DEEP-LEARNING-ENHANCED MULTIPHYSICS FLOW COMPUTATIONS FOR PROPULSION APPLICATIONS

A Dissertation
Presented to
The Academic Faculty

By

Petro Junior Milan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology

December 2021

DEEP-LEARNING-ENHANCED MULTIPHYSICS FLOW COMPUTATIONS FOR PROPULSION APPLICATIONS

Approved by:

Prof. Vigor Yang, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Joseph C. Oefelein
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Prof. Jean-Pierre Hickey
Department of Mechanical and Mechatronics Engineering
University of Waterloo

Dr. Gina M. Magnotti
Energy Systems Division
Argonne National Laboratory

Date Approved: Nov. 17, 2021

To my parents and late grandparents

PREFACE

This dissertation was submitted to the Academic Faculty at the Georgia Institute of Technology (GT), as partial fulfillment of the requirements for the Ph.D. degree. The underlying work of the thesis was carried out at the School of Aerospace Engineering at GT during the period August 2018 to October 2021, and at the Energy Systems (ES) Division at the Argonne National Laboratory (ANL) during the period January to August 2020.

At GT, the work was supervised by Prof. Vigor Yang, to whom I express my sincere appreciation and gratitude for his continuous advice and guidance, wealth of knowledge, and the tremendous support and opportunities he provided throughout my doctoral study. It has been a privilege as well as an inspiring and memorable experience to work with him. During my time in his group ("our" group, as he tells his students), I have not only grown as a researcher but also as a person, and the things I learned from him will stay with me for the rest of my life. I thank him for his trust in me and for giving me the flexibility to work on research projects that interest me. This is the first thesis in our group that deals with deep learning and its application to computational fluid dynamics. While progress has been made, there is still much work to be done in this area, and I hope that future students will follow this direction of research and expand the work.

At ANL, the work was managed by Dr. Gina M. Magnotti and Dr. Roberto Torelli from ES and by Dr. Bethany Lusch from the Argonne Leadership Computing Facility (ALCF), whom I sincerely thank for their mentorship, as well as for many invaluable discussions, and support and collaboration, which significantly impacted this thesis. I learned a lot about multiphase flow modeling and simulation and diesel engine design by working closely with Gina and Roberto, and about deep learning and high performance computing by working closely with Bethany. Moreover, I would like to thank Dr. Romit Maulik from ALCF for his collaboration and insightful discussions. Also, thanks to Dr. Sibendu Som, group manager of the Multi-Physics Computational Research Section at ES, as well as to the rest of my colleagues in the group for their kind help and support during my internship.

Furthermore, I would like to give special thanks to Prof. Jean-Pierre Hickey from the University of Waterloo (Canada) for giving me the opportunity to work with him on collaborative projects, as well as for his advice and technical support, and for many invaluable discussions throughout the period of this thesis. I also would like to thank the committee members, Prof. Joseph Oefelein, Prof. Edmond Chow, Prof. Jean-Pierre Hickey, and Dr. Gina M. Magnotti, for their time and invaluable input to improve the quality of this work.

In addition, I would like to acknowledge Dr. Xingjian Wang and Dr. Yixing Li from our group at GT for their technical support and fruitful discussions on the simulation and modeling of liquid-propellant rocket injector flows. I also would like to thank the rest of my colleagues and friends in the group, in particular Bichuan, Haoxiang, Umesh, and Weiming, for interesting discussions, both regarding research and not, and for a pleasant environment for work and study.

I would like to take the opportunity here to sincerely thank my undergraduate

research advisor at Polytechnique Montreal (Canada), Prof. Huu Duc Vo, who has been a constant source of support and advice over the years.

This work was performed using computing resources and services provided by the Partnership for an Advanced Computing Environment (PACE) at GT, by the Laboratory Computing Resource Center (LCRC) at ANL, and by ALCF, also at ANL. ALCF is a U.S. Department of Energy (DOE) Office of Science User Facility supported under Contract DE-AC02-06CH11357. I also would like to thank Convergent Science for providing CONVERGE licenses and technical support for part of this work.

Financial support for the research carried out at GT was provided by the U.S. Air Force Office of Scientific Research (AFOSR) under Grant No. FA9550-18-1-0216, by the Natural Sciences and Engineering Research Council of Canada (NSERC) under a Post-Graduate Scholarship D, and by the Aerospace Engineering Graduate Fellows Program at GT. The research carried out at ANL was funded by the U.S. DOE under contract LDRD Prime 2020-0094. Any subjective views or opinions that might be expressed in this dissertation do not necessarily represent the views of the sponsoring agencies.

This acknowledgement would be incomplete without the expression of my deepest appreciation and gratitude to my family and close friends. I particularly thank my grandparents, aunts, uncles, and cousins for their support and encouragement over the years. Finally, and most importantly, thanks to my parents and my two brothers, for their endless love, care, and support.

Petro Junior Milan
Atlanta, GA
October 23, 2021

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Numerical simulation is a critical part of research into and development of engineering systems. Engineers often use simulation to explore design settings both analytically and numerically before prototypes are built and tested. Even with the most advanced high performance computing facility, however, high-fidelity numerical simulations are extremely costly in time and resources. For example, a survey of the design parameter space for a single-element injector for a propulsion application (such as the RD-170 rocket engine) using the large eddy simulation technique may require several tens of millions of CPU-hours on a major computer cluster. This is because the flowfields can only be fully characterized by resolving a multitude of strongly coupled fluid dynamic, thermodynamic, transport, multiphase, and combustion processes. The cost is further increased by grid resolution requirements and by the effects of turbulence and high-pressure phenomena, which require treatment of real-fluid physics at supercritical conditions. If such models are used for statistical analysis or design optimization, the total computation time and resource requirements may render the work unfeasible.

Recent developments in deep learning techniques offer the possibility of significant advances in dealing with these challenges and significant shortening of the time-to-solution. The general scope of this thesis research is to set the foundations for new paradigms in modeling, simulation, and design by applying deep learning techniques to recent developments in computational science. More specifically, the research aims at developing an integrated suite of data-driven surrogate modeling approaches and software for large-scale simulation problems. The techniques to be put into practice include: (1) deep neural networks for function approximation and solver acceleration, (2) deep autoencoders for nonlinear dimensionality reduction, and (3) spatiotemporal emulators based on multi-level neural networks for simulator approximation and rapid exploration of design spaces.

A hierarchy of benchmark cases has been studied to generate databases to enable and support the development and verification of the proposed approaches. Emphasis is placed on canonical examples, as well as on engineering problems for aerospace and automotive applications, including supercritical turbulent flows in a rocket-engine swirl injector, and multiphase cavitating flows in a diesel engine injector.

ABBREVIATIONS

ADR	advection-diffusion-reaction
AE	autoencoder
AI	artificial intelligence
ALCF	Argonne Leadership Computing Facility
AMR	adaptive mesh refinement
API	application programming interface
ASOI	time after start of injection
BC	bottom-center
BNN	Bayesian neural network
CAE	convolutional autoencoder
CDE	correlated dynamic evaluation
CFD	computational fluid dynamics
CFL	Courant–Friedrichs–Lewy
CIM	core injection method
CNN	convolutional neural network
CPU	central processing unit
DDM	discrete droplet method
DFNN	deep feedforward neural network
DL	deep learning
DMD	dynamic mode decomposition
DNN	deep neural network
DNS	direct numerical simulation
DoE	design of experiments
ECN	Engine Combustion Network
EE	Eulerian-Eulerian
ELSA	Eulerian-Lagrangian spray atomization
EOS	equation of state
EVD	eigenvalue decomposition
FCAE	fully-connected autoencoder
FDM	finite-difference method
FEM	finite-element method
FLOPS	floating-point operations per second
FOM	full-order model
FVM	finite-volume method
GAN	generative adversarial network

GCLSC	gas-centered liquid-swirl coaxial
GDI	gasoline direct injection
GOX	gaseous oxygen
GP	Gaussian process
GPU	graphics processing unit
HRM	homogeneous relaxation model
HMM	homogeneous mixture model
HPC	high performance computing
ICE	internal combustion engine
ICM	interface capturing method
INRIA	National Institute for Research in Digital Science and Technology
IPC	instructions per cycle
IPS	instructions per second
LE	Lagrangian-Eulerian
LES	large eddy simulation
LOX	liquid oxygen
LPRE	liquid-propellant rocket engines
LSTM	long-short term memory
LVF	liquid volume fraction
ML	machine learning
MFM	multi-fluid model
MPI	message passing interface
MSE	mean-squared error
OWC	one-way coupling
PCA	principal component analysis
PDE	partial differential equation
POD	proper orthogonal decomposition
PR	Peng-Robinson
PSD	power spectral density
ORSC	oxidizer-rich staged-combustion
RANS	Reynolds-averaged Navier-Stokes
RB	reduced-basis
RK	Redlich-Kwong
RNN	recurrent neural network
ROI	rate-of-injection
ROM	reduced-order model/modeling
RNG	re-normalisation group
RPE	Rayleigh-Plesset equation
SciDL	scientific deep learning
SCC	Spray Combustion Consortium
SGS	subgrid-scale
SRK	Soave-Redlich-Kwong
SSE	sum of squared-error
SVD	singular value decomposition

TC	top-center
TFM	two-fluid model
TKE	turbulent kinetic energy
TPU	tensor processing unit
UFPV	unsteady flamelet progress variable
UQ	uncertainty quantification
URANS	unsteady RANS
VAE	variational autoencoder
VOF	volume of fluid
XPI	extra-high pressure injection

MATHEMATICAL NOTATION

This section provides a concise description of the mathematical notation used throughout this thesis. Exceptions to this, if any, will be indicated explicitly in the text. Also, a list of symbols is provided at the end of each chapter in Part II.

Numbers and Arrays

a (or A)	a scalar
$\boldsymbol{a}$ (or $\mathbfcal{A}$)	a vector
$\boldsymbol{A}$	a matrix or tensor

Examples: t is time coordinate (scalar), T is temperature (scalar), $\boldsymbol{q}$ is heat flux vector, $\mathbb{Q}$ is vector of conservative variables, $\boldsymbol{W}$ is weight matrix, and $\boldsymbol{Q}$ is a three-dimensional tensor representing the input to a convolutional autoencoder.

Individual Component Notation

a_i (or $\mathcal{A}_i$)	element i of vector $\boldsymbol{a}$ (or $\mathbfcal{A}$)
A_{ij} (or A_{ijk})	element i, j (or i, j, k) of matrix $\boldsymbol{A}$ (or tensor $\boldsymbol{A}$)

Note that not every subscript in the text necessarily refers to a component of vector, matrix or tensor. For instance, c_p is a scalar representing the specific heat at constant pressure.

Index Notation (or Einstein's Summation Convention)

This is a notational convention for simplifying expressions including summations of vectors, matrices and tensors, thus achieving brevity. We will explicitly state in the text when the index notation is used to avoid confusion with the individual component notation described above. The "rules" of summation convention are:

1. Each index can appear at most twice in any term.
2. Repeated indices are implicitly summed over.
3. Each term must contain identical non-repeated indices.

For example:

a_i (or $\mathcal{A}_i$)	vector $\boldsymbol{a}$ (or $\boldsymbol{\mathcal{A}}$)
A_{ij} (or A_{ijk})	matrix $\boldsymbol{A}$ (or tensor $\boldsymbol{A}$)
δ_{ij}	Kronecker delta matrix, $\delta_{ij} = 1$ for $i = j$, and 0 otherwise
A_{ii}	$\sum_i A_{ii}$ (trace of $\boldsymbol{S}$, which is a scalar)
$a_i x_i$	$\sum_i a_i x_i$ (scalar product of $\boldsymbol{a}$ and $\boldsymbol{x}$)
$\frac{\partial y}{\partial x_i}$	$\nabla_{\boldsymbol{x}} y$ (gradient of y, which is a vector)
$\frac{\partial^2 y}{\partial x_i \partial x_i}$	$\sum_i \frac{\partial^2 y}{\partial x_i^2}$ (Laplacian of y, $\Delta_{\boldsymbol{x}} y$, which is a scalar)
$\frac{\partial a_i}{\partial x_i}$	$\sum_i \frac{\partial a_i}{\partial x_i}$ (divergence of $\boldsymbol{a}$, $\nabla_{\boldsymbol{x}} \cdot \boldsymbol{a}$, which is a scalar)
$\frac{\partial (b a_i a_j)}{\partial x_j}$	$\sum_j \frac{\partial ((b a_j)\boldsymbol{a})}{\partial x_j}$ (vector)

Sets and Graphs

$\mathbb{A}$ (or Ω)	a set
$\mathbb{R}$	the set of real numbers
$\mathbb{N}$	the set of all positive integers including 0
$\{0, 1, \ldots, n\}$	the set of all integers between 0 and n
$[a, b]$	the real interval including a and b
$[a, b)$	the real interval including a but excluding b
$\mathcal{F}$	a graph, object or model

Functions

$f : \mathbb{A} \to \mathbb{B}$	the function f with domain $\mathbb{A}$ and range $\mathbb{B}$
f^{-1}	inverse function
$f \circ g$	composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	a function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$ (sometimes we write $f(\boldsymbol{x})$ and omit $\boldsymbol{\theta}$ to simplify the notation)
$\ln x$	natural logarithm of x
$\|\boldsymbol{a}\|_p$	L^p norm of $\boldsymbol{a}$
$\|\boldsymbol{a}\|$	L^2 norm of $\boldsymbol{a}$
$\|\boldsymbol{A}\|_F$	Frobenius norm of $\boldsymbol{A}$

Linear Algebra Operations

$\boldsymbol{A}^T$	transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A} \otimes \boldsymbol{B}$	element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$

Calculus

$\frac{dy}{dx}$	derivative of y with respect to x
$\frac{\partial y}{\partial x}$	partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	gradient of y with respect to $\boldsymbol{x}$
$\Delta_{\boldsymbol{x}} y$	Laplacian of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{x}} \cdot \boldsymbol{a}$	divergence of $\boldsymbol{a}$ with respect to $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	define integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{A}} f(\boldsymbol{x}) d\boldsymbol{x}$	define integral with respect to $\boldsymbol{x}$ over the set $\mathbb{A}$

CHAPTER 1
INTRODUCTION

1.1 Context and Motivation

Many contemporary problems in the physical sciences and engineering require the simulation of complex systems involving millions of degrees of freedom. Despite improved numerical solvers and the availability of high performance computing (HPC) infrastructures, high-fidelity models still remain computationally too expensive to be used routinely by practitioners and engineers in a design or analysis setting [1, 2, 3, 4]. Of particular interest here is the prediction of fluid flow and flame dynamics for the design of advanced chemical propulsion and power generation systems, such as liquid-propellant rocket engines (LPREs) and diesel engines. Since these devices operate under high pressures and extreme conditions, special care must be taken to describe the underlying processes that dictate system behavior.

To characterize the flowfields in these devices, we need to resolve a multitude of strongly coupled fluid dynamic, thermodynamic, transport, multiphase, and chemical processes to satisfy the governing equations. The problem is exacerbated by the broad range of length and time scales over which interactions occur, due to the presence of turbulence, and by the effects of high-pressure phenomena, which require treatment of real-fluid physics at supercritical conditions [5]. Direct numerical simulations (DNSs) or large eddy simulations (LESs) with very fine grids, long execution times, and high memory usage are required to accurately predict these flowfields. The simulations, however, can take on the order of weeks to months to complete, especially for full-scale practical devices, such as the RD-170 rocket injector element shown in Fig. 1.1, and even when parallelized on modern supercomputers. This combination of challenges significantly complicates the process of model development and scientific discovery. Further, if such methods are used in many-query scenarios such as statistical analysis or design optimization, which often involve running the simulations repeatedly for various configurations and design parameters, the total computation time and resources required may render the work unfeasible.

Accelerating these complex simulations is a key challenge, as it would enable a number of useful numerical tools to enhance scientific computation and improve the design cycle of engineering devices. Beyond the obvious advantages of expedited, or even real-time, decision making and control, the development of numerical methods for fast computations would also allow for the use of entirely new approaches, and techniques that have as yet been too complex for industrial applications [6]. Crucially, better uncertainty quantification (UQ) would become feasible if simulation costs were decreased substantially. Variables of interest for simulation are defined based on inputs estimated from real-world data, even though such data suffer unavoidably

from noise and measurement uncertainties. In certain cases, these uncertainties can propagate through a simulation, and the probability distributions of the variables of interest must be quantified in order to increase confidence in the prediction [6].

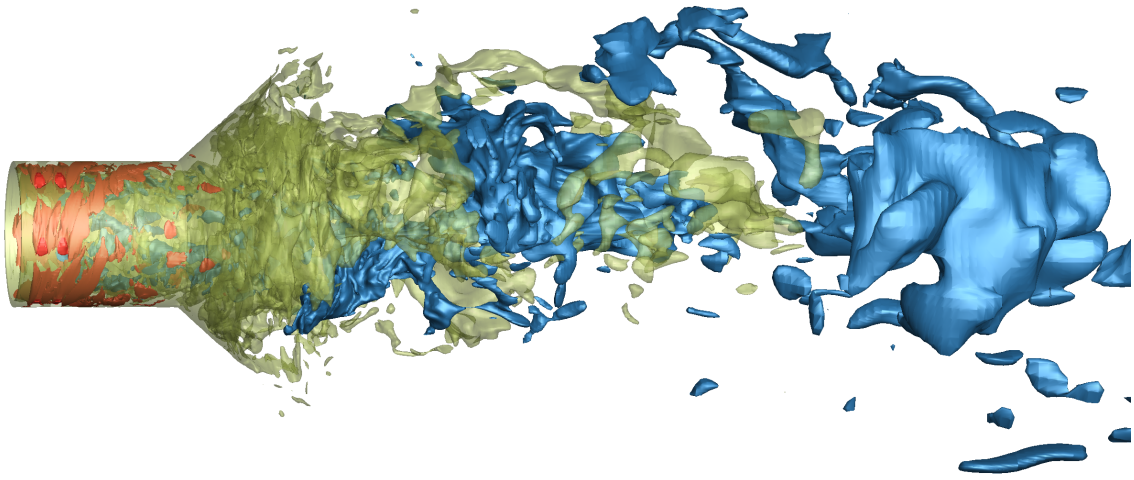


Figure 1.1: Iso-surfaces of azimuthal velocity at values of -70 (red), -20 (yellow) and 10 (blue) m/s obtained from 3D LES for a single-element RD-170 rocket injector fed for GOX/kerosene system operating at $p = 253$ bar. Each simulation takes about 1.2 million CPU-hours per 10 ms of simulated time and generates about 2 TB of data [7, 8].

Surrogate modeling approaches have emerged as an important avenue toward lowering the computational cost of a forward computation or accelerating inference in computationally expensive problems. A *surrogate model* aims to provide a simpler, and hence faster, emulation of the output of a more complex model in function of its input. Typically, surrogates can be classified into three categories, following Refs. [9, 10]: *data-fit models*, *reduced-order models* (ROMs), and *hierarchical models* (though these categories are not mutually exclusive, and a combination is often used). Here, the first two categories are of interest.

Data-fit models are generated using interpolation or regression to approximate the input/output relationships of a high-fidelity model. Traditionally, polynomial-based approximation, Gaussian processes (GPs) [11], and radial basis functions have been used extensively. These approaches treat the forward model as a black box, and thus require careful attention to the modeling choices and hyperparameter tuning that determine the shape of the model [9]. These models can be effective, although their application to high-dimensional problems and large datasets has been somewhat limited, and few studies have demonstrated their promise [12]. Unfortunately the associated computational cost is high, especially for GP-based models, which have a time complexity of $\mathcal{O}(n^3)$, with n being the size of the training set [13].

Reduced-order models are commonly derived using a projection framework to retain the essential physics and features of their corresponding full-order models

(FOMs). Historically, they fall largely within two categories: (1) techniques for projecting the governing equations onto a linear subspace of the original state using Galerkin projections [14], and (2) a posteriori methods for decomposition of the solution of interest into a linear combination of modes using the proper orthogonal decompostion (POD) [15] and dynamic mode decomposition (DMD) [16]. There are challenges in applying ROM to highly nonlinear flows, because a large number of modes are required to accurately reconstruct the flowfield [17]. The development of efficient and reliable ROM techniques for such nonlinear systems is an active area of research.

Recent advances in *deep learning* (DL) techniques, and their successful application to high-dimensional data regression and nonlinear feature fusion in image recognition/reconstruction [18, 19] and natural language processing [20], have stimulated research on DL-based surrogate modeling for high-dimensional nonlinear systems. DL, a subfield of machine learning (ML) and artificial intelligence (AI), involves algorithms that are loosely inspired by neuroscience and are referred to as *deep neural networks* (DNNs). In contrast to classical surrogate modeling approaches, carefully designed DNNs can capture complex high-dimensional nonlinearities, while avoiding overfitting. Further, they are universal approximators of continuous functions [21, 22]. Despite their recent success, the idea of DNNs is not new; they were first proposed by Walter Pitts and Warren McCulloch in the 1940s [23]. Their increased usage and current popularity is related to: (1) advancements in computer hardware, leading to widespread availability of graphics processing units (GPUs) and tensor processing units (TPUs) for accelerated computation and neural-network training, (2) availability of exceptionally large labelled datasets, (3) advances in DL theory and numerics, including improved stochastic optimization techniques for backpropagation, such as Adam [24] and RMSprop, with novel initialization techniques and activation functions, and (4) development of user-friendly and open-source software libraries, such as `Tensorflow` [25] and `PyTorch` [26].

Lately, DL has made its mark in fluid dynamics [27, 28]. In the context of data assimilation and computational reduction, Omata et al. [29] and Murata et al. [30] utilized an autoencoder (AE), a type of DNN, to extract the essential features of free-shear flows. Maulik et al. [17] combined AEs with recurrent neural networks (RNNs) for parametric ROMs of advection-dominated inviscid and low-viscosity systems. These works highlight AEs' ability to extract nonlinear modes with lower reconstruction errors than POD, at the same level of data compression. In the context of regression and emulation, Seong et al. [31] developed a neural network to evaluate liquid holdup and pressure gradient in a horizontal pipe flow. More recently, Xu et al. [32] developed a multi-level neural network framework for parameteric prediction of spatiotemporal dynamics in transient flows over a cylinder and behind a shipstructure. These works demonstrate the ability of DNNs to capture high-dimensional relationships in unsteady flows, but most of them have focused on canonical and laboratory-scale problems, with predictions under a fixed design point or with parametric variations in a single design parameter. Building on the findings from these studies, our work presents a novel contribution to the field by extending the applicability of DL models to large-scale, industry-relevant, fluid problems and

to multi-parametric studies, with a particular focus on high-pressure practical flows.

The general scope of this thesis is to explore new approaches based on deep neural networks for the development of surrogate models that are not only fast but also accurate, and to study the efficacy of these models in predicting the dynamics of complex multiphysics flows. Ultimately this work will enable predictive simulations of advanced concepts, and accelerate the design cycle for engineering devices. The research effort has *theoretical* as well as *numerical* goals:

- Theoretical: to develop novel surrogate modeling approaches based on deep learning for large-scale simulation problems that could impact propulsion applications as well as the broader scientific and engineering communities.
- Numerical: to create software and databases that enable the study and evaluation of the approaches developed.

Figure 1.2 illustrates the placement of the present work at the intersection of four research areas: chemical propulsion, computational fluid dynamics (CFD), DL, and HPC.

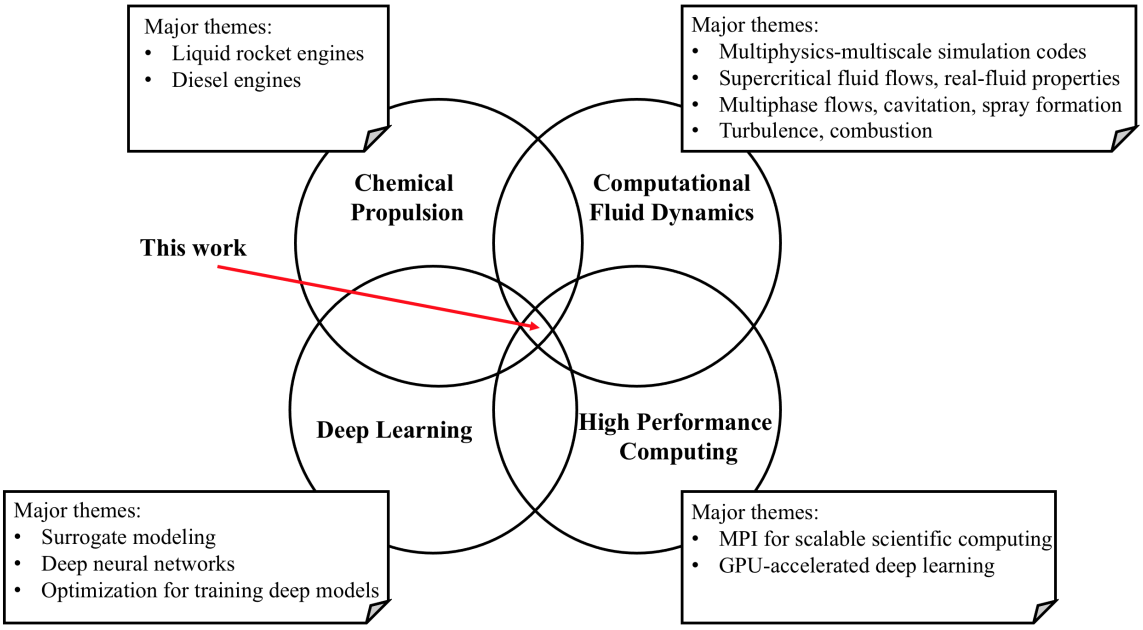


Figure 1.2: Intersection of research areas.

1.2 Aim and Objectives of this Work

The scope of the current work is to develop an integrated suite of surrogate modeling approaches and software to enhance *predictive simulation* and *design* of contemporary and future propulsion engines for automotive and aerospace applications. The proposed work utilizes recent breakthroughs in DL and HPC to substantially improve modeling capabilities at many levels. Techniques to be implemented include: (1) deep neural networks for function approximation and solver acceleration, (2) deep autoencoders for nonlinear dimensionality reduction, and (3) spatiotemporal emulators based on multi-level neural networks for rapid exploration of design spaces and enhanced optimization. A series of benchmark cases, comprising several canonical examples and two industry-relevant problems, has been identified to generate databases that will enable and support development, verification, and validation of the proposed algorithms. A particular focus is on applications involving high-pressure fuel injection processes and coupled multiphysics interactions. The research aim can be summarized as:

Research Aim

Explore and develop novel surrogate modeling approaches and software based on deep learning to accelerate the simulation and design of thermo-fluid systems for propulsion applications.

Several objectives are identified to achieve the overall aim. Each objective is briefly described along with the corresponding sub-objectives, or tasks. Note that at this stage of the document some of the tasks may not seem obvious or lack further explanations/justifications, but further details will be given in Part II.

1. **Generate databases from high-fidelity simulations for deep learning and reduced-order model development.**

   Here, the objective is to perform high-fidelity computer simulations of fluid dynamics in high-pressure propulsion engine environments. Two model problems with realistic geometries and operating conditions are addressed, namely (a) supercritical turbulent flows in a swirl rocket injector, with an operating pressure of 253 bar (cf. Sec. 5.5), and (b) multiphase cavitating flows in a diesel engine injector, with an injection pressure of 1,500 bar and an ambient pressure of 20 bar (cf. Sec. 7.3). Each problem is broken down into the following tasks:

   - Set up design of experiment (DoE). This includes identifying input/design parameters and their ranges, and generating a DoE table based on a given sampling technique. While the other steps are common to both problems, this step is applicable only to the diesel injector problem.
   - Set up CFD cases. This includes selecting an injector geometry, meshing, and selecting appropriate solver options (i.e., numerical scheme, time steps, turbulence model, thermodynamic model, etc.).

- Generate CFD data. This includes compiling the code, and running the numerical simulations on a supercomputer.
- Extract spatiotemporal CFD data at subdomain of interest for model building and/or model validation. This includes data management and post-processing.
- Analyze the flow physics. This includes an investigation of the detailed flow structures and mixing characteristics, and an examination of the effects of various design attributes on the flow.

2. **Develop a coupled deep-learning and physical-simulation framework to accelerate the convergence of high-fidelity simulations.**

   Here, we address the following question: how can DL models be used inside traditional CFD solvers to accelerate fluid simulations on a supercomputer? To do so, a neural network model, referred to as DFNN-BC (cf. Sec. 5.4), is developed for function approximation and solver acceleration. As a demonstration example, the evaluation of real-fluid properties, which is an expensive procedure in the simulation of high-pressure supercritical flows, is considered (cf. Chapter 5). The following tasks are identified:

   - Train and validate a neural network model for real-fluid property estimation. This includes building a separate real-fluid thermodynamic solver to generate a dataset of real-fluid properties at conditions of interest, training a neural network, and exporting the network parameters from the ML library.
   - Integrate boundary information into the model to respect the convergence constraints imposed by the higher-fidelity solution.
   - Build the coupled DFNN-BC and CFD framework. This includes deploying the DFNN-BC model in a flow solver, writing a subroutine for the forward propagation computation, and creating an interface to integrate the DFNN-BC into the solver.
   - Evaluate the proposed approach on the rocket injector problem and on other configurations, including counterflow diffusion flames.
   - Examine the computational efficiency and accuracy of the proposed approach. This includes comparing the outputs of the approach to those of the baseline numerical simulations, code profiling, and assessing the time and memory savings. As a reminder, the baseline simulations are performed as part of Objective 1.

3. **Develop a data-driven deep learning emulation framework for parametric and efficient prediction of spatiotemporal flowfields.**

   Here, we address the following question: how can DL models be used to accelerate the time-to-design? We implement a second framework based on data-driven spatiotemporal emulators to approximate expensive simulators (cf. Secs. 6.3

and 7.3.2). The framework is mainly developed for efficient survey, or exploration, of the full design space. Here, the use of the governing equations during the forward solve of the fluid flow configuration is avoided and replaced by surrogate models. The framework is evaluated on: (a) canonical examples (cf. Chapter 6) and (b) engineering problems (cf. Chapter 7). The following tasks are identified:

- Train and validate an AE-based network for nonlinear dimensionality reduction. Here, a novel ROM is introduced to extract physics-based information and to map high-dimensional snapshots to compressed representations in a nonlinear and universal manner. This step is required before a regression technique can be implemented.
- Train and validate a DNN-based regressor model to learn the relationship between the design parameters and reduced space representations.
- Build the spatiotemporal emulation framework with capabilities for parameter prediction. This involves combining the constituents from the previous tasks into a single framework.
- Port the emulation framework to a GPU-based supercomputer. This allows parallelization of the training of neural networks to handle large-scale datasets.
- Evaluate the framework on canonical one-dimensional and two-dimensional examples. These relatively straightforward problems enable debugging and permit preliminary verification and demonstration of the emulator.
- Evaluate the framework on the diesel injector problem, a realistic engineering application. This includes comparing the spatiotemporal predictions of the emulator at the injector exit to that of the high-fidelity simulations for unseen conditions during training, and assessing computational efficiency. Recall that the baseline simulations are performed as part of Objective 1.

1.3 Outline

The remainder of this thesis is composed of seven chapters organized into three major parts. This modular organization is designed to accommodate a variety of readers, especially graduate students majoring in Aerospace/Mechanical Engineering or Computational Science and Engineering, as well as practitioners and engineers who are interested in implementing some of the methods presented herein in their own software library or platform.

Part I provides the necessary fundamental concepts and terminology, numerical methods, and software tools that are needed to understand the core research projects. This part is composed of three chapters, as follows:

- Chapter 2 provides a basic review of surrogate modeling approaches, including DL and non-DL approaches. It also presents the DL models that form the major core architectural building blocks of the frameworks developed here. Lastly, it

suggests some practical considerations for training neural networks on a laptop or a supercomputer.

- Chapter 3 presents some of the basic theory and numerics of CFD relevant to this work. This includes the governing equations of fluid motion, turbulence and its modeling, the LES approach, and the finite-volume discretization method. It also introduces the various CFD solvers used to run the simulations and generate data for model development and/or model evaluation.

- Chapter 4 presents some of the basic physics and modeling practices for high-pressure fuel injection processes in diesel engines and LPREs, the engineering applications considered in this work.

Part II focuses on the three research projects carried out. The first is concerned with the use of DL in traditional and legacy CFD solvers to accelerate the convergence of fluid simulations, while the last two are concerned with the use of DL to build data-driven spatiotemporal emulators for rapid exploration of design spaces. Each project targets specific applications in fluid dynamics, with emphasis on practical high-pressure flows and fuel injection processes. The applications were mainly chosen for demonstration purposes, but the methodologies can be applied to other areas in computational science and engineering. For each project, the research problem and the underlying computational bottleneck are discussed, along with the successes and limitations of previous studies as reported in the literature. This part is organized as follows:

- Chapter 5 presents a DL-based approach for fast calculation of real-fluid thermophysical properties in numerical simulation of supercritical flows. The method features a DNN with appropriate boundary information and can be coupled to a flow solver in a robust manner. The approach is demonstrated in primitive- and conservative-variable-based solvers and is systematically evaluated on various problems of increasing complexity: zero-dimensional thermodynamics, one-dimensional counterflow diffusion flames, and rocket injector flows.

- Chapter 6 presents a data-driven emulation framework based on DL techniques for efficient prediction of parametric, multivariate, and spatiotemporal flow-fields. The framework features deep AEs, for nonlinear dimensionality reduction, and DNNs for supervised learning of the latent space. Two versions of this framework are proposed, based on two different autoencoders: FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$. The framework is evaluated on two canonical fluid problems: the one-dimensional viscous Burgers equation and the two-dimensional advection-diffusion-reaction (ADR) equation.

- Chapter 7 applies the FCAE-$\mathcal{R}$ framework to a representative engineering problem for automotive propulsion. Specifically, the emulator is used to efficiently predict the multiphase cavitating flow at the orifice exit of a diesel injector for a wide range of design and physical parameters. The predictions from the emulator are used to initialize one-way coupled spray simulations in the combustion chamber.

The last part contains only Chapter 8. In this chapter, conclusions, including an overall summary and highlights of major contributions, as well as recommendations for future work are provided. Supplementary materials and additional insights on specific topics are included in appendices.

Part I

Fundamental Background, Tools, and Techniques

CHAPTER 2
APPROXIMATION METHODS AND DEEP LEARNING

... all models are approximations. Essentially, all models are wrong, but some are useful. However, the approximate nature of the model must always be borne in mind ...

Georges E. P. Box, 1987

... what we want is a machine that can learn from experience.

Alan Turing, 1947

Approximation methods are more and more commonly used in the acceleration of computer simulations, as well as in the optimization of complex engineering systems. The approximation method provides a surrogate model which, once developed, can be called instead of the original expensive model for the purposes of fast computation and optimization.

This chapter provides a review of surrogate modeling approaches and deep learning. In Sec. 2.1, a taxonomy of surrogate models is presented. In Sec. 2.2, we discuss some of the basic models used in deep learning, such as deep feedforward neural networks (DFNNs), convolutional neural networks (CNNs), and autoencoders. We also briefly discuss some more advanced models. Finally, in Sec. 2.3, we provide state-of-the-art practices for training neural networks.

2.1 Taxonomy of Surrogate Models

2.1.1 Classification based on Mathematical Structure

Surrogates can be classed broadly in three categories, based on their mathematical structure [9, 10]: *data-fit models*, *reduced-order models* (ROMs), and *hierarchical models*, as shown in Fig. 2.1. Only the first two classes are employed in the present work, although for the sake of completeness all three are described below.

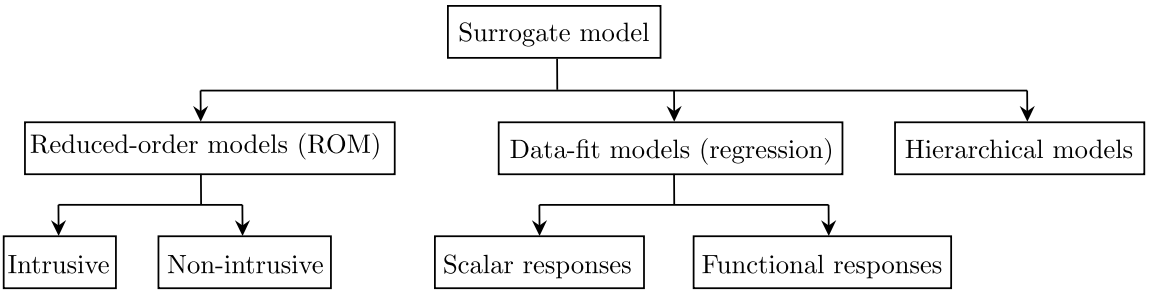


Figure 2.1: Classification of surrogate models (note that this classification is not exhaustive).

2.1.1.1 Data-Fit Models

Data-fit models are generated using interpolation or regression approaches to approximate the input/output relationships of a high-fidelity model. They include polynomial regression, GPs [11], and neural networks. Data-fit models can further be classified into two sub-categories depending on whether the responses are scalar or functional; these are *scalar-response models* and *functional-response models*.

In a scalar-response model, the output is a scalar or a vector of scalars. That is, each sample element is geometrically a point. The model provides statistical approximation of the original function $f^* : \mathbb{R}^{n_{\text{in}}} \mapsto \mathbb{R}^{n_{\text{out}}}$ where $\boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}$ is the input and $\boldsymbol{y}^* \in \mathbb{R}^{n_{\text{out}}}$ is the output.

In a functional-response model, the output is a function or a vector of functions, also called field variable. That is, each sample element is geometrically a curve, surface or anything else varying over a continuum. The physical continuum over which these functions are defined is parametrized by parameters $t_1, t_2, \ldots, t_n$. A particular case is when the physical continuum is defined over the time t and spatial location $\boldsymbol{r}$. In such a case, the output is written as $\boldsymbol{y}^*(t, \boldsymbol{r}) \in \mathbb{R}^{n_{\text{out}}}$, or in discretized form as $\boldsymbol{Y}^* \in \mathbb{R}^{n_t \times n_{\text{grid}} \times n_{\text{out}}}$, where n_t is the number of time snapshots, n_{grid} is the number of grid points in the domain, and n_{out} is the number of output variables (or functions). The model provides statistical approximation of the original mapping $\mathcal{S}^* : \mathbb{R}^{n_{\text{in}}} \mapsto \mathbb{R}^{n_t \times n_{\text{grid}} \times n_{\text{out}}}$ where $\boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}$ is the input and $\boldsymbol{Y}^*$ is the output. In other words, the object $\mathcal{S}^*$ that we are trying to approximate can be thought of as a simulator, which refers to the computational model that explicitly solves a set of differential equations that govern some physical processes over a defined spatial region and time interval. The input can be a vector $\boldsymbol{x}$ defining the design or physical parameters of the simulation, such as the Reynolds number, fluid properties, boundary conditions, etc. The output can be the solution of the simulation, $\boldsymbol{Y}^*$, which is usually represented by one or more spatiotemporal fields, such as the axial velocity field, temperature field, etc.

