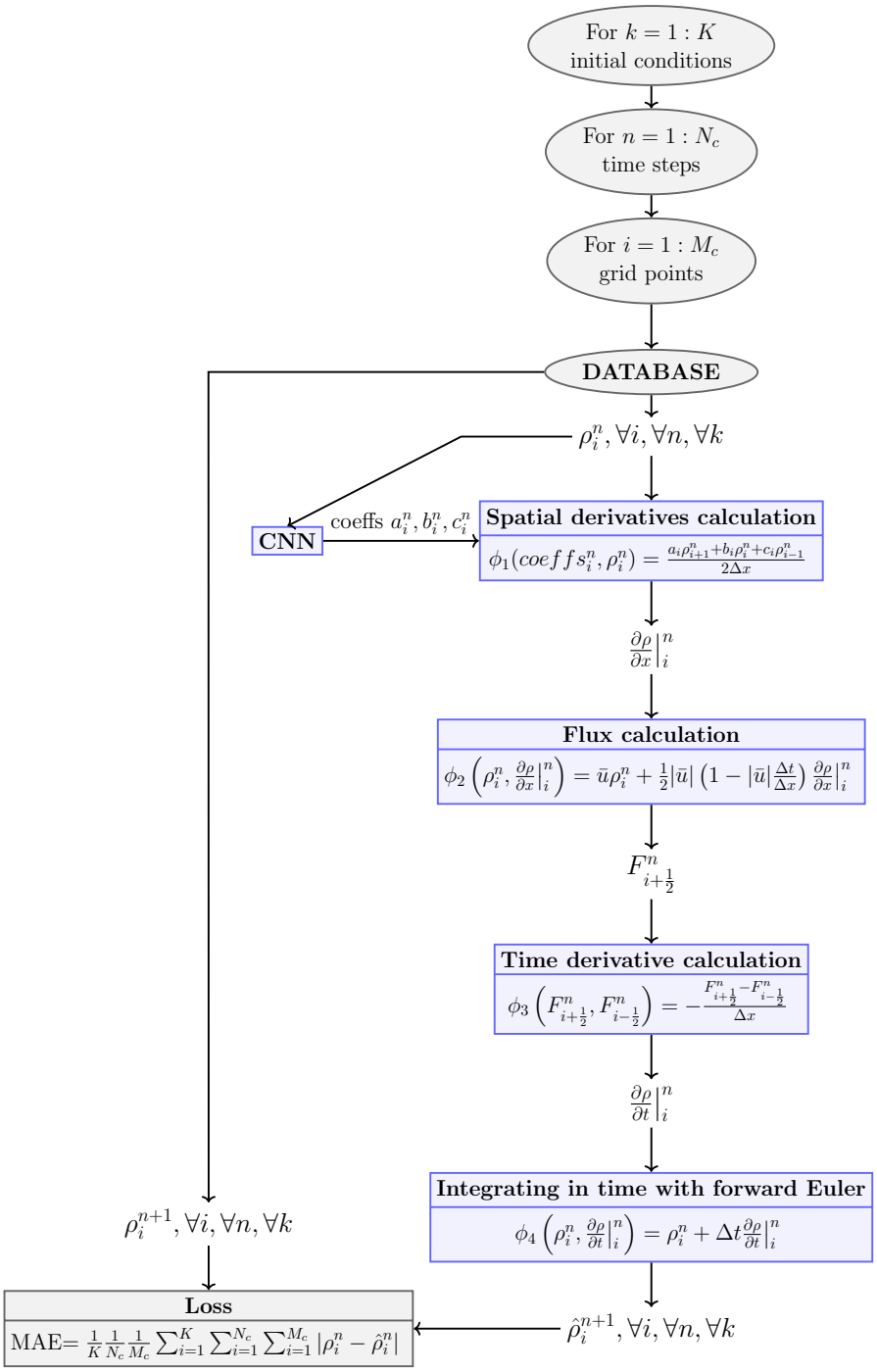


Figure 3.3: *LI method flowchart of the forward pass (see Sec. 2.4) of a single epoch (full-batch for simplicity) of the training phase (Stage 2). The data are high-resolution numerical solutions that have been coarsened. For all initial conditions $k \in [1, K]$, all (coarse) time steps $n \in [1, N_c]$ and every grid point $i \in [1, M_c]$ (coarse grid) the hybrid model, given the n^{th} precomputed snapshot, estimates the $n + 1^{th}$ snapshot. Training is done by comparing all $n + 1^{th}$ estimated snapshots to the $n + 1^{th}$ precomputed (coarsened Hi-Res) snapshots and updating the weights in the direction of the gradient of the Loss function w.r.t. the weights (computed via backprop).*

Steps 2 to 5 are iterated for as many epochs as needed until the Loss function has sufficiently converged. At the end of stage 2, after the training is over, the synaptic weights of the NN are saved. Then the hybrid model can be used to produce solutions for an initial condition out of its training patterns (see sec.3.2.3). However, before analyzing the inference part of this model, the LC method's training stage is also going to be described.

LC method

For all the snapshots in the training input, meaning in general for $K \times (N_c - 1)$ and in the 1D advection case for $30 \times 191 = 5730$ in parallel:

Step 1: The weights of the NN are initialized (Xavier initialization).

Step 2: Firstly, the coarsened field snapshots $\rho_i^n, \forall i$ are fed into the hybrid model (see Fig.3.4). The field snapshots $\rho_i^n, \forall i$ (which are $K \times (N_c - 1)$ in general and 5730 in the discussed case) are processed by the numerical part of the hybrid model which produces temporary new field snapshots (of the same size) at the next time step $\hat{\rho}_i^{n+1}|_{temp}, \forall i$.

Step 3: Then, the 5730 temporary-new field snapshots $\hat{\rho}_i^{n+1}|_{temp}, \forall i$ are fed into the CNN which provides corrections for the $M_c = 48$-points (coarse) grid. This means that $(K \times (N_c - 1)) \times M_c$ or $(30 \times 191) \times 48 = 5730 \times 48$ different corrections $CORR_i^{n+1}, \forall i$ to the temporary-new fields are produced. (This means that at each different point in spacetime a different correction is generated).

Step 4: The new field snapshots $\hat{\rho}_i^{n+1}, \forall i$, are the sum of the temporary-new field snapshot and the produced corrections $\hat{\rho}_i^{n+1} = \hat{\rho}_i^{n+1}|_{temp} + CORR_{\forall i}^{n+1}, \forall i$.

Step 5: The hybrid model predictions $\hat{\rho}_i^{n+1}, \forall i$ ($(K \times (N_c - 1)) \times M_c$ new snapshots in general and 5730×48 in the examined case) are compared with the coarsened Hi-Fi snapshots that are available in the training output data $\rho_i^{n+1}, \forall i$ (obviously of the same size as the predictions). This is done using the Loss function.

Step 6: The weights of the CNN inside the hybrid model are updated, in the direction that the Loss function is minimized, similarly to the LI method.

Steps 2 to 6 are iterated for as many epochs as needed until the Loss function has sufficiently converged as previously discussed in the LI method.

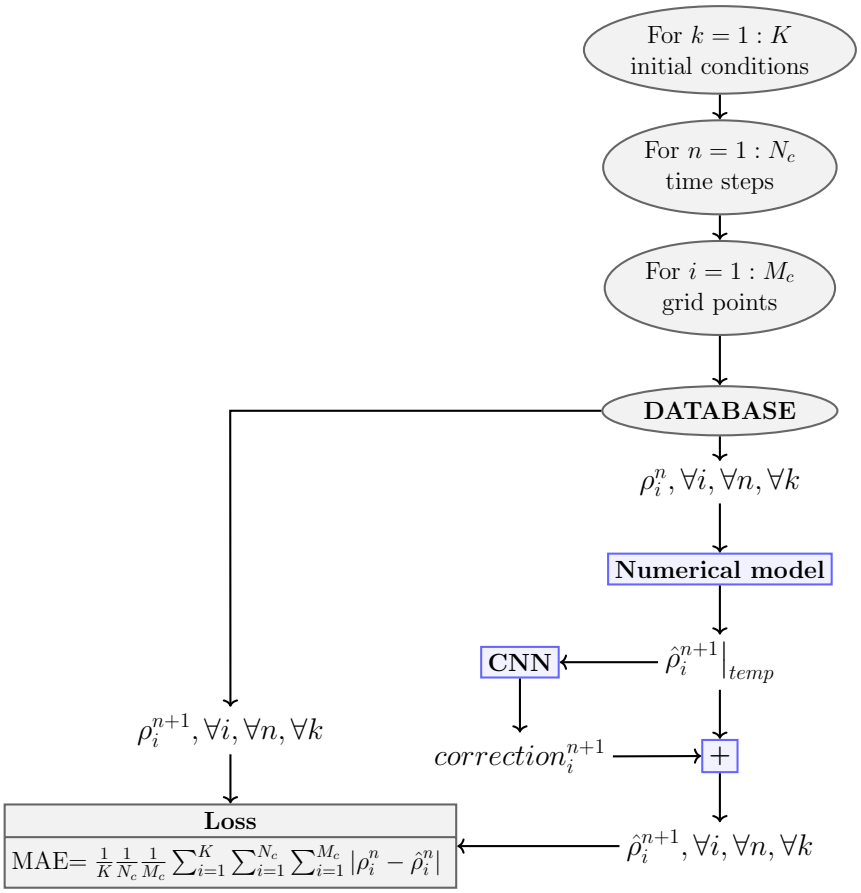


Figure 3.4: *LC method flowchart of the forward pass (see Sec. 2.4) of a single epoch (full-batch for simplicity) of the training phase (Stage 2). The data are high-resolution numerical solutions that have been coarsened. It is very similar to the LI method except for what the CNN is doing. The LC hybrid solver includes the numerical model using standard Taylor-defined constant coefficients, see Fig. 3.1, (FVM second-order scheme with a three-point stencil) and the CNN producing space and time dependent corrections while training.*

3.2.3 Deployment of the hybrid models on new initial conditions

Once the training is finished, each hybrid model is ready to be used. It is wrapped inside a loop which iterates over all desired time steps and thus, a hybrid solver is obtained. This solver, trained on K initial conditions, is used to produce solutions for initial conditions that were not in the training data. This enables the acquisition of high-resolution results (as the CNN was trained on fine-grid results that were coarsened but preserved most of their accuracy) with lower computational costs (as the hybrid model runs on a coarse grid). The hybrid solvers for the LI and LC methods are presented in Figs. 3.5 and 3.6 respectively. The steps involved in each

method are detailed below.

3.2.4 LI method

Step 1: An out-of-sample high-resolution initial condition is coarsened and fed into the hybrid model ($\rho_i^{n=0}, \forall i$).

Step 2: This goes into the -already trained- CNN which produces coefficients for the M_c-points (coarse) grid. This means that in the case of a (S-point stencil), $M_c \times S$ coefficients are produced.

This CNN coefficients-generator has been trained on coarsened field snapshots of close to high fidelity. This means that the produced coefficients construct -in conjuction with the field values of the stencil- such spatial derivatives that the estimated field snapshot at the next time step is going to also be Hi-Fi despite of the solver operating on the coarse grid.

Step 3: The coefficients are fed into the numerical model (FVM second-order scheme with a 3-point stencil and no limiter) which utilized them to approximate spatial derivatives and to eventually compute the field snapshot at the next time step $\hat{\rho}_i^{n+1}, \forall i$. This is the field snapshot estimate of the hybrid model.

Step 4: These predictions $\hat{\rho}_i^{n+1}, \forall i$ are then fed back into the hybrid model, so that $\hat{\rho}_i^{n+2}, \forall i$ is constructed. This is repeated for all the time steps in the user-defined time window.

3.2.5 LC method

Step 1: An out-of-sample high-resolution initial condition is coarsened and fed into the hybrid model ($\rho_i^{n=0}, \forall i$).

Step 2: This is passes into the numerical model which produces a temporary new field snapshot $\hat{\rho}_i^{n+1}|_{temp}, \forall i$.

Step 3: The temporary new field snapshot is fed into the CNN, which produces a corrections field snapshot $CORR_i^{n+1}, \forall i$.

Step 4: The corrections are summed with the temporary new field snapshot to generate the new field snapshot estimate $\hat{\rho}_i^{n+1} = \hat{\rho}_i^{n+1}|_{temp} + CORR_{\forall i}^{n+1}, \forall i$.

Step 5: These predictions $\hat{\rho}_i^{n+1}, \forall i$ are then fed back into the hybrid model, to generate the new field snapshot estimate $\hat{\rho}_i^{n+2}, \forall i$. This is repeated for all the time steps.

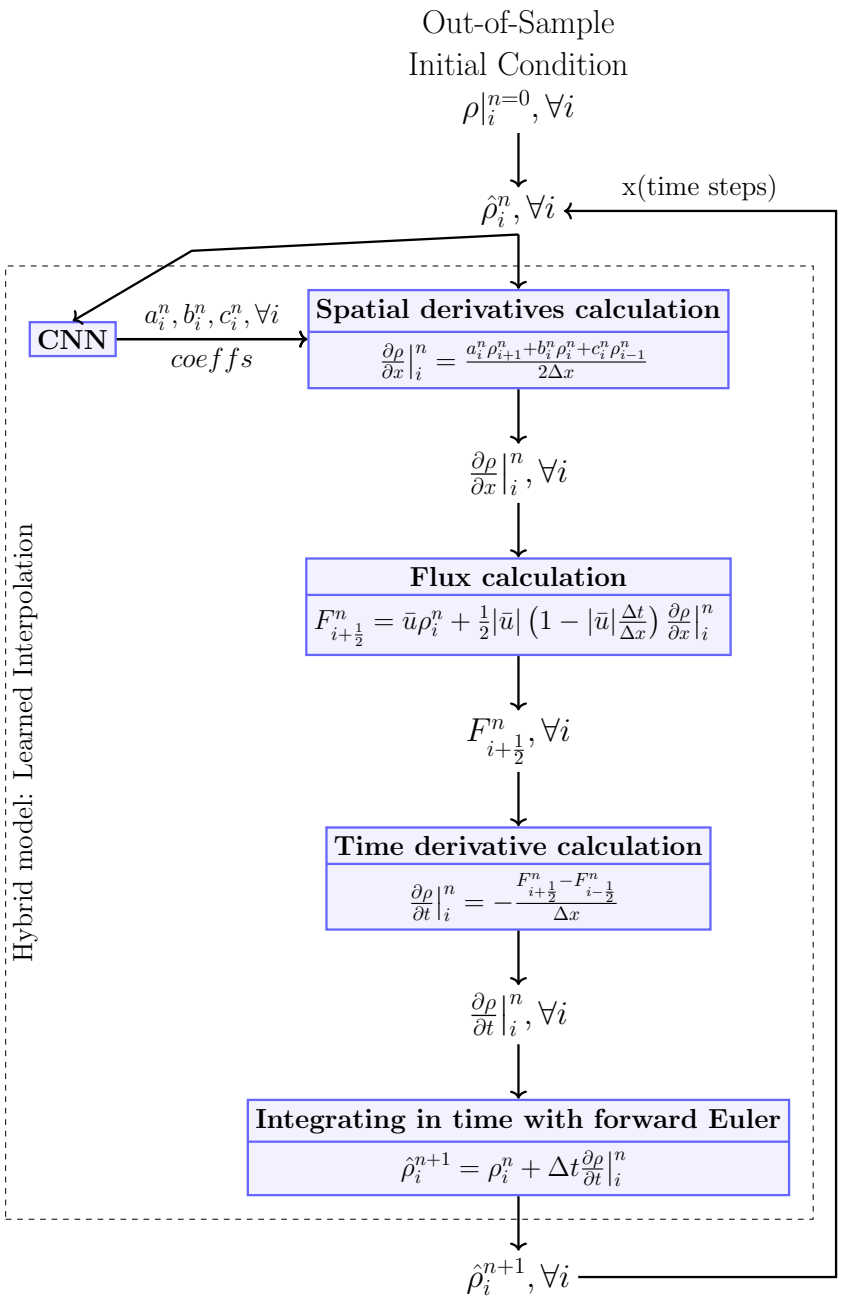


Figure 3.5: *LI hybrid solver deployment. At inference time -after the CNN inside is trained- the solver is fed a Hi-Fi (coarsened) out-of-sample initial condition and produces the solution in a user-defined time window. Each estimated snapshot $\hat{\rho}_i^n, \forall i$ passes through both the CNN that produces space and time dependent coefficients a_i^n, b_i^n, c_i^n, and the numerical model (FVM second-order scheme, no limiter, three-point stencil) that utilizes these varying coefficients. Based on this procedure, the estimated snapshot $\hat{\rho}_i^{n+1}, \forall i$ is predicted. This is looped for all desired time steps.*

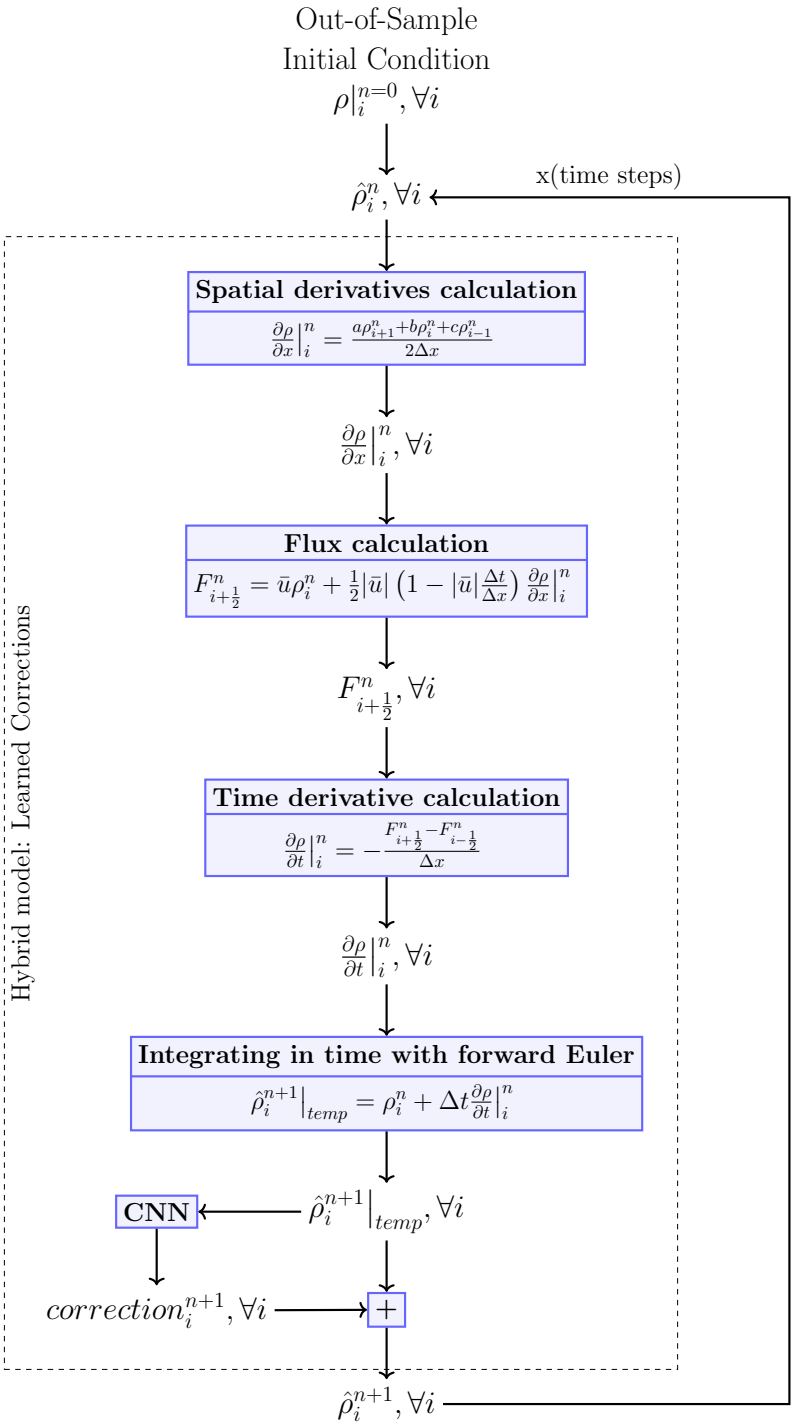


Figure 3.6: *LC hybrid solver deployment. At inference time -after the CNN inside is trained- the solver is fed a Hi-Fi out-of-sample initial condition and produces the solution in a user-defined time window. Each estimated snapshot $\hat{\rho}_i^n, \forall i$ passes through the numerical model (FVM,second-order scheme, no limiter, three-point stencil with Taylor-defined coefficients) and a temporary subsequent snapshot $\hat{\rho}_i^{n+1}|_{temp}, \forall i$ is predicted. Then this temporary snapshot passes through the CNN which predicts space and time dependent corrections $CORR_i^{n+1}, \forall i$ for this field snapshot. Summing the temporary snapshot and the correction, the estimated snapshot $\hat{\rho}_i^{n+1}, \forall i$ is predicted. This is looped for all desired time steps.*

3.3 Additional elements of the hybrid models

The previous sections presented the most fundemental parts of the discussed hybrid methods, but there are additional elements that were proposed in [6, 2, 59, 54] and are paramount for them to perform well.

3.3.1 A multistep Loss function

This technique is applied to both hybrid solvers. Instead of the typical Mean Absolute Error (MAE), the selected Loss function is going to be a multistep MAE. What a multistep Loss function does is that the hybrid model is requested to predict multiple following field snapshots instead of just the next one. In each such multistep prediction, the weights of the embedded NN do not change.

Based on these, the Loss function -for a single snapshot as input- is defined as:

$$\text{MAE} = \frac{1}{Q}\frac{1}{M}\sum_{n=1}^{Q}\sum_{i=1}^{M}|\rho_i^n - \hat{\rho}_i^n|, \tag{3.5}$$

where ρ_i^n is the target snapshot from the Hi-Fi training data and $\hat{\rho}_i^n$ is the snapshot predicted by the hybrid model. The i indexing iterates over spatial points, and n indexing iterates over the set ("quantum") of snapshots that have been stockpiled for Q time-steps.

The use of multiple steps aids the hybrid model recognize that its next prediction is not isolated; it has an impact on future predictions beyond just the next time instance. That way, it produces forecasts that don't blow up in time. The number of snapshots stockpiled is the most important hyperparameter of such models (see Sec.5.5). A quantum of time steps is produced at each model run, so this hyperparameter is refered to as "quantum" or "Q time-steps". A high-level view of the procedure can be seen in Fig.3.7.

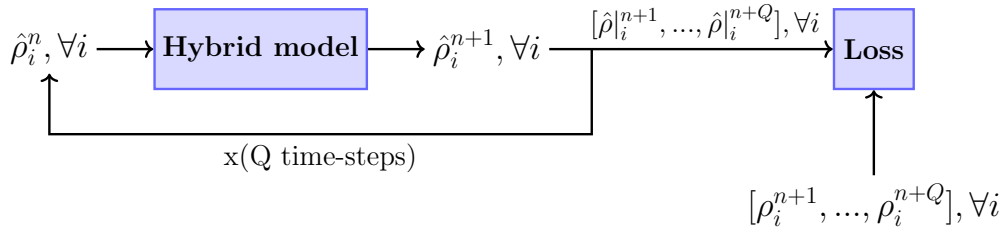


Figure 3.7: *The stockpiled snapshots are fed into the Loss function and compared with the training dataset. Depicted for a single snapshot as input.*

For example, in the case of 1D advection, where the "Q" hyperparameter was set to four, the Loss function can be computed as so:

$$\text{MAE} = \frac{1}{4}\frac{1}{M}\sum_{n=1}^{Q=4}\sum_{i=1}^{M}|\rho_i^n - \hat{\rho}_i^n|, \tag{3.6}$$

This means that the hybrid solvers use an initial snapshot of the coarsened field as input and generate not one but four subsequent snapshots of the field. The hybrid models then utilize the last predicted snapshot out of these four to generate the next four ones and this is repeated for the totality of time steps the user has predefined.

3.3.2 Enforcing a regularizing constraint

This part is relevant only for the LI method. At first, it may seem logical to design the CNN inside the model to generate coefficients c that are arbitrary. However, it could be helpful if these coefficients possessed certain desirable properties. One such property is that they could exhibit a certain level of formal polynomial accuracy [2]. This would offer a valuable guarantee that at least for simple functions the derivative (spatial derivative in the LI method) approximation would be exact. If this objective can be accomplished, it would also serve as an effective, built-in regularizer for the model. In the sense that, loosely speaking, it would constrain the solutions to originate from a specific meaningfull set of coefficients, narrowing down the solution space to a submanifold within the original space.

Accomplishing formal accuracy order coefficients requires that they are are solutions to the following linear system $Ac = b$ of equations, derived using the Taylor expansion. For a given arbitrary stencil s of length N with the order of derivatives $d < N$:

$$\begin{bmatrix} s_1^0 & s_2^0 & \dots & s_N^0 \\ s_1^1 & s_2^1 & \dots & s_N^1 \\ \vdots & \vdots & \vdots & \vdots \\ s_1^{N-1} & s_2^{N-1} & \dots & s_N^{N-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = d! \begin{bmatrix} \delta_{0,d} \\ \delta_{1,d} \\ \vdots \\ \delta_{i,d} \\ \vdots \\ \delta_{N-1,d} \end{bmatrix}$$

If the requested order of accuracy is lower than the maximum that can be achieved for the specific derivative order with the specific stencil, it is enough for coefficients to satisfy a reduced-rows version of this linear system. Namely, for securing m^{th} accuracy order for a d^{th} derivative with a N-sized arbitrary stencil, $N - m - d$ rows must be retained. This would then constitute an underdetermined system.

If someone aims to ensure that the vector c satisfies an underdetermined linear system (which can be interpreted as a set of linear constraints), then this equation for the residual R must hold:

$$R(c) = Ac - b = 0 \tag{3.7}$$

However, if the CNN generates arbitrary numbers as coefficients and is trained as such, it is evident that this constraint would not be met by default.

$$R(c_{\text{CNN}}) \neq 0 \tag{3.8}$$

So, how can the CNN be forced to inherently produce c in a way that naturally satisfies this constraint? There is actually a way around this. It is a Linear Algebra fact that a solution c_{cnstr} which satisfies this underdetermined linear system can be represented as follows:

$$c_{cnstr} = c_{bias} + (\text{arbitrary-weights}) A_{null-basis} \tag{3.9}$$

where, c_{bias} is any solution -of the infinitely many- of the underdetermined system such that:

$$R(c_{bias}) = 0 \tag{3.10}$$

and

$$A_{null-basis} = [v_1 \; v_2 \; ... \; v_M] \tag{3.11}$$

where $[v_1 \; v_2 \; ... \; v_M]$ are the basis vectors of the nullspace.

So based on Eq.3.9, if someone intends to generate constrained coefficients, they should make the CNN produce not the coefficients themselves, but rather these "(arbitrary-weights)" which are essentially some weights to the nullspace basis. By doing this, regardless of what the CNN may initially produce, after passing it through Eq.3.9, the result is coefficients that adhere to the constraint by default. The nullspace basis of the underdetermined system can be obtained through a Singular Value Decomposition (SVD).The right eigenvectors of the SVD of matrix A constitute a basis for the nullspace of A.

In the trial-and-error procedure of the cases presented in the next chapters, it was verified (see also [2, 59]) that when approximating spatial derivatives, enforcing first-order accuracy is the choice that gives both a well converged behavior during training and a stable and generalizable model during inference. For instance, in the cases of this thesis where a first-order spatial derivative is approximated, first-order accuracy just amounts to enforcing that the nodal coefficients produced by the CNN

for each node, sum up to zero. Namely, if $\frac{\partial \rho}{\partial x}|_i^n = a_i^n \rho_{i-1}^n + b_i^n \rho_i^n + c_i^n \rho_{i+1}^n$, then the constraint $a_i^n + b_i^n + c_i^n = 0$ should be enforced.