In summary, a scalar-response model is used to approximate a function f^* whereas a functional-response model is used to approximate a simulator $\mathcal{S}^*$.

2.1.1.2 Reduced-Order Models

Reduced-order models retain the essential physics and features of their corresponding FOMs. They are constructed using a projection framework or via ML and DL approaches. Many different types of ROMs have been developed over the years, and a possible distinction is between those that are *intrusive* and thus require the knowledge of the underlying FOM, such as the governing equations or the numerical solver, and those that are *purely data-driven* and thus *non-intrusive*. In the first sub-category, one can find the reduced-basis (RB) method [33, 34, 35, 36], POD-Galerkin approach [37, 38], and residual minimization principles [39, 40, 41]. In the second category fall the non-intrusive RB method [42, 43, 44], snapshot-based POD (see Appendix A), DMD [16], and autoencoders (see Sec. 2.2.5).

ROMs can also be classified as *linear* and *nonlinear* models, depending on whether the transformation or mapping from the high-dimensional space to the latent space is linear or nonlinear. Examples of linear ROMs include POD and DMD while examples of nonlinear ROMs include autoencoders.

2.1.1.3 Hierarchical Models

Hierarchical models are physics-based models of lower accuracy and reduced computational cost. They are derived from higher-fidelity models using approaches such as simplifying physics assumptions or omitted physics, coarser grids, lower-order discretization methods, and relaxed solver tolerances. This category of models also includes multi-fidelity models [45], which are built by combining different fidelity models.

2.1.2 Data-Driven vs. Physics-Informed Deep Learning

ML models, in particular neural networks, will be explained in detail in Sec. 2.2. However, since the present section deals with classification of surrogate models, it is worth distinguishing at this stage of this document between the following two approaches to scientific DL [46, 47]:

1. *Pure data-driven models*, which aim to model physical systems using neural networks that are trained to fit observed data without knowledge of the underlying physics. Here, the assumption is that given enough data, the neural network should be able to recognize or discover hidden patterns and correlations in the data. In these models, the loss function is mainly composed of two terms: (1) data loss, which is usually defined as the mean-squared error (MSE) or sum of squares for error (SSE) between the ground truth solution and predicted output by the neural network, and (2) regularization term, such as L_1 and L_2 regularization, used to avoid overfitting (see Sec. 2.3.3.1). If trained properly, these models can generalize well to unseen points, or conditions, within the defined design space, however, they perform poorly outside the design space (i.e.,

poor extrapolation). Overall, these models are useful in the *large-data regime*, where a large amount of labeled data is available.

2. *Physics-informed models* (also referred to as *physics-constrained models*, or *knowledge-guided models*), which aim to model physical systems by incorporating physical principles or physics-based knowledge (i.e., governing equations, conservation principles, initial and boundary conditions, symmetries, invariances, etc.) into the neural networks together with data. By exploiting physics, the amount of data needed to get good performance can be reduced considerably compared to that of the pure data-driven approaches, and the trained neural network can not only generalize well but also sometimes extrapolate well. Typically, the physics are imposed on the neural network via *soft* or *hard* constraints. Soft constraints are implemented by adding physics-based penalty terms to the original loss function [48, 49, 50]. These terms act as a physics-based regularizer that penalizes the neural network if it doesn't obey the desired physics. In particular, for problems or tasks involving spatiotemporal emulation, the underlying governing equations can be embedded into the loss function of the neural network via *automatic differentiation* [51], which is a technique used for differentiating the output coordinates of the neural network with respect to its input coordinates and model parameters. The hard constraints, on the other hand, are enforced by modifying the architecture of the neural network such that the desired physics are automatically fulfilled [52, 53]. Overall, physics-informed models are useful in the *small-data regime*, where labeled data can be difficult to get, or when the available data are sparse and noisy.

Note that in this thesis the focus is mostly on data-driven models. However, for certain problems, knowledge-guided models have been exploited to some extent, in particular for incorporating initial and boundary conditions into the loss function of the involved neural network for improved prediction accuracy.

2.2 Neural Network Architectures

In this section, we present a brief overview of neural network approaches including DFNNs, CNNs, and AEs. These approaches form the major core architectural building blocks of DL models currently used, and can also be composed modularly – a little like a "Lego" model – to build new application-specific network architectures [54]. For a more comprehensive review, interested readers may refer to classic textbooks such as Goodfellow et al. [55], Aggarawal [56], and Nielsen [57].

2.2.1 What is Deep Learning?

Deep learning (DL) is an approach to artificial intelligence (AI). Specifically, it is a subfield of *machine learning* (ML), a technique that enables computer systems to improve with experience and data. DL is concerned with algorithms modeled loosely on the human brain called *deep neural networks* (DNNs).

DL (and ML) models are capable of different types of learning, which are usually classified as *supervised learning*, *unsupervised learning*, and *reinforcement learning*. Supervised learning utilizes labeled datasets to categorize or make predictions (known as classification and regression, respectively); this requires some kind of human intervention to label the data appropriately. In contrast, unsupervised learning utilizes unlabeled datasets to analyze and discover hidden patterns in the data for various tasks. This includes clustering them by any distinguishing characteristics (known as clustering), determining the distribution of data within the input space (known as density estimation), and transforming the data from a high-dimensional space into a low-dimensional space (known as dimensionality reduction). Reinforcement learning is concerned with how a model learns to become more accurate for performing an action in an environment based on feedback in order to maximize a reward.

2.2.2 Artificial Neuron and Dense Layer

The basic entity of any neural network is a model of an *artificial neuron*, also called *perceptron*, as shown in Fig. 2.2. Let $\boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}$ be an input. The functional form of an artificial neuron is given by

$$y = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b), \tag{2.1}$$

where $b \in \mathbb{R}$ is a bias scalar, $\boldsymbol{w} \in \mathbb{R}^{n_{\text{in}}}$ is a weight vector, and $y \in \mathbb{R}$ is an output. The activation function, σ, performs a linear or nonlinear transformation between the input and output (more details about activation functions can be found in Sec. 2.2.7). The loss function of an artificial neuron can be defined as the MSE between the actual and predicted values,[1] as follows

$$\begin{aligned} J(\boldsymbol{w}, b) &= \|y^* - y\|_{\text{MSE}} \\ &= \frac{1}{n} \sum_{i=1}^{n} (y^{*,(i)} - y^{(i)})^2, \end{aligned} \tag{2.2}$$

where n is the training dataset size (or size of a minibatch[2]), and $y^{*,(i)}$ and $y^{(i)}$ are the true and predicted output, respectively, for the ith training example $\boldsymbol{x}^{(i)}$. Note that $y^{(i)} = \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b)$, hence the dependence of J on the parameters $\boldsymbol{w}$ and b in Eq. (2.2).

A *dense layer*, also called *fully-connected layer*, is a layer unit composed of many artificial neurons. Its functional form is given by

$$\boldsymbol{y} = f_{\text{dense}}(\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.3}$$

where $\boldsymbol{y} \in \mathbb{R}^{n_{\text{out}}}$ is an output, and $\boldsymbol{\theta}$ a set of parameters consisting of a weight matrix $\boldsymbol{W} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$ and a bias vector $\boldsymbol{b} \in \mathbb{R}^{n_{\text{out}}}$.

[1] Note that other error metrics such as the SSE, can be used instead of the MSE when training the DL models, but this is not done here.

[2] See Sec. 2.3.6 for a definition of *minibatch*.

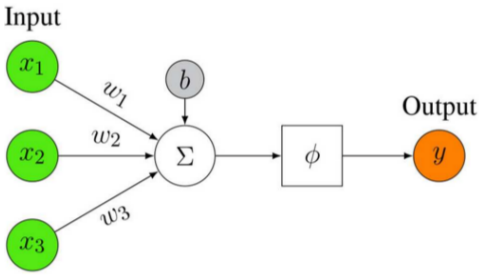


Figure 2.2: Representation of an artificial neuron model . In this schematic, $\boldsymbol{x} \in \mathbb{R}^3$ (reproduced from Ref. [54]).

2.2.3 Deep Feedforward Neural Networks

If all layers in a network are dense layers and interconnected in a sequential way, the network is called a *deep feedforward neural network* (DFNN), or *multilayer perceptron* (MLP). In Fig. 2.3, we show examples for a one-hidden layer architecture and a deep architecture. The computation in the lth layer of a network, $\boldsymbol{h}^{(l)} \in \mathbb{R}^{n_l}$, with $n_l \in \mathbb{N}$, takes the form

$$\begin{aligned} \boldsymbol{h}^{(l)} &= f^{(l)}(\boldsymbol{h}^{(l-1)}; \boldsymbol{\theta}^{(l)}) \\ &= \sigma_l\Big(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\Big), \end{aligned} \tag{2.4}$$

where $f \equiv f_{\text{dense}}$, $\boldsymbol{h}^{(0)} = \boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}$ is the input layer, $\boldsymbol{h}^{(L)} = \boldsymbol{y} \in \mathbb{R}^{n_{\text{out}}}$ is the output layer, $\boldsymbol{\theta}^{(l)} = \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}$ is a set of trainable parameters, and L is the total number of layers (excluding the input layer from the count). Also, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is a weight matrix, where the element $W^{(l)}_{mn}$ denotes the weight of the connection from the nth neuron in the $(l-1)$th layer to the mth neuron in the lth layer, $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ is a bias vector, where the element $b^{(l)}_m$ is the bias for the mth neuron in the lth layer, and σ_l is an activation function applied in the lth layer. Note that the layers $\boldsymbol{h}^{(l)}$, $l = 1, 2, \ldots, L-1$, are called *hidden layers*.

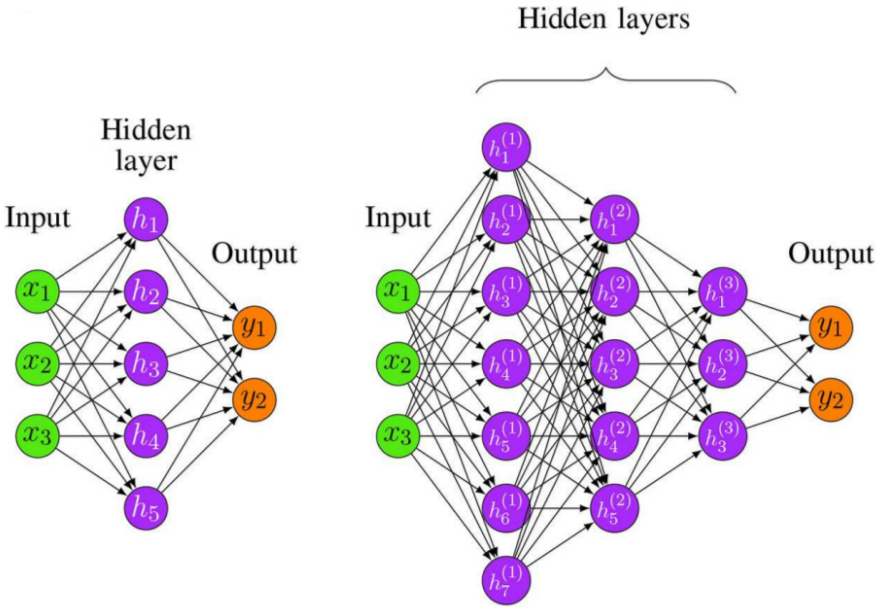
The DFNN can be expressed using a mapping $\mathcal{F}^{\text{DFNN}} : \mathbb{R}^{n_{\text{in}}} \mapsto \mathbb{R}^{n_{\text{out}}}$ as:

$$\boldsymbol{y} = \mathcal{F}^{\text{DFNN}}(\boldsymbol{x}; \boldsymbol{\theta}) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\boldsymbol{x}; \boldsymbol{\theta}^{(1)}), \tag{2.5}$$

where $\circ$ is the function composition operator, and $\boldsymbol{\theta} = \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}; l = 1, 2, \ldots, L\}$ is a set containing all of the trainable parameters of the neural network. The loss function of a DFNN can be defined as the MSE between the actual and predicted values, as follows

$$J(\boldsymbol{\theta}) = \|\boldsymbol{y}^* - \boldsymbol{y}\|_{\text{MSE}}, \tag{2.6}$$

where $\boldsymbol{y}^*$ is the true output.



(a) (b)

Figure 2.3: Representations of: (a) neural network with one hidden layer, and (b) DFNN with 3 hidden layers. In this schematic, $\boldsymbol{x} \in \mathbb{R}^3$ and $\boldsymbol{y} \in \mathbb{R}^2$ (reproduced from Ref. [54]).

2.2.4 Convolutional Neural Networks

A *convolutional neural network* (CNN) [58] is a specialized type of neural network for processing grid-structured inputs, which have strong spatial dependencies in local regions of the grid. Examples include one-dimensional time series data, two-dimensional image data, and three-dimensional video data. An additional dimension can be added to capture the different colors or field variables in the data (referred to as the number of *channels*), which creates a multi-dimensional input volume (or hypervolume). For the purpose of discussion, assume a 32×32 colour image like in the CIFAR-10 dataset [59]. For each of the 32×32 pixels, there is a value for the red, green, and blue colors. Hence, in this case, the CNN interprets one input sample as a $32 \times 32 \times 3$ volume where 3 is the number of channels. In the following, we will limit our discussion to a CNN operating on two-dimensional images.

The CNN works in a similar way to a DFNN, except that it performs local operations on its layers in order to learn spatial hierarchies of features [56]. CNNs utilize *convolution* and *pooling* layers, in addition to *dense* layers. The dense layers are no different from a traditional neural network, and are often added toward the end of the network to map the internal representations to a set of output nodes that conforms to a given problem. Prior to the application of any dense layer, each layer in the network preserves a three-dimensional grid structure, which has a *height*, *width*, and *depth*. Here, the word "depth" refers to the number of channels in the input layer or

the number of feature maps/levels in the hidden layers, and should not be confused with the depth of the network itself.

The convolution layer applies a set of n_{f} learnable *filters*, also called *kernels*. The filter is usually square in terms of its spatial dimension, typically much smaller than that of the layer the filter is applied to. However, the depth of the filter is always the same as that of the layer it is applied to. Each filter passes across the entire input image using a certain stride s, and performs a dot product operation between its parameters and the matching grid in the input volume. To maintain spatial dimensions of the output the same as those from the input after a convolution operation, layers of zeros can be added to the border of the image using a *padding* process. In this case, the resulting output maintains the same height and width of the input layer, but has a different depth equal to the number of filters used in the process.

The functional form of a convolution layer can be symbolically written as

$$\boldsymbol{H}^{(l+1)} = f_{\mathrm{conv}}(\boldsymbol{H}^{(l)}; \boldsymbol{W}, \boldsymbol{b}) = \sigma(\boldsymbol{H}^{(l)} \star \boldsymbol{W} \oplus \boldsymbol{b}), \tag{2.7}$$

where $\boldsymbol{H}^{(l)}$ denotes a three-dimensional input volume with a depth of d_l in the lth layer, $\boldsymbol{H}^{(l+1)}$ is the output representing the $(l+1)$th layer, $\boldsymbol{W}^{(m)}$ is the weights of the mth three-dimensional filter applied to the lth layer, with $m = 1, 2, \ldots, n_{\mathrm{f}}$, $\boldsymbol{b} \in \mathbb{R}^{n_{\mathrm{f}}}$ is the bias term in the $(l+1)$th layer, σ is the activation function applied in the $(l+1)$th layer, and the symbols "$\star$" and "$\oplus$" denote the convolution and addition operations, respectively. At a sliding location (i, j) and filter index m, the output $H_{ijm}^{(l+1)}$ of a convolution layer can be expressed as

$$H_{ijm}^{(l+1)} = \sigma\left(\sum_{k=1}^{d_l} \sum_{p} \sum_{q} H_{i-p,j-q,k}^{(l)} W_{pqk}^{(m)} + b_m \right), \tag{2.8}$$

where $H_{ijk}^{(l)}$ is the input value at point (i, j, k), $W_{pqk}^{(m)}$ the weight at point (p, q, k) in the mth filter, and b_m the bias of the mth filter. It is noted that the output $\boldsymbol{H}^{(l+1)}$ has a depth of n_{f}, the number of filters used in the process. An example of a convolution operation is shown in Fig. 2.4.

The convolution layer is usually followed by a pooling layer that performs compression operations with respect to the data. The pooling operation defines a non-learnable filter (i.e., a filter without any learnable parameters) that is usually square in size and typically much smaller than that of the layer the filter is applied to. The filter is passed across the entire layer using a certain stride s, and produces another layer with the same depth, but with smaller height and width. Unlike with convolution operations, where all d_l feature maps are used in combination with a filter to produce a single feature value, pooling independently operates on each feature map to produce another feature map. Hence, the pooling operation does not change the depth of the layer.

Two types of pooling layers are usually used: *max pooling* and *average pooling*. The max pooling layer computes the maximum element from the region of feature map covered by the filter. Thus, the output of a max pooling layer is a feature map

containing the most dominant features of the previous feature map. On the other hand, the average pooling layer computes the average of the elements present in the region of feature map covered by the filter. Thus, average pooling gives the average of features for each patch of the feature map. An example of a max pooling operation is shown in Fig. 2.5.

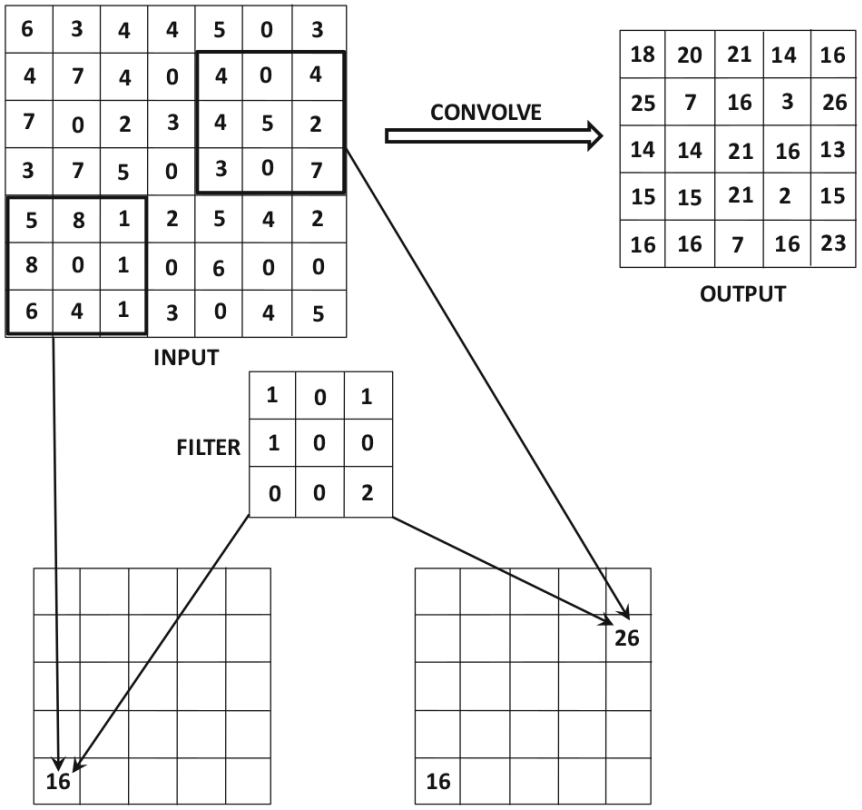


Figure 2.4: Example of a convolution operation between a $7 \times 7 \times 1$ input and a $3 \times 3 \times 1$ filter with stride of 1 and no padding. The resulting output is of size $5 \times 5 \times 1$. It is noted that no activation function nor bias term were used in this example (reproduced from Ref. [56]).

2.2.5 Autoencoders

Autoencoders (AEs) are a type of neural network architecture developed for unsupervised feature extraction. The main motivation for utilizing AEs is to perform dimensionality reduction. Importantly, while POD implements a linear transformation, AEs are nonlinear. Usually, this results in better performance and lower reconstruction errors at the same level of data compression [19]. Here, we distinguish between *fully-connected autoencoders* (FCAEs) and *convolutional neural network autoencoders* (CAEs).

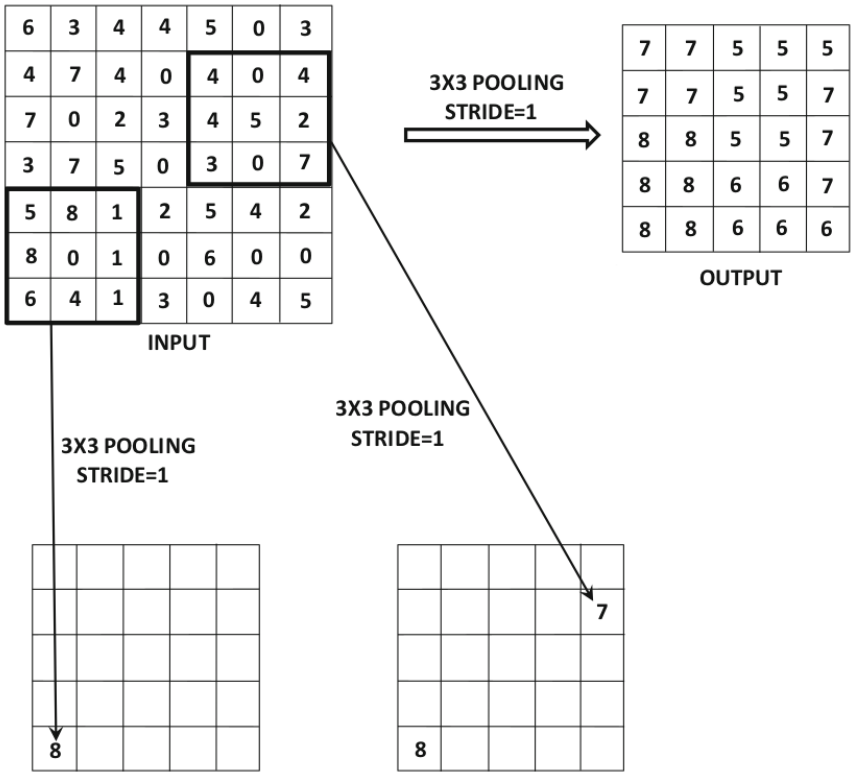


Figure 2.5: Example of a max pooling operation with a window of size $3 \times 3 \times 1$ and stride of 1 applied to a layer of size $7 \times 7 \times 1$. The resulting output is a layer of size $5 \times 5 \times 1$ (reproduced from Ref. [56]).

2.2.5.1 Fully-Connected Autoencoders

A fully-connected autoencoder [19] is a type of DFNN. In this architecture, the input and output are approximately the same, i.e., $\boldsymbol{x} \approx \boldsymbol{y}$, and the neural network has a converging-diverging shape with a low-dimensional hidden layer in the middle. It consists of two parts: an encoder $\mathcal{F}_{\text{enc}}$ and a decoder $\mathcal{F}_{\text{dec}}$, as seen in Fig. 2.6. The encoder is used to map a high-dimensional dataset into a low-dimensional *latent representation* (also referred to as *bottleneck layer*, or *code*). The decoder is then used to expand the dimensionality from that of the latent space back to the one of the original data, reconstructing the dataset. Mathematically, the forward propagation procedure in the FCAE can be described as follows

$$\begin{aligned} \widetilde{\boldsymbol{x}} &= \mathcal{F}_{\text{dec}} \circ \mathcal{F}_{\text{enc}}(\boldsymbol{x}; \boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}) \\ &= \mathcal{F}_{dec}(\boldsymbol{z}; \boldsymbol{\theta}_{\text{dec}}), \end{aligned} \tag{2.9}$$

where $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{n_{\text{in}}}$ is an approximation of the input, and $\boldsymbol{z} \in \mathbb{R}^{n_{\text{latent}}}$ is the latent vector, with $n_{\text{latent}} \ll n_{\text{in}}$. The encoder $\mathcal{F}_{\text{enc}} : \mathbb{R}^{n_{\text{in}}} \mapsto \mathbb{R}^{n_{\text{latent}}}$ is parametrized by $\boldsymbol{\theta}_{\text{enc}}$ and the decoder $\mathcal{F}_{\text{dec}} : \mathbb{R}^{n_{\text{latent}}} \mapsto \mathbb{R}^{n_{\text{in}}}$ is parametrized by $\boldsymbol{\theta}_{\text{dec}}$. The loss function of an FCAE

can be defined as the MSE between the input and output:

$$J(\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{dec}}) = \|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|_{\text{MSE}}. \tag{2.10}$$

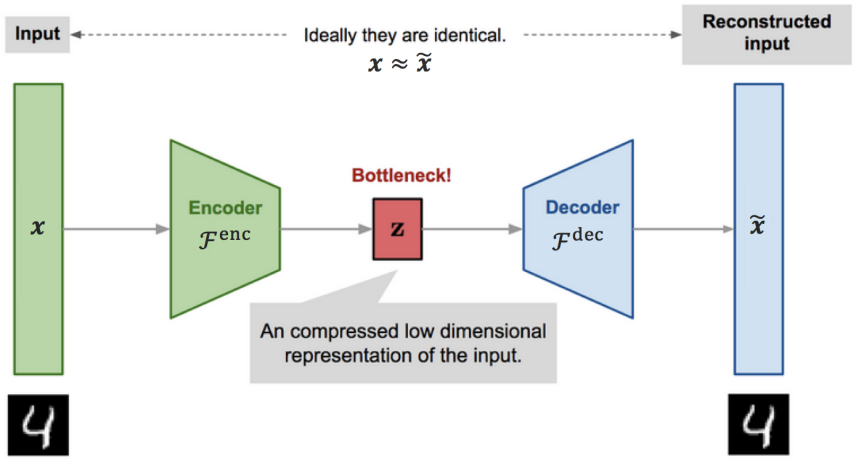


Figure 2.6: Schematic representation of an autoencoder (image modified from Ref. [60]). In this schematic, the input is a vector of pixels representing the digit 4.

2.2.5.2 Convolutional Neural Network Autoencoders

A convolutional autoencoder [61] is a type of CNN utilizing *convolution*, *sampling*, and *dense* layers for dimensionality reduction. Typically, in traditional FCAEs, the input data is flattened to a one-dimensional vector before being processed, which prevents direct learning of spatial coherence. Furthermore, each neuron in a layer is connected to all neurons in the next layer, where each connection is a parameter in the network. This can result in a very large number of trainable parameters. Instead of using only dense layers, a CAE uses some local connectivity between neurons, i.e., a neuron is only connected to nearby neurons in the next layer, which allows to significantly reduce the total number of parameters in the network and to process image data directly, thus retaining spatial information.

As a reminder, the convolution and fully-connected layers were explained in Secs. 2.2.4 and 2.2.2, respectively. The sampling layer performs compression or extension operations with respect to the data, and is usually preceded by a convolution layer. In this work, a max pooling operation is used for compression/downsampling, and a resize operation based on the nearest neighbor interpolation for upsampling [55]. The CAE is composed of a CNN encoder, $\mathcal{G}_{\text{enc}}$, and a CNN decoder, $\mathcal{G}_{\text{dec}}$, as shown in Fig. 2.7. The forward propagation procedure in the CAE can be written as

$$\begin{aligned} \widetilde{\boldsymbol{Q}} &= \mathcal{G}_{\text{dec}} \circ \mathcal{G}_{\text{enc}}(\boldsymbol{Q}; \boldsymbol{\phi}_{\text{enc}}, \boldsymbol{\phi}_{\text{dec}}) \\ &= \mathcal{G}_{\text{dec}}(\boldsymbol{z}; \boldsymbol{\phi}_{\text{dec}}), \end{aligned} \tag{2.11}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n_x \times n_y \times n_c}$ and $\widetilde{\boldsymbol{Q}} \in \mathbb{R}^{n_x \times n_y \times n_c}$ denote the encoder input and decoder output, respectively, with n_c the number of input channels, and $\boldsymbol{z}$ the latent space representation. It is noted that although the same symbol $\boldsymbol{z}$ is used, the latent space obtained from the CAE is different than that from the FCAE. The encoder $\mathcal{G}_{\text{enc}}$ is parametrized by $\boldsymbol{\phi}_{\text{enc}}$, and the decoder $\mathcal{G}_{\text{dec}}$ by $\boldsymbol{\phi}_{\text{dec}}$. The loss function of a CAE is defined in a similar way to that of an FCAE, i.e., as the MSE between the input and output:

$$J(\boldsymbol{\phi}_{\text{enc}}, \boldsymbol{\phi}_{\text{dec}}) = \|\boldsymbol{Q} - \widetilde{\boldsymbol{Q}}\|_{\text{MSE}}. \tag{2.12}$$

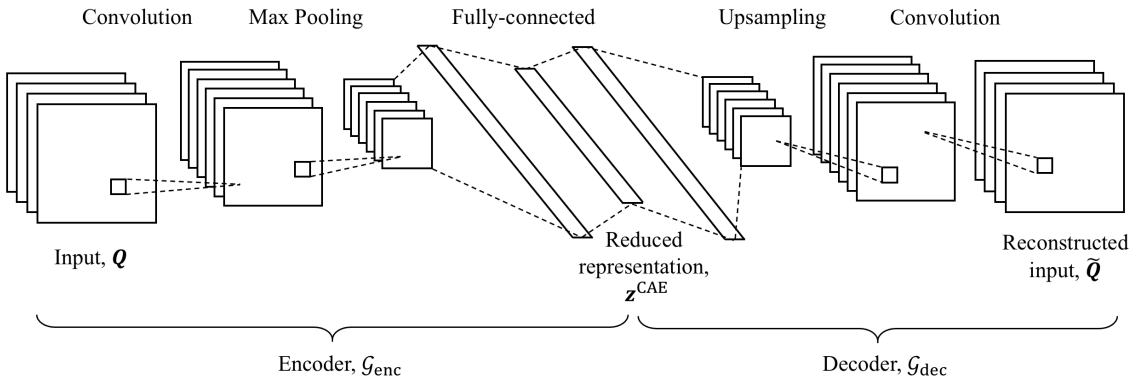


Figure 2.7: Schematic representation of a CAE.

2.2.6 Advanced Models

We would like to emphasize that there are additional models of DL networks, which are outside the core architectures, such as the *recurrent neural network* (RNN) and *Bayesian neural network* (BNN).

RNNs are a class of neural networks that allows previous outputs to be used as inputs while having hidden states. RNNs are particularly useful for cases involving sequential data like natural language and time series. RNNs, however, have limitations. For instance, they do not perform well when handling long-term dependencies [55, 62]. Moreover, they suffer from the gradient vanishing/exploding problem [63]. A *long short-term memory* (LSTM) network is a variant of RNN that is designed to address those limitations of RNNs, making it very useful in practice.

BNNs extend the capability of standard networks with posterior inference. Rather than fitting a point estimate of the network weights, a BNN learns a distribution over the weights in order to capture uncertainty and assign confidence to predictions. The BNN is defined as $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$: given an input $\boldsymbol{x}$, it assigns a probability to each possible output $\boldsymbol{y}$ using a set of parameters $\boldsymbol{\theta}$. The posterior probability $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ is generally computed from Bayes rule by specifying a Gaussian density distribution for the prior parameters and the likelihood. A particular type of BNN is a *variational autoencoder* (VAE), which combines the merit of an AE and a probabilistic model, and thus allows to quantify the amount of information lost during the compression process [64].

Other advanced models include *generative adversarial networks* (GANs) [65] and *transformers* [66]. While these models have their own advantages and disadvantages, their utilization and exploitation are outside the scope of this thesis and are left for future work.

2.2.7 Importance of Nonlinear Activation Functions

Nonlinear activation functions are essential for introducing nonlinearity in the neural network; without them, neural networks reduce to simple linear regression models. The activation functions are mathematical functions attached to each neuron. The activation takes place depending on some rule or threshold, and only the neurons with some relevant information are activated in each layer. A list of commonly used activation functions is provided in Table 2.1.

Table 2.1: Frequently used activation functions.

Activation function	$\sigma(\boldsymbol{x})$	$\sigma'(\boldsymbol{x})$	Range of σ
Identity	$y = x$	1	$(-\infty, \infty)$
Softmax	$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$		$(0, 1)$
Sigmoid/Logistic	$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$	$\sigma(x)(1-\sigma(x))$	$(0, 1)$
Hyperbolic tangent (tanh)	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1-\sigma(x)^2$	$(-1, 1)$
Rectified linear unit (ReLU)	$\text{ReLU}(x) = \max(0, x)$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$[0, \infty)$
Exponential linear unit (ELU)	$\begin{cases} \alpha(e^{x}-1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} \alpha e^{x} & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\alpha, \infty)$
Leaky ReLU	$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish	$x \times \text{sigmoid}(x)$	$\frac{1+e^{-x}+xe^{-x}}{(1+e^{-x})^2}$	$[-0.278\ldots, \infty)$

2.3 Practical Considerations for Training Neural Networks

In the previous section, we introduced various neural network architectures. In this section, we explain the procedure for training neural networks using learning algorithms. We also discuss important considerations, including backpropagation, overfitting, regularization, hyperparameter tuning, and acceleration of model training.

2.3.1 Backpropagation

In order to train neural networks such as the ones defined in the previous section by a gradient-based optimization algorithm, we need to be able to compute the gradient of the loss function with respect to each network parameter. The gradient tells us

how a small change in that parameter will affect the loss function. The gradient can be computed through recursive application of the *chain rule* of calculus; this is referred to as *backpropagation*, or *backprop* for short. Note that the term backpropagation strictly refers only to the procedure for computing the gradient, not how the gradient is used, such as updating the network parameters through an optimization algorithm. However, the term is often used loosely to refer to the entire optimization algorithm [67]. In the following, we show how to compute the gradient for a DFNN. A similar procedure can also be applied to other types of neural networks.

Consider a DFNN with L layers (excluding the input layer from the count). Let $\boldsymbol{W}^{(l)}$ be a weight matrix, where the element $W_{jk}^{(l)}$ denotes the weight of the connection from the kth neuron in the $(l-1)$th layer to the jth neuron in the lth layer, $\boldsymbol{b}^{(l)}$ be a bias vector, where the element $b_j^{(l)}$ is the bias for the jth neuron in the lth layer, and $\boldsymbol{h}^{(l)}$ be the lth layer, where $\boldsymbol{h}^{(0)} = \boldsymbol{x}$ is the input layer, $\boldsymbol{h}^{(L)} = \boldsymbol{y}$ is the output layer, and $\boldsymbol{h}^{(l)}$ with $l = 1, 2, \ldots, L-1$ are the hidden layers. As previously seen in Secs. 2.2.2 and 2.2.3, the functional form of $\boldsymbol{h}^{(l)}$ (for $l = 1, 2, \ldots, L$) is given by

$$\begin{aligned} \boldsymbol{h}^{(l)} &= \sigma(\underbrace{\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}}_{\equiv \boldsymbol{a}^{(l)}}) \\ &= \sigma\Bigg(\sum_k \underbrace{W_{jk}^{(l)} h_k^{(l-1)} + b_j^{(l)}}_{\equiv a_k^{(l)}}\Bigg), \end{aligned} \tag{2.13}$$

where σ is the activation function, and $\boldsymbol{a}^{(l)} \equiv \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}$ is an intermediate quantity in the lth layer before the activation function is applied. Here, we assume that the same activation function is applied to all the hidden layers and output layer. The derivation of backpropagation is easily generalizable, however, to allow different layers to have individual activation functions.

In backpropagation, we want to compute the quantities $\partial J/\partial w$ and $\partial J/\partial b$ with respect to any weight w or bias b in the network, where J is the loss function. As a reminder, J is defined here as the MSE between the input and output (cf. Sec 2.2.3), i.e.,

$$J = \frac{1}{n}\sum_{\boldsymbol{x}\in\mathbb{D}} J_{\boldsymbol{x}}, \tag{2.14}$$

where $\mathbb{D}$ is the training set (or a minibatch from the training set), n is the size of this set, and $J_{\boldsymbol{x}}$ is the loss for an individual training example. Note that

$$\frac{\partial J}{\partial b_j^{(l)}} = \frac{1}{n}\sum_{\boldsymbol{x}\in\mathbb{D}} \frac{\partial J_{\boldsymbol{x}}}{\partial b_j^{(l)}}, \tag{2.15}$$

$$\frac{\partial J}{\partial W_{jk}^{(l)}} = \frac{1}{n}\sum_{\boldsymbol{x}\in\mathbb{D}} \frac{\partial J_{\boldsymbol{x}}}{\partial W_{jk}^{(l)}}. \tag{2.16}$$

Consequently, we can first compute the gradient of the loss function for individual training examples, then take the average to obtain the gradient of the loss function over all the training examples.

An important quantity in the derivation is the vector of errors $\boldsymbol{\delta}^{(l)}$ at the lth layer, which is defined by

$$\boldsymbol{\delta}^{(l)} \equiv \frac{\partial J_{\boldsymbol{x}}}{\partial \boldsymbol{a}^{(l)}}, \tag{2.17}$$

or in individual component form by

$$\delta_j^{(l)} \equiv \frac{\partial J_{\boldsymbol{x}}}{\partial a_j^{(l)}}, \tag{2.18}$$

where $\delta_j^{(l)}$ represents the error of neuron j in layer l.

The four fundamental equations behind backpropagation can be written as follows [57]:

The equations of backpropagation

$$\boldsymbol{\delta}^{(L)} = (\nabla_{\boldsymbol{h}^{(L)}} J_{\boldsymbol{x}}) \odot (\sigma'(\boldsymbol{a}^{(L)})), \tag{2.19}$$

$$\boldsymbol{\delta}^{(l)} = ((\boldsymbol{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}) \odot (\sigma'(\boldsymbol{a}^{(l)})), \tag{2.20}$$

$$\frac{\partial J_{\boldsymbol{x}}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}, \tag{2.21}$$

$$\frac{\partial J_{\boldsymbol{x}}}{\partial W_{jk}^{(l)}} = h_k^{(l-1)} \delta_j^{(l)}. \tag{2.22}$$

Note that Eqs. (2.19)–(2.21) are written in vector form, whereas Eq. (2.22) is written in individual component form. We also note that a proof of these equations is provided in Appendix B, where the chain rule is used substantially. These equations give us a way of computing the error and the gradient of the loss function with respect to the network parameters. In particular, Eq. (2.19) represents the error in the output layer, $\boldsymbol{\delta}^{(L)}$. To get this quantity, we first compute the vector $\nabla_{\boldsymbol{h}^{(L)}} J_{\boldsymbol{x}}$, which in individual component form is given by $\partial J_{\boldsymbol{x}} / \partial h_j^{(L)}$, then take the Hadamard product "$\odot$" with the vector $\sigma'(\boldsymbol{a}^{(L)})$, which in individual component form is given by $\sigma'(a_j^{(L)})$. Equation (2.20) represents the error in the lth layer, $\boldsymbol{\delta}^{(l)}$, as a function of the error in the next layer, $\boldsymbol{\delta}^{(l+1)}$. In this equation, we first compute the vector $(\boldsymbol{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}$, which in individual component form is given by $\sum_k W_{kj}^{(l+1)} \delta_k^{(l+1)}$, then take the Hadamard product with the vector $\sigma'(\boldsymbol{a}^{(l)})$ to get $\boldsymbol{\delta}^{(l)}$. This moves the error backward, hence the name *error backpropagation*.

By combining Eqs. (2.19) and (2.20), we can compute the error $\boldsymbol{\delta}(l)$ for any layer in the network. We start by using Eq. (2.19) to compute $\boldsymbol{\delta}^{(L)}$, then apply Eq. (2.20) recursively to get $\boldsymbol{\delta}^{(L-1)}, \boldsymbol{\delta}^{(L-2)}, \ldots, \boldsymbol{\delta}^{(1)}$. After this, we can substitute these errors into Eqs. (2.21) and (2.22) to get the gradient of the loss function for a single training example with respect to any weight and bias in the network. Finally, the gradient of the loss function over all the training examples can be recovered by using Eqs. (2.15) and (2.16).

2.3.2 Underfitting and Overfitting

An important challenge in DL is to design a learning algorithm that performs well not only on data observed during training, but also on new and unseen data; this ability is referred to as *generalization*. To assess generalization, it is important to have a *held-out test set* that is different from the training set (but can come from the same distribution) and evaluate the model's performance on this set after training is completed. Error measured on the data from the training set is called *training error*, while error on the test set is called *test error* (or *generalization error*).

A DL model is said to perform well if the training error is small as well as the gap between training and test errors [55]. As an aside, note that the test error is almost always larger than the training error. However, it is possible for the test error to be smaller than the training error, particularly if the training set contains "harder" cases to learn or if the sampling is biased.

Alternatively, a model is said to be deficient if at least one of the conditions above is not met. In particular, we distinguish between two cases: *underfitting* and *overfitting*.

Underfitting (i.e., high bias and low variance) happens when the training error is large. In this case, the neural network can neither model the training data nor generalize well to new data. Generally, this is due to the model being too simple, informed by too few features, and/or regularized too much. Hence, the model is not learning enough from the training set, which results in unreliable predictions. Underfitting can be reduced (or avoided) by:

Strategies to handle underfitting

- Increasing model complexity.
- Increasing the number of features.
- Increasing the number of epochs during training (in an attempt to get better results).

On the other hand, overfitting (i.e., low bias and high variance) happens when the gap between the training and test errors is large. In this case, the model performs very well on the training set, but poorly on the test set. Generally, this is due to the model trying to learn all trends from the training set, including secondary patterns and noise, but not the dominant one(s). Overfitting can be reduced (or avoided) by:

Strategies to handle overfitting

- Decreasing model complexity.
- Increasing training data (in an attempt to capture the dominant trends).
- Using regularization methods (see Sec. 2.3.3.1).
- Stopping the training earlier (see Sec. 2.3.3.2).
- Using validation techniques (see Sec. 2.3.4).

2.3.3 Regularization

Regularization is any modification done to the learning algorithm in order to reduce its test error (but not necessarily its training error). Regularization can be added in different ways. One way is to add restrictions directly on the parameters in the neural network, which corresponds to a *hard* constraint on the parameter values. Another way is to add extra terms in the loss function, which corresponds to a *soft* constraint on the parameter values [55]. In this latter category, a generic expression for a regularized loss function can be written as

$$J_{\text{reg}}(\boldsymbol{\theta}; \lambda_{\text{reg}}) = J_{\text{unreg}}(\boldsymbol{\theta}) + \lambda_{\text{reg}}\varphi(\boldsymbol{\theta}), \tag{2.23}$$

where J_{reg} is the regularized loss function, J_{unreg} is the unregularized loss function, $\lambda_{\text{reg}} \geq 0$ is the regularization hyperparameter, and φ is the regularization function.

These penalties and constraints can also be designed to encode prior knowledge in the model, such as physical principles and domain knowledge. While various regularization strategies exist, we describe hereafter only the L_1 and L_2 regularization and early stopping methods.

2.3.3.1 L_1 and L_2 Regularization Methods

L_2 regularization, also known as *ridge regression*, *weight decay*, or *Tikhonov regularization*, consists of adding a regularizer, which is controlled by the hyperparameter λ_{reg} to achieve a bias-variance trade-off. As it turns out, overfitting is often characterized by weights with large magnitudes. Consequently, L_2 regularization tries to reduce the possibility of overfitting by keeping the values of the weights small. The weight penalty term added to the loss function corresponds to the sum of the squared values of the weights. This term penalizes large weight values. The L_2-regularized loss function is given by

$$\begin{aligned} J_{L_2\text{-reg}}(\boldsymbol{\theta}; \lambda_{\text{reg}}) &= J_{\text{unreg}}(\boldsymbol{\theta}) + \lambda_{\text{reg}} \sum_k \|\boldsymbol{W}^{(k)}\|_F^2 \\ &= J_{\text{unreg}}(\boldsymbol{\theta}) + \lambda_{\text{reg}} \sum_k \sum_i \sum_j (W_{ij}^{(k)})^2, \end{aligned} \tag{2.24}$$

where $\boldsymbol{W}^{(k)}$ is the weight matrix between the $(k-1)$th and kth layers in the network, and $\|.\|_F$ the Frobenius norm. In particular, for a DFNN with L layers (excluding the input layer from the count), the loss function becomes

$$J_{L_2\text{-reg}}(\boldsymbol{\theta}; \lambda_{\text{reg}}) = \|\boldsymbol{y}^* - \boldsymbol{y}\|_{\text{MSE}} + \lambda_{\text{reg}} \sum_{k=1}^{L} \|\boldsymbol{W}^{(k)}\|_F^2. \tag{2.25}$$

It is noted that we can leave the bias parameters unregularized because they typically require less data to fit accurately than the weight parameters. This is because the number of biases is much smaller than the number of weights, and therefore we do not induce too much variance by leaving the biases unregularized. Moreover, regularizing the bias parameters can introduce a significant amount of underfitting [55].

An important question is how to choose the hyperparameter λ_{reg}. Ideally, we must achieve a balance between model complexity and model accuracy. In particular, if the value of λ_{reg} is too large, the model will be simple and will likely underfit because it will not learn enough about the training data to make useful predictions. On the other hand, if the value of λ_{reg} is too small, the model will be more complex and will likely overfit because it will try to learn all the particularities of the training data.

Another parameter regularization method is the L_1 *regularization*, also known as *Lasso regression*. This method is similar to the L_2 regularization, except that its penalty term corresponds to the absolute values of the weights, as given by the following expression

$$J_{L_1\text{-reg}}(\boldsymbol{\theta}; \lambda_{\text{reg}}) = J_{\text{unreg}}(\boldsymbol{\theta}) + \lambda_{\text{reg}} \sum_{k} \sum_{i} \sum_{j} |W_{ij}^{(k)}|. \tag{2.26}$$

While L_2 regularization encourages the weight values to be spread out more equally towards zero (but not exactly zero), L_1 regularization encourages the weight values to be zero. Hence, L_1 regularization is useful for feature selection, especially in sparse feature spaces (i.e., when the features contain missing values) or when the number of features is larger than the dataset size.

2.3.3.2 Early Stopping

The *early stopping* method [68] is a form of regularization used to reduce overfitting by allowing a training session to be terminated if the number of epochs (or iterations) exceeds a given threshold without an improvement in the validation error. Early stopping requires a validation set.[3] This method is particularly useful when training large neural networks for which we sometimes observe that the training error decreases steadily in time, but the validation error starts to rise again [55]. In such cases, stopping the algorithm earlier can prevent the model from learning all the particularities of the training data, thus reducing overfitting.

[3]Definitions of *validation set* and *validation error* have not been provided yet at this stage of the document. See Sec. 2.3.4 for an explanation of these terms.

Every time the validation error improves, we save a copy of the model parameters. When the training algorithm is terminated, we return the network parameters that correspond to the best validation error, rather than the parameters from the latest epoch (or iteration).

Finally, note that early stopping can be used either alone or in conjunction with other regularization approaches, such as those defined in Sec. 2.3.3.1.

2.3.4 Data Validation Techniques

Given a dataset, one needs to use this resource for training, tuning, and testing the accuracy of the model [56]. Previously, we talked about how to divide the dataset into a *training set* and a *held-out test set*, and how the test set can be used to compute the generalization error of a model for final evaluation (after the training process has completed). Thus, a question that arises is how to choose the portion of the data for use in model selection and hyperparameter tuning; this portion of the data is referred to as *validation set*.

It is important not to use the test set in any way during training, even when making choices about the model hyperparameters, otherwise, we would partially be mixing the training and test data, and the resulting accuracy would be overly optimistic. Consequently, the validation data has to be constructed from the training set. However, it has to be different as well from the data that is used for model building (i.e., learning the weight and bias parameters) in order to reduce overfitting.

In the following, we distinguish between two data validation techniques: the *hold-out method* and *cross-validation*.

2.3.4.1 Hold-Out Method

In this method, the dataset is split into three different parts: training set, validation set, and test set, as shown in Fig. 2.8. Usually, this is done by first splitting the dataset into two parts: training set and test set. We then split the training data into two subsets with some percentage split (for example, 80:20). The largest of these two subsets is used to build and fit the model, i.e., to learn the parameters; this subset is referred to as *model-building set* (also called *training without validation set*[4]). The other subset is the *validation set*, used to compute the validation error *during* or *after* training. If the validation set is used during training, we save a copy of the model parameters every time the validation error improves. Note that the validation error does not need to be computed at each iteration; rather, it can be computed at a given frequency (for example, at every 10 iterations). When the learning algorithm converges (or after it is terminated), it returns the model parameters that correspond

[4]Strictly speaking, the training set comprises both the model-building set and validation set (if a validation set is used). However, by abuse of language, the model-building set is also referred to as the training set, and we do it here as well. In this case, the intended meaning of the wording "training set" can be determined from the context.

to the best validation error, for a given set of hyperparameters. On the other hand, if the validation set is not used during training, the validation error is computed after the learning algorithm converges (or is terminated).

During the tuning process, we need to choose the set of hyperparameters that results in the smallest validation error rather than that corresponding to the smallest training error. The validation error can be considered as an approximation to the generalization error. Since the validation set is used to guide the selection of hyperparameters, it will underestimate the test error (though usually by a smaller amount than the training error does) [55]. After the hyperparameter optimization process is completed, the generalization error can be computed using the test set.

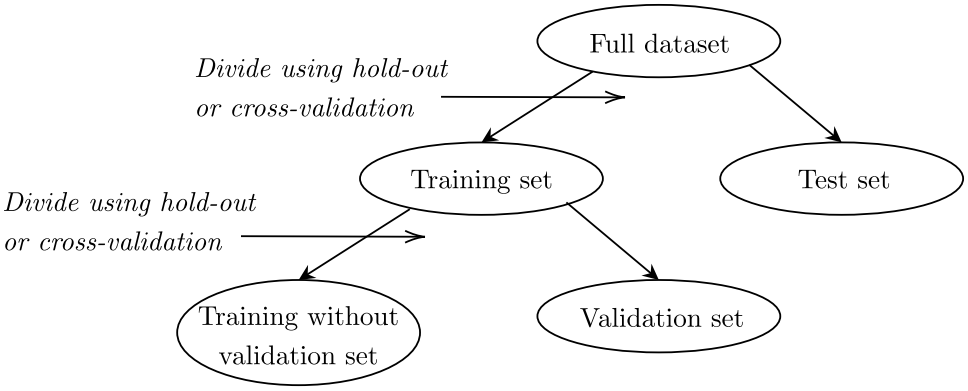


Figure 2.8: Hierarchical division into training, validation and test sets (adapted from Ref. [56]).

2.3.4.2 Cross-Validation

Another way to guide the hyperparameter search is *k-fold cross-validation*. This approach consists of splitting the dataset into k nonoverlapping subsets, also called folds. For each of the k folds, a model is trained using $k-1$ of the folds as training data, and the resulting model is evaluated on the remaining part of the data. At each round, a different fold is selected for testing. The test error may then be computed by taking the average test error across the k rounds. Although k-fold cross-validation is a popular method in ML/DL, it is not used in the current work.

2.3.5 Setup and Initialization Issues

2.3.5.1 Data Preprocessing

In many problems, the raw data may consist of variables that have different units (e.g., Joules, Kelvin, dimensionless, etc.), scales, or distributions. Unscaled data may result in a slow or unstable learning process, and lead to high generalization errors. In particular, unscaled input and target variables on regression-type problems can result in exploding gradients causing the learning process to fail. To stabilize the

learning process, it is recommended to preprocess the data before training the neural networks; this is called *data preprocessing*, *data preparation*, or *data scaling* [55, 56].

Three types of data scaling methods are usually employed in practice, namely *normalization*, *standardization*, and *mean-centering*. Consider observed data $\{y^{(i)}; i = 1, 2, \ldots, n\}$ for a scalar variable $y \in \mathbb{R}$. Normalization is a rescaling of the data from the original range such that all new values are within the range of 0 and 1. The normalization operation s_{N} is given by

$$s_{\mathrm{N}}(y) = \frac{y - y_{\min}}{y_{\max} - y_{\min}}, \tag{2.27}$$

where

$$y_{\min} = \min(y^{(i)}; i = 1, 2, \ldots, n), \tag{2.28a}$$

$$y_{\max} = \max(y^{(i)}; i = 1, 2, \ldots, n). \tag{2.28b}$$

Standardization involves rescaling the distribution of the data such that the mean of the new values is zero and the standard deviation is one. The standardization operation s_{S} is given by

$$s_{\mathrm{S}}(y) = \frac{y - \mu_y}{\sigma_y}, \tag{2.29}$$

where

$$\mu_y = \frac{\sum_i y^{(i)}}{n}, \tag{2.30a}$$

$$\sigma_y = \sqrt{\frac{\sum_i (y^{(i)} - \mu_y)^2}{n}}. \tag{2.30b}$$

Finally, mean-centering involves subtracting the variable's mean from all observations on that variable in the dataset such that the variable's new mean is zero. The mean-centering operation s_{C} is given by

$$y' = s_{\mathrm{C}}(y) = y - \mu_y, \tag{2.31}$$

where y' can be thought of as the fluctuations. It is noted that these operations can also be generalized to a snapshot variable $\boldsymbol{y} \in \mathbb{R}^m$, where $m > 1$. In this work, unless stated otherwise, the data used to train the AEs are normalized, the data used to train the regressors are standardized, and the data used to compute the POD modes are mean-centered.

2.3.5.2 Parameter Initialization

In this section, we will explain how to initialize neural network parameters effectively. Neural network models are trained using an optimization algorithm based on stochastic gradient-descent that incrementally changes the network parameters (i.e., weights and biases) in order to minimize a loss function, with the hope of obtaining

a set of parameters that is capable of making accurate predictions. The optimization algorithm is iterative by nature, and thus requires a starting point in the space of all possible parameter values from which to begin the learning process. The initial point can not only determine whether the algorithm converges at all, but also impact the training cost when learning does converge.

An important consideration of parameter initialization is *symmetry-breaking*: if all the parameters in the neural network are initialized to the same value, it can be difficult (or sometimes impossible) for the parameters to differ as the model is trained. Hence, initializing the model to small random values helps in breaking the "symmetry" and allows different parameters to learn independently of each other. It has also been shown that using more specific heuristics that use information such as the number of inputs to the node and type of activation function can further enhance the training process [55, 69].

Some modern initialization strategies for the network weights are described hereafter:

1. The *Glorot normal* initializer, also called *Xavier normal* initializer, draws each weight from a random normal distribution with a mean of zero and a standard deviation of $\sigma = \sqrt{2/(n_\mathrm{i} + n_\mathrm{o})}$, where n_i is the dimension of the input to the layer (i.e., number of nodes in the previous layer), and n_o is the dimension of the output from the layer (i.e., number of nodes in the current layer).

2. The *Glorot uniform* initializer, also called *Xavier uniform* initializer, draws each weight from a random uniform distribution in the range $[-s, s]$ for $s = \sqrt{6/(n_\mathrm{i} + n_\mathrm{o})}$.

3. The *He normal* initializer draws each weight from a random normal distribution with a mean of zero and a standard deviation of $\sigma = \sqrt{2/n_\mathrm{i}}$.

4. The *He uniform* initializer draws each weight from a random uniform distribution in the range $[-s, s]$ for $s = \sqrt{6/n_\mathrm{i}}$.

It is noted that the biases may be initialized to zero, as long as the asymmetry breaking is provided by the small random numbers in the weights.

Some studies have shown that the Glorot-based methods can be more effective when used to initialize networks that use the sigmoid or tanh activation function, whereas the He-based strategies can be more effective when used on networks with the ReLU activation function or its variants [69]. While this result can be used as a design guideline, one can also treat the initialization strategy as a hyperparameter, and eventually select the method that yields the best performance for a given problem.

2.3.5.3 Hyperparameter Tuning

Hyperparameters are pre-defined variables that determine the network structure and how the network is trained. Examples include the number of hidden layers, the number of neurons per hidden layer, the learning rate, the regularization coefficient,

the convolution filter width, the number of filters in a convolution layer, etc. The performance of a DL model can be highly dependent on the choice of hyperparameters. Therefore, hyperparameter tuning is a crucial part of the model training process.

For each set of hyperparameters, a learning algorithm, such as Adam (see Sec. 2.3.6), is executed to find the network's parameters (i.e., weights and biases) that minimize the loss function. A training error and validation error are then computed for each execution, and the set of hyperparameters that yields the lowest validation error (or at least reaches acceptable level of accuracy) is selected in the final model. The hyperparameters can be tuned *manually* or *automatically*.

Manual tuning is based on a trial-and-error approach, but one must understand the relationship between the hyperparameters and model performance to make appropriate selections. Although simple, manual tuning can be a tedious and time-consuming task for certain problems and use-cases.

On the other hand, automatic tuning involves the use of advanced software like `DeepHyper` [70], developed by the Argonne Leadership Computing Facility (ALCF), or `ModelSearch` [71], developed by Google. These software helps find the optimum set of hyperparameters in the explore search space with minimal human intervention. Although effective, these algorithms are compute-heavy and may require a lot of computing resources.

Note that, for all problems and use-cases considered in this work, manual tuning of the hyperparameters was deemed sufficient to build neural networks with acceptable levels of accuracy, and thus, automatic tuning was not used here.

2.3.6 Gradient-Based Optimization Strategies

The goal of a *learning algorithm*, also called *optimization algorithm*, is to find the parameters $\boldsymbol{\theta}$ such that the loss function J is minimized. Formally, we can write the optimization problem as:

Optimization problem

$$\text{Find } \boldsymbol{\theta}^* = \operatorname{argmin} J(\boldsymbol{\theta}) \text{ for given dataset and hyperparameters.} \tag{2.32}$$

Solving Problem (2.32) analytically is usually difficult and most of the time impossible, mainly because J is a highly nonlinear, nonconvex function of $\boldsymbol{\theta}$. Alternatively, one can attempt at solving the problem numerically using an iterative, gradient-based approach [55].

One basic approach is the method of *steepest descent*, also known as *classical gradient descent*, in which the gradient of the loss function is used to make parameter updates at each iteration. The model training is done in a series of *epochs*, where each epoch consists of one *forward pass* and one *backward pass* over all the samples in the training set. In the forward pass, the loss function is computed, whereas in the backward pass, the gradient is computed. The update rule is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{2.33}$$

where $\epsilon \in (0, 1]$ is the *learning rate*, a positive scalar determining the size of the step, i.e., the amount that the parameters are updated during training. For example, if ϵ is set to 0.01, the parameters will be updated by 1% of the computed error gradient at each epoch. The minus sign in Eq. (2.33) refers to the minimization part of the algorithm. In fact, the gradient represents the direction in which the loss function increases. In the algorithm, we aim to minimize the loss function, therefore at any given point, we need to move in the direction where the function decreases the most, i.e., in the opposite direction of the gradient, which explains the minus sign. Although simple, this method has some big flaws. It does not always point in the best direction of improvement because it uses steps of constant size. Moreover, the method computes the gradient based on the entire training set (known as *full-batch learning*, or simply as *batch learning*), which can slow down the calculations, especially when large training sets are considered.

Calculations can be accelerated by using *stochastic gradient descent* (SGD), which uses *mini-batch learning*. Here, the "true" gradient is no longer computed; rather, an approximation of the true gradient is estimated using randomly selected minibatches from the training set. The number of training samples included in each sub-epoch parameter change is defined as the *batch size*. As a result, the parameters are updated several times over the course of a single epoch. For example, if 1,000 training examples are considered, and the batch size is 50, then it will take 20 iterations to complete one epoch.

Although SGD is faster than steepest descent, the algorithm can sometimes be slow, especially when navigating through areas of high curvature or facing noisy gradients. This can be addressed by adding a *momentum* term [72] in the algorithm to dampen the oscillations in the search; this is referred to as *SGD with momentum*. Rather than using only the gradient of the current iteration to guide the search, the momentum method also accumulates the gradients of previous iterations (more specifically, an exponentially decaying moving average of past gradients) to determine the direction to go. The update rule at each sub-epoch becomes

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta}) \tag{2.34a}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}, \tag{2.34b}$$

where $\boldsymbol{v}$ is the velocity variable representing the momentum (in physics, momentum is mass times velocity, but a unit mass is assumed in the learning algorithm, hence $\boldsymbol{v}$ plays the role of momentum here), $\alpha \in [0, 1)$ is the momentum hyperparameter representing the percentage of the gradient retained at every iteration, and $J_{\text{minibatch}}$ is the loss function calculated over a minibatch. Equation (2.34a) has two parts. The first term represents the gradients retained from previous iterations, while the second represents the gradient from the current iteration.

In the methods described above, the same learning rate is applied to all parameter updates, which can sometimes be problematic, especially for non-stationary problems with very noisy or sparse gradients. To address this issue, *adaptive learning rate* algorithms can be used in which the learning rates of all network parameters are individually adapted [55]. Popular methods in this category include *AdaGrad* [73],

RMSProp, and *Adam* [74]. In particular, in Adam, two variables $\boldsymbol{s}$ and $\boldsymbol{r}$ are used to adapt the learning rate for each network parameter. The variable $\boldsymbol{s}$ corresponds to an estimate of the first-order statistical moment of the gradient, which represents a moving average of the gradients (i.e., the momentum term), while the variable $\boldsymbol{r}$ corresponds to an estimate of the (uncentered) second-order statistical moment of the gradient, which represents a moving average of the squared gradients, both with exponential weighting. Moreover, bias corrections are included in the estimates of both of these moments to account for their initialization at the origin. A pseudo-code for the Adam approach is given in Algorithm 1.

Algorithm 1: The Adam algorithm

Input: Global learning rate ϵ (also called initial learning rate, default: 0.001)
Input: Exponential decay rates for moment estimates, β_1 and β_2 in $[0, 1)$ (default: 0.9 and 0.999, respectively)
Input: Constant δ used for numerical stability (default: 10^{-7})
Input: Initial parameters $\boldsymbol{\theta}$ (obtained by applying some parameter initialization strategy, cf. Sec. 2.3.5.2)
Initialize first and second moment variables: $\boldsymbol{s} = 0, \boldsymbol{r} = 0$
Initialize time step: $t = 0$
while *stopping criterion not met* **do**
  Sample a minibatch of n examples from the training set
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{n} \nabla_{\boldsymbol{\theta}} \sum_i J(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Update time step: $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{g} \odot \boldsymbol{g}$
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \beta_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \beta_2^t}$
  Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations carried element-wise)
  Update network parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while
Return $\boldsymbol{\theta}$

Adam is generally regarded as an algorithm that performs fairly robustly [55]. For this reason, Adam is used as the learning algorithm in all problems and use-cases considered in this work.

2.3.7 GPU Acceleration

Graphic processing units were originally developed to accelerate the rendering of 3D graphics[5] by performing many matrix multiplications in parallel. Over time, they have evolved significantly, enhancing their capabilities. This allowed programmers to use the power of GPUs to dramatically accelerate additional workloads in many other fields including high performance scientific computing and ML.

Neural network implementations require many intensive computations involving vector, matrix, and tensor operations, as seen in Secs 2.2 and 2.3. For example, in a DFNN, each forward propagation involves many matrix-vector multiplications, and in a CNN, each forward propagation involves many convolutions. Further, a dimension that corresponds to the number of samples in the training set (or minibatch) is generally added to the involved arrays, making the operations be between multi-dimensional tensors. Similarly, in backpropagation, matrix (or tensor) multiplication occurs frequently to propagate the errors backward. As a result, GPUs are well-suited to parallelize DL applications.

Widely used DL frameworks such as `PyTorch` and `TensorFlow` (see Sec. 2.3.9) rely on GPU-accelerated libraries such as `cuDNN` [75] and `TensorRT` [76] to deliver high-performance model training and inference. Fortunately for the users, these libraries provide an abstraction of low-level GPU performance tuning and a programming interface that is easy to use with relatively limited rewriting of code. Hence, the changes required to convert a neural network code from its CPU version to a GPU version are often small [56].

The main difference between CPU and GPU is that a CPU is designed to handle a wide variety of tasks quickly, ranging from arithmetic and logical operations to running databases, but is limited in the concurrency of tasks that can be executed. On the other hand, a GPU is best suited for repetitive and highly parallel, but small, computing tasks. That is because a CPU consists of just a few cores (usually between two and eight) with high clock speed[6] and lots of cache memory[7] optimized for sequential serial processing, while a GPU uses thousands of smaller, but weaker, cores with high memory bandwidth[8] to achieve high level of parallelism.

The speedup that can be gained by training a neural network on a GPU architecture is dependent on many factors, including hardware, number of GPUs, number of parameters in the neural network, type of neural network, ML/DL library, etc. Ta-

[5] *3D rendering* is the process of converting three-dimensional models into two-dimensional images on a computer. This is used in many graphics applications, such as computer games.

[6] *Clock speed* is a measure of how many clock cycles a computing unit can perform per second. Note this is different from the number of *instructions per second* (IPS) and *floating-point operations per second* (FLOPS), which refer to how many tasks a computing unit can conduct per second. IPS and FLOPS can be derived by multiplying the number of *instructions per cycle* (IPC) with the clock speed.

[7] *Cache memory* is a temporary storage area that a computing unit can transfer data to and from easily; the data stored in a cache often corresponds to intermediate results.

[8] *Memory bandwidth* refers to the amount of data that can be transferred to and from memory per unit time.

ble 2.2 shows the speedups in training fully-connected and convolutional autoencoders using an NVIDIA Tesla K80 GPU (from the Cooley cluster at ALCF) over an Intel Core i7-10700T CPU @ 2.00 GHz (from a laptop) with the `Keras` API of `Tensorflow 2.4` for various problems considered in this thesis. These problems will be analyzed in greater detail in Chapters 6 and 7, but here we only report the speedups for the purpose of discussion. From this table, we can observe that the training is faster on the CPU when the neural network size is small, as in the one-dimensional Burgers equation. However, for larger neural network sizes, such as those in the ADR equation and A-M1 injector flow problems, a training speedup of up to five times is achieved using the GPU over the CPU. This highlights the effectiveness of GPUs in enhancing large-scale DL applications.

Table 2.2: Comparison of training time of autoencoders between one CPU and one GPU using the `Keras` API of `Tensorflow 2.4` for various problems considered in this thesis.

Problem (type of neural network)	Nb. epochs	Nb. params	CPU-secs	GPU-secs	Speedup
1D Burgers (FCAE)	1,000	39,140	78	272	0.29
1D Burgers (CAE)	1,000	5,014	236	603	0.39
ADR equation (FCAE)	1,000	1,854,702	1,130	626	1.81
ADR equation (CAE)	1,000	28,855	4,986	990	5.04
A-M1 injector flow (FCAE)	2,200	3,589,872	7,412	2,135	3.47

2.3.8 Error Metrics

Various error metrics are used in this work to analyze and evaluate the outputs of neural networks, especially those involving vector or tensor quantities. Let $\boldsymbol{q} \in \mathbb{R}^m$ be the true quantity and $\widetilde{\boldsymbol{q}} \in \mathbb{R}^m$ the predicted quantity. The variable $\boldsymbol{q}$ can represent, for example, a multivariate output of a DFNN in a regression problem, or a variable field (or snapshot) of an AE in a dimensionality reduction problem.

A *mean-squared error*[9] (MSE) is defined as:

$$\text{MSE}(\boldsymbol{q}, \widetilde{\boldsymbol{q}}) = \frac{1}{m}\sum_{j=1}^{m}(q_j - \widetilde{q}_j)^2, \tag{2.35}$$

where q_j and $\widetilde{q}_j$ are the jth component of $\boldsymbol{q}$ and $\widetilde{\boldsymbol{q}}$, respectively.

The error based on the L_2-*norm* is defined as

$$L_2 = \|\boldsymbol{q} - \widetilde{\boldsymbol{q}}\|_2 = \sqrt{\sum_{j=1}^{m}(q_j - \widetilde{q}_j)^2}. \tag{2.36}$$

[9]Note that Eq. (2.35) represents the MSE for an individual sample. To get the averaged loss over all samples in the set, or minibatch, we can write: $J = \frac{1}{n}\frac{1}{m}\sum_{i=1}^{n}\sum_{j=1}^{m}(q_j^{(i)} - \widetilde{q}_j^{(i)})^2$, where $q_j^{(i)}$ and $\widetilde{q}_j^{(i)}$ are the jth component of $\boldsymbol{q}^{(i)}$ and $\widetilde{\boldsymbol{q}}^{(i)}$, respectively, for the ith sample, and n is the total number of samples.

It reduces to absolute value when we plot the L_2 *error distribution* across spatial locations for a snapshot: $\sqrt{(q_j - \widetilde{q}_j)^2} = |q_j - \widetilde{q}_j|$ for all $j = 1, \ldots, m$.

The *relative error* based on the L_2-norm is defined as:

$$\epsilon_{\text{rel}} = \frac{\|\boldsymbol{q} - \widetilde{\boldsymbol{q}}\|_2}{\|\boldsymbol{q}\|_2} = \frac{\sqrt{\sum_{j=1}^{m}(q_j - \widetilde{q}_j)^2}}{\sqrt{\sum_{j=1}^{m} q_j^2}}. \tag{2.37}$$

The relative error provides an easily interpretable measure of the relative quality of the emulator. Multiplying the relative error by 100% results in a percentage error. Please note that the expressions above can be easily generalized to a matrix $\boldsymbol{Q} \in \mathbb{R}^{m_1 \times m_2}$.

2.3.9 Machine Learning Libraries

All the neural network models and training algorithms discussed in this chapter are supported by numerous free and open-source ML and DL frameworks and libraries, such as `TensorFlow` [25] and `Keras` [77], both developed by Google, `PyTorch` [26], developed by Facebook, and `Scikit-learn` [78], developed jointly by Google and INRIA. Some of these libraries allow us to take advantage of GPUs for more efficient training. In the following, a brief description is provided for each library.

`TensorFlow`[10] and `PyTorch` are both low-level ML and DL libraries, giving the user a lot of freedom and control to write custom neural network models. This flexibility, however, brings complexity to the code, often requiring the models to be implemented from scratch, which results in a large number of lines of code. Further, this often requires a deep understanding of DL concepts for model building and fitting. In general, the two frameworks provide comparable performance speed and accuracy. While they both operate on tensors, the main differences between them are in the way in which variables are assigned and the computational graphs are run [79]. Specifically, `Tensorflow` uses static computational graphs while `PyTorch` uses dynamic computational graphs.[11]

`Keras` is a high-level application programming interface (API) built on top of `Tensorflow`, which makes it a wrapper for DL purposes. It abstracts away many implementation details, making the code simpler and more concise than in `TensorFlow` and `PyTorch`, at the cost of limited flexibility and customization as well as slower performance. It is highly modular; neural network models are designed by connecting configurable building blocks together. `Keras` follows best practices to reduce the cognitive load for the user.

[10] Certain references in literature describe `TensorFlow` as a framework supporting both low-level and high-level APIs. Here, however, when we talk about `TensorFlow`, we are referring to its low-level implementation and distinguishing it from its high-level `Keras` API.

[11] In `TensorFlow 1.x`, we first have to define the graph, then execute it. Once the graph is defined, generally it cannot be changed during runtime. In `PyTorch`, however, we can change the structure of the graph during runtime, for example, by adding or removing nodes dynamically. Note that this difference can be overcome in newer versions of `TensorFlow` which provide a way of implementing dynamic graphs through a library called `TensorFlow Fold`.

`Scikit-learn` is a general ML library built on top of `NumPy` [80], the fundamental package for scientific computing in `Python`. It contains a great variety of utilities for general pre- and post-processing of data which can be useful in the development of DL models.

CHAPTER 3
COMPUTATIONAL FLUID DYNAMICS

Computational fluid dynamics (CFD) is the analysis of systems involving fluid flow, heat transfer, and associated phenomena such as phase change and chemical reactions, by means of numerical simulations on computers, or supercomputers. CFD plays an essential role in modeling and simulating many physical and engineering phenomena, such as weather, aerodynamics, and flows inside propulsion engines. Fluid flows are well described by a system of governing equations derived from basic conservation principles and laws, but solving these equations analytically is devilishly hard and in many cases impossible, which is why we resort to numerical approaches. In these approaches, the differential, or integral, equations are discretized and converted into algebraic equations, which are solved at only discrete points, or cells, in the domain. Various techniques are available for this conversion, including the finite-difference method (FDM), finite-volume method (FVM), and finite-element method (FEM).

In this chapter, we introduce some of the basic theoretical and numerical concepts of CFD. The underlying governing equations of fluid motion are presented in Sec. 3.1. Turbulence and its modeling are briefly discussed in Sec. 3.2. The LES approach, which is employed in the present work for modeling turbulent flows, is described in Sec. 3.3. Finally, FVM, which is used in the present work to discretize the equations, is explained in Sec. 3.4.

3.1 Governing Equations

The dynamics of fluid flow are governed by equations that describe the conservation of mass, momentum, and energy. Additional equations can also be used depending on the problem under consideration to describe turbulence transport and the transport of species as well as of scalars and passives. In this chapter, the set of equations governing an unsteady, viscous, single-phase, multi-species, compressible flow are presented.[1] The basic conservation principles and laws used to derive these equations are only briefly summarized here, but more detailed derivations can be found in classic textbooks such as White [81] and Anderson [82].

The equations presented below are written in *differential form* using the *index notation* (also known as *Einstein's summation convention*) in a Cartesian coordinate system.

[1] Multiphase flows are also considered in this thesis. To keep the discussion simple, the equations are presented for a single-phase flow in this chapter. The necessary modifications needed to adapt these equations to multiphase flows will be discussed later in this thesis.

Conservation of Mass: The Continuity Equation

Consider an infinitesimally small element moving with the flow with the same fluid particles inside it (i.e., an infinitesimal control mass[2]). Applying the fundamental physical principle of *mass conservation*, which states that mass cannot be created or destroyed, means that the time rate of change of the mass of the element is zero. The resulting differential equation is

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = 0, \tag{3.1}$$

where t and x_i are independent variables representing the time and spatial coordinates, respectively, ρ is the density, and u_i is the velocity in the direction x_i.

Conservation of Momentum: The Momentum Equations

Applying *Newton's second law* to the moving fluid element states that the net force acting on the element equals its mass times its acceleration. In the present work, body forces are neglected. Consequently, the forces result from the pressure p and viscous stress tensor $\boldsymbol{\tau}$.[3] The resulting differential equations[4] can be expressed as

$$\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j}. \tag{3.2}$$

For a Newtonian fluid, applying Stokes' hypothesis gives a simple expression for the viscous shear stress as a function of the velocity gradient. We get

$$\tau_{ij} = 2\mu (S_{ij} - \frac{1}{3} S_{kk} \delta_{ij}), \tag{3.3}$$

where μ is the dynamic viscosity and δ_{ij} denotes the Kronecker delta. S_{ij} is the strain-rate tensor expressed as

$$S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right). \tag{3.4}$$

Conservation of Energy: The Energy Equation

Applying the fundamental physical principle of *energy conservation*, which is basically the *first law of thermodynamics*, to the moving fluid element states that the rate of

[2]The governing equations can also be derived by applying basic conservation principles and laws to an infinitesimally small element fixed in space (i.e., a control volume). The final set of equations is equivalent to that obtained from the control mass model [82].

[3]In addition to pressure and viscous shear stress, surface tension (which is also a surface force) will be considered in the governing equations in Sec. 7.3.1.3 for modeling the multiphase flow inside the diesel injector nozzle.

[4]In theoretical fluid dynamics, the momentum equations are called the Navier-Stokes equations. In CFD, however, the system of equations comprising the continuity, momentum, and energy equations (species transport equations can also be included) is referred to as the Navier-Stokes equations or extended Navier-Stokes equations.

change of energy inside the element equals the sum of net flux of heat into the element and rate of work done on the element due to the involved forces. The resulting differential equation can be expressed as

$$\frac{\partial(\rho e_t)}{\partial t} + \frac{\partial[(\rho e_t + p)u_i]}{\partial x_i} = -\frac{\partial q_i}{\partial x_i} + \frac{\partial(u_i \tau_{ij})}{\partial x_j}, \tag{3.5}$$

where e_t is the mixture total energy per unit mass (also called mixture specific total energy) and $\boldsymbol{q}$ denotes the heat flux vector. The specific total energy e_t is defined as the sum of mixture specific internal energy and kinetic energy, given by

$$e_t = e + \frac{u_j u_j}{2}. \tag{3.6}$$

The specific internal energy e can be expressed as a function of the mixture specific enthalpy h, pressure p, and density ρ, as follows

$$e = h - \frac{p}{\rho}. \tag{3.7}$$

The mixture specific enthalpy h can be determined from the mixture concentration and partial-mass enthalpies h_n[5]

$$h = \sum_{n=1}^{N} Y_n h_n, \tag{3.8}$$

where N is the total number of species in the system, and Y_n is the mass fraction of the nth species.

Neglecting Dufour and Soret effects, the heat flux can be written as the sum of conductive fluxes, following Fourier's law, and enthalpy fluxes due to species diffusion, as follows

$$q_j = -\lambda \frac{\partial T}{\partial x_j} + \rho \sum_{n=1}^{N} h_n Y_n \mathcal{V}_{n,j}, \tag{3.9}$$

where λ is the heat conductivity, T is the temperature, and $\mathcal{V}_{n,j}$ is the diffusion velocity of the nth species in the direction x_j, which can be obtained from Fick's law.