3.4 Early stopping strategy

One of the inherent challenges in hybrid solvers that integrate NNs into numerical solvers is to attain a stable solver after training. This challenge arises from the unique nature of a solver's operation for unsteady problems during inference. More specifically, at inference time, the hybrid solver does not encounter input that is guaranteed to fall within the training data distribution. The solver, upon receiving an initial condition, iteratively progresses to generate subsequent snapshots of the fluid dynamics. Each snapshot beyond the first is a result of the solver's own predictions, rather than being a direct instance from the training dataset. This iterative process introduces a compounding effect on prediction errors. For instance, the solver's output at the second step (snapshot) includes the errors from the first prediction. When this output is used as the input for the next step, the prediction is based on slightly erroneous data, which were not exactly represented in the training set. This means that the hybrid solver is not guaranteed to produce the third prediction with the Loss value that has been observed in training even for initial conditions that are included in the training dataset. As the solver continues to iterate, these small deviations accumulate, potentially leading to a significant divergence from what the NN has seen during training. This, in turn, may lead to the NNs producing irrational coefficients or corrections (LI and LC methods respectively) which can introduce instability to the hybrid solver.

This could be avoided if the trained CNN does not overfit. This is typically ensured through early stopping using validation data. However it was observed that traditional validation sets do not exhibit the typical behavior of independent validation data in these hybrid solvers. Instead, their Loss curves closely followed the trend of the training Loss curve. This lack of divergence between training and validation Loss curves meant that they fail to provide the necessary feedback to determine any point for early stopping. This should perhaps be expected as the generalization capabilities of hybrid solvers are mostly due to the preserved parts of the numerical solver and not due to the NN.

To effectively implement early stopping and prevent overfitting, thereby ensuring the stability of hybrid solvers, a simple way is to integrate the solver operation into the training phase. This approach involves running the hybrid solver with the current NN weights at periodic intervals during training. However, due to the computational costs associated with this process, a sensible strategy is required.

In this thesis, the hybrid solver is activated during training every some fixed number of epochs, a frequency determined by an additional hyperparameter. While a constant step is utilized here for simplicity, more complex scenarios might require a

statistically informed approach for determining this interval.

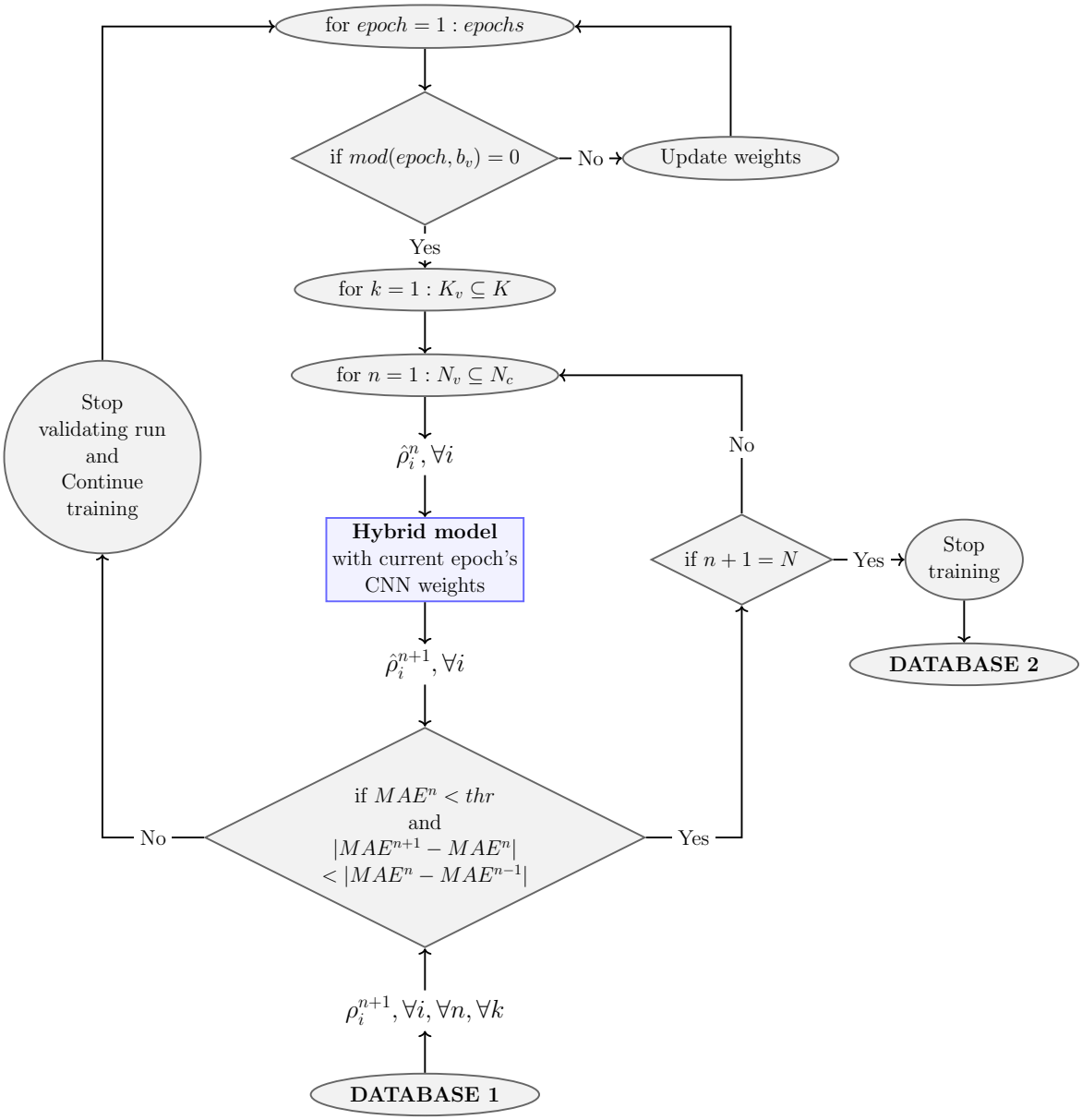


Figure 3.8: *Early stopping strategy for unsteady CFD-DL solvers. In this figure, k is the iterator over the subset (of all initial conditions in the training data K) of validating initial conditions K_v, n is the iterator over the subset (of total time steps in the training data N_c) of validating time steps and i is the index for grid points. Also, b_v and thr define the frequency and thresold hyperparameters of the strategy respectively. When the training stops, the weights are saved and stored.*

For each evaluation, a set of initial conditions from a validation set (or even from the training set) is randomly selected and used for running the solver. For example,

in the linear acoustics case, 5 out of 10 square wave initial conditions of the training set were used. This is a big percentage of the training data but, in the specific case, this is necessary, as using a smaller absolute number for the sample size would significantly compromise the statistical validity of the procedure. For larger datasets, a sample size of approximately 5%-15% of the original training set is suggested, aligning with traditional practices for validation set sizes in other domains.

The solver runs using this set of initial conditions, for all time steps in a predefined temporal domain, until the MAE at the current time step MAE^n reaches a predefined threshold (hyperparameter set by the user) or if the difference in MAEs of subsequent steps keeps growing for some number of time steps (another hyperparameter), which means the solver becomes unstable.

$$\begin{aligned} &|MAE^n| > \text{threshold}, \quad \text{for any } n \\ &\text{or} \\ &|MAE^{n+1} - MAE^n| > |MAE^n - MAE^{n-1}|, \quad \text{for multiple } n \end{aligned} \tag{3.12}$$

These MAEs represent the error between the solver's prediction and the corresponding coarsened Hi-Fi dataset. If the MAE exceeds the threshold or keeps growing before completing all time steps (see Eq.3.12), the solver run is terminated early, and NN training continues. This approach helps reduce unnecessary solver runs. Conversely, if the above MAE criterions are not met for the entire duration of the solver run (e.g. for all time steps), the training stops (early stopping).

This way of checking for instability is possible also due to the similar nature of error progression, in both purely numerical and hybrid solvers (end-to-end NNs are not guaranteed to exhibit this behavior): for each run the error initially exhibits a gradual increase before reaching a point of rapid escalation or explosion.

3.5 Advantages of coefficient prediction in hybrid CFD-DL solvers

Incorporating NNs into existing numerical solvers for CFD presents several advantages when the NNs are tasked with predicting coefficients rather than directly producing spatial or temporal derivatives. First of all, this approach represents a minimal alteration to the numerical solver. This means that the bulk of the physics captured by the solver is retained ensuring the generalizational capabilities of the hybrid solver. Predicting coefficients may also present a more straightforward learning task for the NN as they typically exhibit smaller variance than spatial or temporal derivatives to accurately predict dynamics. This is evidenced by traditional numerical schemes where a limited number of constant or simply varying coefficients

can yield accurate results when spatial and temporal derivatives vary greatly across different points in space and time. This reduction in the learning objective's variance/complexity can lead to more efficient training and improved model stability. Finally, by focusing on coefficient prediction which are combined with local field values, the model intrinsically enforces a sense of locality in the hybrid solver. In CFD, a point in the flow field is predominantly influenced by its immediate surroundings and predicting coefficients that interact with local field values adheres to this principle.

3.6 Advantages of corrections prediction in hybrid CFD-DL solvers

Prediction-Correction schemes are a well established notion in many engineering fields including Numerical Analysis (predictor-corrector schemes), Control Theory (state feedback control) and the Deep Learning field (residual connections). The LC method discussed in this thesis implements this very idea, and there are specific advantages to doing so. First of all, errors have been verified to follow concrete patterns which constitutes them a valid learning target [54]. It is also true that this prediction-correction summation inherently acts as a type of residual connection leading to the same beneficial properties as those of the well-established Resnet type architectures. These include a reduction of the vanishing and exploding gradients problem and the observation that if residual connections are utilized in a NN, training error certainly goes down when the trainable parameters are increased. This is important as, deeper networks (with more parameters) without residual connections can have higher training error (and hence test error) than their shallower counterparts [17], which would mean that one cannot scale the complexity of the NNs to match, for example, the increased complexity of the data. Additionally, in the LC method, the CNN only needs to predict a small correction since much of the dynamics have been captured by the numerical parts of the solver. Because there is a correction at each time step, the error does not get a chance to grow and the necessary correction always remains reasonable and easy to predict. This is the same argument made before for the LI method on minimally modifying numerical solvers. However here, these error corrections might be easier to learn for another reason: in many simulations the scales that the error spans are fewer than the the scales of the full dynamics plus the error scales combined. For example, in the 1D advection case there are roughly two scales in the simulation results. One due to the dynamics of the equation and the other one due to the error. If, given a field snapshot, the CNN had to directly predict the next snapshot (or any other quantity of the field), it would have to handle both propagating the wave in space and also dampen the error in the new place of the wave. This means it would have to deal with two scales when, in the LC method, it only has to deal with one.

Chapter 4

Case 1: 1D Advection Equation

4.1 Introduction

The first case examined aligns with one of the problems presented in [59], which is also based on one of the fundemental methods that were studied in this thesis. This is very important, as it establishes an essential point of reference for the results that follow. This is the standard 1D advection equation:

$$\frac{\partial \rho}{\partial t} + \bar{u}\frac{\partial \rho}{\partial x} = 0, \tag{4.1}$$

where the convecting velocity $\bar{u}$ is considered positive, known and constant.

The spatial domain is defined as $x \in [0, L]$ and the time domain as $t \in [0, t_{final}]$, with periodic boundary conditions (BC) in space:

$$\rho(x = 0, t) = \rho(x = L, t) \tag{4.2}$$

and a typical initial condition of a square wave of some height and width:

$$\rho(x, t = 0) = \begin{cases} height, & \text{if } x_l < x < x_r \\ 0, & \text{elsewhere} \end{cases} \tag{4.3}$$

The analytical solution is a square wave travelling through space unaltered, with

a velocity $\bar{u}$, until it reaches the right end of the spatial domain (as $\bar{u} > 0$). Any portion exiting re-enters from the left end of the domain, and so on and so forth.

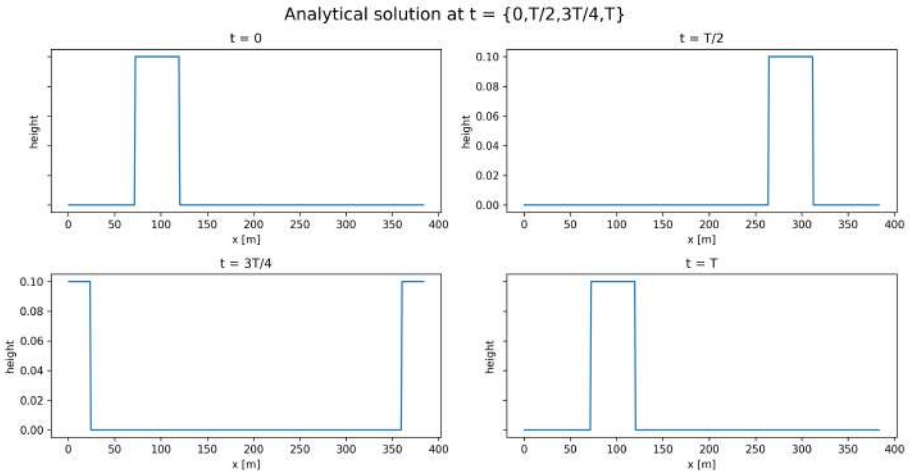


Figure 4.1: *Exact square wave traveling unaltered, at* $t = \{0, T/2, 3T/4, T\}$.

Next, this equation needs to be discretized to be used as the numerical model both for generating high-resolution training data and for parts of it to be integrated inside the hybrid solvers.

4.2 Equation discretization

4.2.1 General formulations

The discretization of Eq.4.1 is based on the FVM due to its advantage of being able to conserve quantities throughout space [29] if properly handled. Firstly, a general formulation of the method of lines, with the spatial derivatives discretized in a FV manner is given:

$$\left.\frac{\partial \rho}{\partial t}\right|_i = \frac{1}{\Delta x}(F_{i+1/2} - F_{i-1/2}) \tag{4.4}$$

For time integration, it is common to use a forward Euler scheme. This leads to:

$$\rho_i^{n+1} = \rho_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^n - F_{i-1/2}^n) \tag{4.5}$$

Choosing the formula of the numerical flux function F defines different FV schemes. Note that, in the following formulas, $\bar{u}^- = \min(0, \bar{u})$ and $\bar{u}^+ = \max(0, \bar{u})$.

- A Typical First-Order Accuracy Scheme (Upwind Method):

$$F^n_{i-1/2} = \bar{u}^- \rho^n_i + \bar{u}^+ \rho^n_{i-1}, \tag{4.6}$$

- Second-Order Accuracy Schemes:

$$F^n_{i-1/2} = \bar{u}^- \rho^n_i + \bar{u}^+ \rho^n_{i-1} + \frac{1}{2}|\bar{u}| \left(1 - |\bar{u}|\frac{\Delta t}{\Delta x}\right) \delta^n_{i-1/2}, \tag{4.7}$$

  where,

$$\delta^n_{i-1/2} = \phi(\theta^n_{i-1/2})\Delta\rho^n_{i-1/2} \tag{4.8}$$

  and,

$$\theta^n_{i-1/2} = \frac{\bar{u}^- \Delta\rho^n_{i-1/2} + \bar{u}^+ \Delta\rho^n_{i+3/2}}{(\bar{u}^- + \bar{u}^+)\Delta\rho^n_{i-1/2}} \tag{4.9}$$

  In the formula of $F^n_{i-1/2}$ in Eq.4.7, the first term enforces the upwind method (as in Eq.4.6), while the second term introduces a correction that exhibits anti-diffusive behavior when the CFL condition is satisfied. The term $\delta^n_{i-1/2}$, given by Eq.4.8, represents a regulator of slope. The function $\phi(\theta)$ represents the imposition of a limiter designed to control the strength of this anti-diffusive term. Further details can be found in [29].

Many common schemes, of either first or second order, with or -degenerately- without limiting, can be implemented using a limiter function. Namely:

- First-order of accuracy:
  - upwind: $\phi(\theta) = 0$
- Second-order of accuracy without limiters:
  - Lax-Wendroff: $\phi(\theta) = 1$
  - Beam-Warming: $\phi(\theta) = \theta$
- Second-order of accuracy with limiters:
  - superbee: $\phi(\theta) = \max(0, \min(1, 2\theta), \min(2, \theta))$
  - monotonized central difference (MC): $\phi(\theta) = \max\left(0, \frac{1+\theta}{2}, 2, 2\theta\right)$
  - Van Leer: $\phi(\theta) = \frac{\theta+|\theta|}{1+\theta}$

The behavior of ϕ with respect to θ is presented in Fig.4.2. The ratio θ which represents the ratio of successive gradients, as defined in Eq.4.9, can be thought of as a measure of the smothness of the quantity that is to be approximated near $x_{i-1/2}$.

If the data is smooth, it is expected that $\theta \approx 1$, whereas near a discontinuity it is expected that θ diverges from 1. It is obvious that one would like that $\phi(\theta \approx 1) = 1$ so that when the quantity is smooth, a pure second-order scheme is achieved. Each limiter then differs in how it handles non-smooth regions, where pure second-order methods fail (see Fig.4.3) respecting the TVD condition. More details on these can also be found in [29].

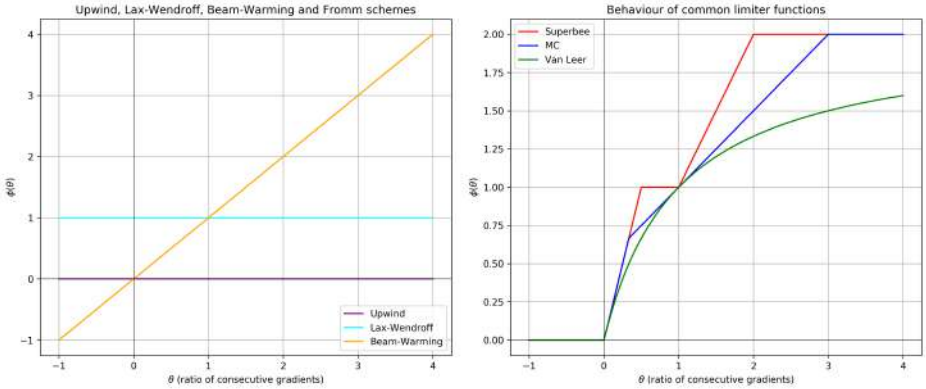


Figure 4.2: *Behavior of ϕ with respect to θ, which concisely describes how schemes handle reconstruction. Upwind is a first-order reconstruction scheme, L-W, B-W schemes employ non-limited second-order reconstruction and Superbee, MC, Van Leer are limited second-order schemes.*

4.2.2 The effect of the chosen scheme on the solution

Before finalizing the discretization, it is important to conduct tests to identify the most suitable scheme and grid resolution for this specific problem. As shown in Fig.4.3 and Fig.4.4, the choice of numerical scheme significantly impacts result quality. In the following snapshots of the numerical solution, the square wave is becoming smeared out as it propagates through space. This diffusion is an inherent artifact of the approximate numerical schemes. This effect can be problematic because it fundamentally alters the underlying physics: sharp discontinuities in the solution are transformed into smooth pulses. For instance, in CFD, such discontinuities may represent shock waves. Their transformation into smooth pulses could then lead to an inaccurate evaluation of the safety of an aerodynamic design. As previously discussed, second-order schemes include an anti-diffusive term that sharpens the solution. Therefore, they are expected to exhibit less smearing effect. However, to avoid a byproduct of second-order schemes (unnatural oscillations), limiters are used. These reduce the intensity of the anti-diffusion term in specific points in the solution to smooth out these oscillations.

In Figs. 4.3 and 4.4 it is obvious that acceptable results are given only by limited second-order accurate methods. The best ones are those with the superbee and the MC limiters.

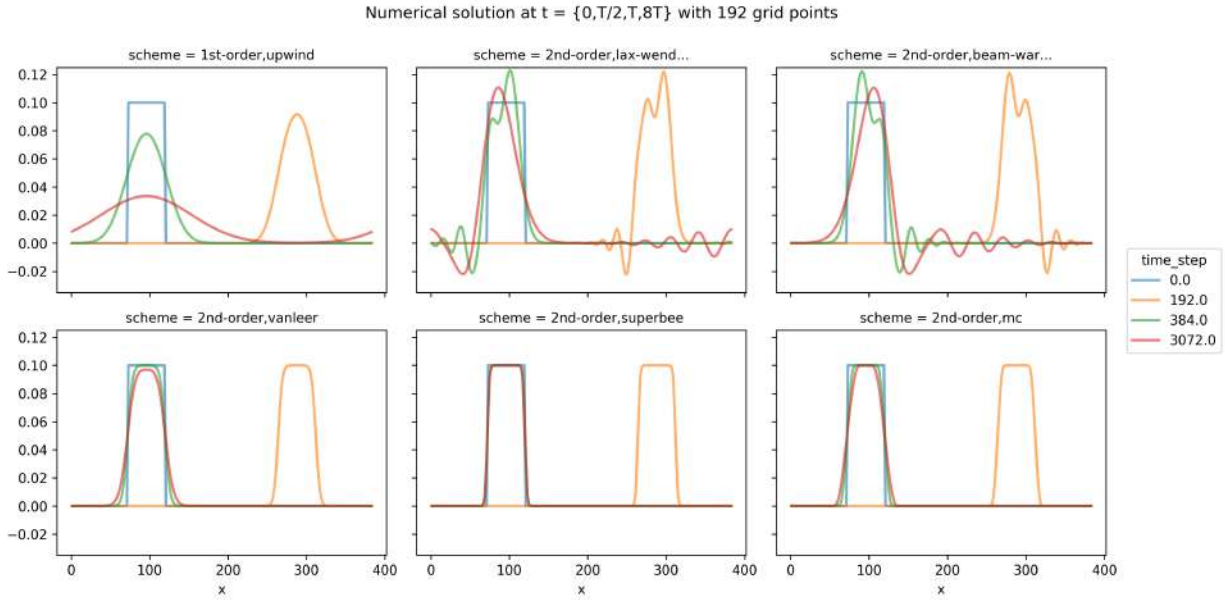


Figure 4.3: *Traveling square wave in 192 grid points resolution, at* $t = \{0, T, 16T, 32T, 64T\}$*. The first-order scheme smears out the equation too much, the non-limited second-order schemes exhibit dispersion in the form of oscillations. The limited second-order schemes give sufficient results. The spatial resolution of 192-points is inadequate in the first row schemes (deformation of the shock's geometry) and borderline adequate at the second row schemes.*

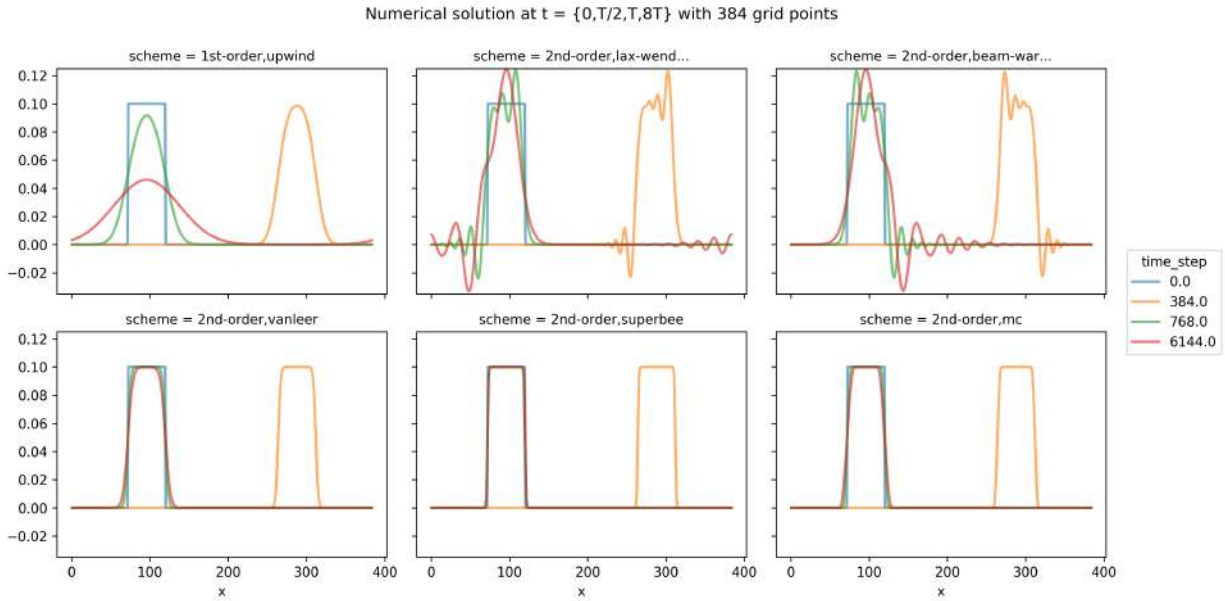


Figure 4.4: *Traveling square wave in 384 grid points resolution, at* $t = \{0, T, 16T, 32T, 64T\}$*. Same comments with Fig.4.3 apply. The only difference is that the spatial resolution of 384-points is deemed adequate in the second-order limited schemes.*

4.2.3 The effect of spatial resolution on the solution

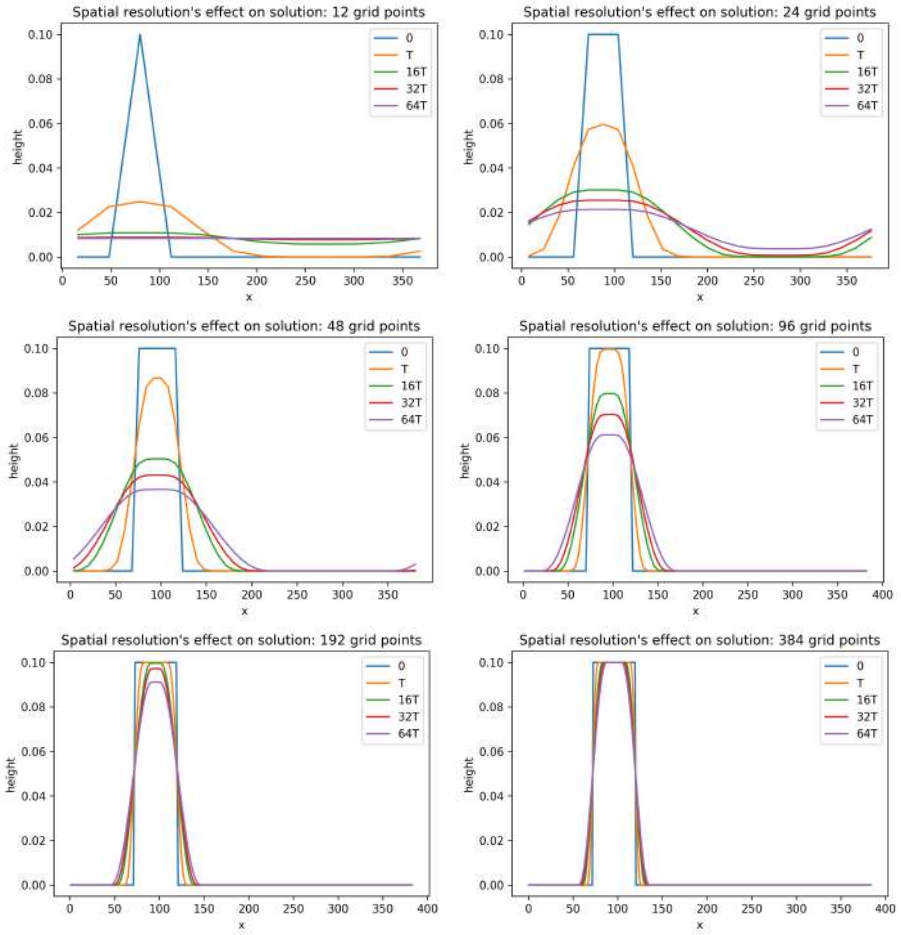


Figure 4.5: *Traveling square wave in {12,24,48,96,192,384} grid points resolutions, at $t = \{0, T, 16T, 32T, 64T\}$. The wave becomes progressively more smeared out with each passing period. At 384 grid points, even after 64 periods have elapsed, the smearing effect is still considered acceptable.*

Firstly, it is important to consider if the spatial discretization is rich enough to capture the continuous phenomena and the full spectrum of scales of the solution. In the current case, the geometry remains constant but shifts in position, so one can tell if the number of points is sufficient by how well the square wave geometry is represented at zero time. In Fig.4.5 it is clear that the simple geometry of a square wave can be captured for more than 24 grid points.