Species Transport Equations

The species transport equations describe the rate of change of the species mass fractions in the flow and can be expressed as

$$\frac{\partial(\rho Y_n)}{\partial t} + \frac{\partial(\rho Y_n u_j)}{\partial x_j} = -\frac{\partial(\rho Y_n \mathcal{V}_{n,j})}{\partial x_j} + \dot{w}_n, \quad \text{for } n = 1, 2, \ldots, N, \tag{3.10}$$

[5]The partial-mass enthalpy of the nth species is defined as: $h_n = \left(\frac{\partial mh}{\partial m_n}\right)_{T,p,m_{j\neq n}}$, where m is the mixture mass and m_j the mass of the jth species. For ideal gases, the partial-mass enthalpy of a species is equal to its specific enthalpy. For non-ideal fluids, however, the two quantities are, in general, different from each other.

where $\dot{w}_n$ is the mass production rate of the nth species. Note that for non-reacting flows $\dot{w}_n$ is zero whereas for reaction flows $\dot{w}_n$ can be determined from the Arrhenius equation combined with suitable reaction mechanisms [83].

In addition to the equations presented above, it is necessary to use an equation of state (EOS) to establish relationships between fluid properties to close the system of equations. Examples of EOS include the ideal-gas EOS and real-fluid EOS (see Sec. 5.3.1). The choice of EOS is dependent on the fluid, or mixture, as well as on the physical conditions under consideration. Finally, we also need expressions to compute transport properties including the heat conductivity, viscosity, and diffusion coefficients.

3.2 Turbulence and its Modeling

Most flows in engineering practice are characterized by high Reynolds numbers, and thus are turbulent. In turbulent flow, the fluid motion undergoes irregular fluctuations, or mixing, in contrast to laminar flow, in which the fluid travels smoothly and in regular paths. Although turbulence has been studied for more than a hundred years, since Osborne Reynolds' experiments at the end of the 19th century, it is still a major challenge in fluid mechanics due to its strong nonlinear behavior [84]. In fact, it is regarded by many as *the last unsolved problem of classical physics*.[6]

In the turbulent regime, the governing equations become highly nonlinear, which complicates our ability to analyze and simulate fluid flows. The nonlinearity creates a wide and continuous range of flow features and scales, characterized by very large energy-containing eddies and very small dissipative eddies. The range of scales increases with the Reynolds number. As a result, at very high Reynolds numbers, the range of temporal and spatial scales becomes so large that the simultaneous representation of both large and small eddies makes for an intractable computational problem, even with today's computational resources and infrastructures.

The complexity of the flow description can be reduced to alleviate this computational burden. This can be done by either averaging or filtering the governing equations such that only the primary features, representing the large-scale motion of the flow, are resolved in detail. These operations, however, introduce additional terms, known as *unclosed terms*, into the equations, making the number of unknowns larger than the number of equations. Closure modeling becomes essential. The three basic methodologies that are widely used in turbulence simulation and research are:

1. *Direct numerical simulation* (DNS), which is the most straightforward approach. In DNS, the governing equations are discretized with enough resolution to resolve all scales of turbulent motion. It does not need the use of a turbulence or

[6] According to an almost certainly apocryphal story, applied mathematician Horace Lamb once said regarding the difficulty of explaining and studying turbulence in fluids: "I am an old man now, and when I die and go to Heaven there are two matters on which I hope enlightenment. One is quantum electro-dynamics and the other is turbulence of fluids. And about the former, I am really rather optimistic." [85].

closure model, thus giving highly accurate solution. DNS has been a very useful technique to gain insight into detailed dynamics of turbulent flows [86]. DNS also provides data for turbulence and turbulent combustion model development and validation. However, to resolve all scales of motion in a three-dimensional setting, the grid size has to be proportional to $Re_L^{9/4}$, with Re_L the Reynolds number associated with the characteristic length of the flow [87], and thus DNS is limited to relatively small Reynolds number flows and is generally infeasible for industrial applications.

2. *Reynolds-averaged Navier-Stokes* (RANS) simulation, in which only statistical quantities, i.e., the ensemble or time-averaged mean quantities, are predicted. The effect of all the scales of motion is modeled (except for Unsteady-RANS, or URANS for short, in which coherent motions are partially resolved [88]). Even though RANS is inherently less expensive and has moderate success in industrial applications, it suffers from one principal shortcoming that the model must account for a very wide range of scales. Based on Kolmogorov's hypothesis, at sufficiently high Reynolds number, the small-scale motions are statistically isotropic and tend to be universal to model [87]. The large-scale motions, however, are strongly dependent on the boundary conditions of the flow, thus it is not possible to develop a universal model for all turbulent flows [84].

3. *Large eddy simulation* (LES), which is a trade-off between the accuracy and computational cost of DNS and RANS; it is more accurate than RANS, but requires less computation time and fewer resources than DNS. In LES, energy-containing large-scale motions that are larger than a prescribed filter width are fully resolved while the effect of small-scale motions is modeled. Since the small-scale motions are more isotropic and universal, they can be modeled in a universal manner with much less *ad-hoc* adjustments in model coefficients compared with the turbulence models for RANS simulations.

Based on the above, LES seems promising to solve turbulent flow problems, especially for practical applications, and thus is used in this work to achieve turbulence closure.

3.3 Large Eddy Simulation

3.3.1 Filtering Operator

In LES, large-scale motions, which carry most of the kinetic energy are fully resolved, while small-scale motions, which are more universal and easier to model, are modeled with subgrid-scale (SGS) models. To separe the large-scale motions from the small-scale ones, a high-pass filtering operation in the physical domain, which is also a low-pass filtering operation in the wavenumber domain, is performed explicitly or

implicitly. The filtered (or resolved) part $\overline{f}(\boldsymbol{x},t)$ of a spatiotemporal variable $f(\boldsymbol{x},t)$ is defined by the relation[7]

$$\overline{f}(\boldsymbol{x},t) = \int_{-\infty}^{+\infty} f(\boldsymbol{x}',t)G(\boldsymbol{x}-\boldsymbol{x}')d\boldsymbol{x}', \tag{3.11}$$

where G is the *filter kernel* (also called *convolution kernel*), which determines the size and structure of the small scales. A filter has to satisfy a certain number of properties, such as conservation of constants, linearity, and commutation with derivation [88]. We can also write

$$f = \overline{f} + f', \tag{3.12}$$

where f' is the sub-filtered portion of f.

Different kernels can be used to perform the desired scale separation, and a thorough discussion on this topic can be found in classic textbooks such as Pope [87], Sagaut [88], and Garnier et al. [89]. In the present work, the filter kernel corresponds to a *Box* (or *top-hat*) filter. For one spatial dimension in physical space, it is given by

$$G(x-x') = \begin{cases} \frac{1}{\Delta} & \text{if } |x-x'| < \frac{\Delta}{2} \\ 0 & \text{otherwise} \end{cases}, \tag{3.13}$$

where Δ is the *filter width* (also called *cutoff scale*). For filtering in three spatial dimensions, the convolution kernel is obtained by tensorial extension of one-dimensional kernels, as follows

$$G(\boldsymbol{x}-\boldsymbol{x}') = \prod_{i=1}^{3} G_i(x_i - x_i'), \tag{3.14}$$

In Eq. (3.14), one can use either a different value of the filter width Δ_i in each direction x_i, or the same characteristic filter width Δ in all the directions.

For stretched non-uniform grids, there may be some advantage in varying the filter cutoff scale to adapt the structure of the solution better instead of setting it to a constant value. As a result, the filter width is usually taken equal to be of the same order as the local grid size.[8] In three-dimensional computations with grid cells of different length Δx, width Δy, and height Δz[9], the cutoff scale is often taken to be the cube root of the local grid cell volume, i.e., $\Delta(\boldsymbol{x}) = \sqrt[3]{\Delta x \Delta y \Delta z}$. In this formulation, any Box-filtered quantity is simply its spatial average in the local computational volume (or cell) $\mathrm{CV}(\boldsymbol{x})$, with $\Delta V(\boldsymbol{x})$ denoting the volume of the cell, i.e.

$$\overline{f}(\boldsymbol{x},t) = \frac{1}{\Delta V(\boldsymbol{x})} \int_{\mathrm{CV}(\boldsymbol{x})} f(\boldsymbol{x}',t)d\boldsymbol{x}'. \tag{3.15}$$

[7]The relation in Eq. (3.11) corresponds to spatial filtering. It is possible to extend this equation to include temporal filtering as well. However, this is not done here because most LES studies are based on spatial filtering [88, 89].

[8]Under this consideration, the commutation of filtering with derivation is no longer strictly valid. The commutation error, however, is usually neglected for moderately stretched grids, or can be accounted for by the SGS models.

[9]Note that, for stretched grids, the grid sizes Δx, Δy, and Δz are usually not constant and vary with the position $\boldsymbol{x}$.

In compressible flows, it is advantageous to apply *Favre-averaged filtering* (or, for short, *Favre filtering*) to the different variables. Favre filtering is a density-weighted approach which simplifies the filtered compressible governing equations by greatly reducing the number of unknowns or SGS terms and can be defined as

$$\widetilde{f} = \frac{\overline{\rho f}}{\overline{\rho}}. \tag{3.16}$$

Any variable f can be decomposed into a *Favre-averaged filtered* (or *resolved*) component $\widetilde{f}$ and a *sub-filtered* (or *unresolved*) component f'', as follows

$$f = \widetilde{f} + f''. \tag{3.17}$$

The $\widetilde{\square}$ operator is linear but does not commute with the derivative operators in space and time, unlike the $\overline{\square}$ operator. Consequently, special care must be taken while deriving the LES governing equations.

3.3.2 Favre-Filtered Governing Equations

Filtering the conservation equations of mass, momentum, energy, and species for a single-phase, compressible, multi-species flow results in the following set of LES equations, which are written using index notation in a Cartesian coordinate system. These are the equations that will be resolved in the numerical simulations. A detailed derivation of these equations can be found in classic textbooks such as Garnier et al. [89].

LES governing equations

$$\frac{\partial \overline{\rho}}{\partial t} + \frac{\partial (\overline{\rho}\widetilde{u}_i)}{\partial x_i} = 0, \tag{3.18a}$$

$$\frac{\partial (\overline{\rho}\widetilde{u}_i)}{\partial t} + \frac{\partial (\overline{\rho}\widetilde{u}_i\widetilde{u}_j)}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \frac{\partial (\overline{\tau}_{ij} - \tau_{ij}^{\text{sgs}})}{\partial x_j}, \tag{3.18b}$$

$$\frac{\partial (\overline{\rho}\widetilde{e}_t)}{\partial t} + \frac{\partial [(\overline{\rho}\widetilde{e}_t + \overline{p})\widetilde{u}_i]}{\partial x_i} = \frac{\partial}{\partial x_i}(-\overline{q}_i + \widetilde{u}_j\overline{\tau}_{ij} - \varphi_i^{\text{sgs}} + \sigma_i^{\text{sgs}}), \tag{3.18c}$$

$$\frac{\partial (\overline{\rho}\widetilde{Y}_n)}{\partial t} + \frac{\partial (\overline{\rho}\widetilde{Y}_n\widetilde{u}_i)}{\partial x_i} = \frac{\partial}{\partial x_i}(-\overline{\rho}\widetilde{Y}_n\widetilde{\mathcal{V}}_{n,i} - \phi_{n,i}^{\text{sgs}} + \theta_{n,i}^{\text{sgs}}) + \overline{\dot{w}}_n, \quad \text{for } n = 1, 2, \ldots, N. \tag{3.18d}$$

The filtered viscous stress matrix $\overline{\tau}_{ij}$ in Eq. (3.18b) and the filtered heat-flux vector $\overline{q}_i$ in Eq. (3.18c) are approximated as

$$\overline{\tau}_{ij} = 2\overline{\mu}\left(\widetilde{S}_{ij} - \frac{1}{3}\overline{S}_{kk}\delta_{ij}\right), \tag{3.19}$$

$$\overline{q}_i = -\overline{\lambda}\frac{\partial \widetilde{T}}{\partial x_i} + \overline{\rho}\sum_{n=1}^{N} \widetilde{h}_k \widetilde{Y}_n \widetilde{\mathcal{V}}_{n,i}. \tag{3.20}$$

Here, $\widetilde{S}_{ij}$ is the resolved strain-rate, given as $\widetilde{S}_{ij} = (\partial\widetilde{u}_i/\partial x_j + \partial\widetilde{u}_j/\partial x_i)/2$, and $\widetilde{\mathcal{V}}_{n,i}$ is the filtered diffusion velocity for the nth species, which can be obtained from Fick's law applied to the resolved variables.

All the subgrid-scale terms, denoted with the superscript "sgs", are unclosed, and thus, require specific modeling. These terms are:[10]

$$\tau_{ij}^{\text{sgs}} = \overline{\rho u_i u_j} - \overline{\rho}\widetilde{u}_i\widetilde{u}_j, \tag{3.21a}$$

$$\sigma_i^{\text{sgs}} = (\overline{u_j \tau_{ij}} - \widetilde{u}_j \overline{\tau}_{ij}), \tag{3.21b}$$

$$\varphi_i^{\text{sgs}} = \overline{\rho e_t u_i} - \overline{\rho}\widetilde{e}_t\overline{u}_i + (\overline{p u_i} - \overline{p}\widetilde{u}_i), \tag{3.21c}$$

$$\phi_{n,i}^{\text{sgs}} = \overline{\rho Y_n u_i} - \overline{\rho}\widetilde{Y}_n\overline{u}_i, \quad \text{for} \quad n = 1, 2, \ldots, N, \tag{3.21d}$$

$$\theta_{n,i}^{\text{sgs}} = \overline{\rho Y_n \mathcal{V}_{n,i}} - \overline{\rho}\widetilde{Y}_n\widetilde{\mathcal{V}}_{n,i}, \quad \text{for} \quad n = 1, 2, \ldots, N. \tag{3.21e}$$

The subgrid momentum flux term τ_{ij}^{sgs} (also called subgrid stress term), subgrid energy flux term φ_i^{sgs}, and subgrid species flux term $\phi_{n,i}^{\text{sgs}}$ result from filtering the corresponding convective terms. The subgrid viscous work term σ_i^{sgs} comes from correlations of the velocity field with the viscous stress tensor, and the subgrid species diffusive flux term $\theta_{n,i}^{\text{sgs}}$ comes from correlations of the species mass fractions with the diffusion velocities. The filtered species mass production rate $\overline{\dot{w}}_n$ is also unclosed.

In addition to the conservation equations, the equation of state must also be filtered. Filtering the equation of state gives us

$$\overline{p} = \overline{\rho}\widetilde{R}\widetilde{Z}\widetilde{T} + T^{\text{sgs}}, \tag{3.22}$$

where $\widetilde{Z}$ is the resolved compressibility factor, and T^{sgs} is a subgrid term defined as $T^{\text{sgs}} = (\overline{\rho R Z T} - \overline{\rho}\widetilde{T}\widetilde{Z}\widetilde{T})$, which is typically neglected. The mixture gas constant $\widetilde{R}$ can be computed as

$$\widetilde{R} = \sum_{n=1}^{N} \widetilde{Y}_n \frac{R_u}{W_n}, \tag{3.23}$$

where R_u is the universal gas constant.

Finally, the filtered specific total energy $\widetilde{e}_t$ can be approximated as

$$\widetilde{e}_t = \widetilde{h} - \frac{\overline{p}}{\overline{\rho}} + \frac{\widetilde{u}_i\widetilde{u}_i}{2} + k^{\text{sgs}}, \tag{3.24}$$

where k^{sgs} is the subgrid kinetic energy, defined as $k^{\text{sgs}} = (\widetilde{u_i u_i} - \widetilde{u}_i\widetilde{u}_i)/2$.

[10] Note that certain references in literature write Eqs. (3.21a)–(3.21e) in a different, but equivalent, form. For example, Eq. (3.21a) can be written as $\tau_{ij}^{\text{sgs}} = \overline{\rho}\widetilde{u_i u_j} - \overline{\rho}\widetilde{u}_i\widetilde{u}_j$.

3.3.3 Closure for SGS terms

The unresolved SGS terms described in the previous section need to be modeled appropriately to represent the influence of the small-scale turbulent motions onto the large-scale structures. As mentioned previously, a major advantage of the LES technique is that the small-scale motions have a universal character. Therefore, the modeling of the unresolved terms is rather simple compared to the RANS approach. Mathematically, closure modeling reduces the number of unknowns in the system by expressing the unresolved terms as a function of the resolved variables such that the number of unknowns becomes equal to the number of equations in the system. In other words, this allows obtaining a well-posed problem.

Many of the SGS closure models employed in literature are based on an eddy-viscosity type model [87]. Examples of these models are the algebraic Smagorinsky model [90], dynamic Smagorinsky model [90], and one-equation model [87]. In these models, the SGS stress tensor τ_{ij}^{sgs} is related to the resolved strain-rate tensor $\widetilde{S}_{ij}$ in analogy to the laminar Stokes' hypothesis. For compressible flows, the SGS stress tensor is modeled as

$$\tau_{ij}^{\text{sgs}} = -2\bar{\rho}\nu_t(\widetilde{S}_{ij} - \frac{1}{3}\widetilde{S}_{kk}\delta_{ij}) + \frac{2}{3}\bar{\rho}k^{\text{sgs}}\delta_{ij}, \tag{3.25}$$

where ν_t is the turbulent eddy viscosity. While the laminar dynamic viscosity ν is a fluid property, ν_t is a flow property. Other SGS models that do not use turbulent viscosity to model the subgrid stress matrix in the momentum equation include the dynamic structure model [91].

In the following, the algebraic Smagorinsky model, which is employed in Chapter 5, is briefly explained.

Algebraic Smagorinsky Model

In this model, the eddy viscosity ν_t and subgrid kinetic energy k^{sgs} are obtained algebraically to avoid solving additional transport equations [90]. This model is based on the *equilibrium hypothesis*, which assumes that the small-scale motions, which have shorter time scales than the large energy-carrying eddies, can adjust more rapidly to perturbations and recover equilibrium nearly instantaneously. Under this assumption, ν_t and k^{sgs} are modelled as

$$\nu_t = C_R(\Delta D^2)\sqrt{2\widetilde{S}_{ij}\widetilde{S}_{ij}}, \tag{3.26}$$

$$k^{\text{sgs}} = C_I(\Delta D^2)\widetilde{S}_{ij}\widetilde{S}_{ij}, \tag{3.27}$$

where C_R and C_I are dimensionless quantities representing the compressible Smagorinsky constants, and Δ is the filter width presented earlier. The Van-Driest damping function D is used to take into account the inhomogeneities near the wall [92] and is expressed as

$$D = 1 - \exp\left(-\left(\frac{y^+}{25}\right)\right), \tag{3.28}$$

where $y^+ = yu_\tau/\nu$ with u_τ the friction velocity.

Substituting Eqs. (3.26) and (3.27) into Eq. (3.25), we can obtain an expression for τ_{ij}^{sgs} as a function of the resolved variables.

The subgrid energy flux term φ_j^{sgs} is modeled based on the gradient transport assumption, as follows

$$\varphi_j^{\text{sgs}} = -\overline{\rho}\frac{\nu_t}{Pr_t}\left(\frac{\partial \widetilde{h}}{\partial x_j} + \widetilde{u}_i\frac{\partial \widetilde{u}_i}{\partial x_j} + \frac{1}{2}\frac{\partial k^{\text{sgs}}}{\partial x_j}\right), \tag{3.29}$$

where Pr_t represents the turbulent Prandtl number, which is set to a standard value of 1.0 in the present study. The SGS viscous work term σ_{ij}^{sgs} is neglected due to its small contribution to the total energy equation [84].

The convective species flux terms is typically modelled as

$$\phi_{n,i}^{\text{sgs}} = -\overline{\rho}\frac{\nu_t}{Sc_t}\frac{\partial \widetilde{Y}_n}{\partial x_i}, \tag{3.30}$$

where Sc_t is the turbulent Schmidt number. The SGS species diffusive flux term $\theta_{n,j}^{\text{sgs}}$ is usually neglected [93].

3.4 Finite-Volume Method

The filtered governing equations are solved numerically by means of a finite-volume method [94, 95]. The computational domain is subdivided into a finite number of small control volumes (also called computational cells) via a suitable grid. Each control volume is assigned a computational node corresponding to its center, and is enclosed by the control volume boundaries (also called control surfaces, or cell faces). An advantage of this method over other numerical approaches, such as the finite-difference method, is that it is easily formulated to allow for complex meshes and geometries. Furthermore, because the flux leaving a computational cell is identical to that entering the adjacent cell, the method is conservative.

In the following, we consider a structured grid (or a block-structured grid) such that each cell in three-dimensional is a hexahedron with six faces and eight corner points. For maintenance of conservation, the control volumes do not overlap; each cell face is unique to the two control volumes which lie on either side of it.

3.4.1 Compact Forms

The system of filtered governing equations (3.18a)–(3.18d) can be written in a compact form suited for CFD, as follows:

$$\frac{\partial \mathbb{Q}}{\partial t} + \frac{\partial(\mathbb{E} - \mathbb{E}_{\text{v}})}{\partial x} + \frac{\partial(\mathbb{F} - \mathbb{F}_{\text{v}})}{\partial y} + \frac{\partial(\mathbb{G} - \mathbb{G}_{\text{v}})}{\partial z} = \mathbb{H}, \tag{3.31}$$

where $\mathbb{Q}$ is the vector of conservative variables, which is given by Eq. (3.34), $\mathbb{E}$, $\mathbb{F}$ and $\mathbb{G}$ are the vectors of inviscid fluxes associated with the operators $\partial/\partial x$, $\partial/\partial y$,

and $\partial/\partial z$, respectively, and $\boldsymbol{\mathfrak{E}}_\mathrm{v}$, $\boldsymbol{\mathfrak{F}}_\mathrm{v}$ and $\boldsymbol{\mathfrak{G}}_\mathrm{v}$ are the vectors of viscous/turbulent fluxes associated with the operators $\partial/\partial x$, $\partial/\partial y$, and $\partial/\partial z$, respectively, and $\boldsymbol{\mathfrak{H}}$ is the source term vector, which is given by Eq. (3.35). We can also write the equations in a more compact form, as follows:

$$\frac{\partial \boldsymbol{\mathfrak{Q}}}{\partial t} + \nabla \cdot (\boldsymbol{F} - \boldsymbol{F}_\mathrm{v}) = \boldsymbol{\mathfrak{H}}, \tag{3.32}$$

where $\boldsymbol{F}$ is the matrix of inviscid fluxes, and $\boldsymbol{F}_\mathrm{v}$ is the matrix of viscous/turbulent fluxes. They are defined as

$$\boldsymbol{F} = \left(\boldsymbol{\mathfrak{E}} \mid \boldsymbol{\mathfrak{F}} \mid \boldsymbol{\mathfrak{G}}\right), \tag{3.33a}$$

$$\boldsymbol{F}_\mathrm{v} = \left(\boldsymbol{\mathfrak{E}}_\mathrm{v} \mid \boldsymbol{\mathfrak{F}}_\mathrm{v} \mid \boldsymbol{\mathfrak{G}}_\mathrm{v}\right). \tag{3.33b}$$

Explicit expressions for $\boldsymbol{F}$ and $\boldsymbol{F}_\mathrm{v}$ are given by Eqs. (3.36) and (3.37), respectively, from which we can also obtain expressions for the vectors $\boldsymbol{\mathfrak{E}}$, $\boldsymbol{\mathfrak{F}}$, $\boldsymbol{\mathfrak{G}}$, $\boldsymbol{\mathfrak{E}}_\mathrm{v}$, $\boldsymbol{\mathfrak{F}}_\mathrm{v}$ and $\boldsymbol{\mathfrak{G}}_\mathrm{v}$. In these expressions, the variables $\widetilde{u}$, $\widetilde{v}$ and $\widetilde{w}$ denote the components of the resolved velocity vector.

$$\mathfrak{Q} = \begin{bmatrix} \overline{\rho} \\ \overline{\rho}\widetilde{u} \\ \overline{\rho}\widetilde{v} \\ \overline{\rho}\widetilde{w} \\ \overline{\rho}\widetilde{e}_t \\ \overline{\rho}\widetilde{Y}_n \end{bmatrix}. \tag{3.34}$$

$$\mathscr{H} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \overline{\dot{w}}_n \end{bmatrix}. \tag{3.35}$$

$$\boldsymbol{F} = \begin{bmatrix} \overline{\rho}\widetilde{u} & \overline{\rho}\widetilde{v} & \overline{\rho}\widetilde{w} \\ \overline{\rho}\widetilde{u}^2 + \overline{p} & \overline{\rho}\widetilde{u}\widetilde{v} & \overline{\rho}\widetilde{u}\widetilde{w} \\ \overline{\rho}\widetilde{u}\widetilde{v} & \overline{\rho}\widetilde{v}^2 + \overline{p} & \overline{\rho}\widetilde{v}\widetilde{w} \\ \overline{\rho}\widetilde{u}\widetilde{w} & \overline{\rho}\widetilde{v}\widetilde{w} & \overline{\rho}\widetilde{w}^2 + \overline{p} \\ (\overline{\rho}\widetilde{e}_t + \overline{p})\widetilde{u} & (\overline{\rho}\widetilde{e}_t + \overline{p})\widetilde{v} & (\overline{\rho}\widetilde{e}_t + \overline{p})\widetilde{w} \\ \overline{\rho}\widetilde{u}\widetilde{Y}_n & \overline{\rho}\widetilde{v}\widetilde{Y}_n & \overline{\rho}\widetilde{w}\widetilde{Y}_n \end{bmatrix}. \tag{3.36}$$

$$\boldsymbol{F}_{\mathrm{v}} = \begin{bmatrix} 0 & 0 & 0 \\ \overline{\tau}_{xx} - \tau_{xx}^{\mathrm{sgs}} & \overline{\tau}_{xy} - \tau_{xy}^{\mathrm{sgs}} & \overline{\tau}_{xz} - \tau_{xz}^{\mathrm{sgs}} \\ \overline{\tau}_{xy} - \tau_{xy}^{\mathrm{sgs}} & \overline{\tau}_{yy} - \tau_{yy}^{\mathrm{sgs}} & \overline{\tau}_{yz} - \tau_{yz}^{\mathrm{sgs}} \\ \overline{\tau}_{xz} - \tau_{xz}^{\mathrm{sgs}} & \overline{\tau}_{yz} - \tau_{yz}^{\mathrm{sgs}} & \overline{\tau}_{zz} - \tau_{zz}^{\mathrm{sgs}} \\ \widetilde{u}\overline{\tau}_{xx} + \widetilde{v}\overline{\tau}_{xy} + \widetilde{w}\overline{\tau}_{xz} - \overline{q}_x - \varphi_x^{\mathrm{sgs}} + \sigma_x^{\mathrm{sgs}} & \widetilde{u}\overline{\tau}_{xy} + \widetilde{v}\overline{\tau}_{yy} + \widetilde{w}\overline{\tau}_{yz} - \overline{q}_y - \varphi_y^{\mathrm{sgs}} + \sigma_y^{\mathrm{sgs}} & \widetilde{u}\overline{\tau}_{xz} + \widetilde{v}\overline{\tau}_{yz} + \widetilde{w}\overline{\tau}_{zz} - \overline{q}_z - \varphi_z^{\mathrm{sgs}} + \sigma_z^{\mathrm{sgs}} \\ -\overline{\rho}\widetilde{\mathscr{V}}_{n,x}\widetilde{Y}_k - \phi_{n,x}^{\mathrm{sgs}} + \theta_{n,x}^{\mathrm{sgs}} & -\overline{\rho}\widetilde{\mathscr{V}}_{n,y}\widetilde{Y}_k - \phi_{n,y}^{\mathrm{sgs}} + \theta_{n,y}^{\mathrm{sgs}} & -\overline{\rho}\widetilde{\mathscr{V}}_{n,z}\widetilde{Y}_k - \phi_{n,z}^{\mathrm{sgs}} + \theta_{n,z}^{\mathrm{sgs}} \end{bmatrix}. \tag{3.37}$$

3.4.2 Approximation of Volume and Surface Integrals

Equation (3.32) is integrated over a computational cell CV enclosed by the surface S in the physical domain, as follows

$$\int_{\mathrm{CV}} \frac{\partial \mathbb{Q}}{\partial t} dV + \int_{\mathrm{CV}} [\nabla \cdot (\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}})] dV = \int_{\mathrm{CV}} \mathbb{H} dV = 0. \tag{3.38}$$

Using the Gauss divergence theorem, we get

$$\int_{\mathrm{CV}} \frac{\partial \mathbb{Q}}{\partial t} dV + \int_{S} [(\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) \cdot \boldsymbol{n}] dS = \int_{\mathrm{CV}} \mathbb{H} dV, \tag{3.39}$$

where $\boldsymbol{n}$ is the outward unit vector normal to infinitesimal surface element dS.

In Eq. (3.39), the volume integrals can be conveniently evaluated using the flow values at the cell center (i, j, k). Thus, we can write the time derivative and source terms as

$$\int_{\mathrm{CV}} \frac{\partial \mathbb{Q}}{\partial t} dV \approx \left. \frac{\partial \mathbb{Q}}{\partial t} \right|_{i,j,k} \Delta V, \tag{3.40a}$$

$$\int_{\mathrm{CV}} \mathbb{H} dV \approx \mathbb{H}|_{i,j,k} \Delta V, \tag{3.40b}$$

where ΔV is the volume of the cell.

Equation (3.39) also contains a surface integral term representing the fluxes; this term requires careful treatment. The net flux through the cell boundary is the sum of integrals over all the cell faces (six faces in 3D), which gives

$$\int_{S} [(\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) \cdot \boldsymbol{n}] dS = \sum_{\text{all cell faces}} \int_{S_l} [(\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) \cdot \boldsymbol{n}_l] dS, \tag{3.41}$$

where l is an index looping over all the cell faces (i.e., $l = 1, 2, \ldots, 6$ in 3D).

In the right-hand side of Eq. (3.41), each surface integral can be evaluated using the *mid-point rule*, i.e., the integral is approximated as a product of the integrand at the cell-face center (which is itself an approximation to the mean value over the surface) and the cell-face area, which gives

$$\sum_{\text{all cell faces}} \int_{S_l} [(\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) \cdot \boldsymbol{n}_l] dS = \sum_{\text{all cell faces}} [(\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) \cdot \boldsymbol{n}_l] \Delta S_l, \tag{3.42}$$

where ΔS_l is the area of cell face l, and $\boldsymbol{n}_l$ is the unit vector normal to cell face l.

Substituting back into Eq. (3.39), we obtain

$$\left. \frac{\partial \mathbb{Q}}{\partial t} \right|_{i,j,k} \Delta V + \sum_{\text{all cell faces}} [(\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) \cdot \boldsymbol{n}_l] \Delta S_f = \mathbb{H}|_{i,j,k} \Delta V. \tag{3.43}$$

In the finite-volume method, all information is available at the center of each computational cell (once the equations are solved numerically). However, cell-faced quantities

are needed by the flux terms in Eq. (3.43) and therefore, some form of interpolation or approximation is needed to evaluate flow variables at the cell faces as a function of cell-centered quantities. The accuracy of the interpolation or approximation controls the overall spatial accuracy of the numerical integration scheme. Further, depending on the manner in which the flux terms are evaluated, a wide variety of central and upwind difference schemes can be used. In addition to these terms, we need expressions for the geometric quantities $\boldsymbol{n}_l$, ΔS_f, and ΔV in order to solve Eq. (3.43), which can be non-trivial, especially for complex grids. Regarding time integration, it can be performed using a multi-stage or multi-step time-stepping method. A local time step can also be used to accelerate convergence. Other important considerations for CFD include initial and boundary conditions, numerical stability, code parallelization, artificial dissipation, etc. These are not discussed here for the sake of brevity. For more details, the reader is referred to the Ph.D. theses of Zong [96] and Huo [97], or to the `Converge` manual [98].

3.5 CFD Solvers

Various CFD codes were employed in this thesis to generate databases for DL model development and validation, as shown in Table 3.1. The discussion above focused mainly on single-phase compressible flows. Differences and specific considerations for each solver are provided in its corresponding section in Part II.

Table 3.1: List of CFD solvers employed in this thesis. The symbol "♮" refers to codes written from scratch in this work.

Code	Type	Case	Numerics	Sec.
`PMBFS`	In-house	GCLSC rocket injector flow	FVM	5.3.2.1
`CONVERGE`	Commercial	A-M1 diesel injector flow	FVM	7.3.1.3
`One-D ThermoCode`	In-house	Quasi-1D counterflow diffusion flame	FVM	5.3.2.2
♮	–	1D Burgers equation	FDM	6.4.1
♮	–	2D ADR equation	FDM	6.5.1

CHAPTER 4
FUEL INJECTION PHENOMENA AND MODELING PRACTICES

Injection, mixing, and combustion under high pressures are very common in many high-performance propulsion and power generation systems. In general, such extreme physical conditions allow to extract the maximum amount of energy from the fuel and convert it to thrust or mechanical power. Examples of engines operating under high pressures are diesel engines, gas turbines, and liquid-propellant rocket engines (LPREs). Here, we focus on diesel engines and LPREs.

This chapter is organized as follows. Section 4.1 provides a classification of combustion engines. Section 4.2 describes the different states of matter for fluids. Finally, Secs. 4.3 and 4.4 present some of the basic physics and engineering concepts related to and modeling approaches for injection and mixing in diesel engines and LPREs, respectively.

4.1 Engine Classification

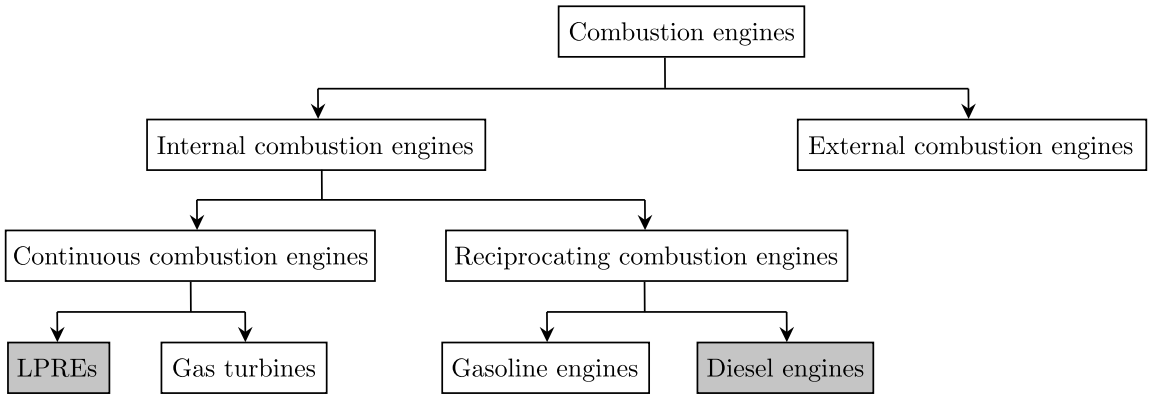


Figure 4.1: Classification of combustion engines. Note that this classification is not exhaustive.

Diesel engines and rocket engines are a type of *internal combustion engines* (ICEs), in which thurst or mechanical power is produced from the heat released by burning or oxidizing the fuel inside the engine. This is different from *external combustion engines*, such as steam engines, which derive its heat from fuel consumed outside the engine. ICEs can be divided into two groups [99]:

1. *Continuous combustion engines*, which are characterized by a steady[1] flow of

[1]Here, the word "steady" does not necessarily mean steady-state. Instead, it is a flow of fuel and oxidizer that is continuously supplied to the combustion chamber rather than periodically or in a cyclic manner. In practice, this flow can be quasi-steady or unsteady.

fuel and oxidizer into the engine and a stable flame. There are mainly two types of continuous combustion engines. These engines differ in how the oxidizer is obtained:

   (a) *Gas turbines*, in which oxygen is obtained from the surrounding air to burn the fuel.

   (b) *LPREs*, in which the oxygen is carried by the vehicle itself to burn the fuel.

2. *Reciprocating combustion engines*, which are characterized by periodic injection and ignition of fuel and air. There are mainly two kinds of reciprocating combustion engines currently in production. These engines differ in how the fuel is supplied and ignited:

   (a) *Gasoline engines* (also called *petrol engines*, or *spark-ignition engines*), in which fuel is mixed with air, compressed by a piston and then ignited by sparks from a *spark plug*, causing combustion.

   (b) *Diesel engines* (also called *compression-ignition engines*), in which the air alone is compressed first, and then the fuel is sprayed through a fuel injector nozzle into the hot compressed air causing it to ignite. Since only air is compressed in this type of engine, the air temperature in the cylinder increases to such a high degree that atomized diesel fuel injected into the combustion chamber evaporates and ignites spontaneously. Diesel engines have better fuel efficiency than gasoline engines, however, they produce more noise and vibrations, as well as higher levels of soot and nitrogen compounds.

4.2 Fluid State Physics

Depending on the local value of pressure and temperature, a fluid can exist in different states:

1. *Liquid*, when the pressure is high and temperature is low. At this condition, from a microscopic view, the fluid particles are close to each other and do not agitate much. The intermolecular attractions are important and particles tend to form temporary clusters. Thermodynamically, state variables can be related using an incompressible EOS (if the density is constant) or a barotropic compressible EOS (if the density is changing).

2. *Ideal gas*, when the temperature is high and pressure is low. At this condition, the fluid particles are very far from each other and in a state of random thermal motion. The intermolecular forces have an insignificant effect on the motion of the individual particles. Thermodynamically, state variables are related via the ideal gas EOS.

3. *Supercritical*, when both the temperature and pressure exceed the critical point. Because surface tension and enthalpy of vaporization, which indicate the liquid-gas boundary, approach zero, there is no longer the possibility of a two-phase region, but instead a single-phase exists. At this condition, fluid properties possess liquid-like density, gas-like diffusivity, and pressure-dependent solubility. In addition, isothermal compressibility and specific heat increase significantly, and fluid properties and their gradients vary continuously. The departure from ideal gas behavior becomes even more significant when the fluid state approaches the critical condition. In the vicinity of the critical point, the thermophysical properties of fluids exhibit anomalous variations and are very sensitive to both temperature and pressure, a phenomenon commonly referred to as *near critical enhancement*. Those phenomena, coupled with intense turbulence and chemical reaction, have significant impact on the dynamics of a given combustion system. Thermodynamically, state variables can no longer be described using the ideal gas EOS. Instead, a real-fluid EOS *must* be used to account for the thermodynamic nonidealities and transport anomalies. As we will see later in Chapter 5, the evaluation of the state equations is an expensive procedure in the simulation of supercritical flows, especially when the system is composed of many chemical species.

The critical properties of the major chemical species considered in this thesis are given in Table 4.1, where p_c, T_c, and v_c are critical values of pressure, temperature, and molar specific volume, respectively, and w is the acentric factor.

Table 4.1: Critical properties (p_c, T_c and v_c) and acentric factor w for the major chemical species considered in this study.