However, a significant factor to consider is that the discretization error in numerical schemes depends on the size of the grid step. In practical terms, this implies that for a fixed spatial domain, as the number of points becomes higher, the grid step becomes smaller, leading to a reduction in error. Essentially, to achieve more accu-

rate solutions, it is necessary to use finer grids. Reducing the discretization error is the primary purpose of the hybrid solvers presented in this thesis. In CFD, a widely used technique for ensuring the quality of a grid is to enforce grid independence. This is typically achieved by placing an upper limit to the error between solutions obtained from successively finer grids. In Fig.4.5, the solution with 384 grid points is regarded as the grid-independent solution. It is clear that refining the grid brings the achieved result closer to the exact solution and diffusion of the wave takes much longer.

4.3 Training data and coarsening

Firstly, regarding the discretization of the spatial and temporal domains, a uniform grid with step size of $\Delta x = 1$ is chosen and the time step is determined using the CFL condition: $c = \bar{u}\frac{\Delta t}{\Delta x}$. By setting the courant number to $c = 0.5$, and the velocity to $\bar{u} = 1m/s$, the time step becomes $\Delta t = 0.5s$.

When generating training data for the model, the discretized equation is solved with a FVM second-order limited (superbee) scheme, on a 384-points grid, using 30 initial conditions of square waves with varying heights and widths. In case 1,

$$height \in [0.1, 1] \text{ with step } 0.1$$

$$width \in [48, 144] \text{ with step 48 grid points (on the fine grid)}$$

Note also that all square waves start from the 96^{th} node of the 384-node grid. The time integration spans two periods: $t_{final} = 2T = 768s$. That means that with a time step of $\Delta t = 0.5s$, the total time steps needed are 1536.

Both methods described in the current thesis require coarsening to harness the benefits of the hybrid models. Actually, the choice of length (and consequently, t_{final}), was made in such a way so as to achieve a grid size with a number of points that follows the pattern 3×2^n (so that progressive coarsening can be executed), and with a grid step size of one ($\Delta x = 1$). Practically, concerning coarsening, while the high-resolution numerical solver is running, results undergo coarsening in time (downsampling) and space (averaging) with a $8\times$ coarsening ratio, which is the maximum allowable one. Averaging coarsening entails the computation of the mean of every eight density values, while the grid must also be adjusted accordingly. It is preferable to other coarsening techniques (e.g. downsampling) for space because it does not break the conservation property of the FVM. The maximum possible coarsening depends on how the initial conditions were set in the grid, and on the scales involved. In the scenario examined here, the initial waves have a minimum width of 48 grid points, which is equivalent to 1/8th of the total grid points. To maintain the geometry of these waves, the maximum coarsening factor that can be applied is $8\times$. A demonstration is given in Fig.4.6:

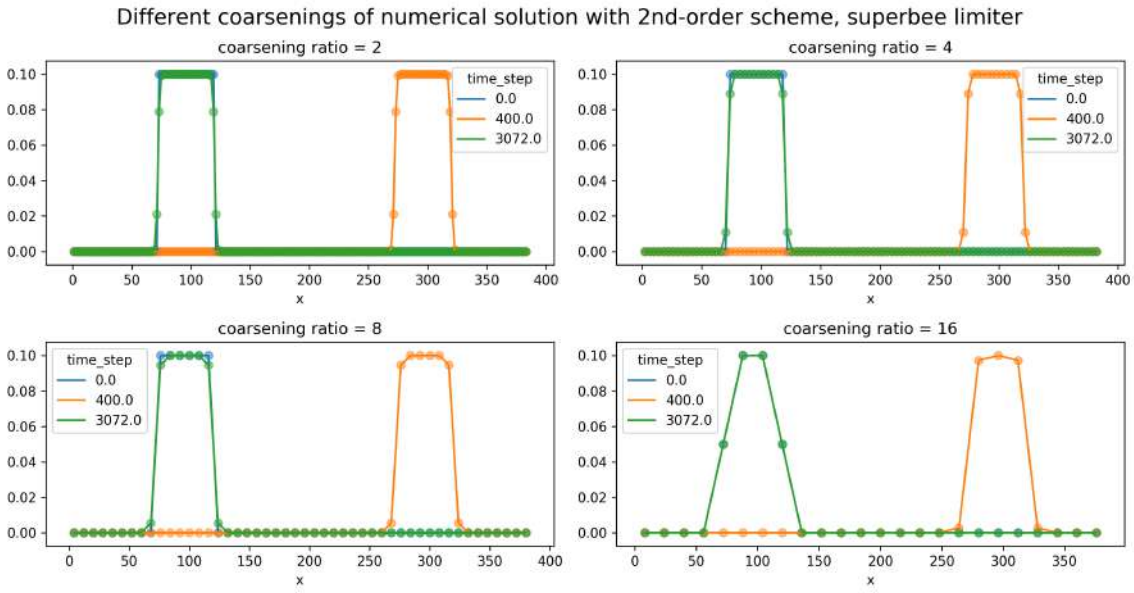


Figure 4.6: *Different coarsening ratios applied to the solution of the discretized equation. Coarsening ratio 16×, creates unnatural geometry at all time steps because of the initial width of the square wave.*

Since the 8× coarsened solution is the largest compression performing well, this is the one to be used. Which means that the 384-points grid is mapped to a 48-points grid and the 1536 time steps are mapped to 192 time steps. Reducing the number of time steps is necessary so that the inherent time step of the hybrid model matches with the time step of the provided data. It is important to note that this represents the most challenging scenario for the hybrid model to perform well because it is based on the coarsest grid. However, it also presents the greatest potential for reducing computational costs.

4.4 Results of the hybrid models

The main parameters' values, as defined in Tab.3.1, and analyzed for case 1 in the previous section, are summarized in Tab.4.1. Additionally, the hyperparameters for each hybrid method are presented in Tab.5.2.

In the following demonstrations, it is shown that the results that have been produced with a runtime cost of a numerical simulation on a 48-points grid plus the CNN inference cost, attain an accuracy close to what would be generated by a 384-points grid one. Some of the capabilities and shortcomings of the model are presented in the following tests.

Firstly, the models' performance is evaluated after being fed the initial conditions of the training data but scaled by a factor of 0.65 in height, changed width and

Parameter	Value
K	30 initial conditions
λ_1	3
λ_2	7
M_f	384 grid points
M_c	48 grid points
N_f	1536 time steps
N_c	192 time steps
S	3-point
CR	8

Table 4.1: *Table with the parameter values for both methods.*

Hyperparameter	LI method	LC method
number of layers	4	4
number of filters	32	32
kernel size	3×1	3×1
batch size	64	64
learning rate	3e-3, 3e-4	3e-3, 3e-4
epochs	102	156
Q time-steps	4	4
optimizer	Adam	
weight initialization	Xavier	
activation functions	ReLU	

Table 4.2: *Chosen Hyperparameters for the hybrid models. All layers have the same number of filters and same activation function. Hyperparameter Q defines how many subsequent snapshots the solver produces per run (see sec.3.3).*

translated by 6 grid points. This tests the behavior of the models to initial conditions of sizes and initial positioning that it has not seen in the training data. It is asked to predict over the time period it has encountered in its training data, as well as for future times. This is then compared to the performance of a pure numerical solution using 48 grid points. The traditional solution employs the same solver that generated the training data. In Figs. 4.7 and4.8, the mean MAE errors between an end-to-end numerical simulation in the fine grid of 384-points and the hybrid models' simulations are showcased. This is done for all snapshots of all time steps up to 32 periods (when the training has taken place for only 2 periods) in time. Note that all fine grid simulations, are projected to the coarse grid (via coarsening) to enable comparison with the coarse ones. The hybrid models' performance is superior to the one of the numerical solution on the 48-points grid for initial conditions involving square waves with heights falling inside the trained range ($height \in [0.1, 1]$ with step 0.1). Spatial shifts of the initial condition do not pose any problems, which probably means that the CNN has manifested translation invariance [28].

It is important to emphasize that, in the specific problem which is periodic, the temporal extrapolation serves as a means to assess the stability of the hybrid models, ensuring that minor adjustments do not lead to solution divergence.

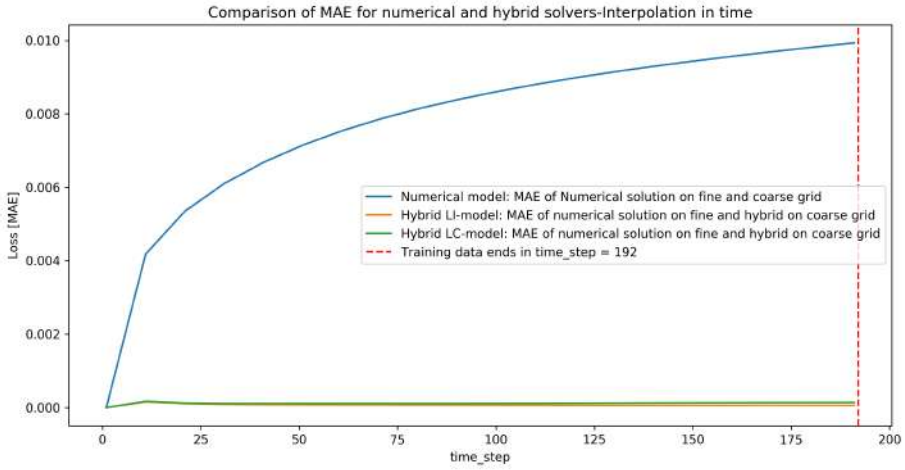


Figure 4.7: *Comparison of end-to-end numerical solver and hybrid models' performances in a MAE sense. MAE is a very important metric in Computational Engineering because outliers -that can complety alter the nature of a phenomenon in physics- must be taken into account when evaluating a simulation. LI is marginally better than LC in the time these models have seen during training.*

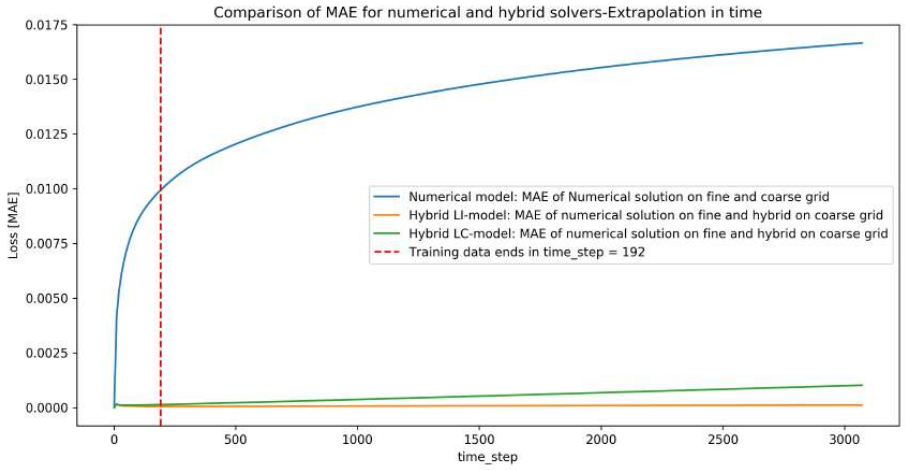


Figure 4.8: *Comparison of end-to-end numerical solver and hybrid model performances in a MAE sense. The hybrid solvers extrapolate 16x as many periods as the ones they have seen in the training data. LI method seems to work a bit better for this case when extrapolating in time.*

To make the quality of the results concrete, the wave propagation resulting from an out-of-sample initial condition (a square wave of $height = 0.39$) is plotted in Figs .4.9 and 4.10 for the LI model and the LC model respectively.

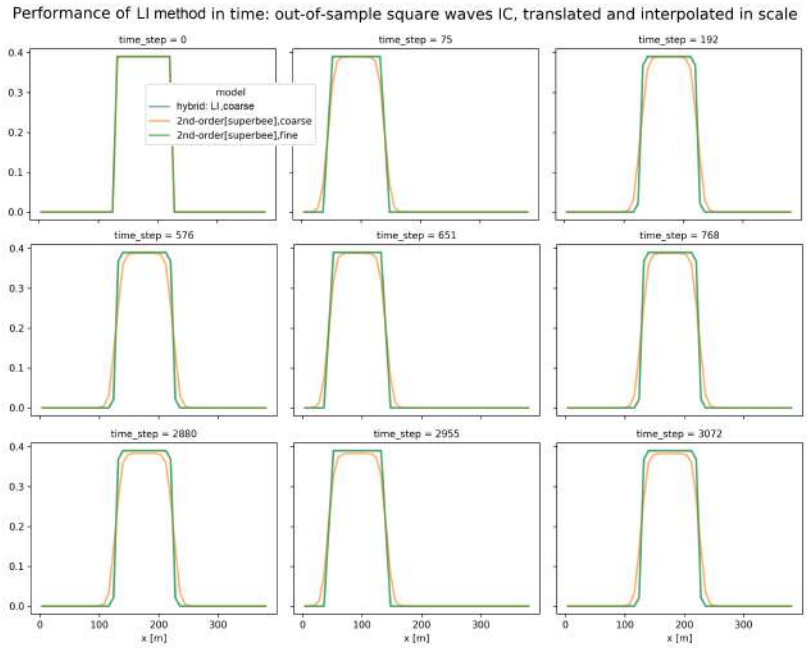


Figure 4.9: *Comparison of a second-order FVM scheme with superbee limiter to the LI hybrid model on the coarse grid of 48-points. The baseline is the numerical solution on a 384-points grid, coarsened to 48 points. The solver is asked to integrate in time for* $16\times$ *the time domain that it has seen in the training data (32 versus 2 periods).*

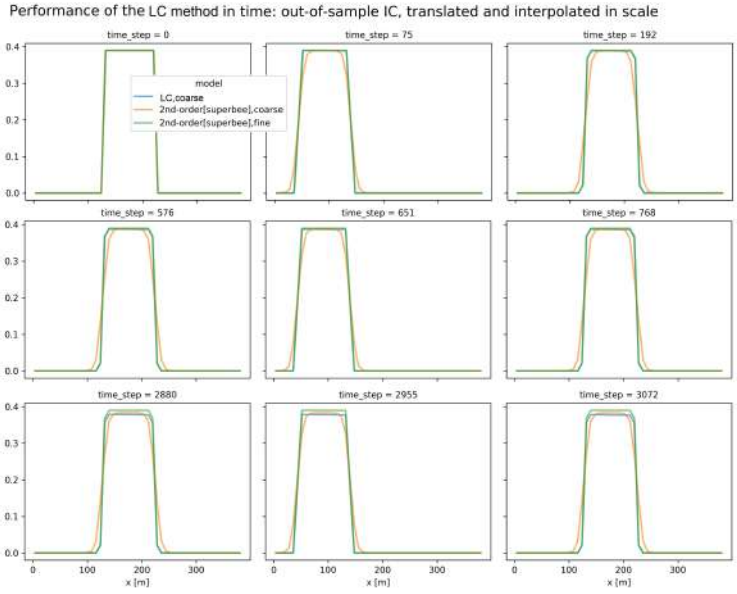


Figure 4.10: *Comparison of second-order FVM scheme with superbee limiter to the LC hybrid model on the coarse grid of 48-points. The baseline is the numerical solution on a 384-points grid, coarsened to 48 points. The solver is asked to integrate in time for* $16\times$ *the time domain that it has seen in the training data (32 versus 2 periods).*

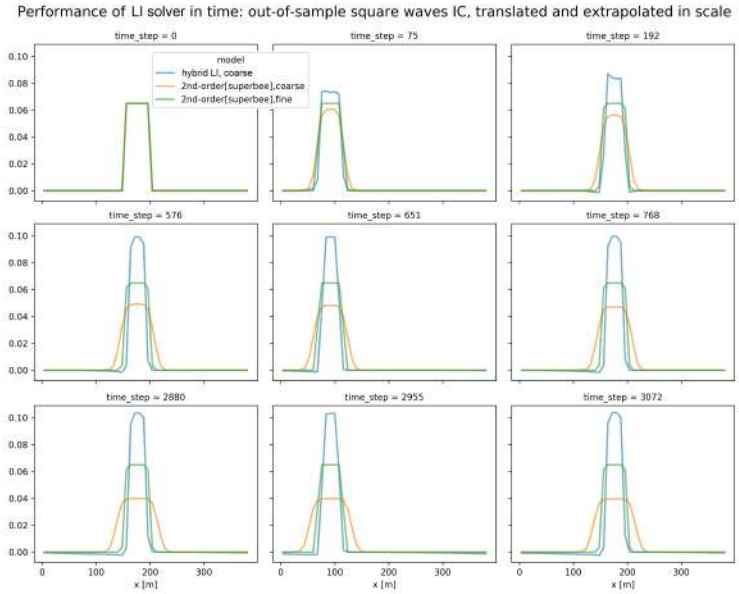


Figure 4.11: *Performance of the LI hybrid solver when given out of sample square waves with smaller size than the ones that it has encountered during training. Having learned an anti-diffusive behavior, it tends to take it to an extreme in this situation.*

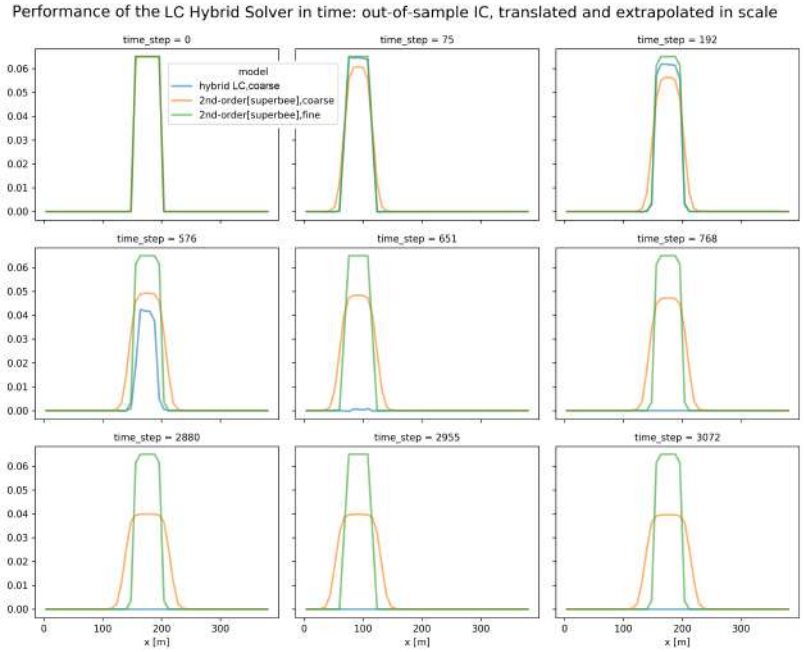


Figure 4.12: *Performance of the LC hybrid solver when given out of sample square waves with smaller size than the ones that it has encountered during training. The behavior it has learnt is mostly diffusive. It takes diffusion to an extreme and completely smooths out the wave to zero height.*

However, both models exhibit poor performance in integrating in time when the out-of-sample initial conditions, are scaled below the smallest sized square waves the hybrid models have been trained on (e.g. $height = 0.065$). Interestingly the LI model seems to have acquired a mostly anti-diffusive behavior and tends to exhibit an excessive version of it in this scenario. This can be seen in Fig.4.11. Whereas in the LC model, in Fig.4.12, which seems to have learnt a mostly diffusive behavior (probably to correct oscillations of the second-order not limited scheme) gradually smears the extrapolated wave's height to zero.

Furthermore, the variation in each coefficient (LI method), for a single grid point, is presented in Fig.4.13 and the variation of each correction (LC method) is presented in Fig.4.14. This is done for two periods in time, which constitute the total time included in training. It is noteworthy that the corrections exhibit small oscillations when the coefficients do not. This observation underlies the fact that the corrections act directly on the field values in an alleviating/therapeutic way to any unwanted occurence in the dynamics of the field (i.e. any divergence from the training data distribution) like the small oscillations produced by second-order not-limited schemes, when the coefficients act in more of a preventive manner by generating solutions that avoid the development of oscillations altogether.

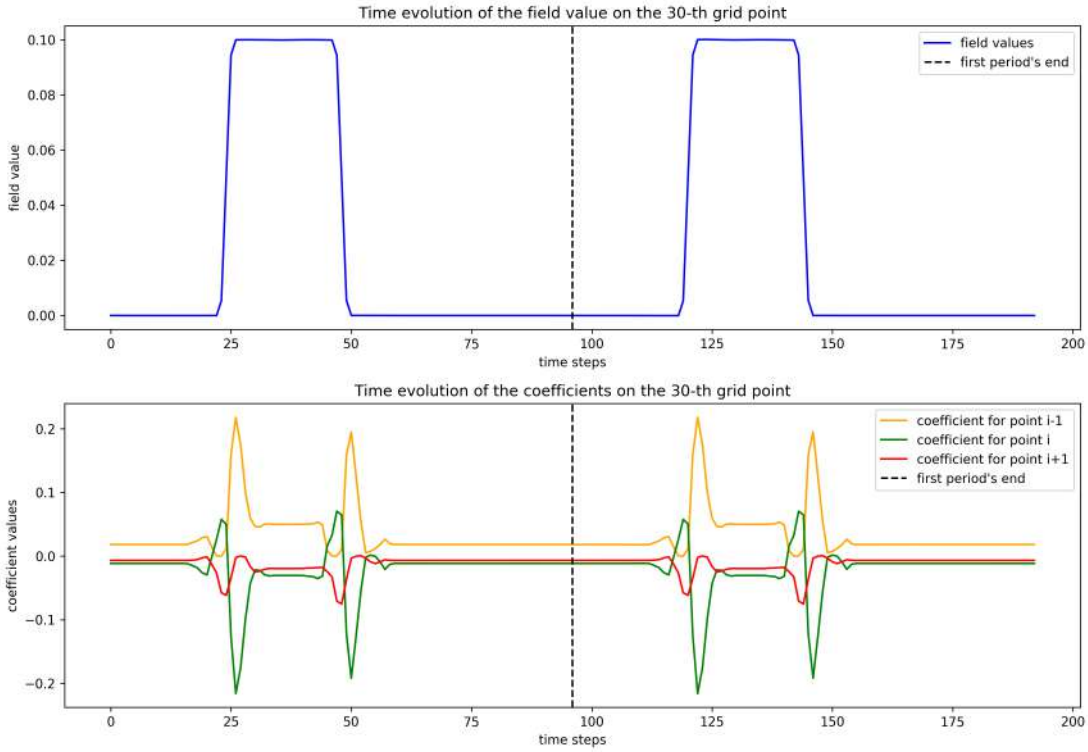


Figure 4.13: *The time evolution of the coefficients of the 30-th grid point in a time span of two periods.*

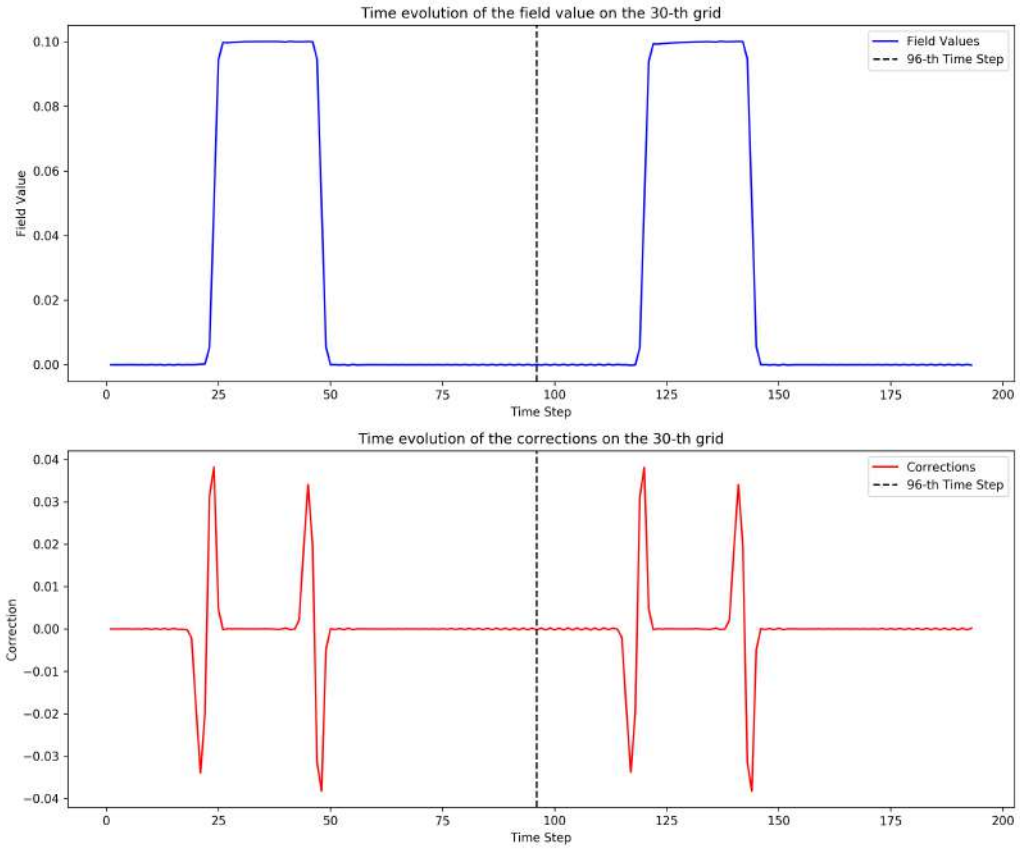


Figure 4.14: *The time evolution of the corrections of the 30-th grid point in a time span of two periods.*

Then, the coefficients that are outputed by the LI model for different time steps are presented in Fig.4.15. It is important to note that these coefficients lead to slopes that interchange between centered, upwind, downwind or custom schemes respectively. The choice of the proper scheme by the hybrid solver depends on the shape of the square wave at every point. It is also noteworthy that, on the sharp drops of the wave, a limiting effect is achieved, when no limiter has been built inside the hybrid model. In this case, where the square wave is just advected in space, optimal coefficients are learned for computing the spatial derivative for the initial square wave and then, "attached" to the point of the wave in which they achieve good results, they also just advect in space.

Finally, the corrections that are outputed by the LC model for different time steps are presented in Fig.4.16. Again, as described for the coefficients, the NN finds some good corrections and seems to attach them to the traveling wave. It has learned that the dynamics is just propagating the wave.

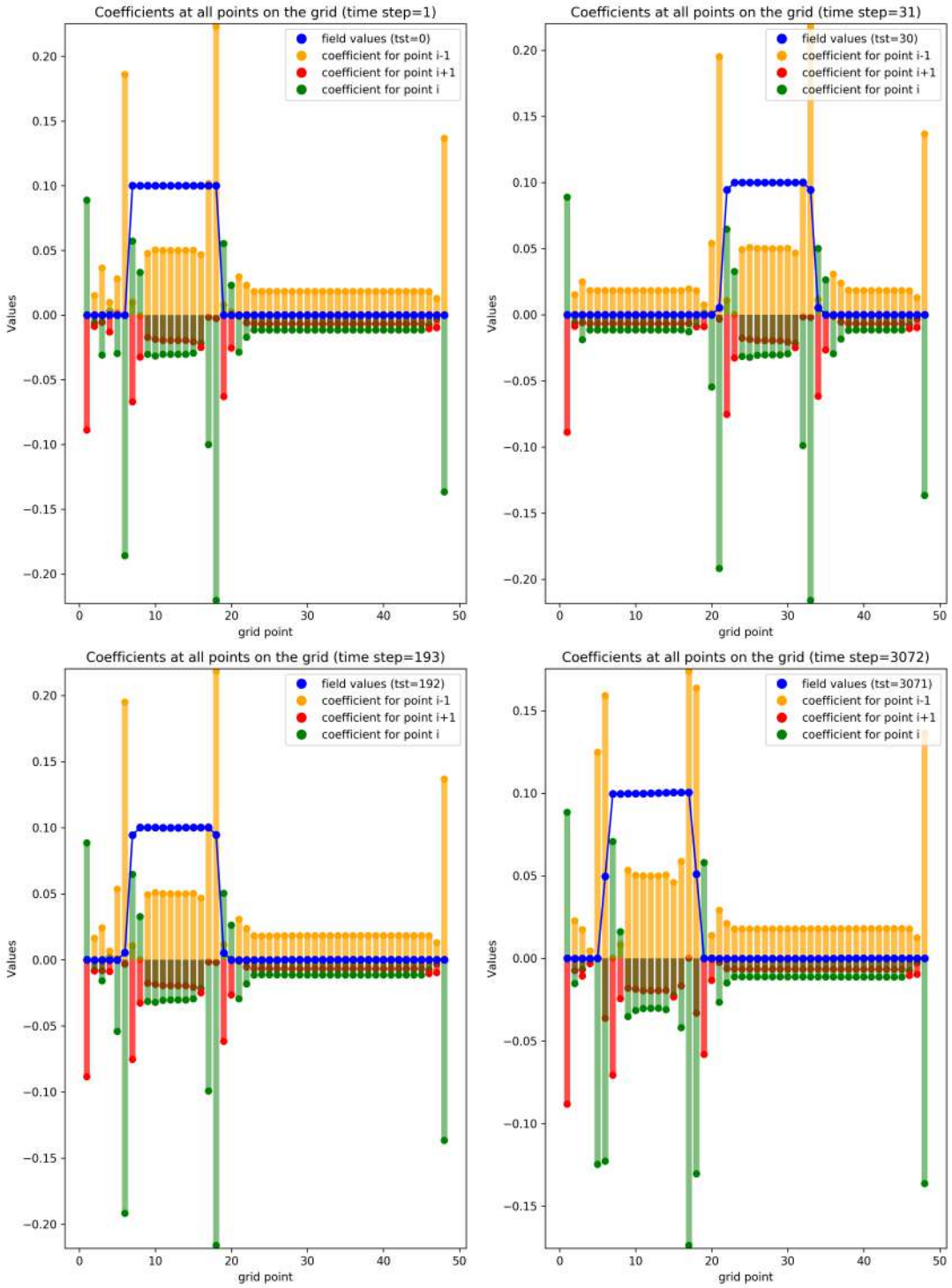


Figure 4.15: *The coefficients generated inside the hybrid model are space and time dependent. The 3-point stencil for the i-th point includes i-1,i, i+1 points. In regions that the solution is constant, the coefficients are identical or almost identical. The values of the coefficients -all scaled by* $\frac{1}{2\Delta x}$*- are depicted on the y-axis of the subplots, with each colour representing the coefficient of one of the points in the stencil. Their values also sum to zero, as a first-order accuracy constraint has been enforced (see sec.3.3.2). Coefficients when interpolating in time, (timesteps 1,31,193), and when extrapolating in time (timestep 3072) are presented.*

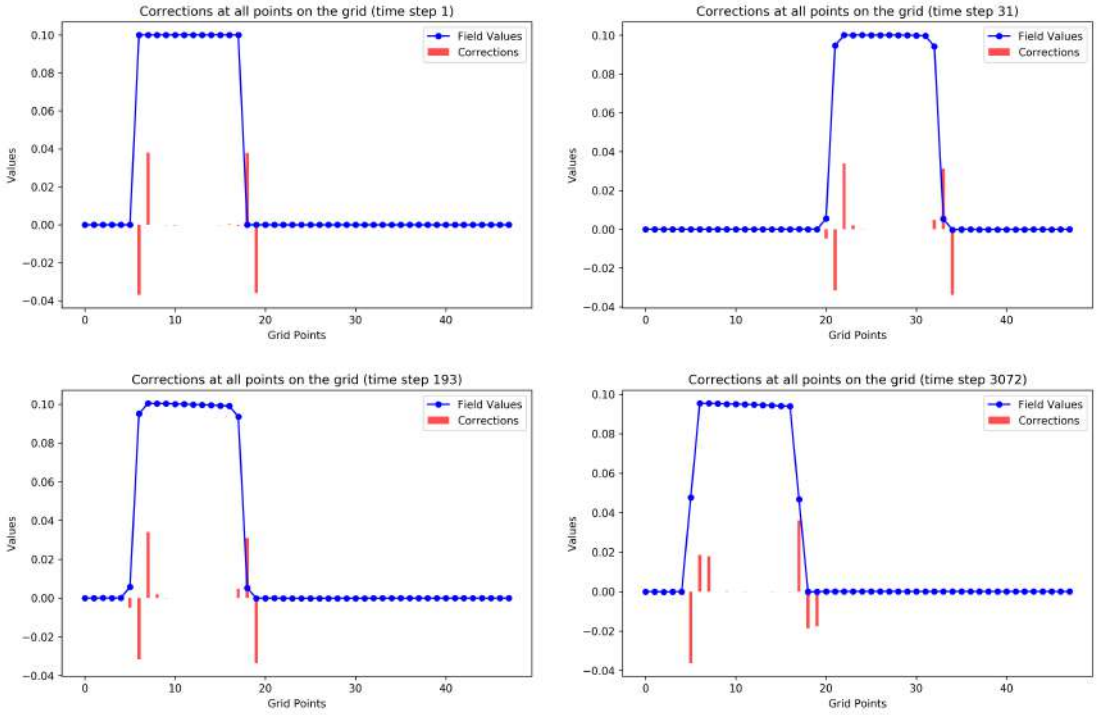


Figure 4.16: *The corrections that are generated inside the hybrid model are space and time dependent. In regions that the solution is constant, the corrections are identical and close to zero. Corrections when interpolating in time, (timesteps 1,31,193), and when extrapolating in time (timestep 3072) are presented.*

In summary, a substantial improvement has been realized. When utilizing a 48-point grid, the hybrid solvers produce results that closely approximate those of a 384-point numerical solution. And they do so in an interpretable way.

Chapter 5

Case 2: 1D Linear Acoustics

5.1 Introduction

The 1D linear acoustic equations, often used to describe sound waves in a fluid medium, consist of a system of two coupled PDEs that represent the conservation of mass (continuity equation) and the conservation of momentum.

$$\frac{\partial}{\partial t}\begin{bmatrix} p \\ u \end{bmatrix} + \begin{bmatrix} 0 & K_0 \\ \frac{1}{\rho_0} & 0 \end{bmatrix} \frac{\partial}{\partial x}\begin{bmatrix} p \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5.1}$$

In these equations:

- $p(x,t)$ is the pressure perturbation from the ambient pressure.
- $u(x,t)$ is the velocity perturbation from the ambient velocity
- $K_0 = \rho_0 {c_0}^2$ is the bulk modulus of the medium (c_0 is the speed of sound)
- ρ_0 is the density of the medium.

These equations assume small perturbations in pressure and velocity. The boundary conditions will be periodic and the initial conditions square waves of different heighs in pressure and zero velocity everywhere:

$$\left\{ \begin{array}{l} p(x=0,t) = p(x=L,t) \\ u(x=0,t) = u(x=L,t) \end{array} \right\} \tag{5.2}$$

$$\left\{ \begin{array}{l} p \\ u \end{array} \right\} (x, t=0) = \left\{ \begin{array}{ll} height_p, & \text{if } x_l < x < x_r, \\ 0, & \text{otherwise} \end{array} \right\} \tag{5.3}$$

The high-resolution solution on the fine grid (for $height_p = 0.6$) for different medium densities ρ_0 is presented in Fig.5.1. These simulation results also serve as a baseline for evaluating the hybrid models in the next sections. It is also clear that for different medium densities, using the previously defined typical initial condition the pressure field retains the exact same dynamics and scale but the velocity fields while having the same dynamics, have its scale change (orange curves in different rows of Fig.5.1).

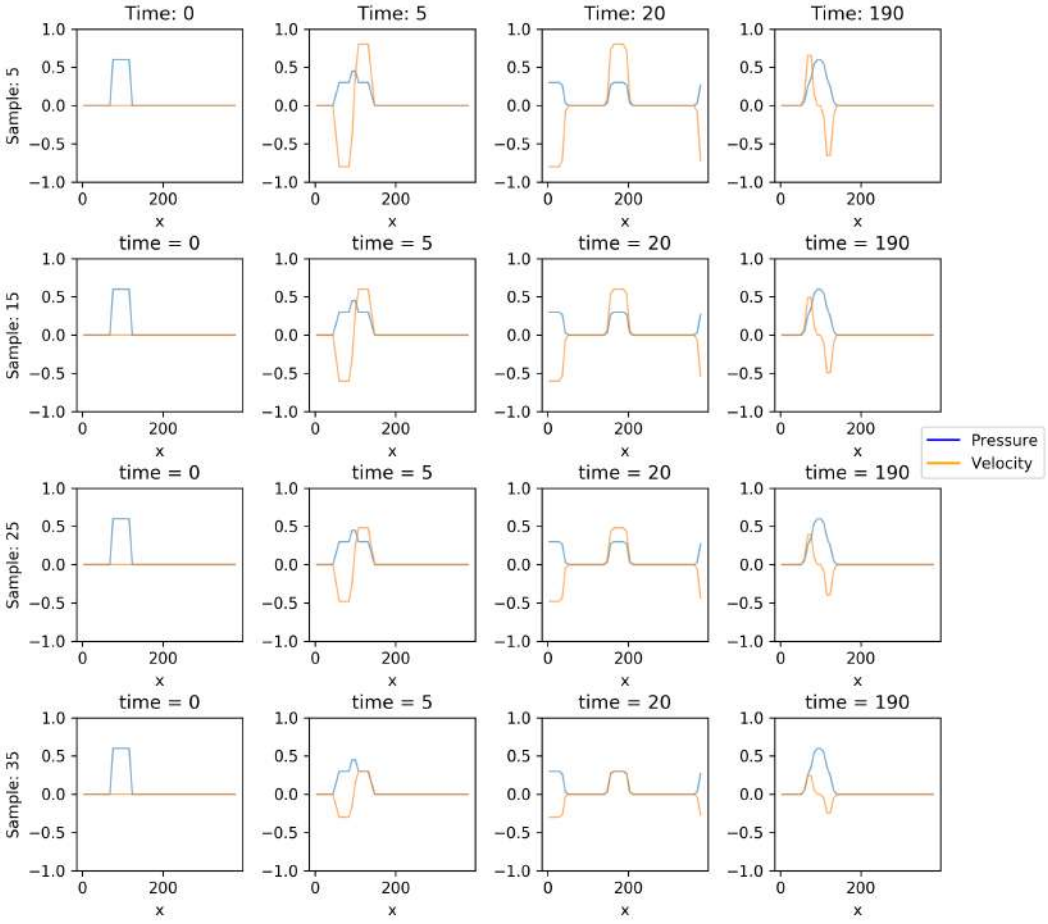


Figure 5.1: *The expected dynamics of the linear acoustic equations. The effect of the change of medium density is a change of the scale in the dynamics of the velocity fields. Hence, the relation between pressure and velocity also changes. The results were produced by the numerical solver on the fine grid. The velocity snapshots are multiplied by a factor of 680 for plotting purposes. The header "time" refers to time steps.*

5.2 Equation discretization

In vector form, the linear acoustics equation can be written as

$$\frac{\partial Q}{\partial t} + A\frac{\partial Q}{\partial x} = 0 \tag{5.4}$$

Its discretization is similar to case 1, only applied to a system. For a second-order scheme, using the Van Leer limiter, this would amount to:

$$F_{i-\frac{1}{2}} = A^+ Q_{i-1} + A^- Q_i + \tilde{F}_{i-\frac{1}{2}} \tag{5.5}$$

where

$$\tilde{F}_{i-\frac{1}{2}} = \frac{1}{2}|A|\left(I - \frac{\Delta t}{\Delta x}|A|\right)\sum_{p=1}^{p=m} \tilde{\alpha}^p_{i-\frac{1}{2}} r^p \tag{5.6}$$

with $A^+ = R\Lambda^+ R^{-1}$, $A^- = R\Lambda^- R^{-1}$ and $|A| = A^+ - A^-$, where R are the right eigenvectors of the A matrix and:

$$\Lambda^+ = \begin{bmatrix} \lambda_1^+ & 0 & \cdots & 0 \\ 0 & \lambda_2^+ & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_m^+ \end{bmatrix} \tag{5.7}$$

$$\Lambda^- = \begin{bmatrix} \lambda_1^- & 0 & \cdots & 0 \\ 0 & \lambda_2^- & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_m^- \end{bmatrix} \tag{5.8}$$

with λ^+ and λ^- being the positive and negative eigenvalues respectively.

Here, the sum $\sum_{p=1}^{p=m} \tilde{\alpha}^p_{i-\frac{1}{2}} r^p$ represents the eigendecomposition of the traveling discontinuitites as predicted by the Rankine-Hugoniot condition. Therefore, r_p represent the right eigenvectors of the matrix A and $\tilde{\alpha}^p_{i-\frac{1}{2}}$ represent the limited eigencoefficients

$$\tilde{\alpha}^p_{i-\frac{1}{2}} = \phi(\theta^n_{i-1/2})\alpha^p_{i-\frac{1}{2}} \tag{5.9}$$

and

$$\theta^p_{i-\frac{1}{2}} = \frac{\alpha^p_{I-\frac{1}{2}}}{\alpha^p_{i-\frac{1}{2}}} \quad \text{with} \quad I = \begin{cases} i-1 & \text{if } \lambda^p > 0, \\ i+1 & \text{if } \lambda^p < 0. \end{cases} \tag{5.10}$$

Finally, the FV method along with Euler time integration would then amount to:

$$Q^{n+1}_i = Q^n_i - \frac{\Delta t}{\Delta x}(F^n_{i+1/2} - F^n_{i-1/2}) \tag{5.11}$$

5.3 Training data

Firstly, regarding the discretization of the spatial and temporal domains, a uniform grid with step size of $\Delta x = 1$ is chosen (for the fine grid) and the time step is determined using the CFL condition: $c = c_0 \frac{\Delta t}{\Delta x}$, where c_0 is the speed of sound and is the maximum information propagation speed for both acoustics waves (traveling in opposite directions). By setting the courant number to $c = 0.5$, and the speed of sound to $c_0 = 340 m/s$, the time step becomes $\Delta t \approx 0.00147 s$.

When generating training data for the model, the discretized equation is solved with a FVM second-order limited (Van Leer) scheme, on a 384-points grid, using 10 initial conditions of square waves (starting from the 96^{th} node of the 384-node grid and has $width = 48$ points) with varying heights in pressure and zero initial velocity always:

$$height_p \in [0.1, 1] \text{ with step } 0.1$$
$$height_u = 0$$

The time integration spans two periods: $t_{final} = 2T \approx 2.26 sec$. That means that with a time step of $\Delta t \approx 0.00147 s$, the total time steps needed are 1536.

As in the first case, the boundary conditions are periodic. Changing the equation but retaining the same initial conditions can lead to safer conclusions about the hybrid methods ability to capture different dynamics without possible interference of the alteration of initial conditions.

In this case, the training data contain the results of the high-resolution numerical simulation (fine grid) of these ten initial conditions for four different medium density values (which leads to changes in the coefficient matrix of the system of PDEs)

$$\rho_0 \in \{0.75, 1, 1.25, 2\} kg/m^3$$

The effect of the change of medium density on the solution has been presented in Fig.5.1. As in case 1, while running the high-resolution numerical solver, the results undergo coarsening in time (downsampling) and space (averaging) with a $8\times$ coarsening ratio, which is the maximum allowable one.

5.4 Results of the hybrid models

Parameter	Value
K	10 initial conditions
λ_1	3
λ_2	7
M_f	384 grid points
M_c	48 grid points
N_f	1536 time steps
N_c	192 time steps
S	3-point
CR	8

Table 5.1: *Table with specified parameters' values for both LI and LC hybrid methods. The symbols have been defined in Tab.3.1.*

Hyperparameter	LI method	LC method
number of layers	5	5
number of filters	64	64
kernel size	5×1	5×1
batch size	64	64
learning rate	3e-3	3e-3, 3e-4
epochs	13	170
Q time-steps	10	15
optimizer	Adam	
weight initialization	Xavier	
activation functions	ReLU	

Table 5.2: *Chosen Hyperparameters for the hybrid models. All layers have the same number of filters and same activation function. Hyperparameter Q defines how many subsequent snapshots the solver produces per run (see sec.3.3).*

The main parameters' values, as defined in Tab.3.1, and analyzed in the previous section, are summarized in Tab.5.1. And the hyperparameters for each hybrid model are presented in Tab.5.2.

The chosen neural network is a CNN of five layers with ReLU activation functions. Even though the problem exhibits more complex behavior, the required NNs to capture its dynamics do not get proportionally bigger. That is a good sign that this technique will be affordable for real world problems.

A key architectural consideration involved the treatment of different variables. In the finally chosen CNN, the receptive field contains both variables as different channels. The approach of using two separate CNNs, each dedicated to one variable,

was explored but resulted in suboptimal performance. Specifically, it was observed that to achieve comparable effectiveness, each CNN required five layers, indicating a significant increase in model complexity. Subsequent design choices pivoted on whether to use the same or different coefficients for the two variables within a single CNN architecture. A trial-and-error procedure showed that the use of distinct coefficients for each variable yields superior outcomes. This is likely attributable to the additional flexibility afforded to the model, which is particularly beneficial in the context of the physically constrained nature of hybrid CFD models. Providing distinct coefficients for each variable allows the CNN to adapt more effectively to the unique characteristics and dynamics of each variable.

As before, the model is firstly asked to predict over the time period it has encountered in its training data, as well as for future times. More specifically, the CNN has seen two periods but is asked to predict four periods in time (to evaluate the stability of the solver). This is then compared to the performance of a traditional numerical solution using 48 grid points. The subsequent figures demonstrate the hybrid solvers' capability to generalize to different equation parameters, a critical property for applying this technique to more complex equations and in more diverse simulation scenarios.

Note that in all following plots, the velocity snapshots have been multiplied by a factor of 680 to bring the pressure and velocity scales closer together and facilitate showcasing them in the same axes.

First of all, Fig.5.2 presents a comparative evaluation where the MAE of the coarse-grid numerical solution with respect to the coarsened fine-grid numerical solution (serving as the baseline MAE) and the MAE of the hybrid solvers' output on the coarse grid with respect to the fine-grid numerical solution. The MAEs for pressure and velocity are plotted separately to showcase the solver's performance in each quantity. In this analysis, both the wave height (IC) and the medium density have not been encountered during training, but belong in the training data distribution.

Then, for LI and LC hybrid solvers respectively, Figs.5.3 and 5.4 showcase multiple snapshots of a wave (out-of-sample initial conditions and medium density) at different time points, including times beyond those seen in training. Its ability to extend its predictions into future timeframes not explicitly presented during training underscores the hybrid solver's stability and the way that, for converged solvers, the error is spread almost evenly across time steps. This visualization illustrates that these solvers closely follow the high-resolution solution. It also seems that the LC model gets better performance than the LI model. However, because NNs are stochastic machines, as they rely on random seeds, initializations and more, direct comparisons for the two models should not be made in a naive manner.

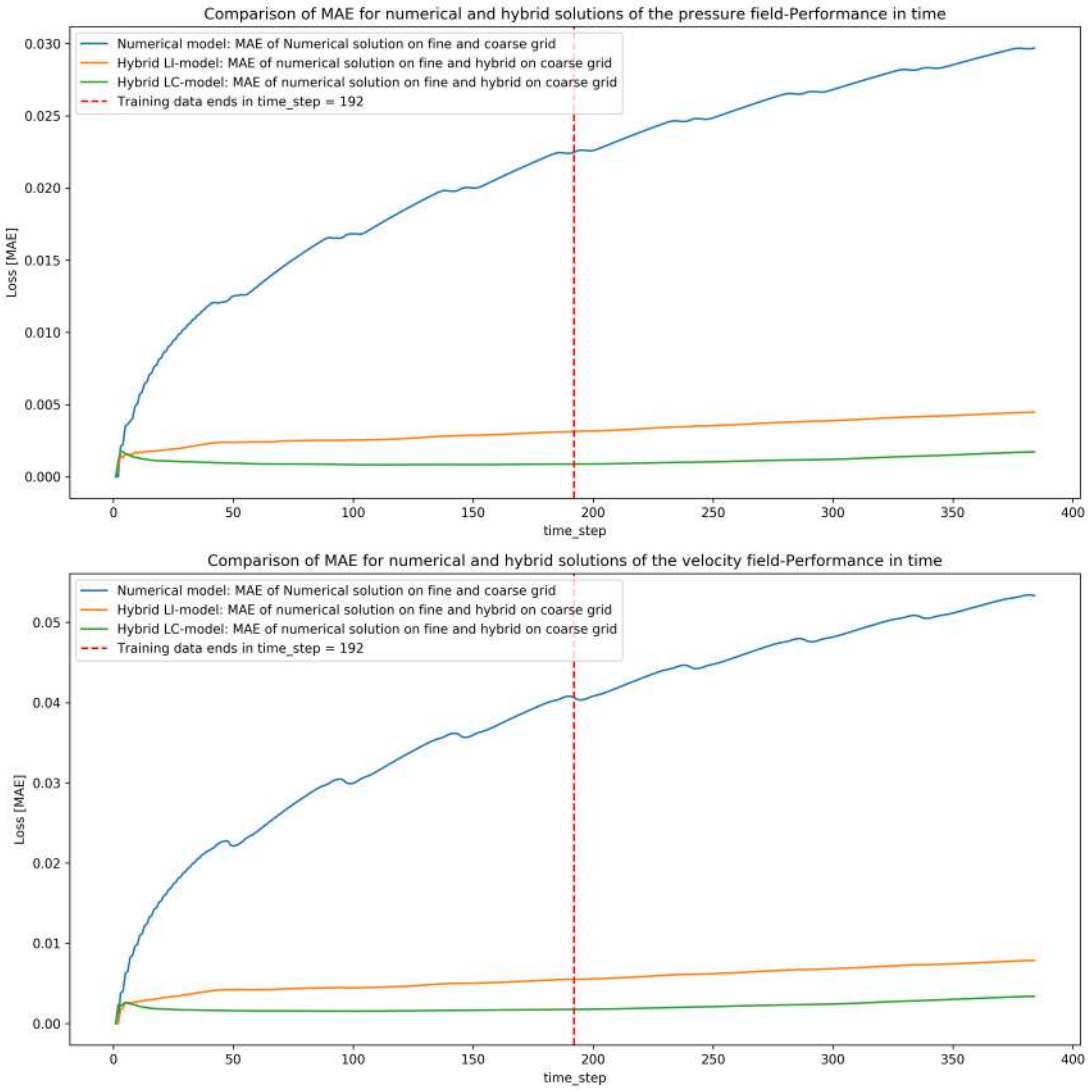


Figure 5.2: *Comparison of the MAE of pressure and velocity fields at every time step by a numerical model with a second-order scheme with Van Leer limiter to the hybrid models on the coarse grid of 48-points. The baseline is the numerical solution of the discretized equation on a 384-points grid, projected to 48 points. This is for initial conditions of square waves with heights not seen during training ((new heights) =* $0.65 * $ *(old heights)) and with a medium of density also not seen in training (*$\rho = 1.5 kg/m^3$*). The solution is also extrapolated for two periods in time. The end-to-end numerical solver makes bigger errors on velocity rather than on pressure snapshots and so does the hybrid solver.*

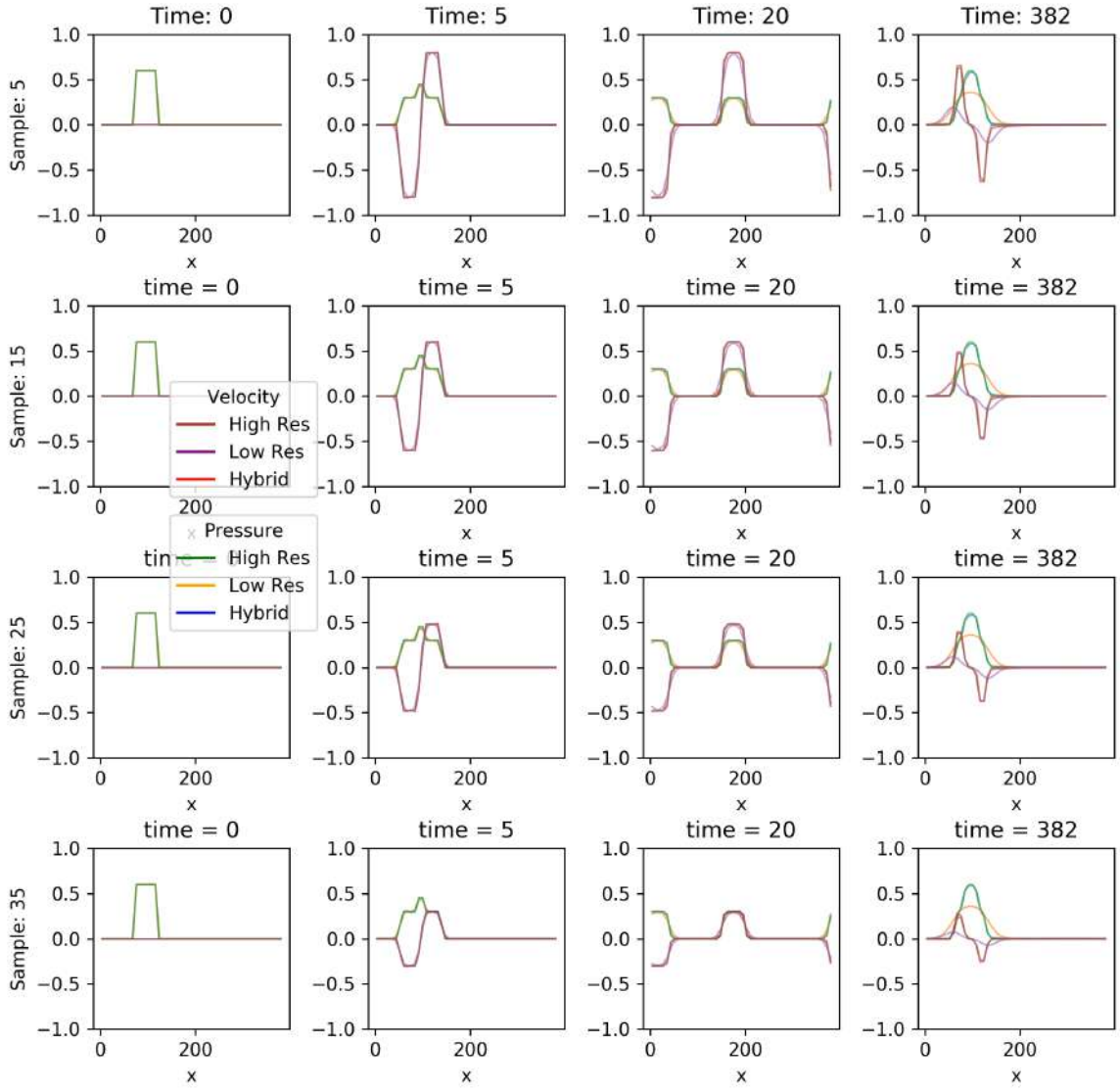


Figure 5.3: *Comparison of multiple pressure and velocity field snapshots solved by a second-order scheme with Van Leer limiter to the LI hybrid model on the coarse grid of 48-points. The baseline is the numerical solution of the linear acoustics equation on a 384-points grid, coarsened to 48 points. Neither the initial square wave has been seen in training nor the specific -interpolated- medium densities (different per row). The solution is also extrapolated for two periods in time. The velocity snapshots are multiplied by a factor of 680 for plotting purposes. The header "time" refers to time steps. Different rows correspond to different (interpolated) out-of-sample medium density values.*