Species	p_c [bar]	T_c [K]	v_c [cm^3/mole]	ω
Oxygen (O_2)	50.43	154.58	73.37	0.025
n-decane ($C_{10}H_{22}$)	21.1	617.7	624	0.49
n-propylbenzene (C_9H_{12})	32	638.35	440	0.345
n-propylcyclohexane (C_9H_{18})	28	639	472.5	0.258
n-dodecane ($C_{12}H_{26}$)	18.1	658.2	754	0.574
Hydrogen (H_2)	12.97	33.25	65	-0.216
Water (H_2O)	220.64	647.14	55.95	-0.344
Nitrogen (N_2)	33.98	126.2	90.10	0.037

4.3 Diesel Engines

4.3.1 Engine Components and Operating Cycles

A diesel engine is composed of several components, including a fuel system, air intake system, exhaust system, lubricating system, and cooling system. In particular, the

fuel system comprises the fuel-injection system and combustion chamber. The fuel-injection system is mainly composed of the fuel tank, delivery pipe, filter, injection pump, and injector nozzle, while the combustion chamber is mainly composed of the cylinder, piston, crankshaft, and connecting rod. For more details on the engine components, the reader is referred to Heywood [100].

Most diesel engines operate using a *four-stroke cycle*,[2] where the piston moves back and forth four times in a cylinder per power cycle, or equivalently the crankshaft rotates two times during the cycle [100, 101], as shown schematically in Fig. 4.2. The four strokes are:

1. An *intake stroke*, which starts with the piston at position top-center (TC) and ends with the piston at position bottom-center (BC). In this stroke, a fresh mixture is introduced into the cylinder through the intake valve, which opens shortly before the stroke starts and closes after it ends.

2. A *compression stroke*, where the piston moves upward to compress the mixture in preparation for ignition. The compression ratios are usually in the range 14:1 to 22:1 in a diesel engine (or in the range 8:1 to 12:1 in a gasoline engine). Both the intake and exhaust valves are closed during this stroke.

3. A *power stroke* (also called *expansion stroke*), which represents the start of the second revolution of the crankshaft, which has already completed a full 360^o revolution. While the piston is at TC, combustion is initiated by heat generated by high compression in a diesel engine (or by a spark plug in a gasoline engine), forcefully returning the piston to BC. This stroke converts the heat released by combustion into mechanical work to turn the crankshaft.

4. An *exhaust* stroke, where the piston returns from BC to TC while the exhaust valve is open to expel the burned gases outside the cylinder. Just after the piston reaches TC, the exhaust valve closes and the cycle starts again.

4.3.2 Types of Diesel Engines

We can distinguish two types of diesel engines:

1. *Direct-injection engines*, in which the fuel is injected directly into the main combustion chamber.

2. *Indirect-injection engines*, in which the fuel is injected into the prechamber followed by the main combustion chamber.

Here, the focus is on direct-injection engines.

[2]Note, however, that engines used in small propulsion applications, such as dirk bikes and outboard motors, usually work by completing a power cycle with only two strokes of the piston (or one revolution of the crankshaft).

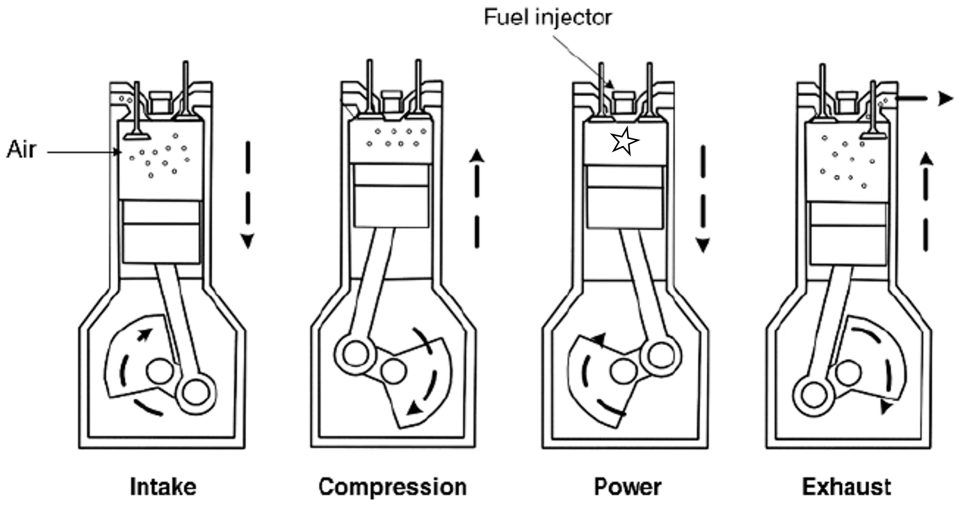


Figure 4.2: Schematic of the four-stroke cycle in a diesel engine (adapted from Ref. [102]).

4.3.3 Fuel Injector Nozzles

The fuel injector nozzle is an integral part of the diesel engine. It consists of a spring-loaded needle valve whose function is to open and close at the correct moment to inject the fuel into the cylinder of the engine with a large pressure differential across the nozzle orifice. The cylinder pressure at injection is typically in the range 20 to 100 bar, while fuel injection pressure is in the range 200 to 1,700 bar, depending on the engine size. These large pressure differences across the injector nozzle are needed to increase the velocity of the injected liquid fuel jet as it enters the chamber to enhance atomization, thus enabling rapid evaporation and improving fuel-air mixing and combustion [100].

Injector nozzles are threaded or clamped into the cylinder head, one for each cylinder (see Fig. 4.2). They are composed of several parts, as shown in Fig. 4.3. These include:

1. An *injector body*, which is the shell of all other parts of the injector nozzle. The inner portion of the injector body embeds an accurately designed passage through which the fuel is introduced from the fuel pump (not shown in Fig. 4.3) into the nozzle.

2. A *needle valve*, which controls the fuel rate into the cylinder chamber. When the needle is open, the fuel flows down through the fuel passage in the injector body toward the nozzle orifice(s). When injection is not occurring, the needle is closed and forced against the valve seat (usually by a spring) to prevent dribbles from the nozzle that can cause undesirable effects such as increase in unburned hydrocarbon emissions. The needle vertical displacement along the nozzle axis is called *needle lift*.

3. A *sac volume*, which is a small cavity in the bottom of the injector nozzle. It is important to keep the amount of fuel left in the sac as small as possible to avoid any fuel flowing into the cylinder after injection is over.

4. One or more *orifices* (also called *holes*) through which the fuel is injected into the cylinder of the engine.

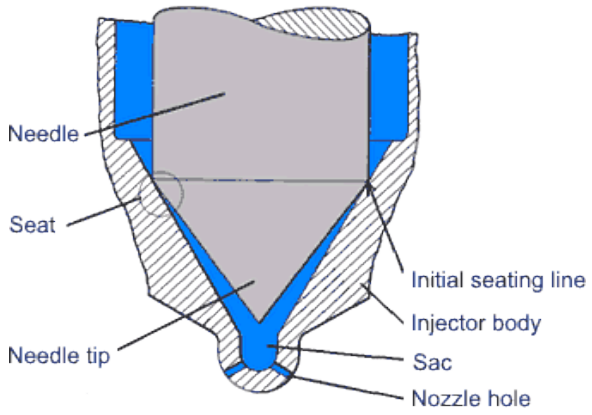


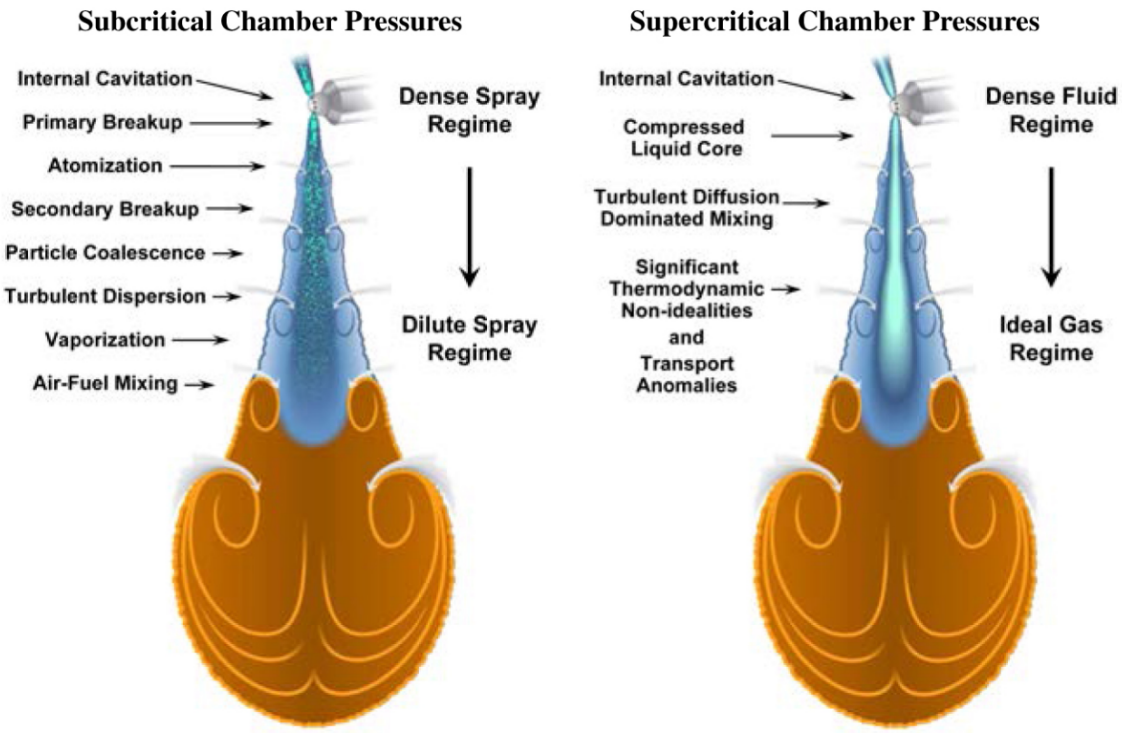
Figure 4.3: Schematic of basic diesel fuel injector nozzle. The blue zones indicate the regions through which the fuel can flow (reproduced from Ref. [103]).

4.3.4 Liquid Jet Behavior

There are two distinctly different sets of processes the liquid jet exhibits as it exits the nozzle, depending on the chamber pressure, as illustrated in Fig. 4.4.

4.3.4.1 Subcritical chamber pressures

At subcritical cylinder pressures, the classical situation exists where a well-defined interface separates the injected liquid from ambient gases due to the presence of surface tension. As the jet exits the nozzle, it will start to break up, as shown in Fig. 4.4(a). The first breakup of the liquid is called *primary breakup*, which results in large ligaments and droplets. The primary breakup is governed by the flow conditions of the liquid inside the injector nozzle, such as turbulence and cavitation (cf. Sec. 4.3.5). After the primary breakup, the droplets will experience another breakup, referred to as *secondary breakup*, caused by the difference in velocity between the droplets and surrounding gas. As a result, the droplets become smaller in size. Droplets within the spray may also change in size due to interactions with one another via *collisions*. Collision can lead to either further breakup of droplets into smaller ones or coalescence to form larger droplets. In the *dense spray region*, particularly near the nozzle, collisions occur more frequently due to the relatively short distances between the droplets. Further downstream as the spray becomes more dilute, i.e.,



(a) (b)

Figure 4.4: Schematic of fuel injection system with different spray sub-processes for: (a) subcritical chamber pressures, and (b) supercritical chamber pressures (reproduced from Ref. [104]).

in the *dilute spray region*, the occurrence of collisions becomes less probable [105]. Important parameters that quantify the fuel spray behavior are the spray cone angle, breakup length, spray tip penetration, and droplet size distribution [100].

The high air temperature in the engine cylinder causes the small droplets to evaporate, producing vapor which mixes with the air to form a combustible mixture which ignites spontaneously due to elevated pressures and temperatures; these mechanisms are referred to as *droplet evaporation*, *fuel-air mixing*, and *ignition*, respectively. During this process, pollutants can be formed as well because of non-ideal processes occurring simultaneously, such as incomplete combustion of diesel fuel, reactions between mixture components under high temperature and pressure, and combustion of engine lubricating oil. Common pollutants include unburned hydrocarbons, carbon monoxide, nitrogen oxides, and particulate matter.

4.3.4.2 Supercritical chamber pressures

At supercritical chamber pressures, the situation becomes quite different. Under these conditions, a distinct gas-liquid interface does not exist, as previously explained in Sec. 4.2. Instead, the injected liquid jet undergoes a transcritical change of state as

interfacial fluid temperatures rise above the critical temperature of the local mixture. For this situation, effects of surface tension and two-phase flow become diminished. The lack of these intermolecular forces promotes diffusion-dominated mixing processes prior to atomization and the respective jet vaporizes, forming a continuous fluid in the presence of exceedingly large thermophysical gradients in a manner that is markedly different from the classical situation explained above.

4.3.5 Cavitation in Fuel Injector Nozzles

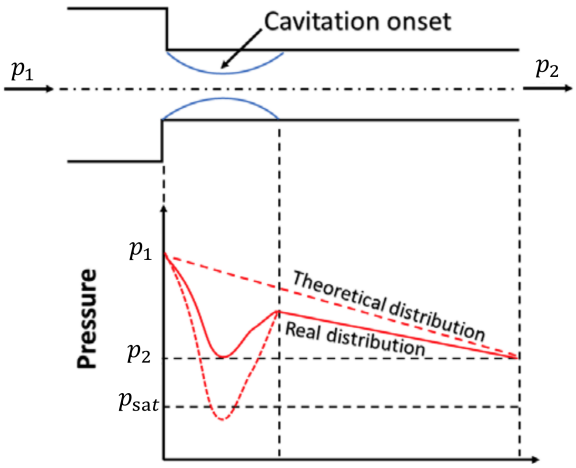


Figure 4.5: Schematic illustration of cavitation formation inside a nozzle orifice (adapted from Refs. [106, 107]).

Cavitation is the process in which the vapor phase of a liquid is generated due to a drop in the pressure at a given temperature. Cavitation is different from *boiling*, which is driven by an increase in the liquid temperature at a given pressure.

Cavitation occurs frequently in fuel injector nozzles. As the fuel flows from the sac region to the orifice, its velocity changes due to losses associated with the area constriction and sharp edges at the orifice inlet. As a result, the fuel velocity increases while the static pressure decreases. In addition to this, the boundary layer tends to separate from the orifice wall, leading to a recirculating flow region near the orifice inlet. At certain locations in the nozzle orifice, when the pressure of the liquid becomes smaller than the saturation pressure (at a given temperature), cavitation is produced and vapor bubbles are formed inside the orifice, as shown schematically in Fig. 4.5.

Typically, four regimes of cavitation are identified based on the cavitation distribution inside the nozzle [107, 106]: (1) *no cavitation*, in which cavitation bubbles are not formed inside the nozzle orifice, (2) *cavitation inception*, in which the cavitation zone appears at the orifice inlet, (3) *developing cavitation*, in which the cavitation zone appears at the orifice inlet and extends further downstream, and (4) *supercavitation*, in which the cavitation zone extends from the inlet up to or close to the nozzle exit. To characterize the global extent of cavitation, a cavitation number Ca

is introduced, as follows

$$Ca = \frac{p_1 - p_{\text{sat}}}{p_1 - p_2}, \tag{4.1}$$

where p_1 is the injection (or upstream) pressure, p_2 is the ambient (or back) pressure, and p_{sat} is the saturation pressure (or vapor pressure) of the fluid. In general, a large value of the cavitation number suggests non-cavitating flow while a small Ca corresponds to strong cavitating flow [106].

Another important quantity for cavitating flows is the nozzle discharge coefficient C_{d}, which represents the hydraulic resistance that the nozzle orifice imposes upon the flow. This parameter is useful for determining the irrecoverable losses through the nozzle orifice. In particular, when C_{d} is reduced, the orifice flow efficiency is reduced as well. C_{d} can be defined as

$$C_{\text{d}} = \frac{\dot{m}}{\dot{m}_{\text{ideal}}}, \tag{4.2}$$

where $\dot{m}$ is the actual mass flow rate through the nozzle, and $\dot{m}_{\text{ideal}}$ is the ideal mass flow rate which can be obtained using the Bernouilli equation. Different expressions for C_{d} are available in literature. An important aspect of their formulation concerns the presence or absence of cavitation in the orifice [107, 106]. For a non-cavitating flow, C_{d} is a function of the Reynolds number Re and geometric information l/d, where l is the orifice length and d the orifice diameter, i.e., $C_{\text{d}} = f(Re, l/d)$. In general, for the flow in a straight nozzle, as Re increases, C_{d} increases as well. On the other hand, for a cavitating flow, C_{d} is determined using both the cavitation number Ca and geometric information, i.e., $C_{\text{d}} = f(Ca, l/d)$. In general, for a cavitating flow in a straight nozzle, as Ca decreases, the level of cavitation becomes stronger, and C_{d} decreases as well.

Cavitation (if occurred) has both desirable and undesirable effects [106]. For subcritical chamber pressures, when the produced bubbles exit the nozzle orifice toward the surrounding air, they will explode, enhancing the spray breakup process. This results in finer droplets and accelerates the fuel evaporation and fuel-air mixing. Therefore, cavitation can promote liquid atomization, as well as reduce drag, which is beneficial for fuel injector nozzles. For supercritical chamber pressures, cavitation can also enhance the behavior of the developing jet leading to improved evaporation and fuel-air mixing. The undesirable effects are usually associated with surface erosion, excessive noise generation, and hydrodynamic losses. The collapse of cavitation bubbles can be violent and damage machinery, which can shorten the life and performance of the injectors. Moreover, cavitation can lead to a reduction in the discharge coefficient, thereby decreasing the flow efficiency. The effect of cavitation can be summarized as follows:

Effect of cavitation in fuel injector nozzles

- *Desirable effects*: enhancement of the liquid atomization process, and drag reduction.
- *Undesirable effects*: surface erosion, noise generation, and reduction in the discharge coefficient.

As a conclusion, it is evident from this section that an in-depth analysis of in-nozzle cavitation and its impact on the developing liquid jet in the cylinder chamber is vital for diesel engine research and design.

4.3.6 Multiphase Modeling of Internal Nozzle Flow

In this section, we examine basic numerical approaches for modeling the internal flow through the injector nozzle. As mentioned previously, cavitation can occur inside the nozzle, and therefore, the internal flow has to be modelled as a *multiphase flow*. The adjective *multiphase* refers to situations where several different phases[3] (or components) – solids, liquids, and gases – are flowing simultaneously. For a cavitating flow, the system is composed of at least two phases:[4] the vapor bubbles and the liquid fuel. The numerical approaches include:

1. *Volume of fluid* (VOF) model [108, 109], in which the phases are assumed to be separated by a distinct interface. Here, we only have one set of conservation equations, and for each phase considered in the model, a variable is introduced as the volume fraction of the phase in the computational cell (the volume fractions of all phases sum up to unity in each cell). The interface is tracked via reconstruction schemes based on the information of the volume fractions of the phases. To accurately predict the interface between the phases, the VOF model must resolve all involved length and time scales in the system.

2. *Multi-fluid model* (MFM) [110, 109], in which the phases are treated as interpenetrating continua. The conservation equations for mass, momentum, and energy are phase-averaged within each computational cell, resulting in a separate set of conservation equations for each phase. The interaction between the phases is included through source/sink terms. A limiting case is the *two-fluid model* (TFM) where only two phases are considered (for example, liquid and vapor).

3. *Homogeneous mixture model* (HMM) [111, 112], in which the phases are assumed to be homogeneously mixed and in local equilibrium, whereby all components

[3] In thermodynamics, a *phase* refers to a chemically and physically uniform quantity of matter that can be separated mechanically from a non-homogeneous mixture. It may consist of a single substance or a mixture of subtances.

[4] Some fuels contain a certain amount of noncondensable gas. Consequently, when cavitation occurs, the system becomes composed of three phases: liquid fuel, noncondensable gas, and fuel vapor.

are assumed to have the same pressure, temperature, and velocity within a given computational cell. Here, we only have one set of conservation equations, and each phase is treated as a species. Hence, this approach treats the multiphase flow in a similar way to a single-phase, multi-species flow. Cavitation is considered by adding the mass transfer between the liquid and vapor to the source/sink terms of the conservation equations.

Note that these models can be framed in any turbulence approach, i.e., RANS, URANS, and LES. For example, in the RANS-HMM, the conservation equations are solved using the Reynolds-averaged Navier-Stokes technique whereas, in the LES-HMM, the conservation equations are solved using the large eddy simulation technique, and so forth.

Although the VOF and MFM models can provide very accurate results given sufficient grid resolution, they are computationally very expensive. The HMM can be seen as a compromise between speed and accuracy and thus is employed in this thesis (see Sec. 7.3.1.3).

4.3.7 Spray Modeling

The majority of spray process modeling methodologies fall into the following three categories:

1. *Eulerian–Eulerian* (EE) approach, in which both the liquid (or dispersed) and gas (or continuous) phases are treated using an Eulerian description. Examples of methods in this category include the VOF, MFM, and HMM presented in Sec. 4.3.6. With regard to sprays, these methods are suitable for the calculation of flows with higher droplet concentrations, such as the dense spray in the near-nozzle field.

2. *Lagrangian-Eulerian* (LE) approach (also referred to as *discrete droplet method* (DDM)) [113], in which Eulerian gas phase equations are solved along with Lagrangian evolution equations for the dispersed phase. This approach injects parcel droplets at the nozzle exit and follows its migration and evolution in the carrier gaseous flow, with several sub-models to capture the physics of spray breakup, droplet drag, collision, coalescence, evaporation, etc. This approach is better suited for the simulation of the dilute spray in the far-nozzle field.

3. *Hybrid* approach [114], which is based on hybridizing between the EE and LE methods. A transition criterion is usually defined to switch from EE to LE methods in the same simulation. This approach is very useful for accurate description of dense-to-dilute spray flows.

4.4 Injection in Liquid-Propellant Rocket Engines

Injection in liquid rocket engines have been studied extensively from a physical and numerical viewpoint by many researchers in the field, and particularly by previous members of our lab, and thus this discussion is kept brief. For a detailed treatment of this topic, readers are referred to classic textbooks such as Sutton and Biblarz [115] and Turner [116], or to the theses of Zong [96], Huo [97], and Lioi [117].

4.4.1 Engine Components and Principle of Operation

Liquid-propellant systems have self-contained liquid propellants. Most of these engines use bipropellant systems, i.e., those in which a fuel and an oxidizer are tanked separately and then transferred into the combustion chamber (also called thrust chamber) through a propellant feed system to mix and burn. The feed system comprises pipes, turbopumps, and fuel and oxidizer manifolds and injectors. The turbopumps raise the pressure above the operating pressure of the combustion chamber, and the propellants are then injected into the chamber in a manner that assures atomization and rapid mixing. Ignition is accomplished either by means of a pyrotechnic igniter or spontaneously, depending on the propellant composition. The hot exhaust produced by combustion is then passed through a supersonic nozzle[5] to accelerate the flow and produce thrust. Liquid-propellant engines have certain advantages over solid systems, in which the propellants are already mixed together and packed into a solid cylinder. These include higher exhaust velocity and specific impulse and better control of operating level in flight (i.e., throttleability). However, they tend to be more complex because of the pumps and storage tanks [115, 118].

4.4.2 Types of Liquid-Propellant Rocket Engines and Power Cycles

Liquid rocket engines may broadly be categorized according to the power cycle they employ do drive their propellant turbopump assembly. Three types of engine cycles are in common use [119]:

1. *Expander cycle*, in which no combustion takes place before the propellants enter the combustion chamber. The fuel is heated and expands as it is circulated around the outside of the combustion chamber.

2. *Gas-generator cycle*, in which a portion of the propellants is burned before entering the main combustion chamber and used to drive the turbopump before being exhausted into the diverging portion of the exit nozzle. This allows for higher pressure head to drive the pump system relative to the expander cycle.

[5] In rocket engines, a *supersonic nozzle* is a converging-diverging nozzle geometry located at the aft of the combustion chamber and is used to convert the thermal energy of the exhaust gas into kinetic energy to generate thrust. Note this is different from the *fuel injector nozzle* in diesel engines which is used to inject the fuel into the cylinder chamber.

3. *Staged-combustion cycle*, in which a larger portion of the propellants is combusted in one or more preburners before being passed through the turbine. This may be done in either oxidizer- or fuel-rich mode. The turbine exhaust is then delivered to the main combustion chamber where it is burned with the remaining portion of the propellants. Staged combustion engines have many advantages over systems employing the other two cycles, in particular lower gas temperatures entering the turbine, and higher chamber pressures. High pressures generally result in better overall efficiency and specific thrust of the engine. Furthermore, staged combustion engines exhibit little to no soot formation. However, they are typically heavier and more complex to design than the other systems.

Oxidizer-Rich Staged-Combustion Cycle

A particular type of the staged-combustion cycle is the oxidizer-rich staged-combustion (ORSC) cycle, shown schematically in Fig. 4.6. Here, all of the oxidizer and a portion of the fuel are fed through the preburner, generating oxidizer-rich gas. After being run through a turbine to power the pumps, the gas is injected into the combustion chamber and burned with the remaining fuel. A well known-example of ORSC engine is the Russian RD-170 LOX/kerosene, shown schematically in Fig. 4.7(a). This engine will be used as a case study in Chapter 5.

4.4.3 Injectors

Propellant injectors and oxidizer manifolds are critical parts of the engine design; they take the propellants and mix them so that they can quickly burn in the preburners or combustion chambers. Typically, injectors are composed of hundreds of injector elements, each consists of a number of small holes or ports through which jets of fuel and oxidizer are injected. A good injector design seeks optimized atomization, mixing, and combustion of the propellants while meeting design requirements and minimizing combustion instabilities. Different types of rocket injectors exist, including jet, swirl, coaxial, and gas-centered liquid-swirl coaxial (GCLSC) injectors. Here, the focus is on GGLSC injectors.

GCLSC Injectors

GCLSC injectors have been extensively used as main combustion chamber injectors in many ORSC rocket engines, including the NK-33, RD-170, and RD-180 engines. In this type of injector, high-temperature gaseous oxygen (originating from the preburner) is axially injected into the oxidizing-gas passage, and low-temperature liquid fuel (originating from the fuel pump) is tangentially delivered into the fuel annulus, as sketched in Fig. 4.7(b). GCLSC injectors offer excellent mixing efficiency and stability behaviors, as well as relatively simple configurations.

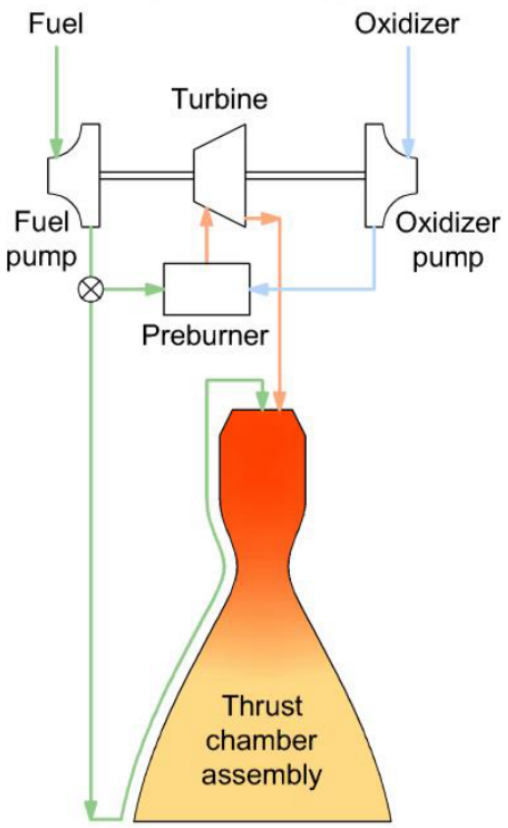
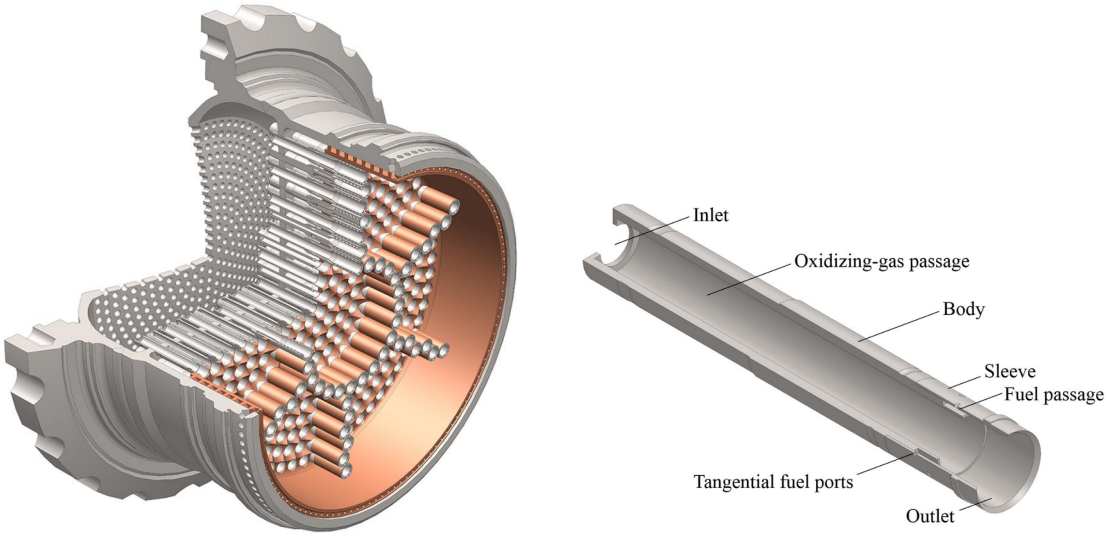


Figure 4.6: ORSC cycle (reproduced from Ref. [120]).



(a) (b)

Figure 4.7: (a) Schematic of main combustion chamber of RD-170 rocket engine, which is a type of ORSC engine. (b) Cross section and zoomed-in view of a main GCLSC injector element (reproduced from Ref. [117]).

4.4.4 Flow Injection Behavior and Modeling

Typically, the operating pressures in ORSC rocket engines exceed the thermodynamic critical points of the fluids involved. Liquid propellants initially injected at a subcritical temperature may heat up and experience a thermodynamic phase transition into the supercritical regime. Many distinctive behaviors occur during the transition and differ from the phenomena encountered at subcritical conditions, such as diminishing of surface tension and absence of droplet formation and a two-phase interface. As a result, single-phase-like, diffusion-dominated mixing takes place between dense and light fluids in the presence of large property gradients in a similar fashion to the situation depicted in Fig. 4.4(b). For this situation, the governing equations for a single-phase, compressible, multicomponent flow with a real-fluid EOS can be used to model the underlying flow [121, 122, 123, 124, 8].

Part II

Research Projects

CHAPTER 5
ACCELERATING THE CONVERGENCE OF REAL-FLUID SIMULATIONS USING DEEP NEURAL NETWORKS

This chapter addresses all items of Objectives 1(a) and 2 (cf. Sec. 1.2). Here, it is shown that using deep learning inside traditional CFD solvers for modeling complex fluid flows and related phenomena can improve the speed of the simulations. The approach is demonstrated on problems involving high-pressure supercritical fluid flows relevant to aerospace propulsion applications. It is robust, and results in considerable computational speedups, and it can be extended to many areas of computationally-intensive science and engineering. Please note that Chapters 2 and 3 as well as Secs. 4.1, 4.2, and 4.4 are prerequisite to this chapter.

The material presented here is adapted from:

- [125] P. J. Milan, J.-P. Hickey, X. Wang, and V. Yang, "Deep-learning accelerated calculation of real-fluid properties in numerical simulation of complex flowfields," *Journal of Computational Physics*, Vol. 444, pp. 1-25, 2021.
- [126] P. J. Milan, X. Wang, J.-P. Hickey, Y. Li, and V. Yang, "Accelerating numerical simulations of supercritical fluid flows using deep neural networks," *AIAA 2020-1157*, pp. 1-12, 2020.

5.1 Abstract

A DL-based approach is developed for efficient evaluation of thermophysical properties in numerical simulations of complex real-fluid flows. The work enables a significant improvement of computational efficiency by replacing direct calculation of the equation of state with a deep feedforward neural network with appropriate boundary information (DFNN-BC). The proposed method can be coupled to a flow solver in a robust manner. Depending on the numerical formulation of the flow solver, the neural network takes in either the primitive or conservative variables, including the chemical composition of the system, and calculates all relevant fluid properties for the subsequent routines in the solver. Two test problems are employed to validate the proposed methodology. The first uses a preconditioning scheme with dual-time integration for the simulation of swirl injector flow dynamics under supercritical conditions. The second uses a conservative-variable-based formulation for the simulation of laminar counterflow diffusion flames for cryogenic combustion. A parametric analysis is performed to optimize the numbers of hidden layers and neurons per hidden layer. The computational accuracy, efficiency, and memory requirements of the neural network are examined. The DFNN-BC model accelerates the evaluation of real-fluid properties by a factor of 2.43 and 3.7 for the two test problems, respectively, and the overall flowfield simulation by 1.5 and 2.3, respectively. In addition, the memory

usage is reduced by up to five orders of magnitude in comparison with the table look-up method. The findings from this work exemplify how high-performance scientific computing can leverage advances in DL to accelerate expensive simulations without compromising accuracy.

5.2 Introduction and Literature Review

Many natural phenomena and engineering applications involve fluid flows at conditions well above their thermodynamic critical points; we refer to these flows as *supercritical flows* (cf. Sec. 4.2). The peculiar behavior of real fluids such as thermodynamic non-idealities and transport anomalies must be taken into account [127], and a generalized EOS valid for the entire thermodynamic fluid regime should be used. Such highly nonlinear EOS is computationally expensive, especially for problems involving multiple species and multiscale physics. For example, in a LES of a supercritical reacting jet in a cross flow using the Peng-Robinson (PR) [128] EOS and a finite-rate kinetic mechanisms of 5 species, the time required for evaluating real-fluid properties accounts for about 50% of the total computation time [129]. Likewise, in a LES of a supercritical reacting mixing layer using the Soave-Redlich-Kwong (SRK) [130] EOS and a flamelet model with 7 species, about 55% of the total computation time [131] is spent on the evaluation of fluid properties.

To render such computations more efficient, thermodynamic tabulation has often been used, see for example Refs. [132, 133, 134, 123]. Fluid properties are pre-computed and stored in a look-up table. They are then retrieved through interpolation during the flow calculation. This method is effective for a small number of chemical species, as shown in the studies of supercritical turbulent mixing layers with 2 species by Wang et al. [123]. As the compositional complexity of the problem increases, however, the dimensionality of the resulting manifold grows dramatically. The look-up table may occupy a significant portion of the memory and may even saturate the system, thereby slowing down the calculation. In addition, the search and retrieval of data from the table become costly. For example, for a multi-dimensional table of discretization of $(n_Z, n_{\widetilde{Z''^2}}, n_C, n_h) = (200, 100, 50, 20)$, where Z, $\widetilde{Z''^2}$, C, and h stand for mixture fraction, variance of mixture fraction, progress variable, and enthalpy, respectively, the evaluation of 15 tabulated variables takes about 2.4 GB of memory [135]. The situation is particularly of concern for homogeneous, parallel, computer codes running on distributed memory architectures with a message-passing interface (MPI), where each core needs to independently access the lookup table. Attempts have been made to reduce the associated memory footprint, including locally-adaptive tabulation [136, 137], unstructured tabulation [138], and the correlated dynamic evaluation (CDE) method [131, 123], as reported in Table 5.1. These efforts, however, still face the inevitable "curse of dimensionality" as more species are considered in the system (especially when the number of chemical species is greater than three, or equivalently when the dimensionality of the table, or manifold, is larger than four), which makes them completely unsuitable for practical problems [131, 135]. An important question is thus: how to efficiently represent the

multi-dimensional manifolds necessary to capture the salient features associated with real-fluid thermodynamics and transport?

Deep learning, a subfield of ML and AI, is concerned with the design of algorithms that are modeled loosely on the human brain. The heart of this discipline lies in the deep neural network, where the term "deep" refers to a network architecture that contains many hidden layers. In recent years, DL has been successfully applied in a variety of fields for image classification, speech recognition, and language translation, rendering many technological breakthroughs with broader impact.

The ability of DNN to learn complex patterns has also been leveraged in the fluid dynamics and combustion fields. The challenges, however, are different from those tackled in other applications [27, 4]. Turbulent flows are characterized by complex, multi-scale, spatiotemporal phenomena with stochastic-like signatures, but the underlying dynamics obey a well-defined set of governing equations. The motivation for the use of DL in the studies of fluid mechanics and combustion centers around two important goals: (1) improved physical predictive modelling of the flow, or (2) reduction of algorithmic or computational expense. DL is more commonly employed to improve physical modeling. For example, Ling et al. [139] employed a DFNN with many hidden layers to model the anisotropic Reynolds stress tensor in RANS. Lapeyre et al. [140] implemented a CNN to predict the subgrid-scale wrinkling of a flame in LES. Nikolaou et al. [141] used a CNN for subgrid-scale modeling in LES of turbulent reacting flows in a deconvolution context. These works use ML algorithms as a means of pattern identification in order to replace the conventional modeling strategies.

Efforts have also been made to use ML as a means of complexity reduction, with the objective of improving the computational efficiency of existing numerical algorithms. In this regard, we note the seminal work by Ihme [142], along with the more recent contributions by Shadram et al. [143], Owoyele et al. [144], and Bhalla et al. [135], in which memory-intensive, multi-dimensional, flamelet look-up tables were replaced with memory-efficient neural networks. In addition, Xing et al. [145] developed an artificial neural network to replace the chemical percolation devolatilization model for accelerated prediction of coal devolatilization kinetics. In this latter category, the ML algorithms focus on improving the computational efficiency of existing algorithms while relying on conventional modelling paradigms.

All of the aformentioned examples demonstrate, in one way or another, that DNN has the ability to learn and approximate nonlinear mathematical functions, and to handle interpolation of points not observed during the training procedure. The technique thus offers the possibility of addressing the "curse of dimensionality" from which conventional low-order techniques like tabulation suffer. The present study attempts to develop a DNN-based framework for efficient evaluation of real-fluid properties in simulations of supercritical flows. To the author's knowledge, this work is *the first of its kind*.

A neural network model, referred to as DFNN-BC, is developed to achieve the desired accuracy when incorporating boundary information (i.e., the flow behavior along the boundaries of the domain). The model can be coupled to a flow solver in a robust manner. The proposed methodology is implemented and tested in two model problems with different numerical formulations: (1) two-dimensional turbulent mixing

of gaseous oxygen and liquid kerosene in a rocket injector using a primitive-variable-based formulation, and (2) a quasi-one-dimensional counterflow diffusion flame for cryogenic hydrogen combustion using a conservative-variable-based formulation. We thus cover two widely-used methods for the simulation of fluid flows at supercritical conditions.

The paper is organized as follows. Section 5.3 describes the equations of state and property evaluation schemes for real fluids, along with the two different numerical codes employed in the simulations. Section 5.4 deals with the DFNN-BC model and its integration into the flow solvers. Sections 5.5 and 5.6 discuss the results for the two model problems, namely, the rocket injector and the counterflow diffusion flame. In these sections, the accuracy, efficiency, and memory requirements of DFNN-BC are discussed and analyzed in comparison to baseline simulations with no property acceleration. Finally, key findings and concluding remarks are made in Sec. 5.7.