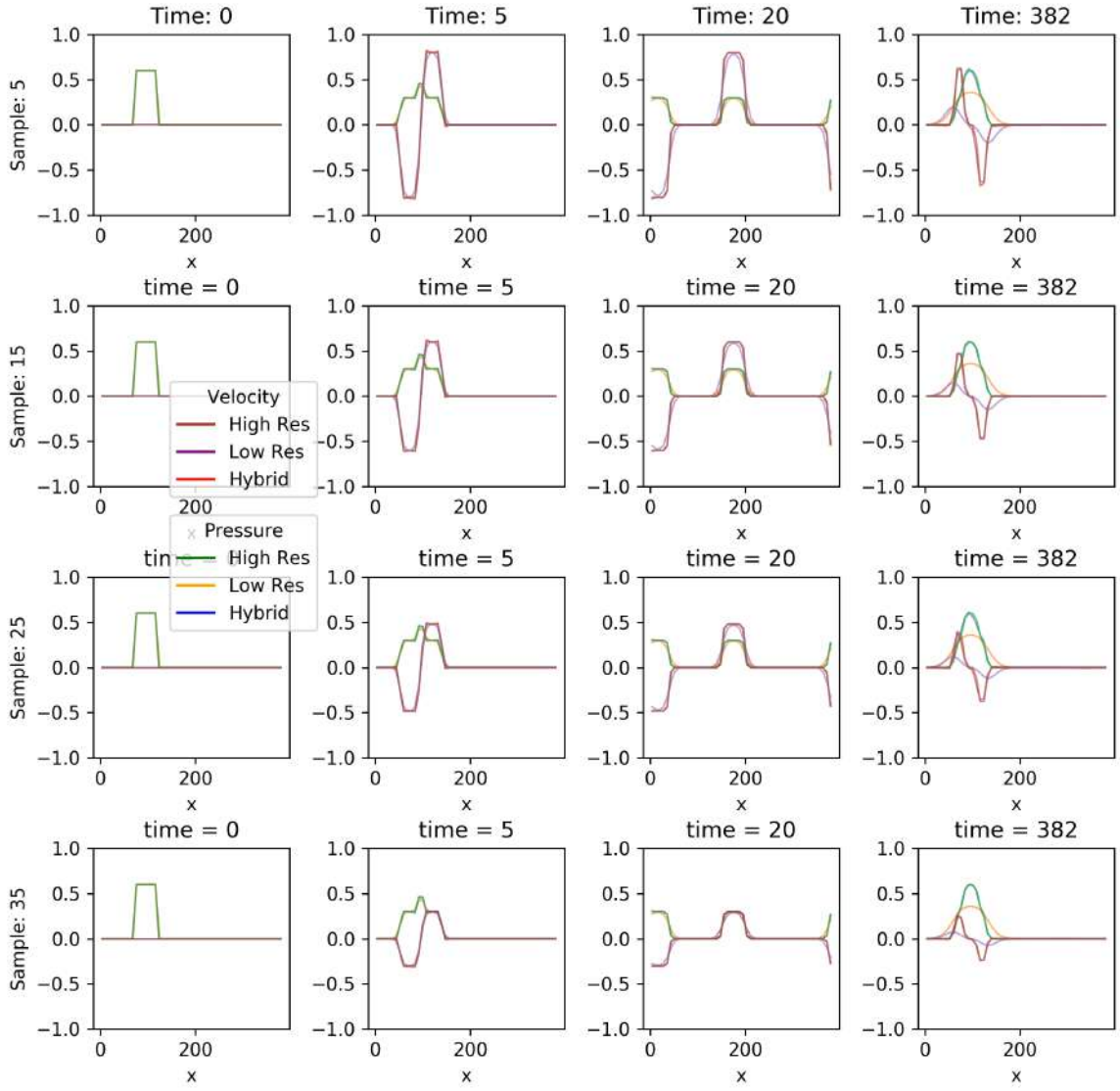


Figure 5.4: *Comparison of multiple pressure and velocity (multiplied by a factor of 680) field snapshots solved by a second-order scheme with Van Leer limiter to the LC hybrid model on the coarse grid of 48-points. The baseline is the numerical solution of the linear acoustics equation on a 384-points grid, coarsened to 48 points. Neither the initial square wave have been seen in training nor these specific medium densities. The solution is also extrapolated for two periods in time. The header "time" refers to time steps. Different rows correspond to different (interpolated) out-of-sample medium density values.*

The previous demonstrations, have shown that the results that have been produced by the hybrid solvers with a 48-points grid, attain an accuracy close to what would be generated by a 384-points grid one when both initial condition and equation parameters change.

5.5 A parametric study on hyperparameters' influence on solvers' performance

5.5.1 LI hybrid solver

In this section, a parametric study is conducted to explore the impact of important hyperparameters (learning rate, batch size, Q time-steps) on the performance of the hybrid solver. The key metric of interest here is the MAE between the hybrid solver's output on the coarse grid and the numerical solution results on the fine grid.

The lowest MAE in simulations is always achieved on the first few time steps, where the solver's (and thus the embedded NN's) input and output closely aligns with the training data. As the simulation progresses, the MAE typically increases, reaching its peak in the later stages. This increase in MAE is expected due to the growing deviation from the initial state, pushing the network into regions less represented in the training data. Because of these qualities of the hybrid solver, MAE is a crucial metric in assessing both the solver's stability and the minimum accuracy achieved at any point during the solver run, offering a comprehensive view of the solver's performance.

The initial aspect of the study focuses on how varying hyperparameter values affect the proportion of training epochs at which stopping would result in a stable solver. This aspect is crucial since the practice of early stopping involves periodic checks every few epochs instead of after every single one to minimize computational costs (see sec.3.4). If only a small percentage of epochs yield a stable solver, it becomes increasingly unlikely to achieve stability. Even if early stopping is omitted, getting stable solvers is a prerequisite for the successful implementation of the techniques discussed in this thesis.

For evaluating the stability and accuracy of the solvers, the maximum MAE for all time steps in the solution produced by a pure numerical solver on a coarse grid, which is approximately 0.033, is defined as the baseline. In Fig.5.5, regions with MAE values meeting or exceeding this baseline value are depicted in blue, indicating solvers that are both stable and accurate. Conversely, orange regions denote solvers that, while stable, fall short in accuracy compared to the baseline numerical solver, rendering them mundane. It is clear that batch size does not significantly correlate with the stability of the solvers. Instead, the learning rate emerges as a more critical factor. Lower learning rates generally lead to increased occurance of stable solvers, but further experimentation with even lower rates (e.g., 3e-5) yields mixed outcomes. This suggests the existence of an optimal learning rate for stability but the way to determine it remains unclear. Practically though, adjusting the learning rate can be a strategic move when seeking to increase occurance of stable solvers. Moreover, it is clear that the number of future snapshots produced by the solver ("quantum steps")

significantly influences stability. This is consistent with the findings of [6, 54, 59].

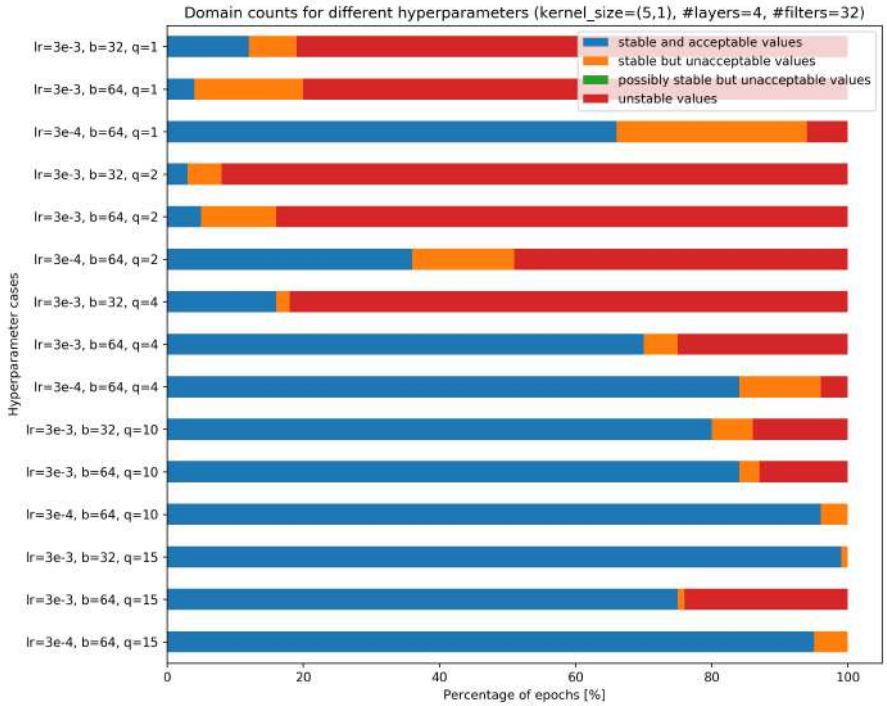


Figure 5.5: *Percentage of epochs when the trained LI hybrid solver is stable. Blue regions represent epochs during which the solver is stable and at least as accurate as the pure numerical solver on the coarse grid. Orange regions represent stable solvers but with accuracy lower than that of the numerical solver on the coarse grid (MAE ranging from 0.03 to 1). Green regions roughly represent possibly stable solvers with unacceptable values (MAE ranging from 1 to 100).These intermediate values do not occur, possibly because instability is quick to happen during running the solver. Finally, red regions represent complety unstable values (MAE ranging from* 100 *to* ∞*). It is apparent that better results are attainable using higher numbers of unrolled steps.*

.

Next, the study presents a comparison in accuracy of all occured solvers across a 100-epoch training period with differing hyperparameter values, as shown in the boxplot in Fig.5.6. This analysis focuses solely on solvers with acceptable MAE (lower than the previously mentioned baseline). It reveals no distinct relationship between accuracy and hyperparameters like learning rate or batch size. However, a notable positive correlation emerges between solver accuracy and the "quantum steps" hyperparameter. This means that, increasing "quantum steps" not only enhances the likelihood of achieving a stable solver (up to 100% as seen in Fig.5.5) but also heavily improves accuracy. These improvements, however, come at the cost of increased training cost. For instance, if "quantum steps" equals 1, the training cost per epoch correlates with the number of training snapshots, say 1000. But if

"quantum steps" is set to 15, the cost escalates, and is proportional to 15,000 for the same number of epochs.

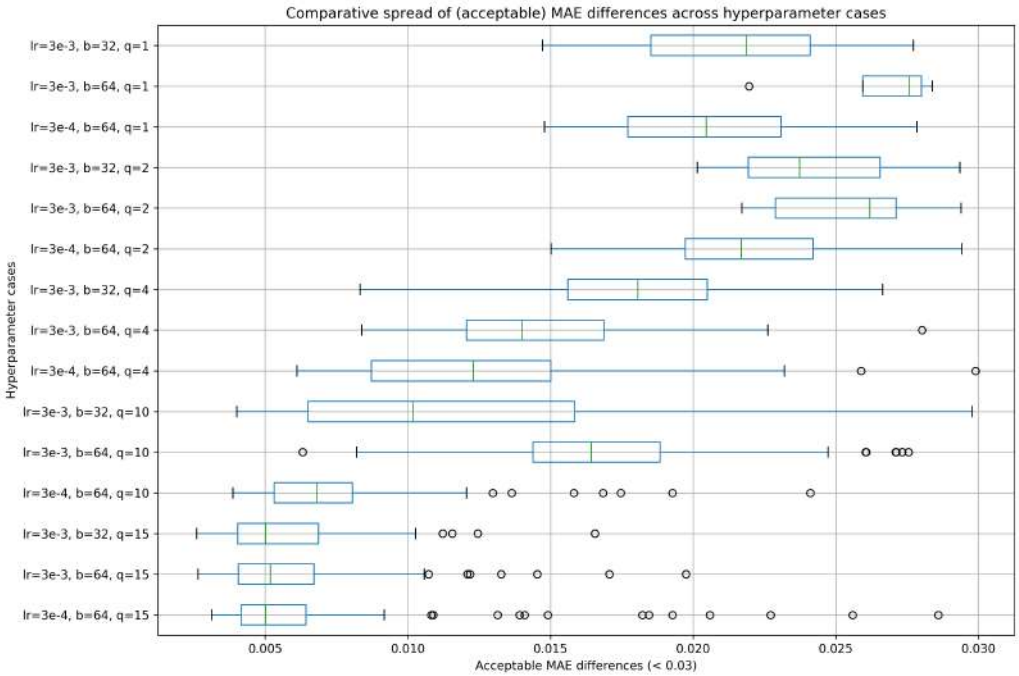


Figure 5.6: *In the boxplot for the LI method, the box shows where the central bulk of the MAE values lies, giving an idea of the typical range. The median line provides a sense of a typical or "middle" value of the data. The whiskers extend to the most extreme MAE values that are not considered outliers, giving a sense of the overall spread of the data. Outlier points represent MAE values that are unusually high or low compared to the rest of the data. In labels, lr represents learning rate, b the batch size, q the quantum steps. In bigger quantum step sizes (e.g. q = 15), the box is far to the left side, indicating achievement of significantly better accuracy, while also being thinner indicating more consistent performance.*

In addition to stability and accuracy, another critical aspect to consider is the time at which representative low values for MAE (here the median value is chosen) are attained, as shown in Fig.5.7. This is crucial as, in order to prevent overfitting and get stable solvers, early stopping can be used. The analysis reveals that, on average, smaller learning rates (indicated by red-colored bubbles) take longer to reach the median MAE value compared to larger ones (blue bubbles), while the effects of batch size (small and big bubbles for 32 and 64 batch size respectively) and the number of quantum steps appear inconclusive. This hinges on a trade-off between the potentially increased likelihood of stability offered by lower learning rates (as noted in Fig.5.5) and the speed at which optimal results are achieved, which has implications for the overall training cost.

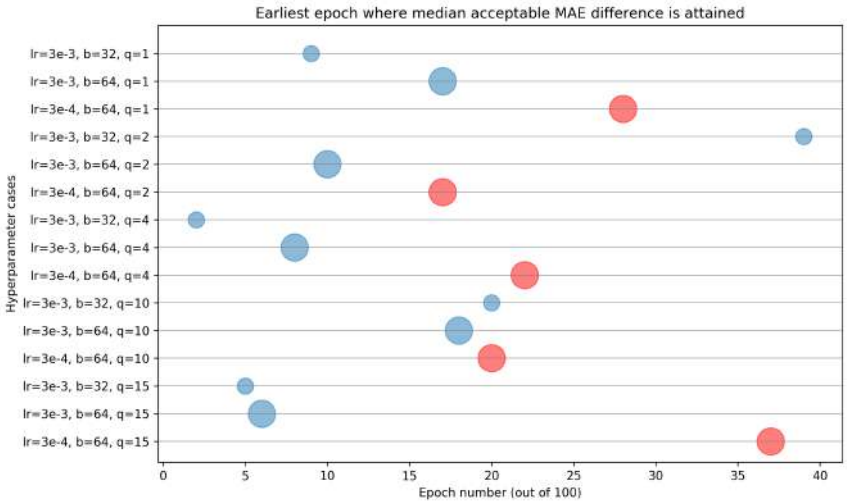


Figure 5.7: *For the LI solver, presentation of the epoch that in each hyperparameters' case the threshold of the median value of the MAE is exceeded. Median value was chosen as a representative of a good result relative to each set of hyperparameters. In labels, lr represents learning rate, b the batch size, q the quantum steps. Learning rates include the typical values of 3e-3 and 3e-4 (blue and red colors respectively) and batch sizes include the typical values of 32 and 64 (small and big bubbles respectively). The speed at which optimal -relative to each hyperparameters set case- results are achieved is better for the bigger learning rates, as higher lr values mean bolder steps during gradient descent optimization.*

The analysis of these figures and prior discussions suggest that the most significant hyperparameter decision in hybrid solvers relates to the number of field snapshots that the solver is tasked to predict recursively. This decision hinges on a trade-off: choosing a higher value for Q-steps typically enhances stability and accuracy, but also increases the computational cost during training. Therefore, determining the optimal value involves balancing the desire for improved solver performance against the practical considerations of training efficiency and resource utilization. Sensibly using early stopping (see sec.3.4), can also weigh in this decision to potentially achieve good performance with less recursive steps. For instance in Fig.5.6 there are quite accurate solvers occuring even when $q = 4$ ($lr = 3e-4, b = 64, q = 4$ case), when for more consistent results during training one would opt for other, more expensive, choices (e.g. $lr = 3e-4, b = 64, q = 15$).

5.5.2 LC hybrid solver

The same parametric study is conducted again for the LC hybrid solver. Based on the following figures, the main conclusions drawn for the LI solver still hold. The Q time-steps are the most significant hyperparameter for achieving stable and accurate solvers. Plus, as before, changing the learning rate is shown to be an

effective strategy to potentially getting different or better results and the batch size correlation with stability and accuracy remain unclear (see Fig.5.8).

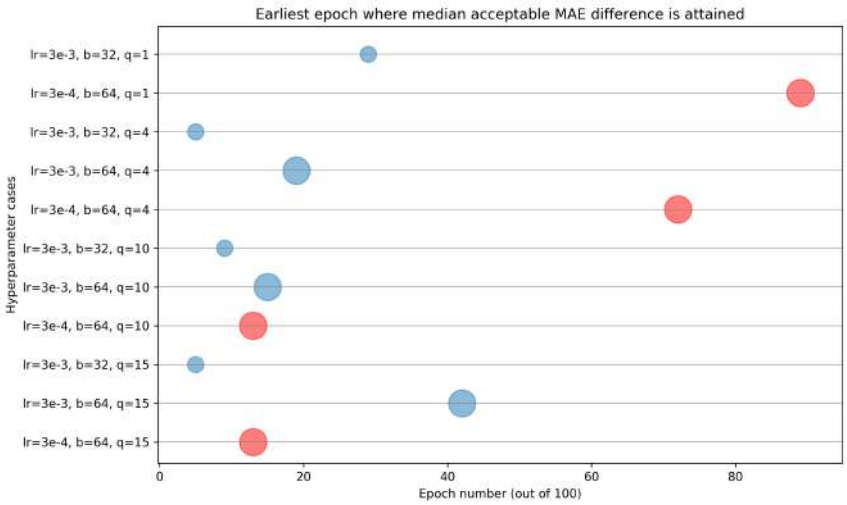


Figure 5.8: *For the LC solver, presentation of the epoch that in each hyperparameters' case the threshold of the median value of the MAE metric is exceeded. In labels, lr represents learning rate, b the batch size, q the quantum steps. Learning rates include the typical values of 3e-3 and 3e-4 (blue and red colors respectively) and batch sizes include values of 32 and 64 (small and big bubbles respectively). The speed at which optimal -relative to each case- results are achieved is better for the bigger learning rates, as higher lr values mean bolder steps during gradient descent optimization. The hyperparameter cases where an acceptable stable solver does not occur are ommited.*

However, some comparative observations of LI and LC methods can be made, at least for the specific case examined here. Firstly, by comparing Figs.5.5 and 5.11, it is clear that the occurence rate of stable solvers is quite higher in the LC method. The reason behind this is that the CNN learns to predict very small corrections (as the deviation of the coarse solution from the coarsened Hi-Fi results at each time step is small) that are in a much smaller scale and of a much lower "energy" than the dynamics of the flow and thus cannot lead to the explosions needed for an unstable solver. It is also possible that this stems from the fact that the LC solver has acquired a mostly diffusive behavior, whereas the LI solver a mostly anti-diffusive one as previously seen in Figs.4.11 and 4.12.

By comparing Figs.5.6 and 5.9, it is also apparent that the LC method, in the same amount of epochs, does not get solvers as accurate as the LI method. It was found however that for more epochs, the solvers emerging from training do get more accurate (see Figs.5.9 and 5.10) . This possibly means that the LC method just converges slower. Of course, slower convergence leads to more epochs, which significantly raises the training cost.

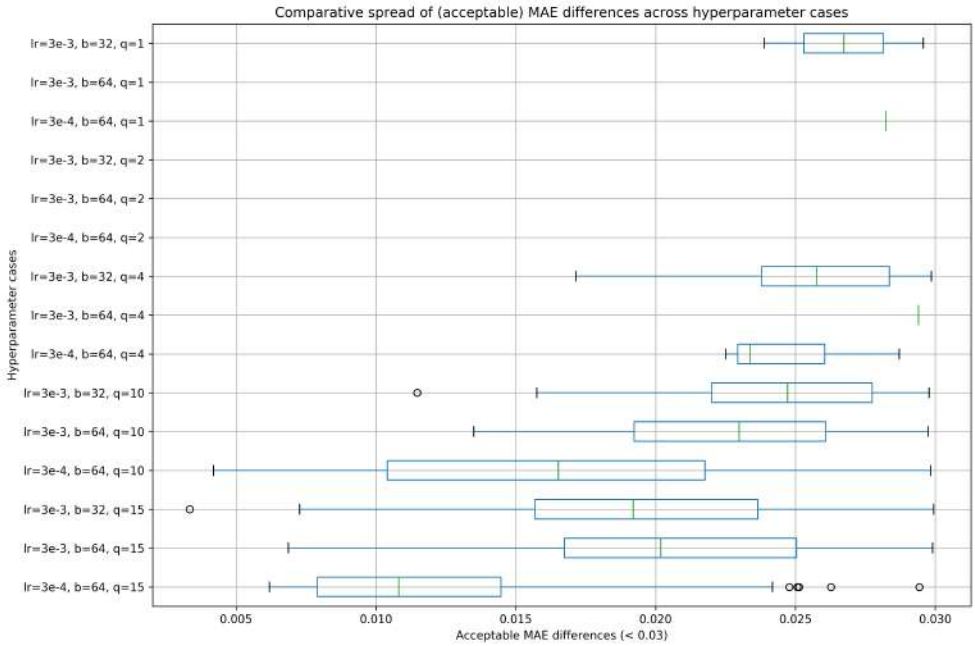


Figure 5.9: *Boxplot for the LC method. The meaning of the box, median line, whiskers and outliers have been explained in Fig.5.6. In labels, lr represents learning rate, b the batch size, q the quantum steps. As for the LI case, in bigger quantum step sizes (e.g. $q = 15$), the box is far to the left side, indicating achievement of significantly better accuracy, while also being thinner indicating more consistent performance.*

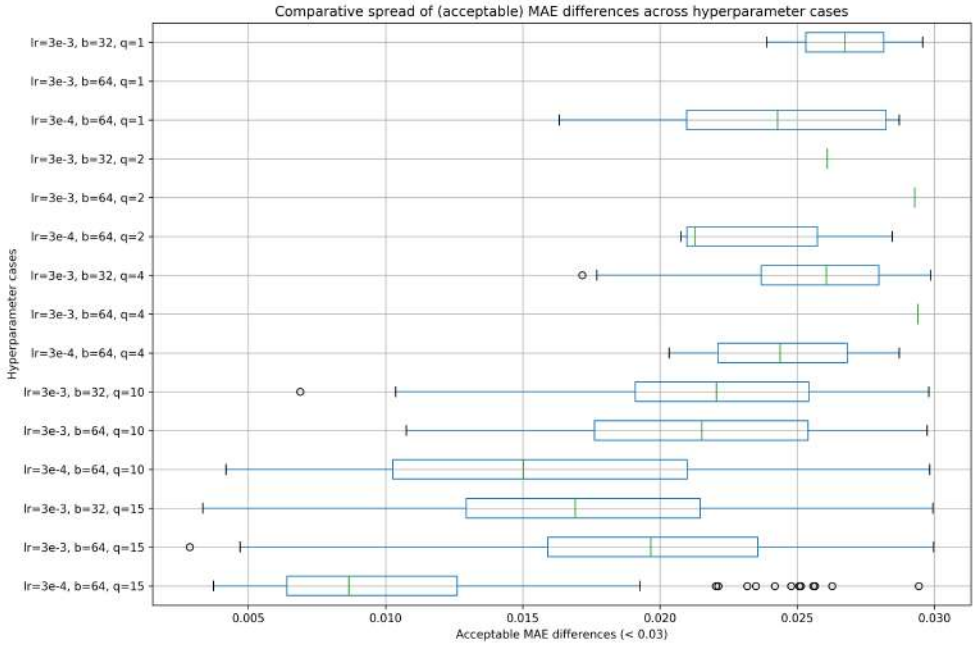


Figure 5.10: *Same plot with Fig.5.9 except for the fact that the training lasts 200 epochs instead of 100. The results get better accross the board.*

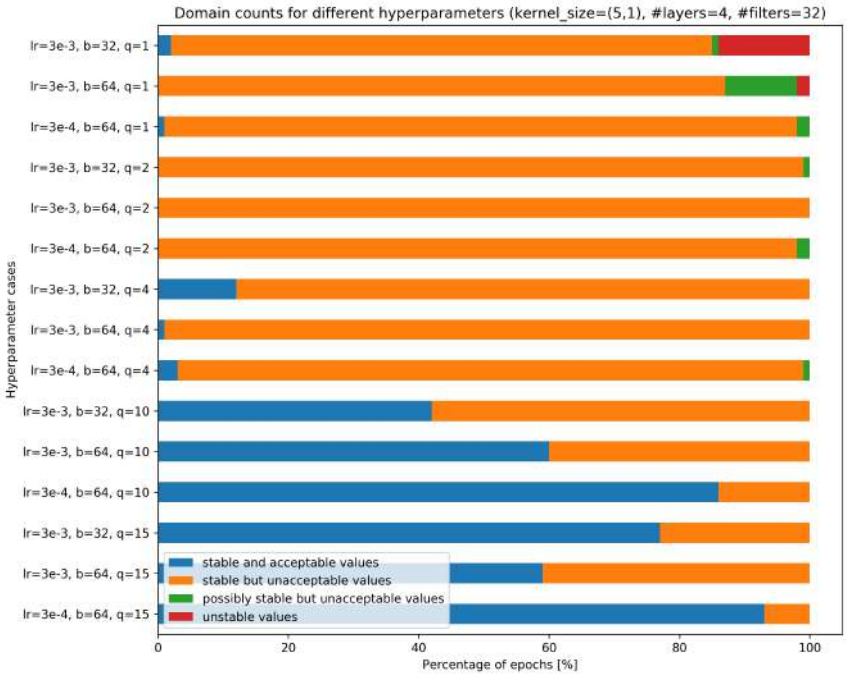


Figure 5.11: *Percentage of epochs when the trained LC hybrid solver would be stable. Blue regions represent epochs during which the solver is stable and at least as accurate as the pure numerical solver on the coarse grid. Orange regions represent stable solvers with accuracy lower than that of the numerical solver on the coarse grid (MAE from 0.03 to 1). Green regions roughly represent possibly stable solvers with unacceptable values (MAE from 1 to 100).These intermediate values do occur in the LC model training. Red regions represent unstable values (MAE from* 100 *to* ∞*). Better results occur for bigger Q-steps values.*

.

Choosing between LI and LC methods is not a straightforward task. Some of the properties of each method have been showcased but the manner and case in which they are implemented is also important. For instance, if the corrections of the LC method were applied on a first-order scheme then its behavior would also be anti-diffusive, leading to a possible convergence with some of the characteristics of the LI method. Generally, the selection between these models should be treated with an experimenting and case-specific mindset.

Chapter 6

Conclusions

This thesis is concerned with the application of two acclaimed hybrid CFD-DL techniques to accelerate accurate solutions of unsteady PDE. The first method involved producing space and time dependent coefficients with a NN for enhancing FVM, and the second one used NNs as on-line correctors to coarse numerical solvers. Both of the techniques were applied to the problems of 1D advection equation and 1D linear acoustics by developing original code using TensorFlow 2.x and Keras. In the pursuit of accumulating technical know-how about such hybrid methods via their implementations, some important conclusions were drawn. Firstly, it is clear that the complexity of the required NNs that are embedded inside the numerical solvers does not scale unfavorably with the increasing intricacies of the problem. This suggests the ability of these techniques to scale to harder problems. Additionally, a range of tests, including a parametric study for the performance of these models, have been conducted to corroborate the findings and behaviors reported in the original research. These tests served to verify the models' robustness, assessing their generalizability in different scenarios with a relatively small amount of training data.

The gains of such hybrid solvers can be summarized as so:

Gains relative to numerical counterparts

- Simulation acceleration: Hybrid models, utilizing NNs, can significantly reduce the computation time, offering faster simulations. Generally, in both end-to-end NNs and in hybrid methods more arithmetic operations are needed at the same resolution than in a purely numerical scheme. However, the nature of these operations in NNs constitutes them extremely parellelizable, which enables them to tap into the full capabilities of modern hardware such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs). The speed-up -in getting solutions of the same quality- provided by the two pre-

sented methods relative to a numerical solver can be approximated as follows:

$$\textit{speed-up} \approx \frac{CR^N}{RoS}$$

where CR is the coarsening ratio, N is the number of spatial dimensions of the PDE and RoS is the Rate of Slowdown of hybrid solvers realtive to pure numerical solvers on the same grid resolution (due to the increased arithmetic operations).

In the literature [54, 59, 23] it has been shown that for 2D flow problems, with the current hardware, the hybrid solvers can be up to 80× faster than their traditional numerical counterparts for attaining solutions of the same quality.

Gains relative to end-to-end NNs

- Generalization: Hybrid models usually exhibit better generalization to unseen data [54, 23].
- Stability: Hybrid methods provide stable unsteady solvers across a variety of conditions.
- Interpretability: Compared to NN models which are essentially black-boxes, hybrid models offer interpretability, as they utilize well-established and well-tested numerical algorithms to handle big part of the solution dynamics. This significantly increases their reliability and credibility.

There are also several challenges in hybrid methods, especially when they are expected to handle real-world, scaled problems. These also represent ideas for future work on the subject.

Challenges

- Extrapolation: The challenge of ensuring models extrapolate well beyond their training distribution. This is an inherent property of pure numerical models. More progress in this area seems possible for hybrid solvers and would be game changing.
- Data representativeness: Achieving extrapolation capabilities may not be easy, so the well-known principle of ML can come into play: with enough data, you always interpolate. However, building large datasets that effectively represent the sample space of initial conditions is also a challenge that can perhaps be addressed using more advanced statistical techniques.
- Scaling up: Efficiently scaling these techniques to handle larger and more complex simulations. These methods have been demonstrated for difficult prob-

lems (e.g. LC to 3D wake flow in [54]), but they must be further generalized (e.g. to unstructured grids or to other important equations in Computational Mechanics).

- Integration of NN architectural advances: CNNs have been shown to be a very effective machine for physical problems because of their translation invariance [28], but there have been impactfull architectural breakthroughs (like Transformers) that could be tested, especially for scaled problems where much data may be available.

- Effective coarsening: Developing strategies for coarsening (e.g. using Autoencoders) that better maps the fine grid fields to the coarse grid would also be significant, as the maximum solution quality that hybrid solvers can achieve is dependent on the quality of coarsening.

- Integration with existing codebases: Hybrid methods are usually implemented in Python-based frameworks because of their implicit need for AD (e.g. TF-Keras, Pytorch) and are "invasive" (especially the LI method). Their integration within the existent industry CFD codebases of Fortran and C++ remains a challenge. TensorFlow C++ API and projects like Enzyme (LLVM that takes arbitrary existing code and computes its derivative or gradient) may be adequate to deal with these problems. However, if the surge in AI research continues to bring significant enhancements to traditional CFD algorithms, the porting of such codes to other emerging programming languages like Julia or Swift (or even Python coupled with JAX) that have inherent AD capabilities while also being very fast, is also conceivable.

On a closing note, in light of the significant advancements in ML, it would perhaps be short-sighted to dismiss the potential dominance of end-to-end NNs, which do not incorporate physics constraints, in simulations in the long term. Research has demonstrated that NNs can learn statistical rules more efficiently than those explicitly provided, leading to superhuman performance in various benchmarks. Computer vision is a prominent example of this. In fluid dynamics, the possibility exists that NNs might learn the physics rules embedded in data in a more compact or generalizable manner than the one provided through physics constraints.

However, the practicality of such end-to-end methods remains uncertain, particularly concerning their economic viability. The increased costs associated with training and inference of bigger NNs which also need more training data, plus the redundancy of forcing the model to acquire knowledge that can be swiftly encoded are tangible drawbacks of end-to-end NNs. It's also crucial to recognize that the mathematical equations representing the underlying physics of fluid dynamics are among the most concise and accurate models humanity has developed. The challenge of surpassing these equations or their numerical counterparts in terms of generalizability should not be understimated.

Hybrid methods, as explored in this thesis, present a promising compromise. They

blend the reliability and generalizability of numerical methods with the computational speed of NNs, harnessing the strengths of both approaches. Current research trends suggest that these hybrid methodologies are likely to be predominant in the short to medium term, offering a balanced solution that leverages the advantages of both traditional numerical methods and the latest developments in neural networks.

Finally, it is essential to recognize the profound impact that emerging hardware technologies like TPUs and specialized GPUs are bound to have on the field of ML, and consequently, on the intersection of scientific computing and ML. The synergy between these hardware advancements and the ongoing surge in AI and statistical algorithm research [37] is expected to create an environment highly fertile to further integrating ML into CFD. This integration will likely lead to more sophisticated, efficient, and accurate CFD models.

Bibliography

[1] Anderson, J.D.J.: Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill, New York, NY, USA (1995)

[2] Bar-Sinai, Y., Hoyer, S., Hickey, J., Brenner, M.: Learning data-driven discretizations for partial differential equations. Proceedings of the National Academy of Sciences **116**(31), 15344–15349 (2019). https://doi.org/10.1073/pnas.1814058116, https://doi.org/10.1073/pnas.1814058116, edited by John B. Bell, Lawrence Berkeley National Laboratory, Berkeley, CA, approved June 21, 2019 (received for review August 14, 2018)

[3] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks **5**(2), 157–166 (1994). https://doi.org/10.1109/72.279181

[4] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 24. Curran Associates, Inc. (2011), https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[5] Bishop, C.: Pattern Recognition and Machine Learning. Springer Science+Business Media, LLC, New York, NY, USA (2006), http://research.microsoft.com/~cmbishop, library of Congress Control Number: 2006922522

[6] Brenowitz, N.D., Bretherton, C.S.: Prognostic validation of a neural network unified physics parameterization. Geophysical Research Letters **45**, 6289–6298 (2018). https://doi.org/10.1029/2018GL078510, https://doi.org/10.1029/2018GL078510, first published: 13 June 2018, Citations: 175

[7] Brunton, S., Noack, B.R., P.Koumoutsakos: Machine learning for fluid mechanics. Annual Review of Fluid Mechanics **52**, 477–508 (2020). https://doi.org/10.1146/annurev-fluid-010719-060214, https://doi.org/10.1146/annurev-fluid-010719-060214, first published as a Review in Advance on September 12, 2019

[8] Chen, W., Chiu, K., Fuge, M.: Aerodynamic design optimization and shape exploration using generative adversarial networks. In: Proceedings of the [Conference Name]. University of Maryland, College Park, Maryland (2019). https://doi.org/10.2514/6.2019-2351, https://www.researchgate.net/publication/330197128, the paper was presented in January 2019 and uploaded by Wei Wayen Chen on 20 March 2019

[9] Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). In: International Conference on Learning Representations (ICLR) 2016. Johannes Kepler University, Linz, Austria (2016)

[10] Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274, https://link.springer.com/article/10.1007/BF02551274

[11] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc. (2015), https://proceedings.neurips.cc/paper_files/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf

[12] Giotis, A., Giannakoglou, K., Periaux, J.: A reduced-cost multi-objective optimization method based on the pareto front technique, neural networks and pvm. In: European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2000). Barcelona, Spain (Sept 2000)

[13] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 9, pp. 249–256. PMLR, Chia Laguna Resort, Sardinia, Italy (13–15 May 2010), https://proceedings.mlr.press/v9/glorot10a.html

[14] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011). vol. 15. JMLR: Workshop and Conference Proceedings, Fort Lauderdale, FL, USA (2011), http://proceedings.mlr.press/v15/glorot11a.html, appeared in Volume 15 of JMLR: Workshop and Conference Proceedings. Copyright 2011 by the authors.

[15] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org

[16] Haley, P., Soloway, D.: Extrapolation limitations of multilayer feedforward neural networks. In: [Proceedings 1992] IJCNN International

Joint Conference on Neural Networks. vol. 4, pp. 25–30 vol.4 (1992). https://doi.org/10.1109/IJCNN.1992.227294

[17] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385

[18] Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2023), https://arxiv.org/abs/1606.08415v5, work done while the author was at TTIC. Code available at github.com/hendrycks/GELUs

[19] Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Improving neural networks by preventing co-adaptation of feature detectors. CoRR **abs/1207.0580** (2012), http://arxiv.org/abs/1207.0580

[20] Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks **4**, 251–257 (1991), received 30 January 1990; revised and accepted 25 October 1990

[21] Hornik, K., M.Tinchcombe, White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**, 359–366 (1989), printed in the USA. All rights reserved.

[22] Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V.B., Tebbutt, W.: A differentiable programming system to bridge machine learning and scientific computing. CoRR **abs/1907.07587** (2019), http://arxiv.org/abs/1907.07587

[23] Kochkov, D., Smith, J., Alieva, A., Wang, Q., Brenner, M., Hoyer, S.: Machine learning–accelerated computational fluid dynamics. Proceedings of the National Academy of Sciences **118**(21) (may 2021). https://doi.org/10.1073/pnas.2101784118, http://dx.doi.org/10.1073/pnas.2101784118

[24] Kontou, M., Kapsoulis, D., Baklagis, I., Trompoukis, X., Giannakoglou, K.: λ-DNNs and their implementation in conjugate heat transfer shape optimization. Neural Computing and Applications **34**, 843–854 (2022)

[25] Kontou, M., Asouti, V., Giannakoglou, K.: DNN surrogates for turbulence closure in CFD-based shape optimization. Applied Soft Computing **143**, 110013 (2023)

[26] Kovani, K., Kontou, M., Asouti, V., Giannakoglou, K.: DNN-driven gradient-based shape optimization in fluid mechanics. In: International Conference on Engineering Applications of Neural Networks (EANN 2023). Leon, Spain (2023)

[27] L., S.: Turbulence and the dynamics of coherent structures part i: Coherent structures. Quarterly of Applied Mathematics **45**(3), 561–571 (Oct 1987)

[28] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (Nov 1998). https://doi.org/10.1109/5.726791

[29] LeVeque, R.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge, United Kingdom (2002), http://www.cambridge.org, first published in printed format

[30] Ling, J., Kurzawski, A., Templeton, J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fluid Mechanics **807**, 155–166 (2016). https://doi.org/10.1017/jfm.2016.615, c Cambridge University Press 2016. This is a work of the U.S. Government and is not subject to copyright protection in the United States. Received 2 August 2016; revised 14 September 2016; accepted 16 September 2016; first published online 18 October 2016

[31] Linnainmaa, S.: Taylor expansion of the accumulated rounding error. BIT Numerical Mathematics **16**, 146–160 (1976), https://api.semanticscholar.org/CorpusID:122357351

[32] Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Mathematical Programming **45**, 503–528 (1989). https://doi.org/10.1007/BF01589116, https://doi.org/10.1007/BF01589116, issue Date: August 1989

[33] Lumley, J.L.: The Structure of Inhomogeneous Turbulent Flows, pp. 166–177 (1967)

[34] M., M., S., P.: Review of "Perceptrons: An Introduction to Computational Geometry". IEEE Transactions on Information Theory **15**(6), 738–739 (Dec 1969). https://doi.org/10.1109/TIT.1969.1054388, https://ieeexplore.ieee.org/document/1054388, iEEE Xplore

[35] Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of the 30th International Conference on Machine Learning (ICML 2013). vol. 28. JMLR: Workshop and Conference Proceedings, Atlanta, Georgia, USA (2013), http://proceedings.mlr.press/v28/maas13.html, copyright 2013 by the author(s)

[36] Martens, J., Sutskever, I.: Training Deep and Recurrent Networks with Hessian-Free Optimization, pp. 479–535. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[37] Maslej, N., Fattorini, L., Brynjolfsson, E., Etchemendy, J., Ligett, K., Lyons, T., Manyika, J., Ngo, H., Niebles, J.C.and Parli, V., Shoham, Y., Wald, R., Clark, J., Perrault, R.: The AI index 2023 annual report. Tech. rep., AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA (April 2023)

[38] Montavon, G., Orr, G., Müller, K.R. (eds.): Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science, vol. 7700. Springer, Heidelberg (2012), https://www.springer.com/gp/book/9783642352881, second Edition

[39] Morgan, N., Bourlard, H.: Generalization and parameter estimation in feedforward nets: Some experiments. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems. vol. 2. Morgan-Kaufmann (1989), https://proceedings.neurips.cc/paper_files/paper/1989/file/63923f49e5241343aa7acb6a06a751e7-Paper.pdf

[40] Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., Sutskever, I.: Deep double descent: Where bigger models and more data hurt. CoRR **abs/1912.02292** (2019), http://arxiv.org/abs/1912.02292

[41] N.Thuerey, Weissenow, K., H.M., Mainali, N., Prantl, L., Hu, X.: Well, how accurate is it? A study of deep learning methods for Reynolds-Averaged Navier-Stokes Simulations. CoRR **abs/1810.08217** (2018), http://arxiv.org/abs/1810.08217

[42] Pawar, S., San, O., Aksoylu, B., Rasheed, A., Kvamsdal, T.: Physics guided machine learning using simplified theories. CoRR **abs/2012.13343** (2020), https://arxiv.org/abs/2012.13343

[43] Pestourie, R., Mroueh, Y., Rackauckas, C., et al.: Physics-enhanced deep surrogates for partial differential equations. Nat Mach Intell **5**, 1458–1465 (2023). https://doi.org/10.1038/s42256-023-00761-y, https://doi.org/10.1038/s42256-023-00761-y

[44] Rabault, J., M.Kuchta, A.Jensen, Réglade, U., Cerardi, N.: Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. Journal of Fluid Mechanics **865**, 281–302 (Feb 2019). https://doi.org/10.1017/jfm.2019.62, http://dx.doi.org/10.1017/jfm.2019.62

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

[46] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review **65**(6), 386–408 (1958). https://doi.org/10.1037/h0042519, https://doi.org/10.1037/h0042519

[47] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986).

https://doi.org/10.1038/323533a0, https://www.nature.com/articles/323533a0, published 09 October 1986

[48] S., W., K., E., P., G.: Principal component analysis. Chemometrics and Intelligent Laboratory Systems **2**, 37–52 (1987), printed in The Netherlands

[49] S.A. Kyriacou, V.A., Giannakoglou, K.: Efficient PCA-driven EAs and metamodel-assisted EAs, with applications in turbomachinery. Engineering Optimization **46**(7), 895–911 (2014). https://doi.org/10.1080/0305215X.2013.812726, https://doi.org/10.1080/0305215X.2013.812726

[50] Shen, Z., Yang, H., Zhang, S.: Optimal approximation rate of relu networks in terms of width and depth. Journal de Mathématiques Pures et Appliquées **157**, 101–135 (2022), https://www.elsevier.com/locate/matpur, received 18 February 2021; Available online 16 July 2021

[51] Springenberg, J., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. In: International Conference on Learning Representations (ICLR) 2015, Workshop Track. University of Freiburg, Freiburg, 79110, Germany (2015), https://arxiv.org/abs/1412.6806v3, arXiv:1412.6806v3 [cs.LG] 13 Apr 2015

[52] Μπεργελές Γ.: Υπολογιστική Ρευστομηχανική. ΣΥΜΕΩΝ (2012)

[53] Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) **58**(1), 267–288 (1996), https://www.jstor.org/stable/2346178

[54] Um, K., Brand, R., Y.R.Fei, P.Holl, N.Thuerey: Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In: Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Vancouver, Canada (2020), https://arxiv.org/abs/2007.00016v2, arXiv:2007.00016v2 [physics.comp-ph] 5 Jan 2021

[55] Wang, Y., Zhang, H., Zhang, G.: cpso-cnn: An efficient pso-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. Swarm and Evolutionary Computation **49**, 114–123 (2019). https://doi.org/https://doi.org/10.1016/j.swevo.2019.06.002, https://www.sciencedirect.com/science/article/pii/S2210650218310083

[56] Wengert, R.E.: A simple automatic derivative evaluation program. Communications of the ACM **7**(8), 463–464 (Aug 1964), received February, 1964

[57] Xiao, X., Yan, M., Basodi, S., Ji, C., Pan, Y.: Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm (2020)

[58] Xu, K., Zhang, M., Li, J., Du, S., Kawarabayashi, K., Jegelka, S.: How neural networks extrapolate: From feedforward to graph neural networks. In: Inter-

national Conference on Learning Representations (ICLR 2021). Massachusetts Institute of Technology (MIT), University of Maryland, University of Washington, National Institute of Informatics (NII) (2021)

[59] Zhuang, J., Kochkov, D., Bar-Sinai, Y., Brenner, M., Hoyer, S.: Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. Physical Review Fluids **6**(064605) (2021). https://doi.org/10.1103/PhysRevFluids.6.064605, received 12 April 2020; accepted 19 April 2021; published 14 June 2021

[60] Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society. Series B (Statistical Methodology) **67**(Part 2), 301–320 (2005). https://doi.org/10.1111/j.1467-9868.2010.00771.x

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Εργαστήριο Θερμικών Στροβιλομηχανών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Υβριδικοί Επιλύτες Υπολογιστικής Ρευστοδυναμικής-Βαθιάς Μάθησης για Χρονικά Μη-Μόνιμα Προβλήματα

Διπλωματική Εργασία - Εκτενής περίληψη

Αντώνης Τζανετάκης

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Εισαγωγή

Τις τελευταίες δεκαετίες, το πεδίο της Υπολογιστικής Ρευστοδυναμικής (ΥΡΔ) έχει συνεισφέρει σημαντικά στην τεχνολογική πρόοδο. Η ευημερία του πεδίου οφείλεται σε δύο καταλυτικούς παράγοντες: την εξέλιξη των υπολογιστών και της ισχύος τους και τη βελτίωση των αριθμητικών αλγορίθμων [10]. Στις μέρες μας, το ίδιο μοτίβο εμφανίζεται για την Τεχνητή Νοημοσύνη (ΤΝ): η εκθετική αύξηση σε διαθέσιμους υπολογιστικούς πόρους και δεδομένα και η πρόοδος των στατιστικών τεχνικών έχουν προκαλέσει την ωρίμανση του πεδίου και την εφαρμογή του σε ποικίλους τομείς της ανθρώπινης δραστηριότητας, συμπεριλαμβανομένου της μηχανικής. Η ΥΡΔ διαθέτει πλήθος ποιοτικών και καλά οργανωμένων δεδομένων, εύρωστους υπολογιστικούς αλγορίθμους και πρόσβαση σε σημαντικούς υπολογιστικούς πόρους. Η συνέργεια αυτών των παραγόντων με την άνθηση της ΤΝ οδηγεί σε αξιόλογους συνδυασμούς της ΥΡΔ και της ΤΝ [4, 12].

Η διπλωματική εργασία εξερευνά το πεδίο υποβοήθησης της ΥΡΔ από τεχνικές Βαθιάς Μάθησης (ΒΜ). Προγραμματίζονται και αξιολογούνται δύο υβριδικές προσεγγίσεις ΥΡΔ-ΒΜ [2, 11], που επιταχύνουν την ακριβή επίλυση χρονικά μη-μόνιμων προβλημάτων. Οι μέθοδοι αυτές στοχεύουν στην αντιμετώπιση του σφάλματος διακριτοποίησης, που εξαρτάται κυρίως απ΄το βήμα του πλέγματος [1], και είναι σημαντική πηγή σφάλματος στις αριθμητικές προσομοιώσεις. Χρησιμοποιώντας Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) εκπαιδευμένα σε δεδομένα υψηλής ανάλυσης, αυτοί οι υβριδικοί επιλύτες ενώ λειτουργούν σε ένα αραιό πλέγμα, επιτυγχάνουν λύσεις που αντιστοιχούν σε πολύ πυκνότερα πλέγματα. Και οι δύο μέθοδοι εφαρμόζονται για να λύσουν δύο 1Δ χρονικά μη-μόνιμα προβλήματα: την εξίσωση μεταφοράς και την εξίσωση γραμμικής ακουστικής.

Τεχνητά Νευρωνικά Δίκτυα

Τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) είναι πολύπλοκες και ευέλικτες συναρτήσεις προσαρμογής [5]. Κατά τη διαδικασία της εκπαίδευσης, τα βάρη του ΝΔ ανανεώνονται μέσω κάποιας διαδικασίας βελτιστοποίησης (συνήθως μέσω της μεθόδου της απότομης κλίσης) μέχρις ότου να ελαχιστοποιηθεί μία συνάρτηση απωλειών. Αυτή η συνάρτηση απωλειών μετρά το σφάλμα μεταξύ της πρόβλεψης του ΤΝΔ και της πραγματικής τιμής στα δεδομένα εκπαίδευσης.

Η μαθηματική τους περιγραφή μπορεί να συνοψισθεί ως μία σύνθεση γραμμικών μετασχηματισμών ανάμεσα στους οποίους παρεμβάλλονται μη-γραμμικές συναρτήσεις (‘‘συναρτήσεις ενεργοποίησης¨).

$$NN(x, W) = f_N \circ f_{N-1} \circ ... \circ f_1(x, W) \tag{1}$$

όπου x είναι το διάνυσμα εισόδου και W είναι το μητρώο των παραμέτρων (βάρη και πολώσεις) του ΝΔ, και θεωρώντας ίδιες συναρτήσεις ενεργοποίησης για κάθε στρώση, μπορεί να γραφτεί:

$$f_n(x, W_n) = \sigma(W_n x + b_n) \tag{2}$$

που περιγράφει την πράξη μέσα σε κάθε στρώση, όπου W_n είναι το μητρώο βαρών για την ν-οστή στρώση, b_n είναι η πόλωση για την ν-οστή στρώση και σ είναι η συνάρτηση ενεργοποίησης.

Στην διπλωματική εργασία, χρησιμοποιείται μία υποκατηγορία των ΤΝΔ που ονομάζεται Συνελικτικά Νευρωνικά Δίκτυα (ΣΝΔ). Τα ΣΝΔ χρησιμοποιούν πράξεις συνέλιξης αντί για πολλαπλασιασμούς πινάκων και είναι η πιο συνηθισμένη επιλογή σε προβλήματα χωροχρονικών δεδομένων και άρα και σε πλεγματικές τοπολογίες. Ο κύριος λόγος που τα καθιστά κατάλληλα για αυτά τα προβλήματα είναι ότι από κατασκευής τους είναι μεταφορικά αδιάφορα [6], δηλαδή όταν ένα χαρακτηριστικό αναγνωρισθεί/μαθευτεί σε μία τοποθεσία, αναγνωρίζεται ως το ίδιο χαρακτηριστικό και σε οποιαδήποτε άλλη τοποθεσία. Αυτή τους η ιδιότητα εξασφαλίζει εξοικονόμηση σε απαραίτητα δεδομένα και σε χρόνο εκπαίδευσης σε σχέση με τις γενικές Πλήρως Συνδεδεμένες (ΠΣ) αρχιτεκτονικές, για τα ειδικά αυτά προβλήματα.

Υβριδικοί Επιλύτες

Και οι δύο εξεταζόμενες τεχνικές χρησιμοποιούνται για την επίλυση διακριτοποιημένων χρονικά μη-μόνιμων Μερικών Διαφορικών Εξισώσεων (ΜΔΕ). Η μέθοδος Παραγωγής Συντελεστών (ΠΣ) αντικαθιστά το κλασικό βήμα ανακατασκευής της μεθόδου των Πεπερασμένων Όγκων (ΠΟ) (βλ. [7]) με μια διαδικασία που βασίζεται σε ΤΝΔ. Συγκεκριμένα, το ΤΝΔ εκπαιδεύεται να παράγει χωροχρονικά μεταβαλλόμενους συντελεστές διακριτοποίησης, που σε κάθε χρονική στιγμή, συνδυάζονται με τις κομβικές τιμές του πεδίου σε ένα αραιό πλέγμα προσεγγίζοντας τις χωρικές παραγώγους της διακριτοποιημένης ΜΔΕ στο κέντρο των κελιών. Αυτές χρησιμοποιούνται για τον υπολογισμό των fluxes στα όρια των κελιών (βήμα ανακατασκευής), και εκείνα μέσω ΠΟ με κάποιο σχήμα χρονικής ολοκλήρωσης βρίσκουν τα πεδία ροής. Τελικά, επιτυγχάνονται αποτελέσματα που αντιστοιχούν σε λύσεις πολύ πυκνότερων πλεγμάτων σε σχέση με εκείνο στο οποίο λειτουργεί το αριθμητικό μέρος του επιλύτη.

Στη μέθοδο Παραγωγής Διορθώσεων (ΠΔ), επιλύεται με έναν συμβατικό αριθμητικό επιλύτη η διακριτοποιημένη ΜΔΕ σε ένα αραιό πλέγμα, παράγοντας λύσεις χαμηλής ανάλυσης. Σε κάθε χρονικό βήμα, σε αυτές προστίθεται μία διόρθωση (διαφορετική για κάθε κόμβο του αραιού πλέγματος) από ένα εκπαιδευμένο ΤΝΔ. Έτσι, το σφάλμα της αριθμητικής λύσης διορθώνεται σταδιακά χωρίς να προλάβει να γιγαντωθεί. Με αυτόν τον τρόπο παράγονται αποτελέσματα υψηλής ανάλυσης, αντίστοιχα ενός πολύ πυκνότερου πλέγματος σε σχέση με εκείνο στο οποίο λειτουργεί το αριθμητικό μέρος του επιλύτη.

Ο ορισμός και οι τιμές των βασικών παραμέτρων για τα δύο προβλήματα που θα παρουσιαστούν περιλαμβάνονται στον Πιν.1. Ο αριθμός σημείων του πλέγματος θα δίνεται από τη σχέση $\lambda_1 \times 2^{\lambda_2}$, το οποίο είναι βοηθητικό για τον πειραματισμό με διάφορους συντελεστές αραίωσης. Οι υβριδικοί επιλύτες που παρουσιάζονται (βλ. και [2, 11])

συμπεριλαμβάνουν τα ακόλουθα στάδια.