Table 5.1: Studies on approximation methods for accelerated evaluation of non-ideal thermophysical properties in numerical simulation of real-fluid flows. Here, n_i refers to the number of inputs in the approximation method, and N represents the number of species in the system under consideration.

Approximation method	Problem description	EOS/Database	$n_\mathrm{i}(N)$	Reference	Year
Curve/suface fitting	Hypersonic 3D corner, hypersonic gap-seal	NASA CEA	2(1)	Coirier [146]	1990
Look-up table	Transonic impeller, multi-stage turbine	GE-NP/NIST	2(1)	Boncinelli et al. [132]	2004
Look-up table	Turbulent boundary layer	REFPROP	2(1)	Kawai et al. [147]	2015
Look-up table	Turbulent mixing layer	SRK	2	Wang et al. [123]	2018
Look-up table	Reacting mixing layer	SRK	3(8)	Milan et al. [131]	2019
Consistent look-up table	Transonic turbine, supersonic nozzle	FluidProp	2(1)	Pini et al. [134]	2015
AMR look-up table	Subsonic 1D nozzle, hypersonic 2D nozzle	REFPROP	2(1)	Xia et al. [136]	2007
AMR look-up table	Subsonic 2D nozzle, 2D heat transfer	NIST	2(1)	Liu et al. [137]	2014
Unstructured look-up table	Supersonic 2D nozzle, transonic 2D turbine, supersonic 3D turbine	FluidProp	2(1)	Rubino et al. [138]	2018
CDE	Turbulent mixing layer	SRK	2	Wang et al. [123]	2018
CDE	Reacting mixing layer	SRK	3(7)	Milan et al. [131]	2019

5.3 Real-Fluid Properties and Numerical Framework

5.3.1 Real-Fluid Properties

A real-fluid EOS is needed to account for the intermolecular forces and volumetric effects in the fluid [127]. A generic two-parameter cubic EOS [127, 130] can be written as

$$p = \frac{R_u T}{\frac{W}{\rho} - b} - \frac{a\alpha}{(\frac{W}{\rho})^2 + \sigma\frac{W}{\rho} + \epsilon} = \rho\left(\frac{R_u}{W}\right) ZT, \tag{5.1}$$

where p is the pressure, T the temperature, ρ the density, Z the compressibility factor, and R_u the universal gas constant. The molecular weight of the mixture, W, is given by

$$W = \left(\sum_{i=1}^{N} \frac{Y_i}{W_i}\right)^{-1}, \tag{5.2}$$

where W_i and Y_i are the molecular weight and mass fraction of the ith species, respectively, and N the total number of species. The parameters σ and ϵ define the specific EOS of concern. Two different state equations are used in this study: the SRK [130], and PR [128] EOSs. Their respective parameters are given in Table 5.2. The difference between these two EOSs is minor; SRK is more accurate when the reduced temperature is smaller than unity, since its parameters are obtained by curve-fitting of vapor pressure data, whereas PR is more accurate when the reduced temperature is greater than unity. The coefficients $a\alpha$ and b, which account for intermolecular forces and volume of molecules, depend primarily on the critical properties and binary interaction parameters.

Table 5.2: Parameters of cubic equations of state

Parameter	SRK	PR
σ	b	$2b$
ϵ	0	$-b^2$

Additional thermodynamic quantities, such as internal energy, enthalpy, specific heats, and speed of sound, are computed using departure functions that are derived from the selected EOS. The mixing rules and the formulation of the departure functions using SRK EOS are presented in Appendix C.1 and Appendix C.2, respectively. For the real-fluid implementation based on PR EOS, the interested reader can consult Refs. [148, 149]. The critical properties of the major chemical species considered in this study are given in Table 4.1.

The extended corresponding-states principle [127, 150] is used to estimate the transport properties of a multicomponent mixture over a broad range of fluid states.

Chung et al.'s method [151, 152] is used to evaluate the dynamic viscosity μ and thermal conductivity λ. Fuller et al.'s empirical correlation [153], combined with Takahashi's high-pressure correction [154], is used to compute the binary mass diffusion coefficients D_{jk}. For mixtures with more than two species, we adopt a mixture-averaged treatment of the diffusion coefficients [155], as follows

$$D_k = \frac{1 - Y_k}{\sum_{j \neq k}^{N} \chi_j / D_{jk}}, \tag{5.3}$$

where χ_k and D_k are the mole fraction and mixture diffusion coefficient of the kth species, respectively. Detailed derivation of these variables is given in Refs. [97, 156, 157].

5.3.2 Numerical Framework

Two different CFD codes are employed in this study. The first one, referred to as `PMBFS`, uses a primitive-variable-based formulation combined with the SRK EOS. The second, `One-D ThermoCode`, uses a conservative-variable-based formulation with the PR EOS.

5.3.2.1 PMBFS (Parallel Multi-Block Flow Solver)

`PMBFS` [123, 158] treats the Favre-filtered form of the conservation equations of mass, momentum, total energy, and species for a compressible multi-species flow (cf. Sec. 3.3.2). Turbulence closure is achieved by means of the LES technique (cf. Sec. 3.2) based on the algebraic Smagorinsky model (cf. Sec. 3.3), which resolves large-scale motions while modelling small-scale contributions to the flow evolution. A preconditioned dual-time stepping strategy [158, 159, 160] is employed to circumvent numerical stiffness resulting from eigenvalue disparity, especially in the low Mach number regime. Within each physical time step, the pseudo-time derivative is integrated until local convergence is achieved. The physical solution for the next time step is then attained. The governing equations with *preconditioning* take the following form

$$\boldsymbol{T}\frac{\partial \mathfrak{P}}{\partial \tau} + \frac{\partial \mathfrak{Q}}{\partial t} + \nabla \cdot (\boldsymbol{F} - \boldsymbol{F}_{\mathrm{v}}) = \mathfrak{S}, \tag{5.4}$$

where t is the physical time, τ is the pseudo-time, $\mathfrak{S}$ is the source term (set to zero here since a non-reacting flow is considered in this section), $\boldsymbol{F}$ and $\boldsymbol{F}_{\mathrm{v}}$ are the matrices of inviscid and viscous/turbulent fluxes, respectively, given by Eqs. (3.36) and (3.37), and $\mathfrak{P}$ and $\mathfrak{Q}$ are the vectors of primitive and conservative variables, respectively, given by

$$\mathfrak{P} = \begin{bmatrix} p_{\mathrm{g}} & u & v & w & T & Y_n \end{bmatrix}^T, \tag{5.5}$$

$$\mathfrak{Q} = \begin{bmatrix} \rho & \rho u & \rho v & \rho w & \rho e_t & \rho Y_n \end{bmatrix}^T, \tag{5.6}$$

where $\boldsymbol{u} = \begin{bmatrix} u & v & w \end{bmatrix}^T$ is the velocity vector, e_t the mass-specific total energy ($e_t = e + 1/2\ \boldsymbol{u} \cdot \boldsymbol{u}$), and p_g the gauge pressure. It is noted that the operators denoting the low-pass filtered and Favre-filtered quantities, commonly represented by the symbols $\overline{\square}$ and $\widetilde{\square}$, respectively (cf. Sec. 3.3), are not used here for the simplification of mathematical notation.

$\boldsymbol{T}$ is the preconditioning matrix for the pseudo-time derivative and can be expressed as [158, 159]:

$$\boldsymbol{T} = \begin{bmatrix} \xi & 0 & 0 & 0 & -\frac{A_T}{A_\rho} & -\frac{A_{Y_j}}{A_\rho} \\ \xi u & \rho & 0 & 0 & -\frac{A_T}{A_\rho}u & -\frac{A_{Y_j}}{A_\rho}u \\ \xi v & 0 & \rho & 0 & -\frac{A_T}{A_\rho}v & -\frac{A_{Y_j}}{A_\rho}v \\ \xi w & 0 & 0 & \rho & -\frac{A_T}{A_\rho}w & -\frac{A_{Y_j}}{A_\rho}w \\ \xi h_t + \frac{T}{\rho}\left(\frac{\partial \rho}{\partial T}\right)_{p,Y_k} & \rho u & \rho v & \rho w & \rho c_p + h_t\left(\frac{\partial \rho}{\partial T}\right)_{p,Y_k} & \rho B_{Y_j} - \frac{A_{Y_j} e_t}{A_\rho} \\ \xi Y_i & 0 & 0 & 0 & -\frac{A_T}{A_\rho}Y_i & \rho(1-\delta_{N-1,j}) - \frac{A_{Y_j}}{A_\rho}Y_i \end{bmatrix}, \tag{5.7}$$

where ξ is the parameter that re-scales the eigenvalues, h_t the mass-specific total enthalpy ($h_t = h + \boldsymbol{u} \cdot \boldsymbol{u}/2$), and $\boldsymbol{\delta}$ the Kronecker delta matrix. The parameters A_T, A_ρ, A_{Y_k}, and B_{Y_k} are defined as follows:

$$A_T = \left(\frac{\partial p}{\partial T}\right)_{\rho_k}, \tag{5.8}$$

$$A_\rho = \left(\frac{\partial p}{\partial \rho}\right)_{T,Y_k}, \tag{5.9}$$

$$A_{Y_k} = \rho\left[\left(\frac{\partial p}{\partial \rho_k}\right)_{T,\rho_{j\neq k}} - \left(\frac{\partial p}{\partial \rho_N}\right)_{T,\rho_{k\neq N}}\right], \tag{5.10}$$

$$B_{Y_k} = (\breve{e}_k - \breve{e}_N) - \left(\sum_{k=1}^{N} Y_k \breve{e}_k - e\right)\left(\frac{\partial \rho}{\partial p}\right)_{T,Y_k} \times \left[\left(\frac{\partial p}{\partial \rho_k}\right)_{T,\rho_{j\neq k}} - \left(\frac{\partial p}{\partial \rho_N}\right)_{T,\rho_{j,\neq N}}\right], \tag{5.11}$$

where $\breve{e}_k$ is the partial-density internal energy for the kth species, given by

$$\breve{e}_k = \left(\frac{\partial \rho e}{\partial \rho_k}\right)_{T,\rho_{j\neq k}}. \tag{5.12}$$

Details for the calculation of the partial derivatives and preconditioning terms are provided in Appendix C.3 and Appendix C.4, respectively. Finally, the system is closed with the SRK EOS, where density is obtained analytically from the primitive variables by solving a cubic equation [161]:

$$\rho = f(p, T, Y_k). \tag{5.13}$$

A major bottleneck in the computation of thermophysical variables results from the double summation operator over the number of chemical species, as evidenced in the mixing rules and their partial derivatives (see Eqs. (C.1), (C.15) and (C.18)). The cost of solving the EOS in primitive-variable-based solvers is $\mathcal{O}(N^2)$. Thus, if the number of chemical species in the mixture is doubled, the cost will be quadrupled.

The system of governing equations is discretized using a finite-volume methodology in generalized curvilinear coordinates (cf. Sec. 3.4). Temporal discretization in physical time is achieved by means of an implicit second-order backward difference scheme, and the inner-loop pseudo-time term is integrated with an explicit three-step Runge-Kutta scheme. Spatial discretization is obtained using a fourth-order central difference scheme. Fourth-order matrix dissipation [162] is implemented to enhance numerical stability in regions of large gradients. Finally, a multi-block domain decomposition technique associated with the `MPI` standard library [163] in `Fortran 90` is applied to achieve parallel computation. The algorithmic flowchart corresponding to this implementation is provided in Algorithm 2. In this flowchart, the superscripts n and m represent indices for the physical-time and pseudo-time steps, respectively. The total number of pseudo-time steps, M, is typically set to 20-40. The output variables in Steps 4–6 of Algorithm 2 are provided in Table 5.3. These variables are considered as the output layer when building the DFNN-BC model and coupling it to the `PMBFS` solver.

Algorithm 2: Time marching in physical time from t^n to t^{n+1} in `PMBFS`

for τ^m, *where* $m = 1, M$ **do**
  for *each stage of the Runge-Kutta scheme* **do**
    Step 1: Face reconstruction and flux computation at τ^m
    Step 2: Compute preconditioner $\boldsymbol{T}^m$ and Jacobians $\boldsymbol{J}^m$
    Step 3: Update primitive variables $\mathfrak{P}^{m+1}$
    Step 4: Compute ρ^{m+1} analytically
    Step 5: Compute other thermodynamic quantities and partial derivatives at τ^{m+1}
    Step 6: Compute transport quantities at τ^{m+1}
    Step 7: Compute turbulence closures at τ^{m+1}
  end for
end for
$\mathfrak{P}^{n+1} \leftarrow \mathfrak{P}^{M+1}$

Table 5.3: Output variables in Steps 4–6 of Algorithm 2.

Step #	Output variables
4	Z, ρ
5	W, e, h_k, h, c_v, c_p, γ, a, A_T, A_ρ, A_{Y_k}, B_{Y_k}
6	μ, λ, D_k

5.3.2.2 *One-D ThermoCode*

The `One-D ThermoCode` is a generalizable, compressible flow solver with a highly modular thermodynamic implementation [164]. This code was built as a high-order flow solver intended for fundamental studies of complex thermodynamics based on the complete conservation equations. A flamelet model is included as a limiting case. In the present study of counterflow diffusion flames, a quasi-one-dimensional mapping about the axis of symmetry (x-axis) is made by polynomial expansion and symmetry implementation. The resulting governing equations take the form below [164, 165]:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + 2\rho V = 0, \tag{5.14}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u u}{\partial x} + \frac{\partial p_0}{\partial x} + 2\rho u V = \frac{\partial}{\partial x}\left[\frac{4}{3}\mu\left(\frac{\partial u}{\partial x} - V\right)\right] + 2\mu\frac{\partial V}{\partial x}, \tag{5.15}$$

$$\frac{\partial \rho V}{\partial t} + \frac{\partial \rho u V}{\partial x} + 3\rho V^2 = -2p_2 + \frac{\partial}{\partial x}\left[\mu\frac{\partial V}{\partial x}\right], \tag{5.16}$$

$$\begin{aligned}\frac{\partial \rho h_t}{\partial t} - \frac{\partial p_0}{\partial t} + \frac{\partial \rho u h_t}{\partial x} + 2\rho h_t V = \frac{\partial}{\partial x}\left[\lambda\frac{\partial T}{\partial x}\right] - \sum_k \frac{\partial \rho Y_k V_{k,x} h_k}{\partial x} \\ + \frac{\partial}{\partial x}\left[\frac{4}{3}\mu\left(\frac{\partial u}{\partial x} - V\right)u\right] + 2\mu\frac{\partial V}{\partial x}u - \frac{4}{3}\mu\left(\frac{\partial u}{\partial x} - V\right)V,\end{aligned} \tag{5.17}$$

$$\frac{\partial \rho Y_k}{\partial t} + \frac{\partial \rho u Y_k}{\partial x} + \frac{\partial \rho V_{k,x} Y_k}{\partial x} + 2\rho V Y_k = \dot{\omega}_k, \tag{5.18}$$

where u and V are the axial and radial velocity components, respectively, and $\dot{\omega}_k$ the mass production rate of the kth species. The pressure expansion preserves the second-order term, thereby leading to the terms p_0 and p_2 in the above equations. The total enthalpy, h_t, corresponds to the sum of the enthalpy and kinetic energy. The code is coupled to an extended version of the `Cantera` package [166] for the computation of heat release (for reacting flow) and real-fluid thermophysical properties. The PR EOS is used here. The conservative formulation of the governing equations requires an iterative solver such as the Newton-Raphson method for solving for temperature for a given mixture:

$$T = f(\rho, e, Y_k). \tag{5.19}$$

The procedure differs from primitive-variable-based solvers, in which no sub-iterations are needed to retrieve thermophysical properties. Here, for each iteration, thermodynamic departure functions and intermediary variables need to be calculated over the entire mixture composition space. Consequently, the time complexity associated with the direct evaluation of the EOS in conservative-variable-based solvers can be even higher than $\mathcal{O}(N^2)$.

The theoretical model is numerically treated with a finite-volume formulation. A four-step Runge-Kutta scheme is employed for temporal discretization, along with

high-order flux reconstruction technique at the cell interfaces. The flux at the interfaces is computed based on the averaged flux of two alternatively biased polynomial interpolations. Such dual flux calculation provides additional numerical dissipation through flux averaging. The flux reconstruction allows for an efficient interface parallelization of the code with a high spatial resolution of complex thermodynamics. The code is written in `Python 2.7` and can be run either with only one core, or in parallel using the `mpi4py` library. The flowchart for the numerical implementation is provided in Algorithm 3. The output variables in Steps 3–5 of Algorithm 3 are listed in Table 5.4. These variables are considered as the output layer when incorporating the neural network into the `One-D ThermoCode`.

Algorithm 3: Time marching in physical time from t^n to t^{n+1} in `One-D ThermoCode`

for *each stage of the Runge-Kutta scheme* **do**
- Step 1: Face reconstruction and flux reconstruction at t^n;
- Step 2: Update conservative variables $\mathbb{Q}^{n+1}$;
- Step 3: Iteratively solve for T^{n+1} given ρ^{n+1}, e^{n+1} and Y_k^{n+1}:
  - Sub-step 3.1: Set $j = 0$ and assume an initial value $T_j^{n+1} = T^n$;
  - Sub-step 3.2: Compute intermediary thermodynamic departure functions;
  - Sub-step 3.3: Compute e_j^{n+1} knowing T_j^{n+1}, ρ^{n+1} and Y_k^{n+1} ;
  - Sub-step 3.3: Compute departure from target value $de = e_j^{n+1} - e^{n+1}$;
  - Sub-step 3.4: Correct estimate of T_{j+1}^{n+1} ;
  - Sub-step 3.5: Continue to step 3.2 until convergence ;
- Step 4: Compute all other thermodynamic quantities and partial derivatives at t^{n+1} knowing T^{n+1};
- Step 5: Compute transport quantities at t^{n+1};

end for

Table 5.4: Output variables of Steps 3-5 of Algorithm 3.

Step #	Output variables
3	T
4	p, W, h, h_k, a
5	μ, λ, D_k

5.4 DFNN-BC Model Specification

We first give a brief overview of deep neural networks, then present a model specification for DFNN-BC. Two canonical test cases are presented to demonstrate the improved predictive performance of the proposed method over standard DFNN models. Finally, the integration of DFNN-BC in the flow solvers is explained.

5.4.1 Overview of Deep Neural Networks

In the present study, a DFNN is used to approximate real-fluid thermodynamic and transport properties for accelerated calculation of supercritical flows. The goal of a DFNN is to approximate some function $\boldsymbol{y}^* = f^*(\boldsymbol{x})$ that maps an input $\boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}$ to an output $\boldsymbol{y}^* \in \mathbb{R}^{n_{\text{out}}}$. A DFNN defines an application $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ in such a way as to make f as close as possible to f^* (cf. Sec. 2.2.3). The model is associated with a direct acyclic graph that consists of an input layer with n_{in} neurons, followed by a certain number of hidden layers, and finally, an output layer with n_{out} neurons, as shown schematically in Fig. 2.3(b). The number of neurons per hidden layer in all the hidden layers is denoted by $\boldsymbol{\mathcal{H}} \in \mathbb{N}^{L_{\text{hidden}}}$, where $\mathcal{H}_j$ is the number of neurons in the jth hidden layer, with L_{hidden} the number of hidden layers.

The forward propagation step of the neural network is given by Eqs. (2.4) and (2.5). After the input data is fed into the network, each hidden layer processes it, and then passes the result to the following layer. The set of parameters, $\boldsymbol{\theta}$, containing all the weights and biases of the model – they are *a priori* unknown – can be expressed as $\boldsymbol{\theta} = \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}; l = 1, 2, \ldots, L\}$, where L is the total number of layers (excluding the input layer from the count), i.e., $L = L_{\text{hidden}} + 1$. Those parameters are adapted during the training process to minimize an objective function J. The latter is taken as the MSE loss between the actual and predicted values, as given by Eq. (2.6). To reduce or completely avoid overfitting and make sure that the model performs well not just on the training data but also on unseen inputs, L_2 regularization is used; it is controlled by the parameter λ_{reg}, which is chosen ahead of time to achieve a bias-variance trade-off (cf. Sec. 2.3.3.1). The Adam optimization algorithm (cf. Algorithm 1), which is a stochastic, first-order, gradient-based approach with an adaptive learning rate, is used to obtain the parameters of the network. The learning rate is initialized with the standard value of $\lambda_{\text{lr}} = 0.001$ throughout this chapter. The performance of a trained network is assessed by computing the MSE (or R^2-score[1]) on a validation dataset, which contains samples kept unseen during training. We note that in this chapter the validation set is used after training, i.e., for each trained model, the validation error is computed after the learning algorithm is terminated (cf. Sec. 2.3.4.1). While it is possible to automate the process of finding the right neural architecture and tuning the hyperparameters, this is not done in the current study. Rather, a manual search is performed based on the trial-and-error method (cf. Sec. 2.3.5.3).

5.4.2 Application to Real-Fluid Properties

In this study, the training database consists of thermodynamic and transport properties that span the thermophysical region of interest. It is generated using an EOS

[1]The R^2-score, which is defined by $R^2 \equiv 1 - \frac{\sum_i (y_i^* - y_i)^2}{\sum_i (y_i^* - \overline{y})^2}$ for exact data y_i^*, predicted values y_i and mean $\overline{y}$, is a statistical measure that gives information about how well the model predictions approximate the exact data points. It is between zero (poor predictive accuracy) and one (excellent).

(see Sec. 5.3). An important question is how to select the variables in both the input and output layers to build the correspondence between the flowfield and the DFNN. The state postulate indicates that only two independent intensive properties, along with the chemical composition, are required to determine the thermodynamic state (that is, a unique description of all thermophysical properties) of a multicomponent mixture in thermal and mechanical equilibrium [167]. For a mixture of N species, $N+1$ independent variables are needed.

Since the DFNN model is to be coupled to a CFD solver, the selection of the independent variables will depend on the formulation of the solver itself. For solvers with a primitive-variable-based formulation like `PMBFS`, a natural choice is the group of primitive variables, $\{T, p, Y_k;\ k = 1, 2, \ldots, N-1\}$. For solvers with a conservative-variable-based formulation like `One-D ThermoCode`, the group of conservative variables, $\{\rho, e, Y_k;\ k = 1, 2, \ldots, N-1\}$, is taken as the input layer of the DFNN. The selection of variables in the output layer is also solver-dependent. For example, if enthalpy and specific heats are needed in the calculations, then they should be included in the output layer, and so forth.

After the database is generated, it is split into two sets – training and validation – with an 80:20 split.[2] The data in each set are standardized using the mean and standard deviation of the set to stabilize the learning process and remove any bias (cf. Eq. (2.29)), particularly when the variables have different units (e.g, Joules, Kelvin, dimensionless, etc.). For example, in a combustion problem such as the one in Sec. 5.6, the mass fractions are in the range of 0-1 whereas the temperature is in the range of 120-3,500 K. The weights of the neural network are initialized to small random values (cf. Sec. 2.3.5.2) and updated, at each epoch, via the Adam optimization algorithm in response to the MSE estimates on the training dataset. The design and training of DFNN are accomplished using `PyTorch` and `scikit-learn` (cf. Sec. 2.3.9).

5.4.3 Boundary Information

Boundary information on f^* is usually available from governing physical laws or domain expertise. We consider here Dirichlet boundary conditions for specifying values of f^* along certain boundaries of the domain. To encode such boundary information as prior information in the DFNN-BC model, each of the boundary points of interest is repeated multiple times in the training data. Two loss functions are defined, denoted by J_{bc} and J_{int}, to compute the errors on the boundary and interior points, respectively [48, 168]. They are given by

$$J_{\text{bc}} = \frac{1}{N_{\text{bc}}} \sum_{j=1}^{N_{\text{bc}}} (f^*(\boldsymbol{x}^{(j)}) - f(\boldsymbol{x}^{(j)}))^2, \tag{5.20}$$

[2] Note that a test set was not considered in the *a priori* analysis. Evaluation of generalization of the neural network is carried out in the *a posteriori* analysis (i.e., when the neural network is implemented in CFD).

$$J_{\text{int}} = \frac{1}{N_{\text{int}}} \sum_{j=1}^{N_{\text{int}}} (f^*(\boldsymbol{x}^{(j)}) - f(\boldsymbol{x}^{(j)}))^2, \tag{5.21}$$

where N_{bc} stands for the number of boundary points (including repetitions) in the training data and N_{int} the number of interior points. In the present work, N_{int} is taken to be a large number $\sim \mathcal{O}(10^5\text{-}10^6)$, while N_{bc} is a small number $\sim \mathcal{O}(10^2\text{-}10^3)$. These loss functions are monitored during the model training process to ensure high accuracy throughout the entire input domain.

In the following, the DFNN-BC model is evaluated *offline* (or *a priori*) through two canonical examples in zero-dimensional (i.e., with no moving fluid). In each of these examples, the system is chosen for the numerical applications based on the `PMBFS` primitive-variable-based algorithm.

5.4.4 Canonical Example 1: Zero-Dimensional Thermodynamics

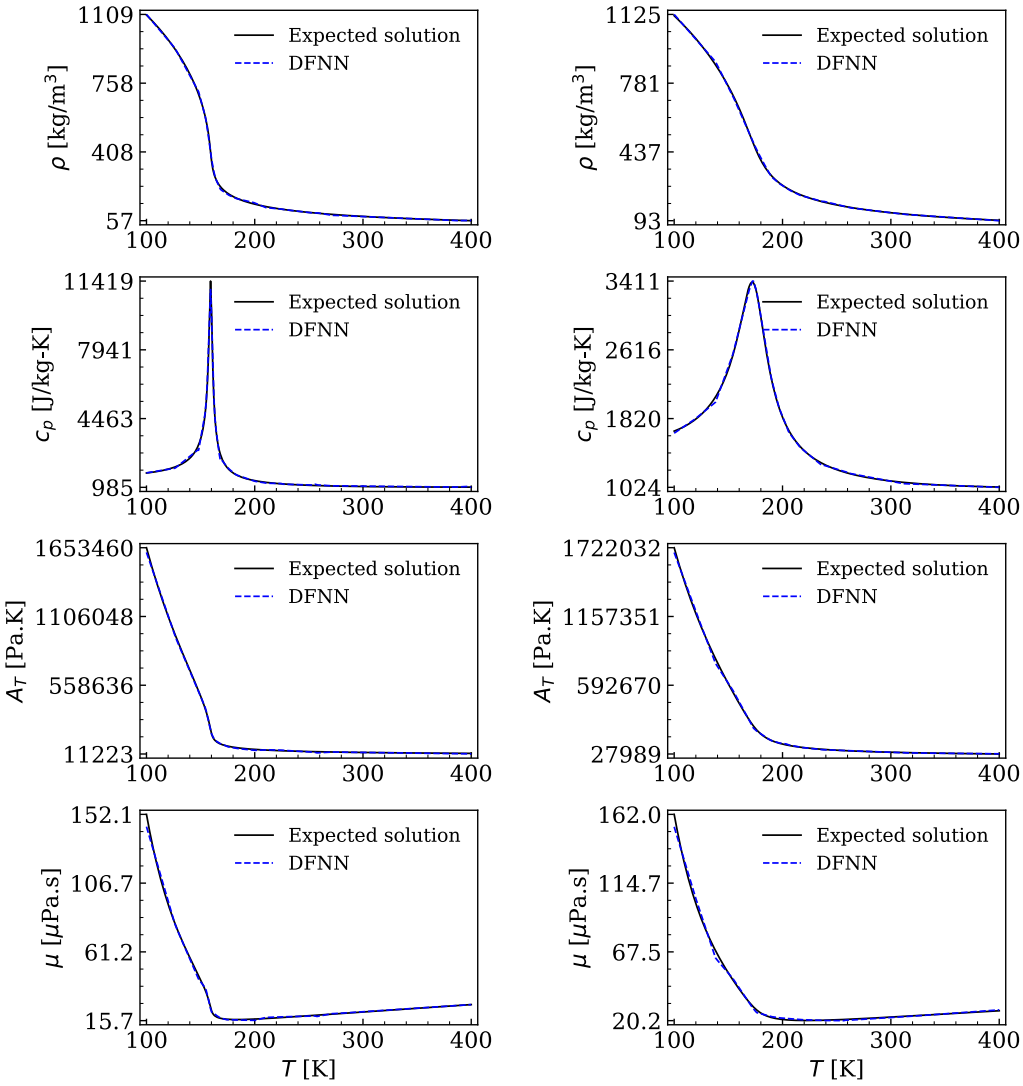
The neural network is first evaluated through a canonical example consisting of a single species in zero-dimensional. Pure oxygen is considered with temperature between 100 K and 400 K at two different pressures (6 MPa and 10 MPa). Both pressures are supercritical pressures, with the first condition closest to the critical pressure of oxygen ($p_{\text{c}} = 5.43$ MPa).

Two neural networks are designed, one for each pressure condition. The input layer of each neural network is designed with one channel, being the temperature. The output layer corresponds to the vector of variables in Table 5.3, referring to Algorithm 2, which results in a total of 12 variables for a single-species system. The training data is generated from the equations described in Sec. 5.3 with a uniform sampling over the manifold of interest. The total number of training points (before splitting the dataset) is $N_{\text{train}} = 300$, and thus the temperature axis is discretized with a step of approximately 1 K.

A standard DFNN is built following the methodology explained in Sec. 5.4.1. An architecture comprising 3 hidden layers and 9 neurons per layer is selected by manual search with the following hyperparameters: $\lambda_{\text{reg}} = 0.001$ and $\sigma = \text{ReLU}$ (in all the hidden layers). The R^2-scores on validation data are 0.998 and 0.999 for the two neural networks. Figures 5.1 compares the predictions from the neural networks to the ground truth solutions for selected variables. As seen from this figure, the neural network predictions are in excellent agreement with the reference solutions, even for the variable c_p which presents the strongest nonlinearity, especially at $p = 6$ MPa. We note that the boundary points did not need to be repeated here due to the simplicity of the example, unlike in the next example.

5.4.5 Canonical Example 2: Zero-Dimensional Thermodynamics

In this section, the model is evaluated through another, but more complicated, canonical example in zero-dimensional. A kerosene-oxygen mixture composed of four



(a) $p = 6$ MPa.

(b) $p = 10$ MPa.

Figure 5.1: Comparison of DFNN and expected solutions for density (ρ), specific heat at constant pressure (c_p), pressure derivative with respect to temperature (A_T), and viscosity (μ). Results are shown for: (a) $p = 6$ MPa, and (b) 10 MPa.

species is considered: O_2, $C_{10}H_{22}$, C_9H_{12}, and C_9H_{18}, and lies in the p-T-Y_i manifold where T ranges between 492.2 K and 687.7 K, Y_{O_2} between 0 and 1, $Y_{C_{10}H_{22}}$ between 0 and 0.7674, and $Y_{C_9H_{12}}$ between 0 and 0.1314. The pressure is assumed to be constant and is set to 253 bar. These physical states correspond to the operating conditions of the rocket injector discussed in Sec. 5.5. The input layer of the neural network is designed with the following four channels: $T, Y_{O_2}, Y_{C_{10}H_{22}}$, and $Y_{C_9H_{12}}$. The mass fraction of the 4th species, $Y_{C_9H_{18}}$, can be obtained by mass conservation (i.e., $\sum_i Y_i = 1$), and thus is not included in the input layer. The output layer corresponds to the vector of variables in Table 5.3, referring to Algorithm 2. The total number of

these variables is 27. The training data is generated from the equations described in Sec. 5.3 with a uniform sampling over the manifold of interest. The total number of training points (before splitting the dataset) is $N_{\text{train}} = 329,300$.

A standard DFNN is built following the methodology explained in Section 5.4.1. While the numbers of neurons in the input and output layers are fixed, and are usually determined by the problem under consideration, the appropriate numbers of hidden layers and neurons per hidden layer are more difficult to find, especially for a system with multiple species. By manual search, we found that a network architecture comprising 4 hidden layers and 9 neurons per hidden layer, i.e., $\boldsymbol{\mathcal{H}} = \begin{bmatrix} 9 & 9 & 9 & 9 \end{bmatrix}^T$, along with hyperparameters $\lambda_{\text{reg}} = 0.001$ and $\sigma = \text{ReLU}$ (in all the hidden layers), is suitable for estimating the thermophysical properties with greatly reduced cost. The R^2-scores on training and validation data are 0.988 and 0.987, respectively. The parametric analysis for guiding the design process and justifying the chosen neural network is reported in Appendix D.

Table 5.5 compares the prediction of the chosen DFNN to the expected solution for two different inputs. The first one ($p = 253$ bar, $T = 525.17$ K, $Y_{O_2} = 0.47$, $Y_{C_{10}H_{22}} = 0.41$, $Y_{C_9H_{12}} = 0.069$) is an interior point in the p-T-Y_i manifold, while the second one ($p = 253$ bar, $T = 687.7$ K, $Y_{O_2} = 1.0$, $Y_{C_{10}H_{22}} = 0$, $Y_{C_9H_{12}} = 0$) is a boundary point. The DFNN solution agrees well with the prediction for the interior point, with a relative error less than 5.5% for all the thermophysical variables. For the boundary point, the DFNN prediction indicates larger errors. For example, the density and internal energy have relative errors of -35.61% and 36.4%, respectively. Implementation in CFD of DNN models with such large errors at the boundaries will pose serious numerical challenges.

To alleviate the situation, the DFNN-BC model is implemented to provide accurate predictions at the boundaries, as explained in Sec. 5.4.3. Two boundary points are particularly of interest in the study of the rocket injector case in Sec. 5.5. The first corresponds to pure oxidizer ($p = 253$ bar, $T = 687.7$ K, $Y_{O_2} = 1.0$, $Y_{C_{10}H_{22}} = 0$, $Y_{C_9H_{12}} = 0$), and the second to pure fuel ($p = 253$ bar, $T = 492.2$ K, $Y_{O_2} = 0.0$, $Y_{C_{10}H_{22}} = 0.74$, $Y_{C_9H_{12}} = 0.15$). A sensitivity analysis is performed to determine the number of times for which each boundary point needs to be repeated in the training data. The resulting MSE losses, J_{bc} and J_{int}, are reported in Table 5.6 for different numbers of boundary training data, N_{bc}. The network architecture is fixed at 4 hidden layers and 9 neurons per hidden layer, and the number of interior training data points at $N_{\text{int}} = 329,298$. The prediction accuracy for the boundaries substantially increases with increasing N_{bc} from 2 to 10,002, as evidenced by the decrease in J_{bc}. The associated J_{int} increases by a small amount; the prediction accuracy for the interior points is not compromised much. For larger values of N_{bc} (i.e., 20,002 and 30,002), J_{int} increases considerably, suggesting that the predictions of the neural network become biased as more boundary points are introduced into the training data. Therefore, the case $N_{\text{bc}} = 10,002$ where each of the two boundary points is repeated 5,001 times is selected.

Table 5.7 compares the prediction of the chosen DFNN-BC to the expected solution for the two different inputs listed in Table 5.5. The DFNN-BC model consider-

Table 5.5: Comparison of DFNN and expected solutions at interior and boundary points.

Test 1: Interior point			
Inputs	Expected solution	DFNN solution	Relative error, %
$p = 253$ bar	$\rho = 299.41$ kg/m^3	$\rho = 299.58$ kg/m^3	0.058
$T = 525.17$ K	$e = -642,620.0$ J/kg	$e = -647,062.77$ J/kg	0.69
$Y_{O_2} = 0.47$	$c_p = 2,011.72$ J/kg-K	$c_p = 1,998.1$ J/kg-K	-0.68
$Y_{C_{10}H_{22}} = 0.41$	$a = 382.17$ m/s	$a = 389.57$ m/s	1.94
$Y_{C_9H_{12}} = 0.069$	$A_T = 91,447.34$ Pa.K	$A_T = 93,939.46$ Pa.K	2.73
	$A_\rho = 115,170.26$	$A_\rho = 120,176.59$	4.35
	$\mu = 3.016 \times 10^{-5}$ Pa.s	$\mu = 3.047 \times 10^{-5}$ Pa.s	1.02
	$\lambda = 7.0597 \times 10^{-2}$ W/m-K	$\lambda = 7.087 \times 10^{-2}$ W/m-K	0.39
	$D_{C_9H_{12}} = 2.062 \times 10^{-8}$ m^2/s	$D_{C_9H_{12}} = 1.950 \times 10^{-8}$ m^2/s	-5.41
Test 2: Boundary point			
Inputs	Expected solution	DFNN solution	Relative error, %
$p = 253$ bar	$\rho = 130.73$ kg/m^3	$\rho = 84.18$ kg/m^3	-35.61
$T = 687.7$ K	$e = 188,442.5$ J/kg	$e = 256,998.65$ J/kg	36.4
$Y_{O_2} = 1.0$	$c_p = 1,074.43$ J/kg-K	$c_p = 1,011.78$ J/kg-K	-5.83
$Y_{C_{10}H_{22}} = 0$	$a = 535.98$ m/s	$a = 512.35$ m/s	-4.41
$Y_{C_9H_{12}} = 0$	$A_T = 38,779.59$ Pa.K	$A_T = 21,831.41$ Pa.K	-43.70
	$A_\rho = 210,363.05$	$A_\rho = 199,132.66$	-5.33
	$\mu = 4.036 \times 10^{-5}$ Pa.s	$\mu = 3.344 \times 10^{-5}$ Pa.s	-17.14
	$\lambda = 6.217 \times 10^{-2}$ W/m-K	$\lambda = 6.415 \times 10^{-2}$ W/m-K	3.20
	$D_{C_9H_{12}} = 1.192 \times 10^{-7}$ m^2/s	$D_{C_9H_{12}} = 9.830 \times 10^{-8}$ m^2/s	-17.56

ably improves the prediction accuracy for the boundaries over the conventional DFNN model; all thermophysical variables are estimated with a relative error smaller than 5.5%. In addition, the predictions for the interior points are not compromised much. Figure 5.2 shows the regression plots for selected output variables with the DFNN-BC predictions on the y-axis and the expected solution on the x-axis. Also indicated is the R^2-score on validation data for each variable. The scatter points are colored by the local relative error, where overprediction is indicated by a positive value of the error and underprediction by a negative value. Overall, the R^2-score for each variable is above 0.93. The predictions of DFNN-BC agree well with the expected solution. It is noted that, although the manual search process yielded a high-performing neural network, as indicated by the high R^2-scores, it is possible that the optimal architecture and set of hyperparameters have not yet been found. Consequently, there are ways to further improve the accuracy of the predictions if needed. For instance, through the use of automated tools, such as `DeepHyper` [70], a more comprehensive search can be conducted to further optimize the neural network. Moreover, while a single neural network was used herein to approximate all the thermophysical variables, an alternative method would be to design separate/customized neural networks for each variable or group of correlated variables. The implementation and exploitation of these techniques, however, are left for a future study.

Table 5.6: MSE losses for different numbers of boundary training data points, N_{bc}. Here, "Repetition" indicates the number of times for which each boundary point of interest (i.e., pure oxidizer and pure fuel) is repeated in the training data.

Repetition	N_{bc} ($2 \times$ Repetition)	N_{train} ($N_{\text{int}} + N_{\text{bc}}$)	J_{int} (Eq. 5.21)	J_{bc} (Eq. 5.20)
1	2	329,300	0.0125	0.377
1,001	2,002	331,300	0.0130	0.366
5,001	10,002	339,300	0.0178	0.365
10,001	20,002	349,300	0.0258	0.336
15,001	30,002	359,300	0.0324	0.327

Table 5.7: Comparison of DFNN-BC and expected solutions at interior and boundary points.

Test 1: Interior point			
Inputs	Expected solution	DFNN-BC solution	Relative error, %
$p = 253$ bar	$\rho = 299.41$ kg/m^3	$\rho = 304.51$ kg/m^3	1.71
$T = 525.17$ K	$e = -642,620.0$ J/kg	$e = -642,148.47$ J/kg	-0.07
$Y_{O_2} = 0.47$	$c_p = 2,011.72$ J/kg-K	$c_p = 1,990.50$ J/kg-K	-1.05
$Y_{C_{10}H_{22}} = 0.41$	$a = 382.17$ m/s	$a = 391.04$ m/s	2.32
$Y_{C_9H_{12}} = 0.069$	$A_T = 91,447.34$ Pa.K	$A_T = 938,68.34$ Pa.K	2.65
	$A_\rho = 115,170.26$	$A_\rho = 117,885.60$	2.36
	$\mu = 3.016 \times 10^{-5}$ Pa.s	$\mu = 3.04 \times 10^{-5}$ Pa.s	0.84
	$\lambda = 7.060 \times 10^{-2}$ W/m-K	$\lambda = 7.05 \times 10^{-2}$ W/m-K	-0.13
	$D_{C_9H_{12}} = 2.062 \times 10^{-8}$ m^2/s	$D_{C_9H_{12}} = 2.084 \times 10^{-8}$ m^2/s	1.06
Test 2: Boundary point			
Input	Expected solution	DFNN-BC solution	Relative error, %
$p = 253$ bar	$\rho = 130.73$ kg/m^3	$\rho = 132.53$ kg/m^3	1.38
$T = 687.7$ K	$e = 188,442.53$ J/kg	$e = 198,509.77$ J/kg	5.34
$Y_{O_2} = 1.0$	$c_p = 1,074.43$ J/kg-K	$c_p = 1,088.35$ J/kg-K	1.30
$Y_{C_{10}H_{22}} = 0$	$a = 535.98$ m/s	$a = 532.60$ m/s	-0.63
$Y_{C_9H_{12}} = 0$	$A_T = 38,779.59$ Pa.K	$A_T = 37,964.82$ Pa.K	-2.1
	$A_\rho = 210,363.05$	$A_\rho = 203,204.20$	-3.40
	$\mu = 4.036 \times 10^{-5}$ Pa.s	$\mu = 3.929 \times 10^{-5}$ Pa.s	1.02
	$\lambda = 6.217 \times 10^{-2}$ W/m-K	$\lambda = 6.179 \times 10^{-2}$ W/m-K	-0.61
	$D_{C_9H_{12}} = 1.192 \times 10^{-7}$ m^2/s	$D_{C_9H_{12}} = 1.172 \times 10^{-7}$ m^2/s	-1.71

5.4.6 Integration in Flow Solvers

After a plausible network topology is determined to meet the desired performance, the DFNN-BC model is implemented in the time integration loop of a flow solver. The procedure is as follows. At each time step and at each cell of the numerical grid, the flow variables are marched forward in time and subsequently provided as inputs to the DFNN-BC, which, in turn, will update the thermophysical variables of interest, as shown in Algorithm 4. The parameters of the network, $\boldsymbol{\theta}$, are obtained from

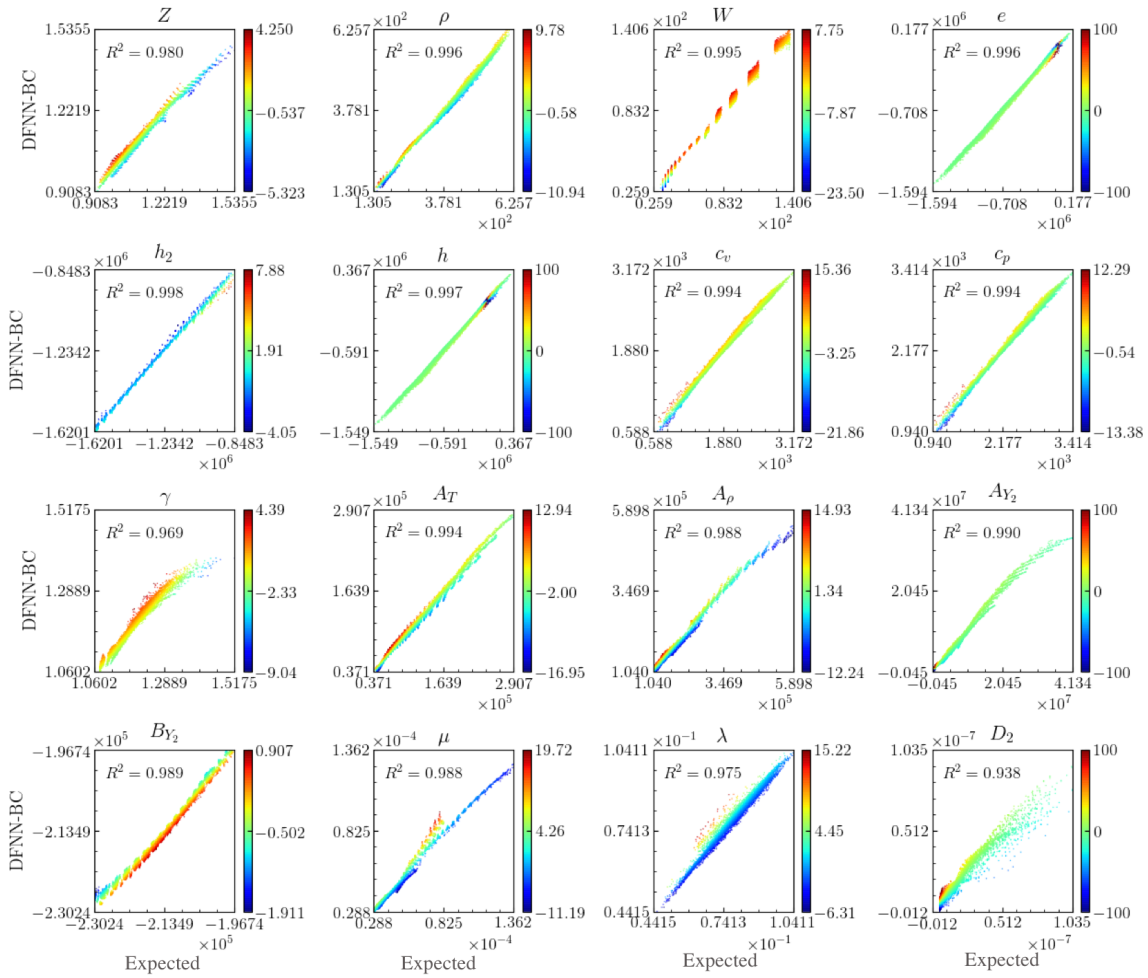


Figure 5.2: Regression plots of selected output variables (16 out of 27). The R^2-score on validation data is indicated for each variable. The color bars represent the local relative error in percentage. The range of the colorbar is set to the min/max values of the corresponding error. All variables are expressed in SI units.

`PyTorch` (after an offline training is completed) and imported into the flow solver. In the present study, we replace Steps 4-6 of Algorithm 2 in `PMBFS`, or Steps 3-5 of Algorithm 3 in `One-D ThermoCode`, by a single call to the subroutine in Algorithm 4. The entire process is summarized in Fig. 5.3.

Algorithm 4: Evaluation of real-fluid properties using DFNN-BC

Input: Flow variables $\boldsymbol{x}$, parameters of neural network $\boldsymbol{\theta} = \{\boldsymbol{W}^{(i)}, \boldsymbol{b}^{(i)}; i = 1, 2, ..., L\}$
Output: Thermodynamic and transport properties $\boldsymbol{y}$
Standardize $\boldsymbol{x}$;
$\boldsymbol{z} \leftarrow \boldsymbol{x}$;
for $i = 1$ *to* L **do**
  $\boldsymbol{z} \leftarrow \texttt{ReLU}(\boldsymbol{W}^{(i)}\boldsymbol{z} + \boldsymbol{b}^{(i)})$
end for
$\boldsymbol{y} \leftarrow \boldsymbol{z}$;
De-standardize $\boldsymbol{y}$;
Return $\boldsymbol{y}$;

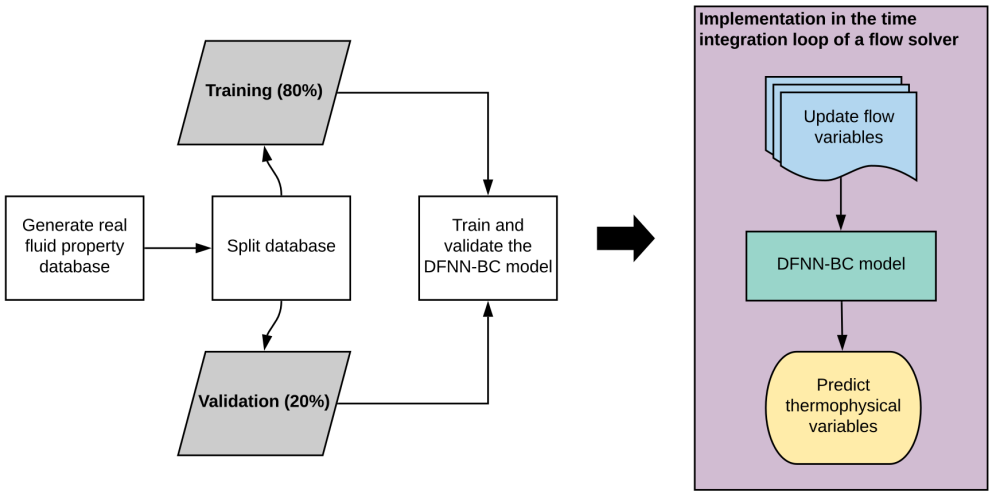


Figure 5.3: Flowchart summarizing the training process of DFNN-BC and its integration in a flow solver.

5.5 Swirl Rocket Injector at Supercritical Conditions

5.5.1 Geometry and Computational Setup

A GCLSC injector (cf. Sec. 4.4.3) is considered in this section. It consists of four regions: the center cylindrical tube (also referred to as the GOX post), coaxial fuel annulus, recess region, and taper region, as shown schematically in Fig. 5.4. The geometric parameters of this injector are provided in Table 5.8. Gaseous oxygen (GOX) is injected axially into the center tube at a mass flow rate of $\dot{m}_{\mathrm{o}} = 1.33$ kg/s and temperature of $T_{\mathrm{in,o}} = 687.7$ K, while liquid kerosene is tangentially introduced into the coaxial annulus at a mass flow rate of $\dot{m}_{\mathrm{f}} = 0.477$ kg/s and temperature of $T_{\mathrm{in,f}} = 492.2$ K through a slit located at $\Delta l = 2.0$ mm downstream of the annulus head. The injection pressure is set to 253 bar (supercritical condition), and thus the mixture can be treated as a single-phase flow (cf. Secs. 4.2 and 4.4.4). The Reynolds

number, Re, is about 2.50×10^5 based on the GOX flow condition and post thickness. Table 5.9 summarizes the injection operating conditions.

The computational domain consists of the injector interior (20 R_o in the axial direction) and a downstream zone (30 R_o and 8 R_o in the axial and radial directions, respectively). Calculations based on a cylindrical sector of the injector with an azimuthal span of three degrees are carried out. Periodic boundary conditions are applied in the azimuthal direction. Although such simplification leads to the exclusion of the vortex-stretching/tilting mechanisms responsible for turbulent energy transfer from large to small eddies, as well as azimuthal wave dynamics, many salient features of flow and mixing dynamics can still be captured by simulations based on a sector configuration [8]. Closure for the subgrid-scale terms is attained using the algebraic eddy viscosity model [87]. The grid contains a total of about 2.3×10^5 finite-volume cells, which are partitioned as follows: 448×128 in the GOX post, 122×48 in the fuel passage, 320×224 in the mixing cup, and 320×320 in the downstream zone. A grid convergence study for this configuration was performed in Ref. [122]. The smallest grid size of 5 μm is comparable with the Taylor scale (10.88 μm) within the injector, and allows for the resolution of the turbulent kinetic energy (TKE) in the subinertial range according to the Kolmogorov-Obukhov theory. An acoustically non-reflecting boundary condition [169] is implemented at the GOX and kerosene entrances. The downstream boundary of the computational domain is treated with sponge layers [170] in both the axial and radial directions to eliminate unphysical wave reflections. Adiabatic and no-slip boundary conditions are enforced at all solid walls. A three-component surrogate (n-decane/n-propylbenzene/n-propylcyclohexane with 74/15/11% by volume) for kerosene is used. Details on the computational setup can be found in Refs. [122, 124].

The numerical simulation, which is performed using the `PMBFS` solver, is initiated by delivering the GOX through the center cylindrical tube. Liquid kerosene is injected into the coaxial annulus after 8.0 ms of physical time, when the GOX flow has reached its stationary state. The time $t_0 = 0$ refers hereafter to the instant when the fuel injection commences.

Table 5.8: Injector parameters.

δ [mm]	h [mm]	R_o [mm]	R_f [mm]	Δl [mm]	L_1 [mm]	L_2 [mm]	L_s [mm]	L_r [mm]	α
0.60	0.745	5.62	7.03	2.0	93	113.1	7.5	8.5	42^o

5.5.2 Neural Network Design

The multicomponent mixture and the physical condition for the rocket injector problem are identical to those in the canonical example (cf. Sec. 5.4.5). The same DFNN-BC model thus is used in both cases. It should be noted, however, that a separate neural network is required if the problem involves different mixtures and/or

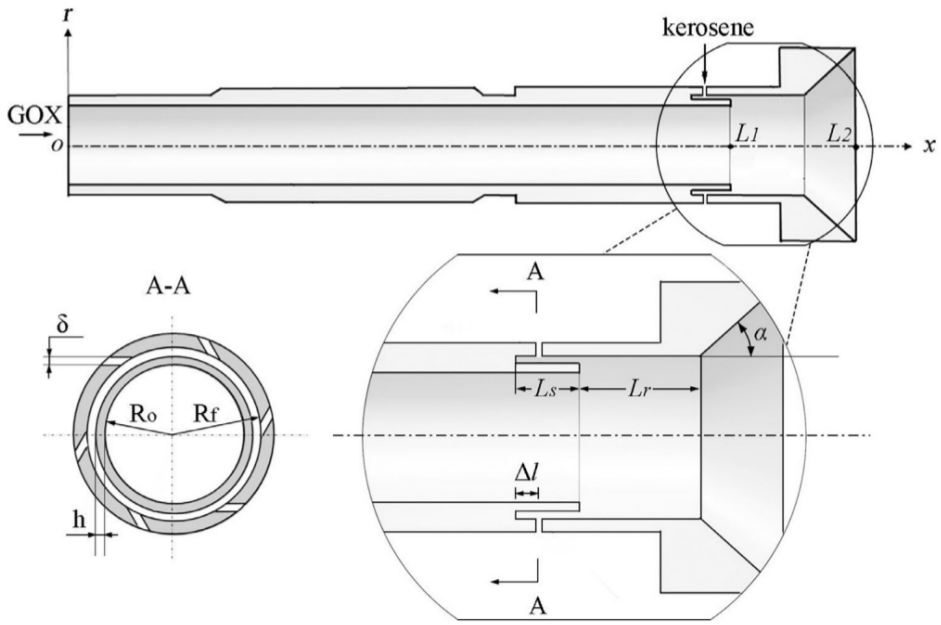


Figure 5.4: Schematic of GCLSC injector geometry [122, 124].

Table 5.9: Injector flow conditions.

	Oxidizer	Fuel
Fluid	GOX	Liquid kerosene
Mass flow rate (kg/s)	1.33	0.477
Temperature	687.7	492.2
Pressure (bar)	253	253

physical conditions. The neural network implemented herein consists of 4 hidden layers and 9 neurons per hidden layer. The input layer is designed with four channels, $T, Y_{O_2}, Y_{C_{10}H_{22}}$, and $Y_{C_9H_{12}}$, since the pressure variation in the injector is minimal [122, 124]. The output layer corresponds to the vector of variables in Table 5.3 (see Algorithm 2). A total of 27 variables are considered. The preconditioning variables A_{Y_4} and B_{Y_4} remain zero, and are omitted from the output layer. The number of boundary data points is set to $N_{bc} = 10,002$ based on the offline training result (cf. Sec. 5.4.5). The R^2-scores of the selected model are 0.982 and 0.980 on training and validation data, respectively. Table 5.10 compares the memory required to export the model parameters of DFNN-BC (from `PyTorch`) versus that required to export from look-up tables into data files for subsequent implementation in the flow solver. The look-up tables, which are five-dimensional here, consume a lot of memory, and a small refinement in the table content would significantly increase the memory footprint. For example, a table with a discretization of $50 \times 30 \times 20 \times 20 \times (4+27)$ occupies 240 MB whereas one with a discretization of $75 \times 50 \times 40 \times 40 \times (4+27)$ consumes 2.415 GB. This is the "curse of dimensionality". On the other hand, the memory occupied by

the DFNN-BC parameters is minimal, about four to five orders of magnitude smaller than its tabulation counterpart. This is a significant advantage of neural networks, which can considerably reduce the memory usage.

Table 5.10: Comparison of memory requirement between DFNN-BC and tabulation methods.

Model	Architecture	Memory [MB]
DFNN-BC	$n_{\text{in}} = 4,\ \mathscr{H} = \begin{bmatrix} 9 & 9 & 9 & 9 \end{bmatrix}^T,\ n_{\text{out}} = 27$	5.6×10^{-2}
Look-up table	$50 \times 30 \times 20 \times 20 \times (4 + 27)$	2.40×10^{2}
Look-up table	$75 \times 50 \times 40 \times 40 \times (4 + 27)$	2.415×10^{3}

5.5.3 Evaluation of the Coupled DFNN-BC and LES Approach

Two separate simulations are carried out with identical numerical setup except for the evaluation of real-fluid properties: Case 1 – Baseline, where properties are computed using the brute force equations described in Sec. 5.3; Case 2 – DFNN-BC, where properties are computed using the DFNN-BC model. The comparison between the two cases is conducted in terms of both instantaneous and time-averaging flow features.

5.5.3.1 Instantaneous Flowfield

Figure 5.5 shows the evolution of the density field in the mixing cup region between $t = 0$ (kerosene injection) and $t = 1.0$ ms. Qualitatively, the results for the two cases are very similar. Kerosene mixes quickly with the GOX flow while entrained downstream. The resultant structures for the two cases appear to be similar; but, their exact positions are not. In addition, finer scales are observed at the exit of the fuel annulus in the baseline simulation compared to the DFNN-BC case. These phenomena may be attributed to the numerical errors introduced by approximating real-fluid properties in a neural network. The errors propagate from the state equations to the conservation equations and cause the damping of the flow motions. Nevertheless, most of the salient features of the flowfield are well captured by the DFNN-BC model. Figure 5.6 shows snapshots of the density field at $t = 16$ ms. First, the wall shear layers from the GOX post and fuel annulus interact and merge together to form a new shear layer convecting downstream, a situation known as the axial shear-layer instability. In the meantime, the liquid film fluctuates in the axial direction. At the exit of the recess region, kerosene is largely entrained into the GOX stream. Both cases show the formation of a recirculation zone caused by the adverse pressure gradient in the taper region.

Figure 5.7 shows the power spectral density (PSD) of pressure fluctuations for the two cases. The PSD is calculated as follows:

$$\text{PSD}(f) = \frac{2 \times |\text{DTFT}(p(t))|}{N_p}, \tag{5.22}$$

where DTFT is the discrete-time Fourier transform operator, f the frequency, and N_p the number of samples in the pressure signal. Also shown in Fig. 5.7 is the instantaneous density field of the baseline case.

Probe 1 is located within the GOX post, where the acoustic oscillation is primarily longitudinal. The frequencies of the dominant modes of pressure fluctuations are about 2.8 and 3.2 kHz for the baseline and DFNN-BC, respectively. Probe 2 is located in the kerosene stream. The measured frequency spectrum is similar to that of Probe 1. Probe 3 is located at the beginning of the taper region where intensive mixing happens. The dominant pressure modes at this location have frequencies of about 10.8 and 10.9 kHz for baseline and DFNN-BC, respectively. As the mixture moves downstream and spreads outwards, the pressure oscillations become much smaller, as indicated by the signal at Probe 4. Overall, the DFNN-BC model captures well the dynamics of the flow. In addition, it is observed from Fig. 5.7 that the maximum pressure fluctuation is less than 1 bar, about 0.4% of the operating pressure in the injector (253 bar). This justifies why pressure was not included in the input layer of the neural network.

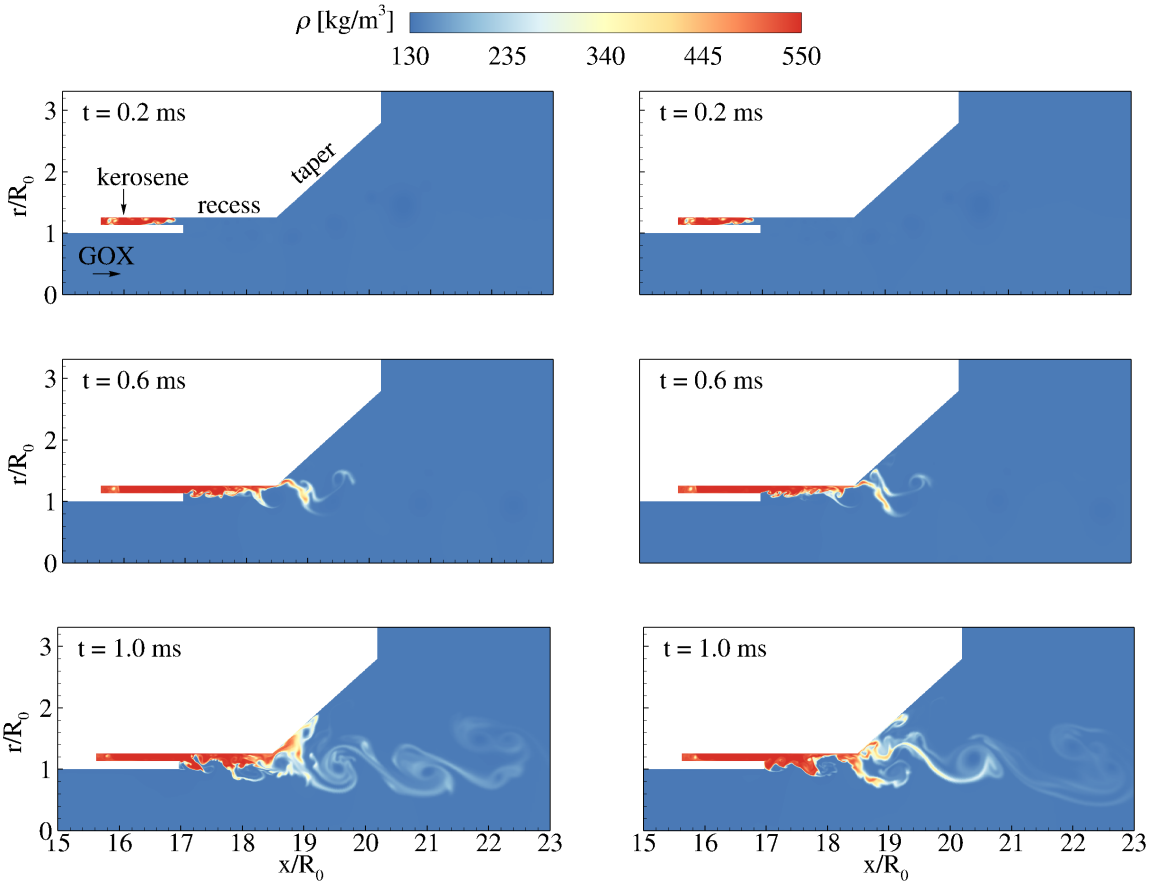


Figure 5.5: Temporal evolution of the density field for the baseline (left) and DFNN-BC (right) cases.

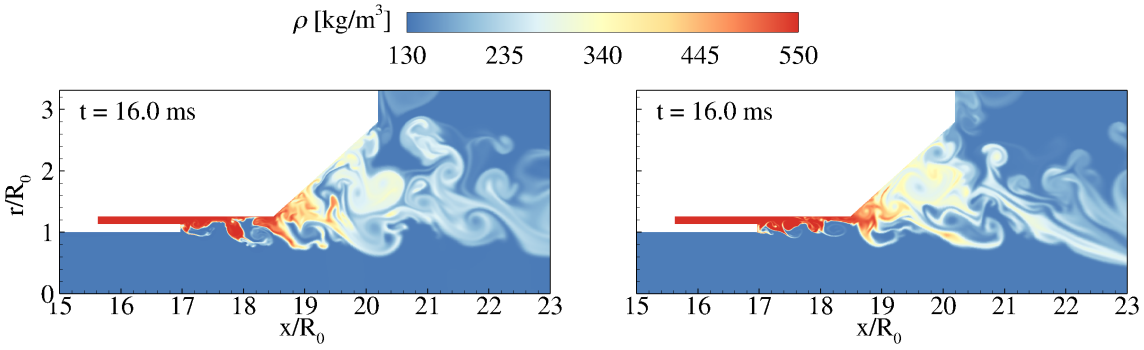


Figure 5.6: Snapshots of the evolved density field at $t = 16$ ms for the baseline (left) and DFNN-BC (right) cases.

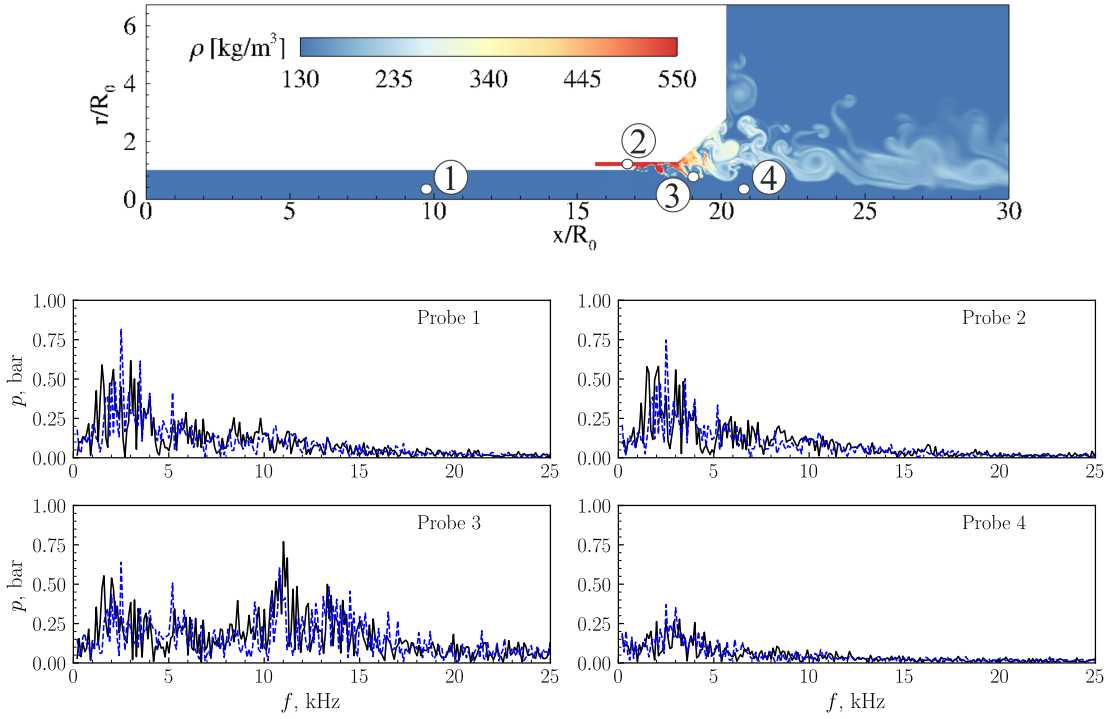


Figure 5.7: Frequency spectra of pressure fluctuations at different positions for the baseline (solid, black lines) and DFNN-BC (dashed, blue lines).

5.5.3.2 Time-Averaged Flowfield

Figure 5.8 shows the distributions of the time-averaged mixture fraction and temperature for the baseline and DFNN-BC cases. Averaging for a period of 10 ms was performed after the flowfield had reached a quasi-stationary state at $t = 5.0$ ms. Good agreement is achieved between the two cases. The propellant mixing is well predicted by the DFNN-BC model. The isoline of $Y_F = 0.85$ (fuel dominant) starts at the exit of the fuel annulus and disappears slightly downstream at the entrance of the taper

region. The central GOX stream, enclosed by the isoline of $Y_F = 0.05$, flows all the way to the downstream region. The mixing effectiveness, indicated by the isoline of $Y_F = 0.25$ (close to the stoichiometric mixture ratio), ends at $x/R_0 = 27$ for the baseline case, compared to $x/R_0 = 27.4$ for the DFNN-BC case. The spatial evolution of the three temperature isolines ($T = 550$ K, 610 K, and 670 K) also suggests the accuracy of the prediction by the DFNN-BC model.

Further analysis is conducted in terms of radial profiles of the time-averaged quantities. Figure 5.9 shows the distributions of the axial velocity, temperature, mixture fraction, and density at three different axial positions in the injector, $x/R_0 = 18,\ 20,$ and 25, corresponding to the recess, taper, and downstream regions, respectively. Very good agreement is observed among the profiles. Some minor discrepancies, however, are noted, in particular at $x/R_0 = 20$, where a local geometric change induces strong flow distortions, and at $x/R_0 = 25$, where the peaks in the temperature and density profiles are underestimated by the DFNN-BC model. For completeness, the relative errors for the time-averaged radial profiles predicted by the DFNN-BC model at different axial locations are summarized in Table 5.11. The relative error based on the L_2-norm, ϵ_{rel}, is defined as:

$$\epsilon_{\text{rel}} = \frac{\|\boldsymbol{q}^* - \boldsymbol{q}\|_2}{\|\boldsymbol{q}^*\|_2} = \frac{\sqrt{\sum_i (q_i^* - q_i)^2}}{\sqrt{\sum_i q_i^{*2}}}, \tag{5.23}$$

where $\boldsymbol{q}^*$ is a vector representing the true profile and $\boldsymbol{q}$ is a vector representing the profile predicted by the coupled DFNN-BC and CFD approach. Overall, the prediction errors are less than 1% for T, less than 6% for u and ρ, and less than 9% for Y_F.

Table 5.11: Relative errors in percentage (%) for the time-averaged radial profiles predicted by the DFNN-BC model at different axial locations in the injector. Results are shown for different field variables.

Variable	$x/R_0 = 18$	$x/R_0 = 20$	$x/R_0 = 25$
u	1.51	5.13	5.95
T	0.82	0.87	0.70
Y_F	8.85	6.19	8.59
ρ	5.45	4.13	2.32

5.5.4 Computational Cost

The assessment of computational cost was performed locally on a Linux cluster with the following specifications per node: 128 GB of memory and 64 cores of AMD Opteron 6276 and 6279 CPU. Each simulation is carried out using a total of 469 cores. The cost of 1 ms of physical time is about 2,300 CPU-hours for the baseline case. Figure 5.10(left) shows the time distribution of the different kernels of the `PMBFS` solver for one pseudo-time step. Here, only the computational time is taken

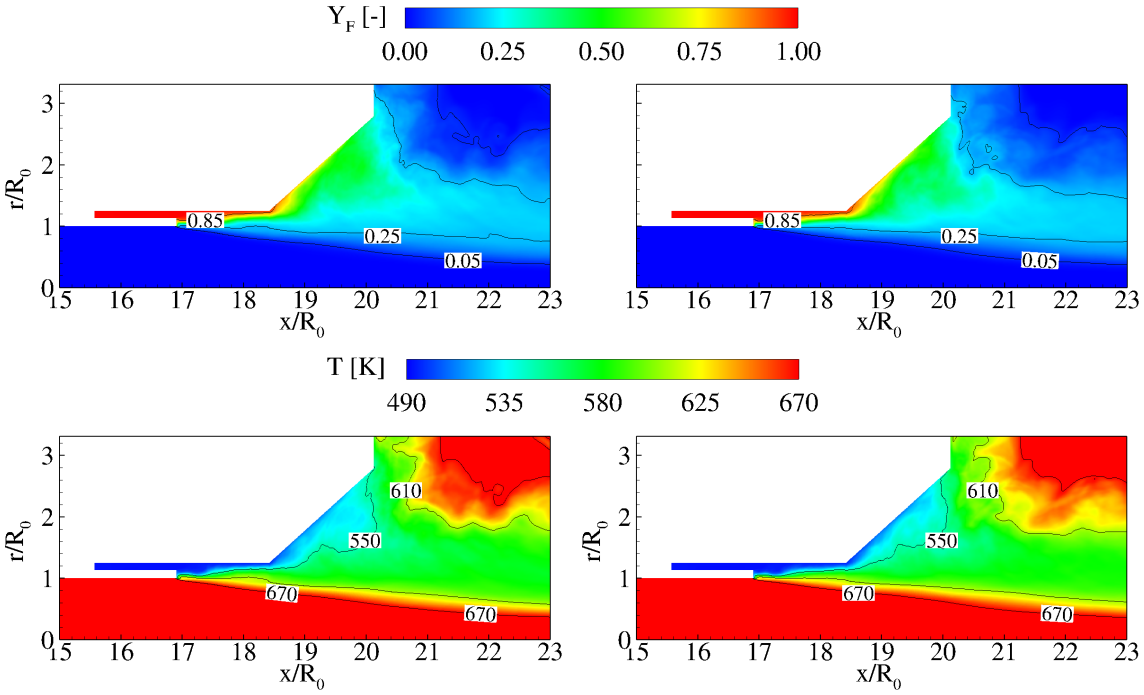


Figure 5.8: Time-averaged distributions of fuel mass fraction and temperature fields for the baseline (left) and DFNN-BC (right) cases.

into account; the communication time in the MPI process is not included. The total computational time for one pseudo-time step in the baseline case is about 0.142 s wall time. The distribution is as follows: 39.2% for fluxes and other basic routines like update of flow variables (Steps 1 and 3 of Algorithm 2), 2.5% for the Jacobian and preconditioning matrix (Step 2), 56.3% for real-fluid properties (Steps 4-6), and 2.0% for turbulence closures (Step 6). The evaluation of real-fluid properties clearly dominates the use of computational time. Figure 5.10(right) shows the time distribution of the different components of the property kernels: 16.2% for the compressibility and density (Step 4), 77.2% for the remaining thermodynamic quantities and their derivatives (Step 5), and 6.7% for transport properties (Step 6). Step 5 poses a severe limitation to the computational efficiency of the `PMBFS` solver. Table 5.12 compares the computational times between the baseline and DFNN-BC cases. Significant improvement is achieved in the efficiency of property evaluation using the neural network. The speedup is 2.43 times. The time saving for the entire simulation is 1.5 times.

Table 5.12: Computational time distributions of kernels per pseudo-time step.

	Total time (s)	Properties (s)	Remaining routines (s)
Baseline	0.142	0.080	0.062
DFNN-BC	0.095	0.033	0.062

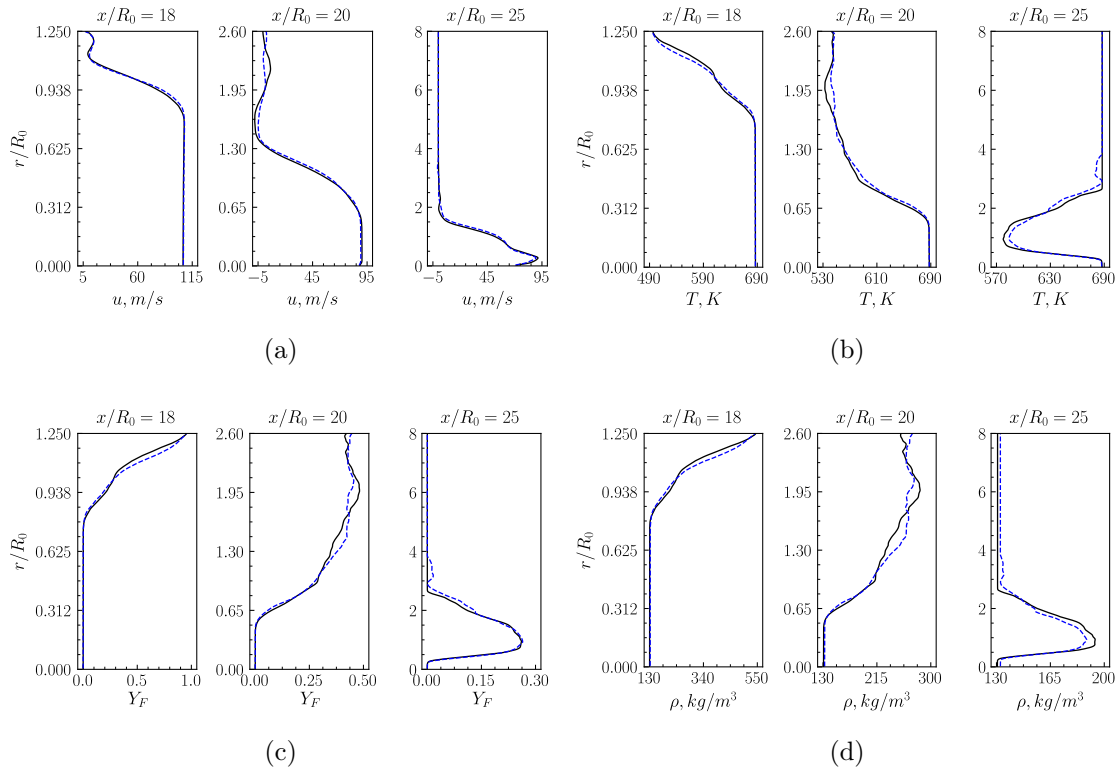


Figure 5.9: Radial distributions of time-averaged axial velocity (a), temperature (b), fuel mass fraction (c) and density fields (d) at different axial locations ($x/R_0 = 18,\ 20$ and 25) in the injector for the baseline (solid, black lines) and DFNN-BC (dashed, blue lines) cases.