Παράμετρος	**Περιγραφή**	**Τιμή**
K	αρ. διαφορετικών αρχικών συνθηκών	30 για το προβλ. 1 και 10 για το προβλ. 2
CR	λόγος πύκνωσης $= 2^{\beta}, \beta \in \mathbb{N}^* \backslash \beta < \lambda_2$	8
λ_1	παράγων του αρ. των πλεγματικών σημείων	3
λ_2	εκθέτης του αρ. των πλεγματικών σημείων	7
M_f	αρ. σημείων στο πυκνό πλέγμα $= \lambda_1 \times 2^{\lambda_2}$	384
M_c	αρ. σημείων στο αραιό πλέγμα $= \frac{M_f}{CR}$	48
N_f	αρ. χρονικών βημάτων στο αραιό πλέγμα	1536
N_c	αρ. χρονικών βημάτων στο πυκνό πλέγμα	192
S	μέγεθος στένσιλ	3-σημείων

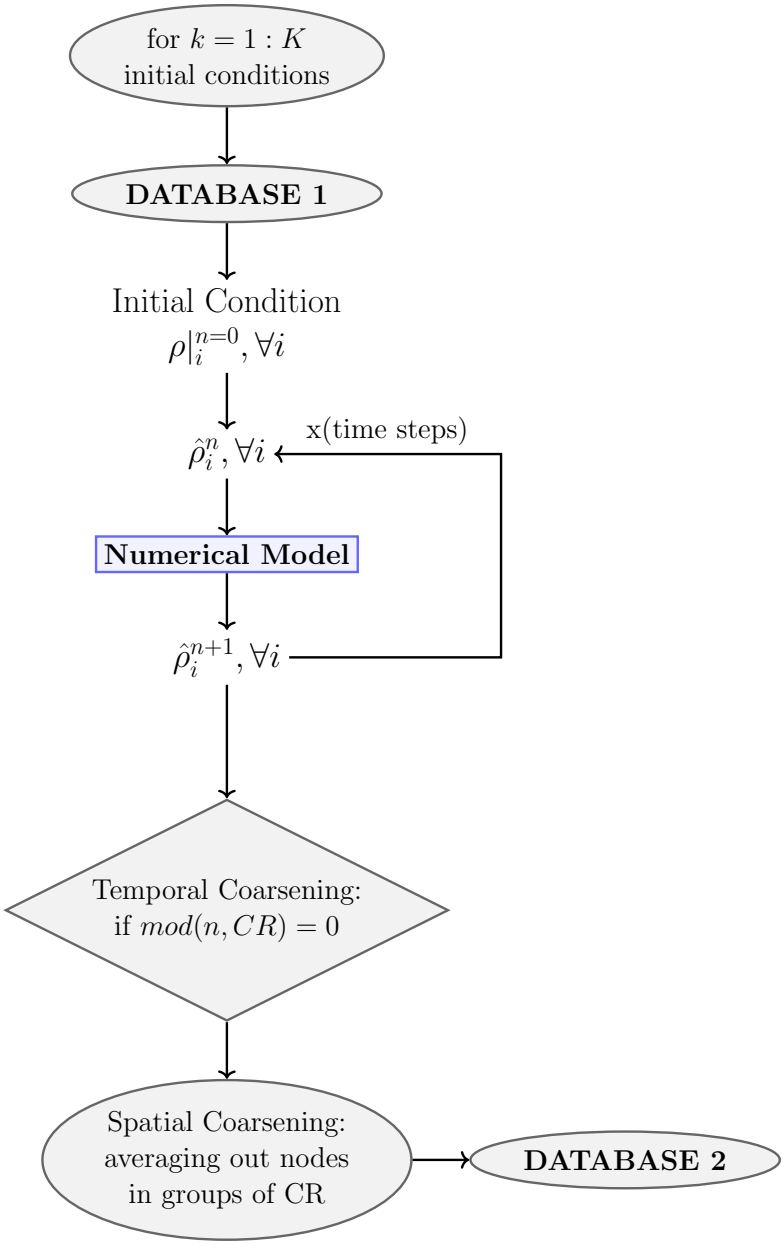
Πίνακας 1: *Πίνακας με σημαντικές παραμέτρους, τις περιγραφές τους και τις τιμές που παίρνουν για τα δύο προβλήματα. Το μόνο που αλλάζει είναι οι αρχικές συνθήκες. Σημειώνεται ότι ο δείκτης f θα χρησιμοποιείται για παραμέτρους σχετικές με το πυκνό πλέγμα και ο δείκτης c για παραμέτρους σχετικές με το αραιό πλέγμα.*

Στάδιο 1ο: Για μία αρχική συνθήκη, ολοκλήρωση στον χρόνο της διακριτοποιημένης ΜΔΕ στο πυκνό πλέγμα (M_f σημεία), για N_f χρονικά βήματα μέχρι όλα τα χαρακτηριστικά της λύσης να έχουν εκδηλωθεί (λ.χ. να περιλαμβάνεται και η μεταβατική και η περιοδική μόνιμη κατάσταση ενός προβλήματος). Καθώς γίνεται η ολοκλήρωση, αποθηκεύονται, αραιωμένα χωροχρονικά σε M_c σημεία και N_c χρονικά βήματα, τα αποτελέσματα της ολοκλήρωσης. Η ίδια διαδικασία γίνεται -εν δυνάμει με παράλληλο τρόπο- για K αρχικές συνθήκες (π.χ. τετράγωνα κύματα διαφορετικού ύψους). Τα τελικά δεδομένα μεγέθους $K \times N_c \times M_c$ που αποθηκεύτηκαν θα χρησιμοποιηθούν σαν δεδομένα εκπαίδευσης. Το στάδιο αυτό περιγράφεται στο Σχ.1.

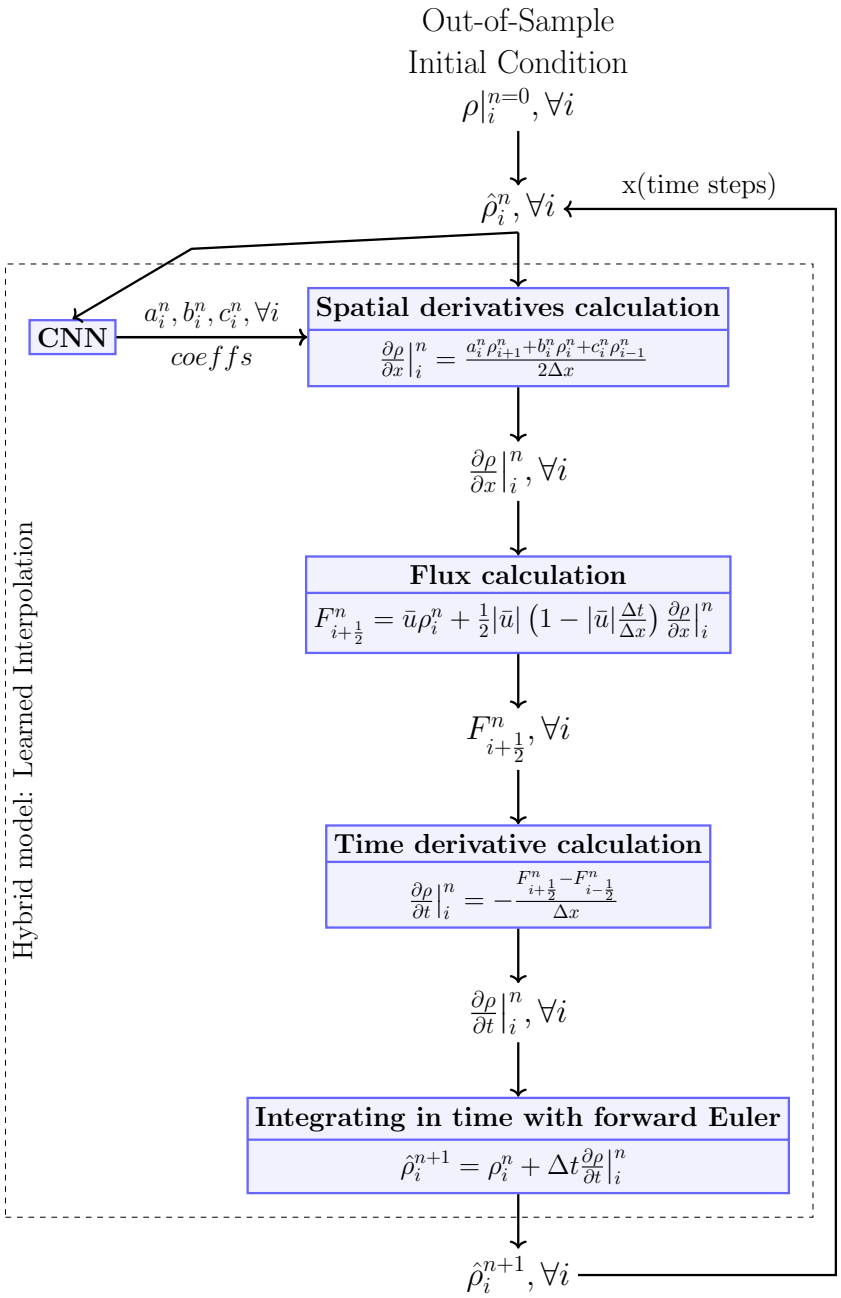
Στάδιο 2ο: Εκπαίδευση του ΣΝΔ στα αραιωμένα δεδομένα. Η έξοδος του ΣΝΔ είναι συντελεστές διακριτοποίησης για την μέθοδο ΠΣ ή διορθώσεις των πεδιακών τιμών για τη μέθοδο ΠΔ.

Στάδιο 3ο: Χρήση του εκπαιδευμένου ΣΝΔ στον υβριδικό επιλύτη για νέες αρχικές συνθήκες που δεν έχουν χρησιμοποιηθεί κατά την εκπαίδευση. Το ΣΝΔ θα παράξει είτε συντελεστές διακριτοποίησης (για την μέθοδο ΠΣ) είτε πεδιακές διορθώσεις (για τη μέθοδο ΠΔ) και τα αριθμητικά κομμάτια του επιλύτη θα χρησιμοποιήσουν είτε το

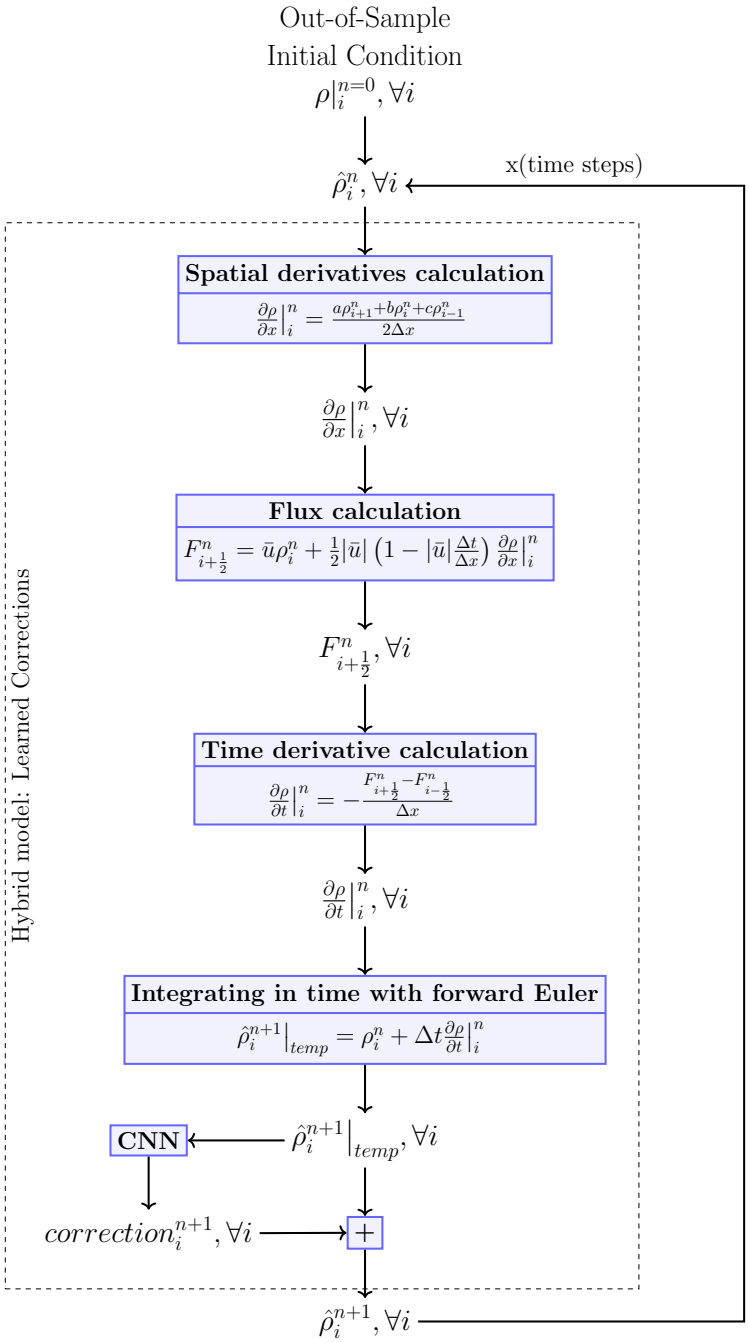
ένα είτε το άλλο αντίστοιχα για να παράξουν το πεδίο της ροής την επόμενη χρονική στιγμή.



Σχήμα 1: *Διάγραμμα ροής του 1ου σταδίου: Επιλύεται με το συμβατικό αριθμητικό επιλύτη (ΠΟ, 2ης τάξης ακρίβειας, πυκνό πλέγμα) η διακριτοποιημένη ΜΔΕ, γίνεται χωροχρονική αραίωση και συλλέγονται τα δεδομένα. Για την 1Δ εξίσωση μεταφοράς (όπου $CR = 8$), κάθε 8ο χρονικό βήμα (αντιστοιχεί σε χρονική πύκνωση, βλ. κόμβο με το εάν), το πεδίο της ροής αραιώνεται χωρικά και αποθηκεύεται. Η χωρική αραίωση γίνεται λαμβάνοντας τον μέσο όρο για κάθε ομάδα από $CR = 8$ κόμβους.*



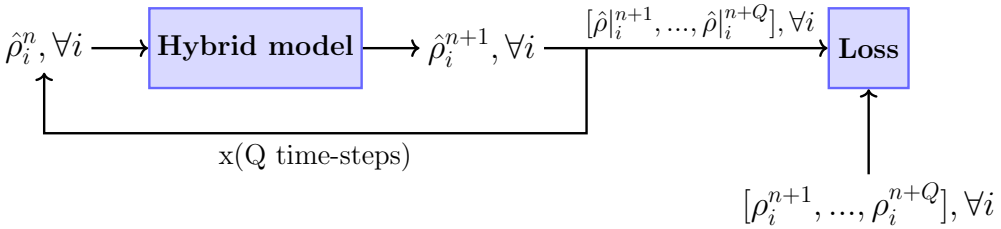
Σχήμα 2: *Αξιοποιήση του υβριδικού επιλύτη ΠΣ, αφού έχει γίνει η εκπαίδευση του ΣΝΔ. Μία υψηλής ανάλυσης αρχική συνθήκη που δεν εμπεριέχεται στα δεδομένα εκπαίδευσης αραιώνεται και τροφοδοτείται στον υβριδικό επιλύτη. Αυτός παράγει την λύση για ένα συγκεκριμένο χρονικό διάστημα με τον ακόλουθο τρόπο: Κάθε εκτιμώμενο πεδίο ροής σε μία χρονική στιγμή $\hat{\rho}_i^n, \forall i$ περνά τόσο μέσα από το ΣΝΔ που παράγει χωροχρονικά μεταβαλλόμενους συντελεστές όσο και από το αριθμητικό μοντέλο (ΠΟ, 2ης τάξης σχήμα, χωρίς περιοριστή, στένσιλ 3-σημείων) που χρησιμοποιεί αυτούς τους συντελεστές, για την προσέγγιση των χωρικών παραγώγων. Έτσι, προβλέπεται το πεδίο στο νέο χρονικό βήμα $\hat{\rho}_i^{n+1}, \forall i$. Αυτό επαναλαμβάνεται για όλα τα χρονικά βήματα.*



Σχήμα 3: *Αξιοποιήση του υβριδικού επιλύτη ΠΔ, αφού έχει γίνει η εκπαίδευση του ΣΝΔ. Μία υψηλής ανάλυσης αρχική συνθήκη που δεν εμπεριέχεται στα δεδομένα εκπαίδευσης αραιώνεται και τροφοδοτείται στον υβριδικό επιλύτη. Αυτός παράγει την λύση για ένα συγκεκριμένο χρονικό διάστημα με τον ακόλουθο τρόπο: Κάθε εκτιμώμενο πεδίο ροής σε μία χρονική στιγμή $\hat{\rho}_i^n, \forall i$ περνά μέσα από το αριθμητικό μοντέλο (ΠΟ, 2ης τάξης σχήμα, χωρίς περιοριστή, στένσιλ 3-σημείων με του κλασικούς συντελεστές Taylor) και προβλέπεται ένα προσωρινό πεδίο για την επόμενη χρονική στιγμή $\hat{\rho}_i^{n+1}|_{temp}, \forall i$. Τότε, αυτό το προσωρινό πεδίο περνά μέσα από ένα ΣΝΔ που γεννά χωροχρονικά μεταβαλλόμενες διορθώσεις $CORR_i^{n+1}, \forall i$ για το πεδίο αυτό. Προσθέτοντας το προσωρινό πεδίο και την διόρθωση, γίνεται η πρόβλεψη του (τελικού) επόμενου χρονικά πεδίου $\hat{\rho}_i^{n+1}, \forall i$. Αυτό επαναλαμβάνεται για όλα τα χρονικά βήματα.*

Πρόσθετα στοιχεία των Υβριδικών Επιλυτών

Και για τους δύο υβριδικούς επιλύτες, επιλέγεται ως συνάρτηση απωλειών μία πολυβηματική εκδοχή του κλασικού Μέσου Απόλυτου Σφάλματος [3]. Μια υψηλού επιπέδου όψη της διαδικασίας φαίνεται στο Σχ.4. Η χρήση πολλών βημάτων βοηθά τα μοντέλα να αναγνωρίσουν ότι η πρόβλεψη της επόμενης χρονικής στιγμής επηρεάζει και τις μελλοντικές τους προβλέψεις και είναι ένα από τα βασικά στοιχεία που καθιστά τους επιλύτες αυτούς ευσταθείς.



Σχήμα 4: *Τα συσσωρευμένα πεδία για Q στον αριθμό χρονικών στιγμών συγκρίνονται με τα δεδομένα εκπαίδευσης στη συνάρτηση απωλειών.*

Για τον υβριδικό επιλύτη ΠΣ, επιβάλλεται επίσης ελάχιστη τάξης πολυωνυμικής ακρίβειας στο προκύπτον σχήμα πεπερασμένων διαφορών που προσεγγίζει την χωρική παράγωγο και καθορίζεται από τους χωροχρονικά μεταβαλλόμενους συντελεστές που παράγει το ΣΝΔ. Οι δοκιμές της παρούσας εργασίας συμφωνούν με τα ευρήματα των [2, 13] ότι η βέλτιστη ελάχιστη ακρίβεια είναι η 1η τάξη ακρίβειας. Για την 1ης τάξης χωρική παράγωγο που χρησιμοποιείται στην παρούσα εργασία, αυτό σημαίνει πρακτικά ότι οι παραγόμενες ομάδες (τριάδες εν προκειμένω) συντελεστών από το ΣΝΔ πρέπει να αθροίζουν στο μηδέν. Η τήρηση αυτού του περιορισμού επιβάλλεται στην πράξη με έναν κατάλληλο γραμμικό μετασχηματισμό της εξόδου του ΣΝΔ.

Έγκαιρη διακοπή της διαδικασίας εκπαίδευσης

Ο τυπικός τρόπος να αποφευχθεί η υπερπροσαρμογή στα δεδομένα είναι με την έγκαιρη διακοπή της εκπαίδευσης με χρήση δεδομένων επικύρωσης [9]. Στις συγκεκριμένες μεθόδους αυτό δεν λειτουργεί λόγω της φύσης των μη-μόνιμων επιλυτών. Μία λύση σε αυτό μπορεί να είναι η χρήση του ίδιου του υβριδικού επιλύτη κατά τη διάρκεια της εκπαίδευσης, με έναν όσο το δυνατόν πιο οικονομικό τρόπο. Το κριτήριο που επιβάλλεται για την διακοπή της εκπαίδευσης είναι το εξής: σε ένα μικρό αλλά ικανό κλάσμα του χρονικού διαστήματος που επιθυμείται να δουλεύει καλά ο επιλύτης, πρέπει το Μέσο Απόλυτο Σφάλμα μεταξύ υβριδικής λύσης στο αραιό και αραιωμένης αριθμητικής λύσεις να μένει σταθερό στον χρόνο, άρα να μηδενίζεται σχεδόν εντελώς το σχετικό σφάλμα λόγω αριθμητικής διακριτοποίησης μεταξύ αραιού και πυκνού πλέγματος.

Δεδομένα προβλημάτων

Και για τα δύο προβλήματα που εξετάζονται, το χωρικό πεδίο ορίζεται ως $x \in [0, L]$ και το χρονικό πεδίο ως $t \in [0, t_{final}]$, με περιοδικές οριακές συνθήκες στον χώρο:

$$\rho(x=0,t) = \rho(x=L,t) \quad \& \quad \begin{cases} p(x=0,t) = p(x=L,t) \\ u(x=0,t) = u(x=L,t) \end{cases} \tag{3}$$

Οι αρχικές συνθήκες έχουν μορφή τετράγωνου κύματος κάποιου ύψους και πλάτους για την εξίσωση μεταφοράς:

$$\rho(x, t=0) = \begin{cases} height, & \text{αν } x_l < x < x_r \\ 0, & \text{αλλού} \end{cases} \tag{4}$$

και μορφή τετράγωνου κύματος κάποιου ύψους στην πίεση και μηδενική ταχύτητα για την εξίσωση ακουστικής:

$$\left\{ \begin{array}{c} p \\ u \end{array} \right\} (x, t=0) = \left\{ \begin{array}{ll} height_p, & \text{αν } x_l < x < x_r, \\ 0, & \text{αλλού} \end{array} \right\} \tag{5}$$

Τα χαρακτηριστικά των ΣΝΔ που εκπαιδεύτηκαν παρουσιάζονται στον Πιν.2.

Κατηγορία	**Πρόβλ. 1 (ΠΣ)**	**Πρόβλ. 1 (ΠΔ)**	**Πρόβλ. 2 (ΠΣ)**	**Πρόβλ. 2 (ΠΔ)**
αριθμός στρώσεων	4	4	5	5
αριθμός φίλτρων	32	32	64	64
μέγεθος πυρήνα συνέλιξης	3×1	3×1	5×1	5×1
μέγεθος του batch	64	64	64	64
βήμα εκμάθησης	3e-3, 3e-4	3e-3, 3e-4	3e-3	3e-3, 3e-4
εποχές	102	156	13	170
αρ. χρονικών βημάτων Q	4	4	10	10
βελτιστοποιητής	Adam			
αρχικοποίηση βαρών	Xavier			
συν. ενεργοποίησης	ReLU			

Πίνακας 2: *Τα χαρακτηριστικά των ΣΝΔ για κάθε μέθοδο και πρόβλημα. Η υπερπαράμετρος Q καθορίζει το πόσα χρονικά βήματα προβλέπει ο επιλύτης ανά τρέξιμο.*

1Δ εξίσωση μεταφοράς

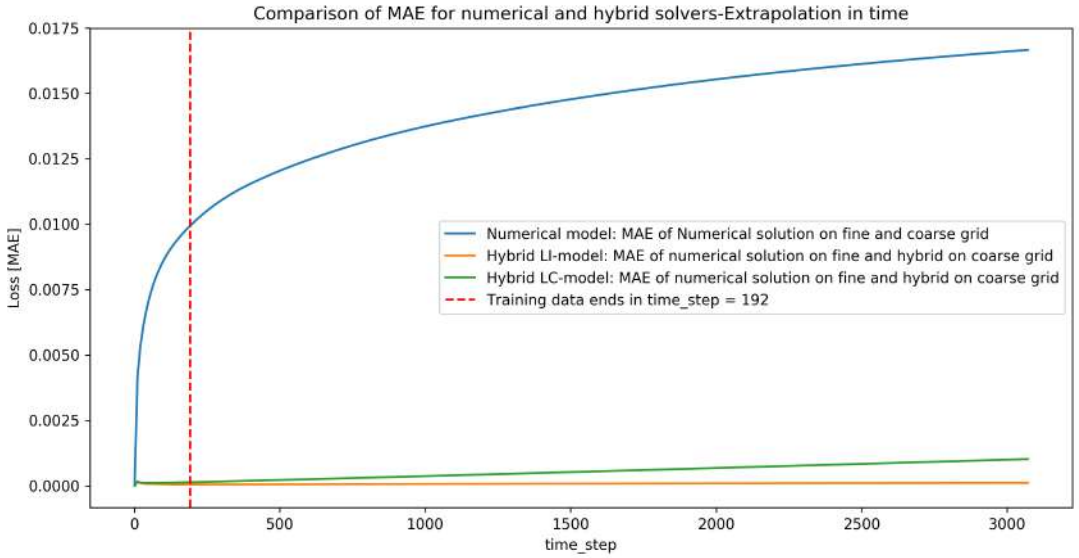
Η 1Δ εξίσωση μεταφοράς γράφεται:

$$\frac{\partial \rho}{\partial t} + \bar{u}\frac{\partial \rho}{\partial x} = 0, \tag{6}$$

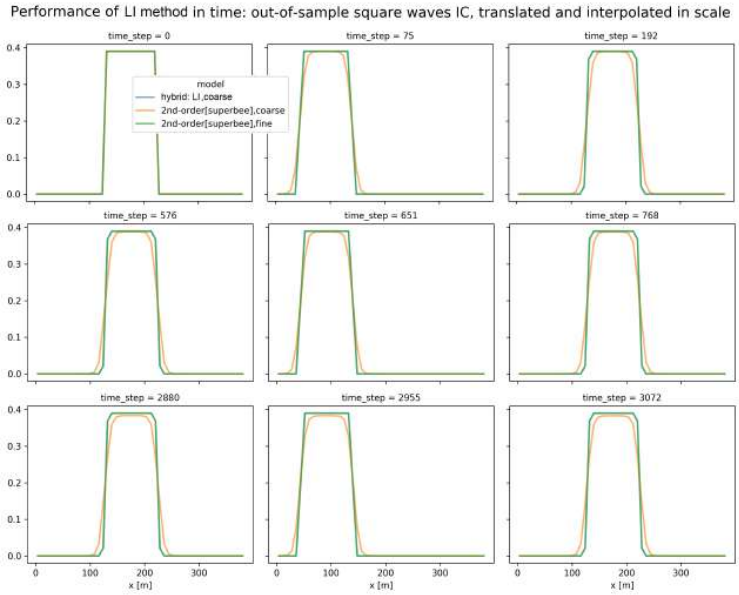
όπου η ταχύτητα μεταφοράς $\bar{u}$ θεωρείται σταθερή και θετική.

Οι παράμετροι του προβλήματος έχουν οριστεί στον Πιν.1.

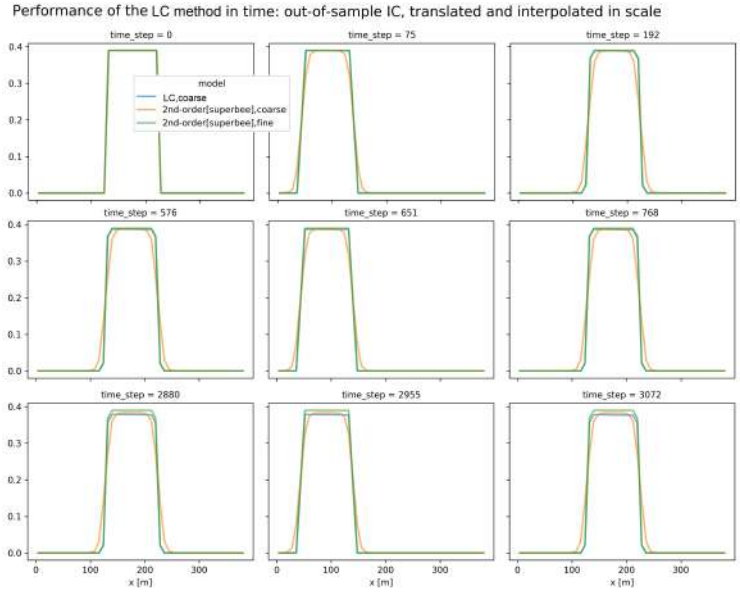
Σε αυτό το πρόβλημα, οι αρχικές συνθήκες είναι 30 τετράγωνα κύματα με διαφορετικά ύψη και πλάτη. Ζητείται από τους υβριδικούς επιλύτες να προβλέψουν την εξέλιξη για ένα τετράγωνο κύμα με διαφορετικό ύψος και πλάτος, μετατοπισμένο στο πλέγμα κατά την αρχική χρονική στιγμή. Η πρόβλεψη τους ελέγχεται σε μεγαλύτερο χρονικό διάστημα απ΄ όσο έχουν εκπαιδευθεί για την διασφάλιση της ευστάθειας τους.



Σχήμα 5: *Σύγκριση του αποκλειστικά αριθμητικού επιλύτη στο αραιό πλέγμα και των υβριδικών στο αραιό πλέγμα, βάσει του Μέσου Απόλυτου Σφάλματος (ΜΑΣ) τους σε σχέση με την αποκλειστικά αριθμητική λύση στο πυκνό πλέγμα -προβεβλημένη στο αραιό- για τετράγωνο κύμα ύψους που δεν έχει χρησιμοποιηθεί στην εκπαίδευση και με αρχική θέση μετατοπισμένη κατά 6 κόμβους στο πλέγμα. Η μέθοδος ΠΣ φαίνεται να είναι οριακά καλύτερη από τη μέθοδο ΠΔ ειδικά όταν γίνεται ολοκλήρωση στον χρόνο πέρα από το χρονικό κομμάτι που έχει χρησιμοποιηθεί στην εκπαίδευση.*



Σχήμα 6: *Σύγκριση ενός 2ης τάξης σχήματος με superbee περιοριστή σε αραιό πλέγμα 48-σημείων (πορτοκαλί), με τον υβριδικό επιλύτη ΠΣ σε αραιό πλέγμα 48-σημειών (μπλέ). Η σύγκριση γίνεται με βάση την αριθμητική επίλυση σε πυκνό πλέγμα 384-σημείων που προβάλλεται στο αραιό (πράσινο). Προεκβολή στον χρόνο κατά 16 φορές το χρονικού πεδίο που υπάρχει στα δεδομένων εκπαίδευσης.*



Σχήμα 7: *Σύγκριση ενός 2ης τάξης σχήματος με superbee περιοριστή σε αραιό πλέγμα 48-σημείων (πορτοκαλί), με τον υβριδικό επιλύτη ΠΔ σε αραιό πλέγμα 48-σημειών (μπλέ). Η σύγκριση γίνεται με βάση την αριθμητική επίλυση σε πυκνό πλέγμα 384-σημείων που προβάλλεται στο αραιό (πράσινο). Προεκβολή στον χρόνο κατά 16 φορές το χρονικού πεδίο που υπάρχει στα δεδομένων εκπαίδευσης.*

Τα αποτελέσματα μπορούν να συνοψισθούν στην εξής πρόταση: χρησιμοποιώντας ένα αραιό πλέγμα 48 σημείων, οι υβριδικοι επιλύτες επιτυγχάνουν λύσεις κοντινές σε αυτές που παράγονται από (καθαρά) αριθμητικούς επιλύτες σε πυκνό πλέγμα 384-σημείων που προβάλλεται στο αραιό. Στο Σχ.5 γίνεται εμφανές ότι η υβριδική λύση ακολουθεί πιστά την υψηλής ποιότητας λύση. Τα σχήματα αυτά στο σύνολο τους αποδεικνύουν τόσο την ποιότητα των λύσεων που επιτυγχάνονται όσο και την ευστάθεια του προκυπτόμενου επιλύτη.

1Δ εξίσωση γραμμικής ακουστικής

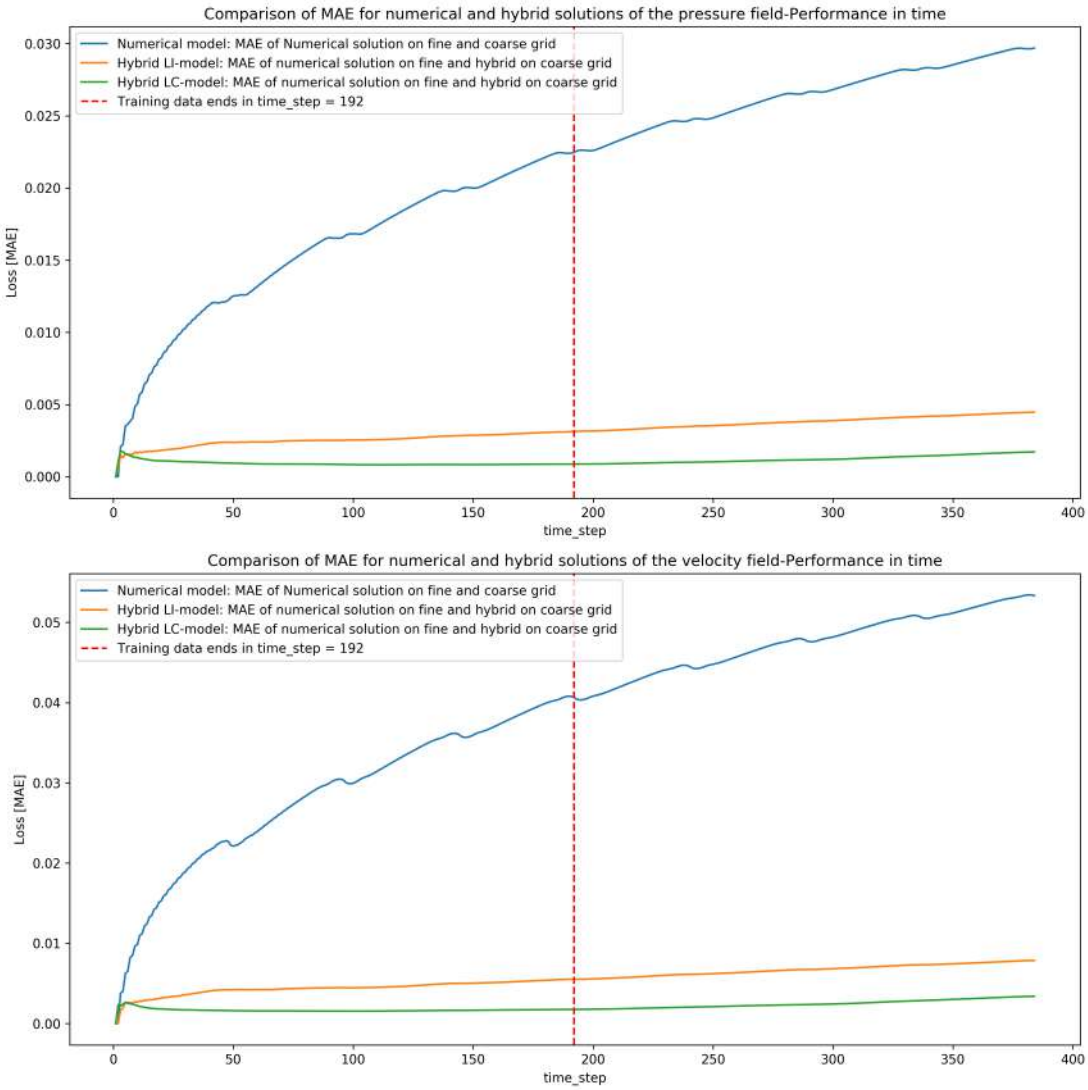
$$\frac{\partial}{\partial t}\begin{bmatrix} p \\ u \end{bmatrix} + \begin{bmatrix} 0 & K_0 \\ \frac{1}{\rho_0} & 0 \end{bmatrix} \frac{\partial}{\partial x}\begin{bmatrix} p \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7}$$

Όπου:

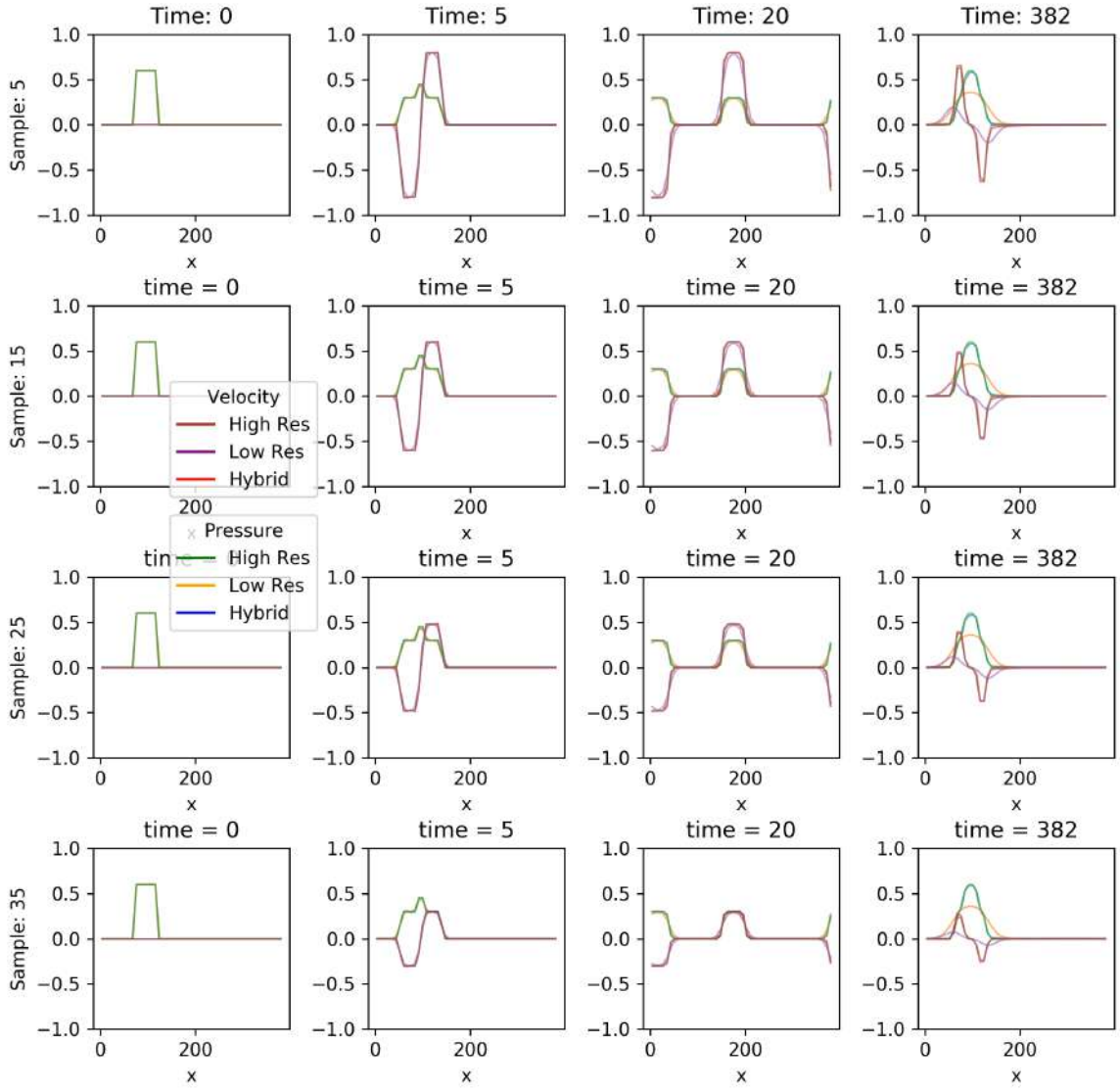
- $p(x,t)$:: διαταραχή της πίεσης σε σχέση με το περιβάλλον
- $u(x,t)$: διαταραχή της ταχύτητας σε σχέση με το περιβάλλον
- $K_0 = \rho_0 {c_0}^2$: το όριο συμπιεστότητας του μέσου (c_0: ταχύτητα του ήχου)
- ρ_0: η πυκνότητα του μέσου

Σε αυτό το πρόβλημα, οι αρχικές συνθήκες στις οποίες εκπαιδεύτηκε το ΤΝΔ είναι 10 τετράγωνα κύματα με διαφορετικά ύψη και κυμαινόμενη πυκνότητα του μέσου μετάδοσης σε τέσσερις διακριτές τιμές $\rho = \{0.75, 1, 1.5, 2\} kg/m^3$. Ζητείται από τους υβριδικούς επιλύτες να προβλέψουν την εξέλιξη για ένα τετράγωνο κύμα με διαφορετικό ύψος (ενδιάμεσο των άλλων) και με διαφορετική (ενδιάμεση) πυκνότητα μέσου. Και ελέγχεται η πρόβλεψη τους σε μεγαλύτερο χρονικό διάστημα απ΄ όσο έχουν εκπαιδευθεί για διασφάλιση της ευστάθειας τους.

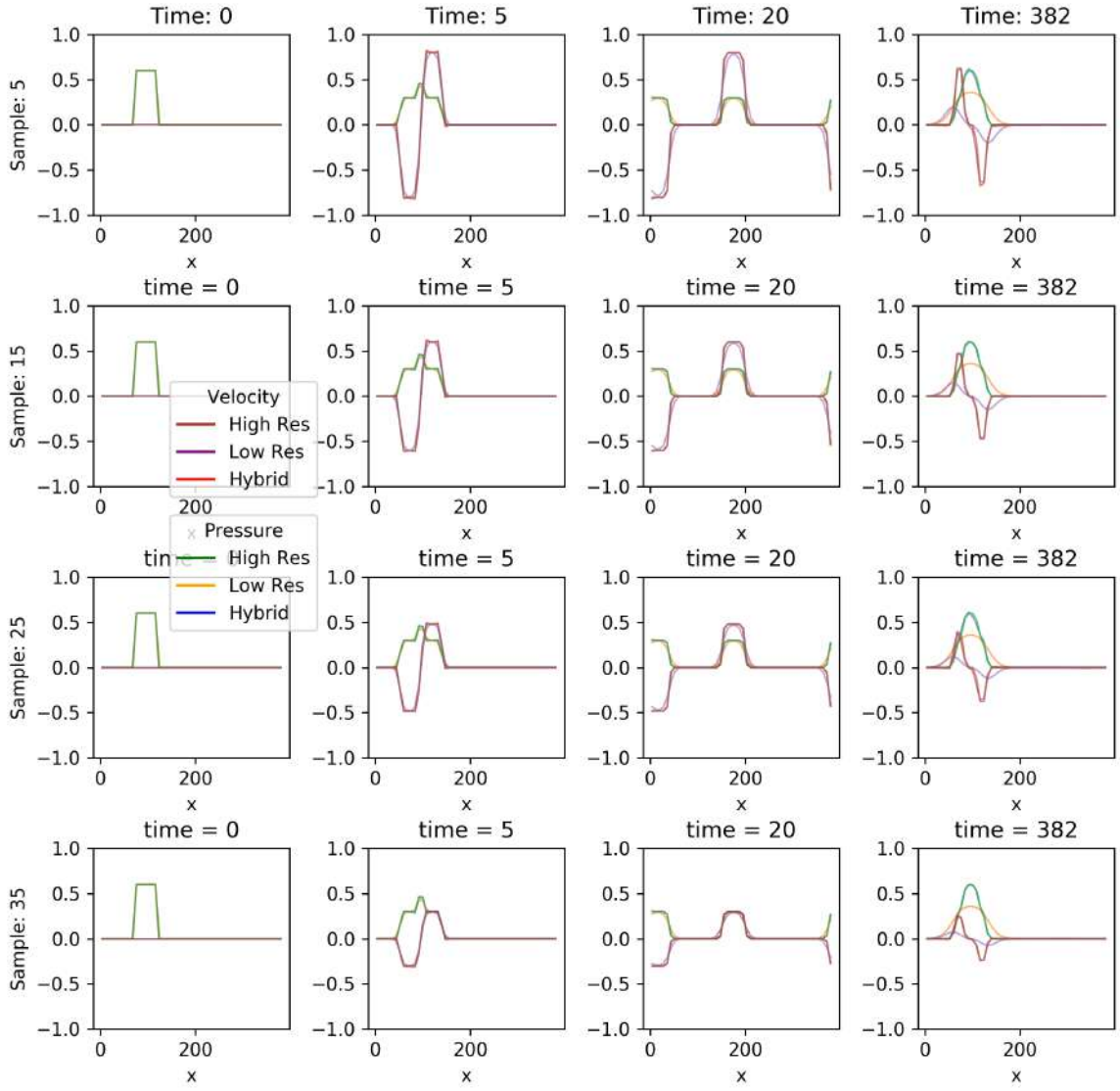
Επιβεβαιώνονται τα αποτελέσματα του πρώτου προβλήματος. Χρησιμοποιώντας ένα αραιό πλέγμα 48-σημείων, οι υβριδικοι επιλύτες γεννούν λύσεις κοντινές σε αυτές που θα παράγονταν από (καθαρά) αριθμητικούς επιλύτες σε πυκνό πλέγμα 384-σημείων -αφού προβληθούν στο αραιό-, απαλείφοντας σχεδόν το σφάλμα διακριτοποιήσης στο βαθμό που αυτό μπορεί να γίνει αντιληπτό από το ΤΝΔ μέσα από τα δεδομένα εκπαίδευσης του.



Σχήμα 8: *Σύγκριση των πεδίων πίεσης και ταχύτητας (πάνω και κάτω σχήματα αντίστοιχα) του (καθαρά) αριθμητικού επιλύτη στο αραιό πλέγμα 48-σημείων (μπλέ) και των υβριδικών στο αραιό πλέγμα 48-σημείων (μέθοδος ΠΣ: πορτοκαλί και μέθοδος ΠΔ :πράσινο), βάσει του Μέσου Απόλυτου Σφάλματος (ΜΑΣ) τους σε σχέση με την (καθαρά) αριθμητική λύση στο πυκνό πλέγμα 384-σημείων που έχει προβληθεί στο αραιό. Αυτό γίνεται για αρχικές συνθήκες τετράγωνων κυμάτων με ύψη που δεν έχουν χρησιμοποιηθεί στην εκπαίδευση ((νέα ύψη) $= 0.65 *$ (παλιά ύψη)) και με πυκνότητα μέσου ($\rho = 1.5kg/m^3$) που πάλι δεν έχει χρησιμοποιηθεί στην εκπαίδευση. Γίνεται προέκταση στον χρόνο για δύο ακόμα περιόδους (διπλάσιος χρόνος από αυτόν στα δεδομένα εκπαίδευσης). Εδώ η μέθοδος ΠΔ φαίνεται να είναι κάπως καλύτερη από την ΠΣ αλλά και οι δύο παράγουν καλά αποτελέσματα και ευσταθείς επιλύτες.*



Σχήμα 9: *Για τον επιλύτη ΠΣ: σύγκριση πολλών πεδίων πίεσης (χρωματική ομάδα πράσινο, πορτοκαλί, μπλέ) και ταχύτητας (χρωματική ομάδα: μπορντό,μωβ, κόκκινο) σε διάφορα χρονικά βήματα (ανά στήλη) και για τέσσερις διαφορετικές πυκνότητες (ανά σειρά). Ούτε το αρχικό τετράγωνο κύμα, ούτε οι συγκεκριμένες πυκνότητες μέσου έχουν χρησιμοποιηθεί στην εκπαίδευση. Η σύγκριση γίνεται για ένα 2ης τάξης σχήμα με Van Leer περιοριστή σε αραιό πλέγμα 48-σημείων (μωβ και πορτοκαλί) και για τον υβριδικό επιλύτη ΠΣ σε αραιό πλέγμα 48-σημειών (κόκκινο και μπλέ). Η σύγκριση γίνεται με βάση την αριθμητική επίλυση σε πυκνό πλέγμα 384-σημείων που έχει προβληθεί στο αραιό (μπορντό και πράσινο). Η λύση της ροής έχει προεκταθεί και για δύο περιόδους στον χρόνο (διπλάσιος χρόνος από αυτόν στα δεδομένα εκπαίδευσης). Τα πεδία ταχύτητας είναι πολλαπλασιασμένα με έναν παράγοντα* 680 *για την οπτικοποίηση των αποτελεσμάτων στο γράφημα. Η επικεφαλίδα "time" αναφέρεται σε διαφορετικά χρονικά βήματα.*



Σχήμα 10: *Για τον επιλύτη ΠΔ: Σύγκριση πολλών πεδίων πίεσης (χρωματική ομάδα πράσινο, πορτοκαλί, μπλέ) και ταχύτητας (χρωματική ομάδα: μπορντό,μωβ, κόκκινο) σε διάφορα χρονικά βήματα (ανά στήλη) και για τέσσερις διαφορετικές πυκνότητες μέσου (ανά σειρά). Παρόμοια αποτελέσματα με το Σχ.9.*

Παραμετρική μελέτη για την συμπεριφορά των υβριδικών επιλυτών

Πολύ σημαντική για την ευστάθεια και γενικότερη επίδοση των υβριδικών αυτών μοντέλων -κατά την εκπαίδευση και την χρήση τους- είναι η επιλογή των υπερπαραμέτρων. Από την μικρή παραμετρική μελέτη που διεξάγεται, συμπεραίνεται ότι η πιο σημαντική υπερπαράμετρος τόσο για την ευστάθεια του προκυπτόμενου υβριδικού επιλύτη, όσο και για την ακρίβεια των αποτελεσμάτων του είναι ο αριθμός των προβλέψεων που κάνει άνα ένα τρέξιμο του ο υβριδικός επιλύτης. Είναι εμφανές ότι απαιτείται ένας συμβιβασμός. Περισσότερα αναδρομικά βήματα οδηγούν σε καλύτερη ακρίβεια και ευ-

στάθεια αλλά αυξάνουν σημαντικά το κόστος κατά την εκπαίδευση. Η αυστηρότητα του κριτηρίου της έγκαιρης διακοπής όπως περιγράφηκε παραπάνω επίσης είναι μία παράμετρος που πρέπει να συνυπολογιστεί, μιας και μπορούν πιθανώς να ληφθούν καλά αποτελέσματα με λιγότερα τέτοια αναδρομικά βήματα.

Το βήμα εκμάθησης και το μέγεθος του batch εκπαίδευσης φαίνεται να παίζουν μικρό ρόλο (το βήμα μεγαλύτερο) και χωρίς κάποια καθαρή ένδειξη στο πώς να επιλεγούν εφόσον τα αποτελέσματα σε μία δοκιμή δεν είναι καλά.

Συμπεράσματα

Κατά τον προγραμματισμό και αξιολόγηση των δύο περιγραφόμενων υβριδικών επιλυτών, αποκτήθηκε τεχνογνωσία γύρω από το πεδίο τομής της ΥΡΔ και της ΒΜ καθώς και αντλήθηκαν χρήσιμα συμπεράσματα. Αρχικά, επιβεβαιώθηκε ότι η πολυπλοκότητα των απαιτούμενων ΤΝΔ που ενσωματώνονται μέσα στους αριθμητικούς επιλυτές δεν αυξάνεται δυσμενώς με την αύξηση της πολυπλοκότητας του προβλήματος. Αυτό υποδεικνύει την δυνατότητα εφαρμογής αυτών των τεχνικών σε μεγαλύτερα και πιο δύσκολα προβλήματα. Επιπλέον, μία σειρά από δοκιμές χρησίμευσαν για να ελέγξουν την ευστάθεια των μοντέλων σε διάφορες υπερπαραμέτρους και να επαληθεύσουν την ανθεκτικότητα της εκπαίδευσης και της ποιότητας των αποτελεσμάτων τους σε ποικίλες συνθήκες χρησιμοποιώντας σχετικά μικρή ποσότητα δεδομένων εκπαίδευσης. Το κέρδος των επιλυτών αυτών -στο αραιό πλέγμα- σε ταχύτητα σε σχέση με τους αριθμητικούς -στο πυκνό πλέγμα- εξαρτάται από τον λόγο αραίωσης, τον αριθμό των χωρικών διαστάσεων και από το κατά πόσο χρησιμοποιείται υλικό που αξιοποιεί παράλληλους υπολογισμούς.

Οι τρέχουσες τάσεις της σχετικής βιβλιογραφίας υποδηλώνουν ότι υβριδικές μεθοδολογίες όπως αυτές που παρουσιάστηκαν είναι πιθανό να επικρατήσουν σε βραχυπρόθεσμο έως μεσοπρόθεσμο διάστημα, προσφέροντας μια ισορροπημένη λύση που εκμεταλλεύεται την αξιοπιστία και γενικευσιμότητα των παραδοσιακών αριθμητικών μεθόδων και την ταχύτητα των ΤΝΔ.

Η συνέργεια προόδων στο υλικό (TPUs και εξειδικευμένες GPUs) και η συνεχιζόμενη άνθηση της έρευνας των στατιστικών αλγορίθμων [8] αναμένεται να δημιουργήσει ένα ιδιαίτερα γόνιμο περιβάλλον για την περαιτέρω ενσωμάτωση της ΤΝ στην ΥΡΔ. Και αυτό φαίνεται ότι θα οδηγήσει μακροπρόθεσμα σε πιο αποδοτικά και ακριβή μοντέλα ΥΡΔ.

Bibliography

[1] Anderson, J. D., J.: Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill, New York, NY, USA (1995)

[2] Bar-Sinai, Y., Hoyer, S., Hickey, J., Brenner, M.: Learning data-driven discretizations for partial differential equations. Proceedings of the National Academy of Sciences **116**(31), 15344–15349 (2019). https://doi.org/10.1073/pnas.1814058116, https://doi.org/10.1073/pnas.1814058116, edited by John B. Bell, Lawrence Berkeley National Laboratory, Berkeley, CA, approved June 21, 2019 (received for review August 14, 2018)

[3] Brenowitz, N.D., Bretherton, C.S.: Prognostic validation of a neural network unified physics parameterization. Geophysical Research Letters **45**, 6289–6298 (2018). https://doi.org/10.1029/2018GL078510, https://doi.org/10.1029/2018GL078510, first published: 13 June 2018, Citations: 175

[4] Brunton, S., Noack, B.R., P.Koumoutsakos: Machine learning for fluid mechanics. Annual Review of Fluid Mechanics **52**, 477–508 (2020). https://doi.org/10.1146/annurev-fluid-010719-060214, https://doi.org/10.1146/annurev-fluid-010719-060214, first published as a Review in Advance on September 12, 2019

[5] Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks **4**, 251–257 (1991), received 30 January 1990; revised and accepted 25 October 1990

[6] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (Nov 1998). https://doi.org/10.1109/5.726791

[7] LeVeque, R.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge, United Kingdom (2002), http://www.cambridge.org, first published in printed format

[8] Maslej, N., Fattorini, L., Brynjolfsson, E., Etchemendy, J., Ligett, K., Lyons, T., Manyika, J., Ngo, H., Niebles, J.C.and Parli, V., Shoham, Y., Wald, R., Clark, J., Perrault, R.: The ai index 2023 annual report. Tech. rep., AI Index

Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA (April 2023)

[9] Morgan, N., Bourlard, H.: Generalization and parameter estimation in feedforward nets: Some experiments. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems. vol. 2. Morgan-Kaufmann (1989), https://proceedings.neurips.cc/paper_files/paper/1989/file/63923f49e5241343aa7acb6a06a751e7-Paper.pdf

[10] Γ. Μπεργελές: Υπολογιστική Ρευστομηχανική. ΣΥΜΕΩΝ (2012)

[11] Um, K., Brand, R., Y.R.Fei, P.Holl, N.Thuerey: Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In: Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Vancouver, Canada (2020), https://arxiv.org/abs/2007.00016v2, arXiv:2007.00016v2 [physics.comp-ph] 5 Jan 2021

[12] Vinuesa, R., Brunton, S.: Enhancing computational fluid dynamics with machine learning. Nature Computational Science **2**(6), 358–366 (Jun 2022). https://doi.org/10.1038/s43588-022-00264-7, http://dx.doi.org/10.1038/s43588-022-00264-7

[13] Zhuang, J., Kochkov, D., Bar-Sinai, Y., Brenner, M., Hoyer, S.: Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. Physical Review Fluids **6**(064605) (2021). https://doi.org/10.1103/PhysRevFluids.6.064605, received 12 April 2020; accepted 19 April 2021; published 14 June 2021