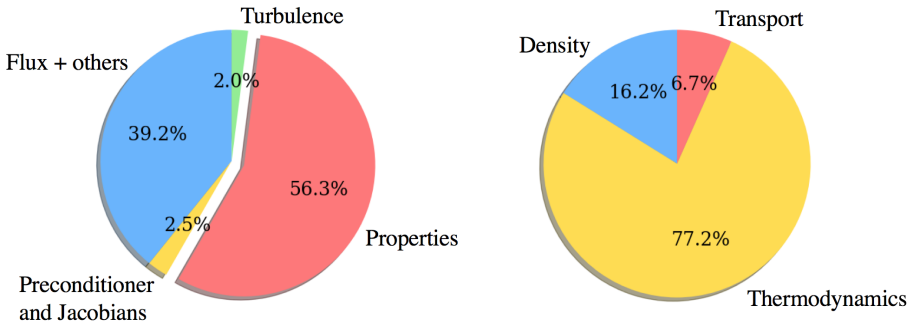


Figure 5.10: Computational time distribution of the different kernels in the `PMBFS` solver (left). Computational time distribution of the different components of the property kernels (right).

5.6 Counterflow Diffusion Flame

5.6.1 Computational Setup

Counterflow diffusion flames have been extensively studied over the last several decades [171, 172, 173, 174]. In this section, we consider a counterflow diffusion flame for cryogenic hydrogen combustion with an oxygen/nitrogen mixture ($0.5N_2/0.5O_2$ by mole fraction) at high pressures, as shown schematically in Fig. 5.11. This test case is significantly different from the rocket injector problem discussed in Sec. 5.5. The flow is quasi-one-dimensional, laminar, and reacting. Calculations are carried out using the `One-D ThermoCode` conservative-variable-based algorithm, described in Sec. 5.3.2.2. The separation distance between the fuel and oxidizer is $L = 0.01$ m. Their inlet velocities are $u_\mathrm{f} = 2.742$ m/s and $u_\mathrm{o} = -0.258$ m/s, respectively, and the inlet temperatures $T_\mathrm{f} = 295$ K and $T_\mathrm{o} = 140$ K, respectively. This gives a strain rate $\epsilon = 300$ s^{-1}. The operating pressure is $p = 7$ MPa (70 bar). The high-pressure $H_2/N_2/O_2$ chemical-reaction mechanism developed by Burke et al. [175] is employed. It contains 9 species (H, H_2, O, OH, H_2O, O_2, N_2, HO_2, and H_2O_2) and 27 reactions. The numerical grid contains 992 cells, and the CFL number is set to 1.2 for numerical stability, leading to a time step about 5.6×10^{-10} s.

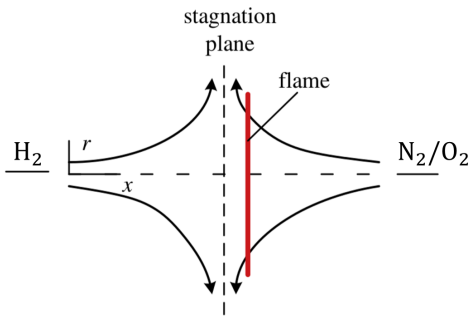


Figure 5.11: Schematic of $H_2/N_2/O_2$ counterflow diffusion flame [173].

5.6.2 Neural Network Design

The input layer contains the list of conservative variables, $(\rho, e, Y_k; k = 1, 2 \ldots, N-1)$, from which the thermophysical properties are retrieved as prescribed by Algorithm 3. While the total number of species is $N = 9$, the mass fractions of HO_2 and H_2O_2 are much smaller compared to the other species, and therefore, their effect can be neglected. The input layer thus consists of the following eight nodes: ρ, e, Y_H, Y_{H_2}, Y_O, Y_OH, $Y_{\mathrm{H}_2\mathrm{O}}$, Y_{O_2}, with the following ranges for training: $\rho \in [4.254, 642.114]$ kg/m^3, $e \in [-3031.692, -19.682]$ kJ/kg, $Y_\mathrm{H} \in [0, 0.00104]$, $Y_{\mathrm{H}_2} \in [0, 1]$, $Y_\mathrm{O} \in [0, 0.00867]$, $Y_\mathrm{OH} \in [0, 0.05430]$, $Y_{\mathrm{H}_2\mathrm{O}} \in [0, 0.51272]$, and $Y_{\mathrm{O}_2} \in [0, 0.53320]$. The mass fraction of nitrogen, Y_{N_2}, can be obtained by mass conservation (i.e., $\sum_i Y_i = 1$) and is not

included in the input layer. The output layer consists of all the variables listed in Table 5.4, referring to Algorithm 3. The total number of these variables is 21. The same methodology described in Sec. 5.4.5 is employed to design the DFNN-BC model except that the EOS is different (the PR EOS). A DFNN-BC comprising of 2 hidden layers and 25 neurons per layer is chosen by manual tuning. The R^2-scores for the training and validation data are 0.953 and 0.952, respectively.

5.6.3 Evaluation of the Integrated DFNN-BC and CFD Approach

Two separate simulations are carried out with identical numerical setup except for the evaluation of real-fluid properties: Case 1 – Baseline, where properties are computed using the brute force equations described in Sec. 5.3; Case 2 – DFNN-BC, where properties are computed using the DFNN-BC model. Figure 5.12 shows the spatial profiles of density, internal energy, temperature, and mass fractions of selected species for both the baseline and DFNN-BC cases. Excellent agreement is observed between the two cases. The maximum flame temperature is about 3,460 K for the DFNN-BC case compared to 3,450 K for the baseline case. This corresponds to a relative error of 0.3%. The flame structure is also well predicted by the DFNN-BC case, as evidenced by the distributions of species' mass fractions.

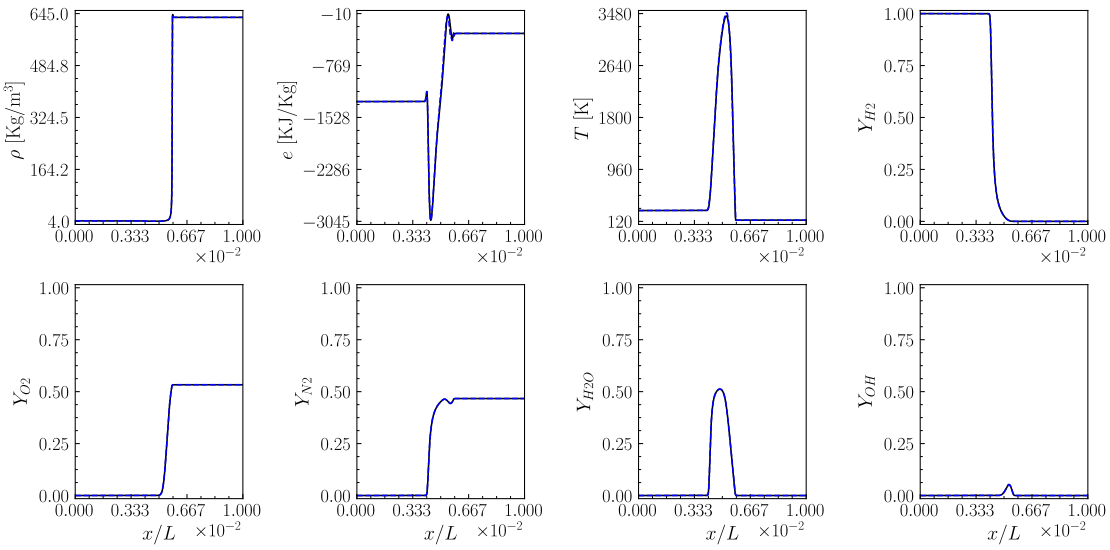


Figure 5.12: Spatial profiles of density, internal energy, temperature and mass fractions of selected species for the baseline (solid, black lines) and DFNN-BC (dashed, blue lines) cases.

5.6.4 Computational Cost

The computational cost was assessed on a Linux laptop that has the following specifications: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz. Each simulation was carried

out using 2 cores. Table 5.13 compares the computational time distribution of different kernels between the baseline and DFNN-BC cases. Significant improvement is achieved in the efficiency of property evaluation using the neural network. The speedup is about 3.7 times, while that for the entire simulation is 2.3.

	Total time (s)	Properties (s)	Remaining routines (s)
Baseline	0.159	0.123	0.036
DFNN-BC	0.069	0.033	0.036

Table 5.13: Computational time distributions of kernels per Runge-Kutta step.

5.7 Summary

The development of numerical methods for efficient evaluation of EOS has been identified as an important step for effective and accurate simulations of supercritical fluid flows. In this chapter, a neural network model, referred to as DFNN-BC, was proposed to replace the computationally expensive solution of the EOS. It allows for fast calculations of real-fluid thermophysical properties at conditions of interest. The DFNN-BC features a deep feedforward neural network that has provably improved prediction accuracy when incorporating boundary information into the model. The scheme can be coupled to a CFD solver in a robust manner. Such improvement was obtained by repeating each boundary point multiple times in the training data and monitoring two loss functions during the model training process, one for the interior points and the other for the boundary points, to ensure high accuracy throughout the entire input domain.

The method was evaluated with two model problems for supercritical fluid flows. The first treated LES of supercritical turbulent mixing in a GCLSC injector using a primitive-variable-based formulation and the SRK EOS. A kerosene/oxygen mixture composed of 4 species was considered. A neural network with 4 hidden layers and 9 neurons per hidden layer was designed to approximate the thermophysical variables, with temperature and chemical composition used as channels in the input layer. The R^2-score of the selected neural network was 0.98. The calculated result from the integrated DFNN-BC and LES approach compared well to the baseline simulation. The prediction errors of time-averaged data were less than 1% for temperature, less than 6% for axial velocity and density, and less than 9% for fuel mass fraction.

The second problem treated quasi-one-dimensional simulations of laminar counterflow diffusion flames for cryogenic combustion using a conservative-variable-based formulation and the PR EOS. A hydrogen/oxygen/nitrogen mixture composed of 7 major species was considered. A neural network with 2 hidden layers and 25 neurons per hidden layer was developed to achieve the desired approximation, with density, internal energy and chemical composition taken as channels in the input layer. The R^2-score of the selected neural network was 0.952 on validation data. The numerical prediction obtained using the integrated DFNN-BC and CFD approach showed very

good agreement with the baseline simulation. The maximum flame temperature was predicted with a relative error of 0.3%.

In addition to the *a priori* and *a posteriori* analyses, a time complexity analysis was conducted. Results showed that the DFNN-BC model accelerated the evaluation of real-fluid properties by a factor of 2.43 and 3.7 for the two test cases, respectively, and the overall flow simulation by 1.5 and 2.3, respectively. Further, the memory usage was reduced by up to five orders of magnitude in comparison with the table look-up method.

List of Main Symbols

Latin Symbols

a	speed of sound
$a\alpha$, b	variables accounting for intermolecular forces and volume of the molecules
$A_T, A_\rho, A_{Y_k}, B_{Y_k}$	preconditioning related terms
$\boldsymbol{b}$	bias vector
c_p	specific heat at constant pressure
c_v	specific heat at constant volume
C	progress variable
D_{jk}	binary mass diffusion coefficient between the jth and kth species
D_k	mixture diffusion coefficient of the kth species
e	mass-specific internal energy of the mixture
e_t	mass-specific total energy of the mixture
f	generic function, frequency
$\boldsymbol{F}$	matrix of inviscid fluxes
$\boldsymbol{F}_\mathrm{v}$	matrix of viscous/turbulent fluxes
h	mass-specific enthalpy of the mixture
h_k	partial-mass enthalpy of the kth species
h_t	mass-specific total enthalpy of the mixture
J	loss function
$\boldsymbol{J}$	Jacobian matrix
l	number of hidden layers
L	total number of layers (excluding input layer from the count)
L_hidden	number of hidden layers
$\dot{m}$	mass flow rate
M	total number of pseudo-time steps
n_i	number of input parameters to the table or manifold
n_in	size of input layer
n_out	size of output layer
N	number of species, number of samples
p	pressure
p_g	gauge pressure
r	radial coordinate

R_0	injector interior radius
Re	Reynolds number
R_u	universal gas constant
t	physical time
T	temperature
$\boldsymbol{T}$	preconditioning matrix
u	axial component of velocity
$\boldsymbol{u}$	velocity vector
v, V	transverse component of velocity
w	velocity component in the z direction
$\dot{w}$	mass production rate
W	molecular weight of the mixture
W_k	molecular weight of the kth species
$\boldsymbol{W}$	weight matrix
x, y	spatial Cartesian coordinates
$\boldsymbol{x}$	input layer of neural network
$\boldsymbol{y}$	output layer of neural network
Y_k	mass fraction of the kth species
Z	compressibility factor, mixture fraction
$\widetilde{Z''^2}$	variance of mixture fraction

Calligraphic Symbols

$\boldsymbol{\mathcal{H}}$	vector of neurons per layer in all the hidden layers
$\boldsymbol{\mathcal{P}}$	vector of primitive variables
$\boldsymbol{\mathcal{Q}}$	vector of conservative variables
$\boldsymbol{\mathcal{S}}$	vector of source terms
$\boldsymbol{\mathcal{V}}_k$	diffusion velocity vector of the kth species

Greek Symbols

γ	specific heat ratio
$\boldsymbol{\delta}$	Kronecker delta matrix
ϵ	parameter defining the specific EOS
ϵ_{rel}	relative error
ξ	parameter that re-scales the eigenvalues
$\boldsymbol{\theta}$	parameters of neural network
λ	heat conductivity
λ_{lr}	initial value for learning rate
λ_{reg}	control parameter for L_2 regularization
μ	dynamic viscosity
ρ	density of the mixture
ρ_k	density of the kth species

σ	parameter defining the specific EOS, activation function
τ	pseudo-time
χ_k	mole fraction of the kth specis

Subscripts

bc	boundary points in the thermophysical domain
c	critical values
f, F	fuel
in	inlet
int	interior points in the thermophysical domain
o	oxidizer
train	training set

Superscripts

sgs	subgrid-scale
$\square^*$	ground truth variable
$\breve{\square}$	partial-density variable

CHAPTER 6
DATA-DRIVEN DEEP LEARNING EMULATORS FOR PARAMETRIC AND EFFICIENT PREDICTION OF FLUID FLOW PROBLEMS

In the previous chapter, deep learning was used to replace only certain subroutines in the CFD solver, and thus acted as a software accelerator for numerical simulations. This made possible a considerably faster coupled deep-learning and physical-simulation framework, while maintaining high-fidelity results. The flow equations, however, were still marched forward in time during the simulation, which limited the maximum possible speedup.

In this chapter, another area of scientific deep learning, data-driven surrogate modeling, is explored in order to address Objective 3(a) of this thesis (cf. Sec. 1.2). Here, basically, deep learning is used to replace the full CFD solver. Such an approach can enable significant simulation speedups and accelerate the process of design space exploration and optimization in many-query studies. Here, the theoretical and numerical formulations of the proposed deep learning spatiotemporal surrogates are laid down. The models are analyzed in detail on representative canonical problems (models are applied to an engineering problem in Chapter 7). Lastly, please note that this chapter builds directly on Chapter 2.

Some of the material presented here is adapted from:

- P. J. Milan et al., "Data-driven deep learning surrogates for spatiotemporal emulation of fluid systems" *(manuscript in preparation)*.
- [176] P. J. Milan, G.M. Magnotti, and V. Yang, "Data-driven deep learning surrogates for parametric prediction of reacting flows," *ILASS-Americas 2021*, Paper ID 49, pp. 1-13, 2021.

6.1 Abstract

Surrogate modeling has become a popular approach to efficient design optimization and UQ of highly dimensional flowfields. Even with the progress that has been achieved to date, however, successful application of surrogate models to multiscale multiphysics flows remains a challenge. The major issues can be attributed to the wide range of spatiotemporal scales, strongly coupled thermophysical processes, and complex geometries. The present work develops a novel data-driven surrogate framework based on deep learning techniques for efficient prediction of spatiotemporal flow dynamics. Specifically, we first train an emulator model from training data generated from a full-order model (FOM). The emulator features a fully-connected autoencoder (FCAE), for nonlinear dimensionality reduction of the flowfields, and a regression-based neural network ($\mathcal{R}$) for supervised learning of the latent space representations.

The proposed emulator is compared with classical surrogate modeling approaches. Subsequently, we also implement a variant of this framework in which the FCAE is replaced by a convolutional autoencoder (CAE) that can retain spatial coherence of the input data by means of convolution layers. The two emulators, referred to as FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively, are validated in two nonlinear fluid flow problems. The first is the one-dimensional viscous Burgers equation representative of the advection of a shock profile. The second is the two-dimensional advection-diffusion-reaction (ADR) equation representative of the evolution of a premixed hydrogen flame. In each problem, the emulators are examined for their computational efficiency and their ability to predict the flow features for unseen parameter instances in the design space.

6.2 Introduction and Related Work

Numerical simulations of multiscale multiphysics flows described by systems of nonlinear partial differential equations (PDEs) provide considerable insight into underlying processes for fundamental and applied sciences. The numerical solutions to these PDEs often depend on a set of input parameters, which comprises physical conditions, fluid properties, and geometrical parameters, and such PDEs are referred to as *parametrized PDEs*. In many scenarios, such as sensitivity analysis and design optimization, high-fidelity solutions of parametrized PDEs need to be computed for many different points in the parameter domain. A single run of a high-fidelity simulation, however, often requires very fine spatiotemporal grids, long running times, and high memory usage, to adequately characterize the flowfields. Consequently, performing several executions of a high-fidelity simulation to explore the full parameter domain can be computationally infeasible, especially for large-scale engineering and industrial problems.

To alleviate this situation, surrogate modeling approaches have been proposed to replace expensive computer simulations. The common goal of these approaches is to build a fast-running, yet accurate, surrogate model that can be called instead of the original high-fidelity model to predict approximate solutions for new input parameter instances. In general, such models are developed via data-driven techniques in four major steps: (1) extract snapshots from high-fidelity simulations conducted at designated sampling points in a given parameter domain, or design space, using a DoE method, (2) compress the high-dimensional snapshots to reduced representations using a ROM technique, (3) fit a regression model to learn the relationship between the input parameters and the reduced data, and (4) integrate the previous modules into a unified framework along with a reconstruction algorithm to allow for new predictions. Traditionally, Step 2 has relied on POD, DMD, or Galerkin projections (or a variant of these methods) while Step 3 has relied on GPs or polynomial-based approximations [177, 178, 179, 180, 181, 182]. Although these methods can be effective at times, they have many drawbacks. POD, DMD, and Galerkin projections, for instance, are linear methods; they are challenging to apply to ROM for highly nonlinear flows, because a large number of modes is required to accurately reconstruct the flowfield [17]. GPs and polynomial-based curve/surface fitting require certain prior assumptions, which may limit the representation capacity of the model and give rise to robustness

issues [13]. Another drawback is the associated computational cost, especially for GP-based models, which have a time complicity of $\mathcal{O}(n^3)$, with n being the size of the training set [13]. The development of efficient and reliable surrogate models for such nonlinear systems is an active area of research.

In recent years, deep learning has been widely applied in various fields of science and technology due to its ability to account for nonlinear relationships within the data [55, 183]. Autoencoders, a type of DNN, can be used for dimensionality reduction. In their seminal work, Hinton and Salakhutdinov [19] used a multi-layer autoencoder to extract codes, i.e., compressed representations, for all of the handwritten digits in the MNIST training set [184], and showed that the autoencoder can produce higher-quality reconstructions than PCA. Similar results were also obtained in the more recent contribution by Almotiri et al.[185], who have used parallel processing to reduce the training time considerably. In the context of fluid problems, Omata et al. [29] and Murata et al. [30] utilized a CAE to extract the essential features of free-shear flows. Maulik et al. [17] combined CAEs with RNNs for parametric ROMs of one-dimensional and two-dimensional advection-dominated inviscid systems. Agostini et al. [186] combined the autoencoder network with a clustering algorithm to provide a probabilistic low-dimensional dynamical model applied to unsteady flow around a cylinder. These works highlight that autoencoders are able to extract nonlinear modes with lower reconstruction errors at the same level of data compression than those of POD; however, most of these studies used the autoencoders in a standalone manner under a fixed design point, or with parametric variations in a single input parameter.

Building on these promising results, a new data-driven surrogate framework that is entirely based on deep learning techniques is proposed in this chapter to efficiently emulate complex spatiotemporal flows in a multi-parametric setting, as shown in Fig. 6.1. The surrogate model is trained using a database drawn from numerical simulations for a set of sampling points. An FCAE is then built to compress the representation of the flowfield data. Following the autoencoder, a regression-based neural network ($\mathcal{R}$) is used to learn the encoded reduced-space representations for flowfield reconstruction at new design settings. We also propose a variant of this framework where a CAE is used in lieu of the FCAE to retain the spatial coherence of the flowfield data by means of convolution layers, while keeping the regression method unchanged. CAEs have shown tremendous success in compressing large-scale image data for computer vision applications [187, 61], and thus are promising tools to further enhance surrogate modeling capability.

The surrogate models considered in this study are investigated in two nonlinear problems: the one-dimensional viscous Burgers equations, and the two-dimensional numerical simulation of a hydrogen-air premixed flame governed by the ADR equation. These representative problems allow us to carefully assess the performance of the models in simplified settings before moving to more complex engineering flows in the next chapter.

The rest of the chapter is structured as follows. Section 6.3 introduces the various data-driven strategies, including the proposed FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$ frameworks. Section 6.4 is concerned with the application of the surrogates to the one-dimensional

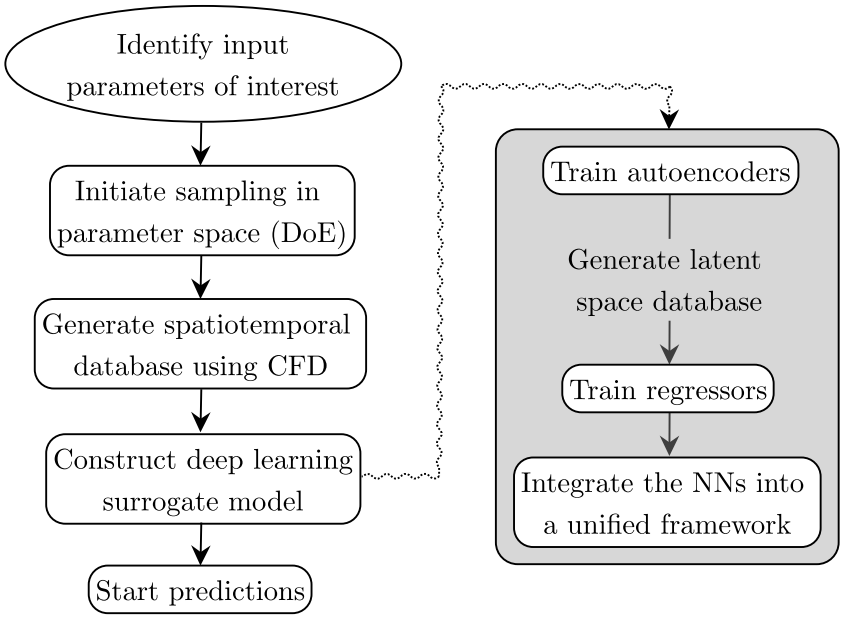


Figure 6.1: Flowchart of building the deep learning surrogate model.

viscous Burgers equation, while Sec. 6.5 is concerned with the application to the two-dimensional ADR equation. In each problem, the FOM formulation, including the governing equations, computational setup, and DoE, and the results are presented. Finally, the conclusions are reported in Sec. 6.6.

6.3 Deep Learning Emulators

A high-level overview is first presented for the two deep learning surrogates that are developed in this chapter. Descriptions are then provided for the deep learning architectures for the autoencoders and regressors. Finally, the algorithm for the prediction phase is explained.

6.3.1 Overview of the Emulator Models

Let $\mathcal{S}$ be a surrogate that approximates the FOM, i.e., $\mathcal{S} \approx \text{FOM}$, and $\boldsymbol{w}$ be a vector comprised of the spatiotemporal field variables of interest, i.e., temperature and velocity components. Two variants of $\mathcal{S}$ are developed hereafter, namely (1) FCAE-$\mathcal{R}$, and (2) CAE-$\mathcal{R}$. In both models, sequences of snapshots at several known design points are used as training data, and the sequence of snapshots at a new parameter in the same time interval is predicted. In the prediction phase, the input is a new parameter $\boldsymbol{\mu}'$, and the output is $\boldsymbol{w}_{\text{pred}}(\boldsymbol{x}, t; \boldsymbol{\mu}')$, which is an approximation to $\boldsymbol{w}(\boldsymbol{x}, t; \boldsymbol{\mu}')$ obtained from the FOM.

Each model comprises two levels of neural networks to approximate the spatiotemporal flow in the physical problem under consideration, as shown in Fig. 6.1. In the

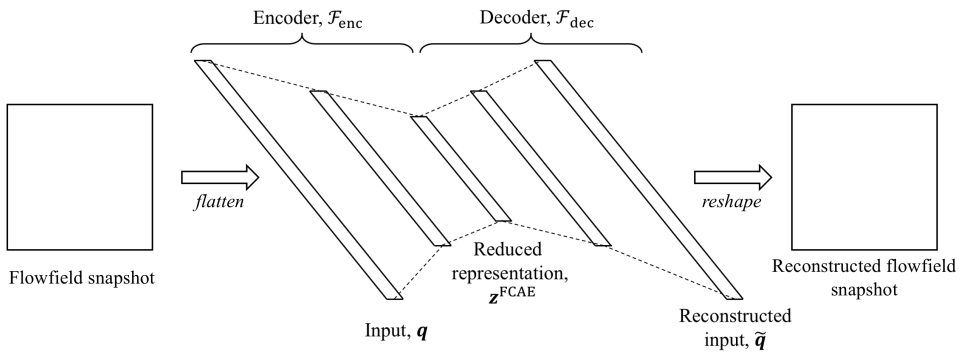


Figure 6.2: Schematic representation of an FCAE.

first level, an autoencoder is used to perform spatial compression of the sequence of high-dimensional snapshots into a sequence of latent vectors. In FCAE-$\mathcal{R}$, the autoencoder has a fully-connected neural network architecture, whereas in CAE-$\mathcal{R}$, it has a convolutional neural network architecture. Differences between the FCAE and CAE are discussed in more details in Secs. 6.3.2.1 and 6.3.2.2. Following the autoencoder, a regressor $\mathcal{R}$ is used in the second level to learn the relationship between the design parameters and latent space from the training data, and to predict the sequence of latent vectors for a new parameter $\boldsymbol{\mu}'$. Lastly, the flowfield in physical space is reconstructed from the latent vectors using the decoder function of the autoencoder. Both surrogate models are implemented using the `Keras` API in `TensorFlow 2.4` (cf. Sec. 2.3.9).

6.3.2 Constituent Parts

6.3.2.1 Fully-Connected Autoencoder

The encoder is comprised of an input layer as well as multiple hidden layers with activation functions, while the decoder is defined as the mirror image of the encoder. The nonlinearity of the activation functions has been noted as a key attribute of autoencoders in extracting nonlinear features with low reconstruction errors (cf. Secs. 2.2.5 and 2.2.7). The forward propagation procedure in the autoencoder was previously described in Sec. 2.2.5.1, and is repeated here for clarity:

$$\begin{aligned}\widetilde{\boldsymbol{q}} &= \mathcal{F}_{\text{dec}} \circ \mathcal{F}_{\text{enc}}(\boldsymbol{q}) \\ &= \mathcal{F}_{\text{dec}}(\boldsymbol{z}),\end{aligned} \tag{6.1}$$

where $\boldsymbol{q} \in \mathbb{R}^{n_{\text{in}}}$ and $\widetilde{\boldsymbol{q}} \in \mathbb{R}^{n_{\text{in}}}$ denote the encoder input and decoder output, respectively, and $\boldsymbol{z} \in \mathbb{R}^{n_{\text{latent}}}$ is the latent vector, with n_{in} and n_{latent} being the dimensions of the input layer and latent vector, respectively. The loss function is taken as the regularized MSE between the input and output. Please note the following:

- FCAEs can be applied to any type of data (structured or unstructured, regular or irregular) since the snapshots are flattened to one-dimensional vectors before being processed by the network.
- In this study, we will train one FCAE per field variable, and consequently n_{in} will correspond to the number of grid points n_{grid} in the subdomain of interest for emulation. Alternatively, we could have also stacked the field variables into a single vector, and in that case, n_{in} would have been equal to $n_{\text{grid}} \times n_{\text{var}}$, where n_{var} is the number of field variables. However, such an approach can result in large vector sizes and complicate the process of model fitting and hyperparameter search, and thus it was not pursued here.

6.3.2.2 Convolutional Neural Network Autoencoder

A CAE is a type of CNN utilizing convolution, sampling, and dense layers for dimensionality reduction (cf. Sec. 2.2.5.2). Typically, in traditional FCAEs, the input data is flattened to a one-dimensional vector before being processed, which prevents direct learning of spatial coherence. Furthermore, each neuron in a layer is connected to all neurons in the next layer, where each connection is a parameter in the network [54]. This can result in a very large number of trainable parameters. Instead of using only dense layers, a CAE uses some local connectivity between neurons, i.e., a neuron is only connected to nearby neurons in the next layer, which allows to significantly reduce the total number of parameters in the network and process image data directly, thus retaining spatial information.

The CAE is composed of a CNN encoder, $\mathcal{G}_{\text{enc}}$, and a CNN decoder, $\mathcal{G}_{\text{dec}}$, as shown in Fig. 2.7. The forward propagation procedure in the CAE was previously described in Sec. 2.2.5.2, and is repeated here for clarity:

$$\begin{aligned} \widetilde{\boldsymbol{Q}} &= \mathcal{G}_{\text{dec}} \circ \mathcal{G}_{\text{enc}}(\boldsymbol{Q}) \\ &= \mathcal{G}_{\text{dec}}(\boldsymbol{z}), \end{aligned} \tag{6.2}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n_x \times n_y \times n_c}$ and $\widetilde{\boldsymbol{Q}} \in \mathbb{R}^{n_x \times n_y \times n_c}$ denote the encoder input and decoder output, respectively, and $\boldsymbol{z}$ the latent space representation. The loss function is taken as the MSE between the input and output. Please note the following:

- Although the same symbol $\boldsymbol{z}$ is used, the latent space obtained from the CAE is different than that from the FCAE.
- Classical CAEs (such as the ones used here) can only be applied to structured data defined on rectangular domains with uniform grid spacing. The formulation, however, can be modified to allow for treatment of non-uniform grids and irregular domains, but this is left as future work.
- In this study, we will train one CAE per field variable, and consequently n_c will bet set to 1 in each autoencoder. Alternatively, we could have also trained one CAE for all the field variables (and set n_c to n_{var}), however, this was not done here with the intent to be consistent with the FCAE.

6.3.2.3 *Regressor for Latent Space*

A regressor, $\mathcal{R}$, is constructed to learn the relationship between the set of selected input parameters, $\boldsymbol{\mu}$, and the temporal evolution of low-dimensional flowfield, $\boldsymbol{z}(t)$, generated by the autoencoder; this is done for both the FCAE and CAE. Formally, the regressor $\mathcal{R}$ can be written as:

$$\mathcal{R} : (\boldsymbol{\mu}, t) \in \mathbb{R}^{n_{\boldsymbol{\mu}}} \times \mathbb{R} \mapsto \boldsymbol{z} \in \mathbb{R}^{n_{\text{latent}}}. \tag{6.3}$$

In this study, $\mathcal{R}$ is modeled as a fully-connected neural network, as shown schematically in Fig. 6.3. The regularized MSE error is used as the loss function, $J_{\mathcal{R}}$, to train the model, i.e.,

$$J_{\mathcal{R}}(\boldsymbol{\phi}) = ||\boldsymbol{z}_{\text{true}} - \boldsymbol{z}_{\text{pred}}||_{\text{MSE}} + \zeta_{\text{reg}} \sum_{k} ||\boldsymbol{W}_{\mathcal{R}}^{(k)}||_F^2, \tag{6.4}$$

where $\boldsymbol{z}_{\text{true}}$ is the true latent vector, $\boldsymbol{z}_{\text{pred}}$ is the latent vector predicted by $\mathcal{R}$, and $\boldsymbol{\phi} = \{\boldsymbol{W}_{\mathcal{R}}^{(1)}, \boldsymbol{b}_{\mathcal{R}}^{(1)}, \boldsymbol{W}_{\mathcal{R}}^{(2)}, \boldsymbol{b}_{\mathcal{R}}^{(2)}, \dots\}$ contains the parameters of the regression model. ζ_{reg} is the L_2-regularization hyperparameter.

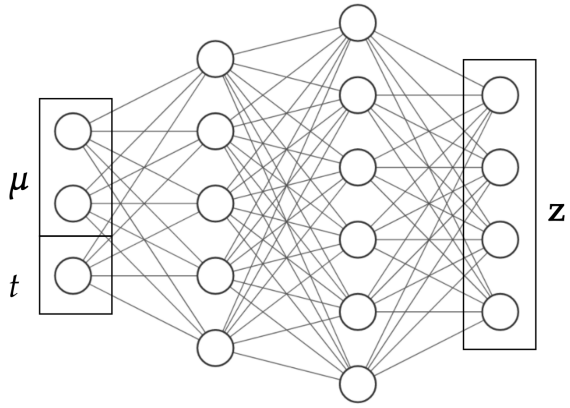


Figure 6.3: Schematic representation of a latent space regressor.

6.3.3 Training and Prediction Procedures

For each emulator framework and field variable of interest, an autoencoder and a regressor are trained following the methodology explained in Sec. 2.3; the models with the minimum validation errors are selected as the final models. To stabilize the learning process, data preprocessing operations are used (cf. Sec. 2.3.5.1). Specifically, the data is normalized in the range 0-1 for the autoencoders,[1] and is standardized for the regressors.

[1]Note if the data is already in the range 0-1, or close to it, for example, 0-0.5, there is no need to normalize the data.

After training, the flowfield for new design parameters $\boldsymbol{\mu}'$ can be predicted. The procedure proceeds as follow. The trained regressors are used first to predict the corresponding time sequence of latent vectors, i.e., $\boldsymbol{z}_{\text{pred}}(t;\boldsymbol{\mu}') = \mathcal{R}(\boldsymbol{\mu}', t)$. This sequence is then passed to the trained decoders to obtain the emulated flowfield in physical space. For the FCAE-$\mathcal{R}$ emulator, the decoding function $\mathcal{F}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}^{\text{FCAE}}(t;\boldsymbol{\mu}'))$ gives a one-dimensional vector $\boldsymbol{q}_{\text{pred}}(t;\boldsymbol{\mu}') \in \mathbb{R}^{n_{\text{in}}}$, whereas for the CAE-$\mathcal{R}$ emulator, the decoding function $\mathcal{G}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}^{\text{CAE}}(t;\boldsymbol{\mu}'))$ gives a tensor $\boldsymbol{Q}_{\text{pred}}(t;\boldsymbol{\mu}') \in \mathbb{R}^{n_x \times n_y \times n_c}$. Data scaling/unscaling operations are applied appropiately at each step. The pseudo-codes for the training and prediction procedures are given in Algorithms 5–7.

Algorithm 5: Autoencoder training

Load CFD/true flowfield data (e.g., temperature field)
Normalize the data (cf. Sec. 2.3.5.1), and save the normalization parameters (i.e., min and max values)
Shuffle and split the data into training, validation, and test sets using the hold-out method (cf. Sec. 2.3.4.1)
Set the number of epochs n_{epoch} and the batch size n_{batch}
Set the other hyperparameters (i.e., network architecture, n_{latent}, λ_{reg}, λ_{lr}, weight/bias initializer, activation function, early stopping criterion, etc.)
Set up the autoencoder network (i.e., create the network layers and model, and define the loss function)
for *each epoch index* i **do**
    Generate data batches of size n_{batch} from the training set
    Train autoencoder by feeding these data batches to minimize the loss function $J_{\text{minibatch}}$ using the Adam optimizer (cf. Algorithm 1)
    Compute the error on the validation set J_{valid}
    if J_{valid} *is improved* **then**
        Save autoencoder model
    end if
end for
Return autoencoder model with smallest J_{valid}

Algorithm 6: Regressor training

Load latent space data obtained from the autoencoder, and more specifically, from the encoder function (this data is considered here as the true data)
Standardize the data (cf. Sec. 2.3.5.1), and save the standardization parameters (i.e., mean and standard deviation values)
Shuffle and split the data into training, validation, and test sets using the hold-out method (cf. Sec. 2.3.4.1) (cases in the test set for regressor are same as those for autoencoder)
Set the number of epochs n_{epoch} and the batch size n_{batch}
Set the other hyperparameters (i.e., network architecture, λ_{reg}, λ_{lr}, weight/bias initializer, activation function, early stopping criterion, etc.)
Set up the regressor network (i.e., create the network layers and model, and define the loss function)
for *each epoch index* i **do**
    Generate data batches of size n_{batch} from the training set
    Train regressor by feeding these data batches to minimize the loss function $J_{\text{minibatch}}$ using the Adam optimizer (cf. Algorithm 1)
    Compute the error on the validation set J_{valid}
    if J_{valid} *is improved* **then**
        Save regressor model
    end if
end for
Return regressor model with smallest J_{valid}

Algorithm 7: FCAE-$\mathcal{R}$ (or CAE-$\mathcal{R}$) emulator prediction

Input: New design parameters $\boldsymbol{\mu}'$
Input: Trained regression model $\mathcal{R}$ with corresponding scaling parameters
Input: Trained decoder model $\mathcal{F}_{\text{dec}}$ (or $\mathcal{G}_{\text{dec}}$) with corresponding scaling parameters
Standardize design parameters: $\hat{\boldsymbol{\mu}}' \leftarrow s_{\text{S}}(\boldsymbol{\mu}')$
for *each snapshot/time index* i **do**
    Standardize time variable: $\hat{t}^{(i)} \leftarrow s_{\text{S}}(t^{(i)})$
    Predict standardized latent space: $\hat{\boldsymbol{z}}^{(i)} \leftarrow \mathcal{R}(\hat{\boldsymbol{\mu}}', \hat{t}^{(i)})$
    De-standardize latent space: $\boldsymbol{z}^{(i)} \leftarrow s_{\text{S}}^{-1}(\hat{\boldsymbol{z}}^{(i)})$
    Reconstruct normalized flowfield in physical space: $\hat{\boldsymbol{q}}^{(i)} \leftarrow \mathcal{F}_{\text{dec}}(\boldsymbol{z}^{(i)})$ (or $\hat{\boldsymbol{Q}}^{(i)} \leftarrow \mathcal{G}_{\text{dec}}(\boldsymbol{z}^{(i)})$)
    De-normalize flowfield in physical space: $\boldsymbol{q}^{(i)} \leftarrow s_{\text{N}}^{-1}(\hat{\boldsymbol{q}}^{(i)})$ (or $\boldsymbol{Q}^{(i)} \leftarrow s_{\text{N}}^{-1}(\hat{\boldsymbol{Q}}^{(i)})$)
end for
Return all the $\boldsymbol{q}^{(i)}$ (or all the $\boldsymbol{Q}^{(i)}$)

In the following, we introduce the two representative problems that are used to assess the proposed surrogate frameworks.

6.4 One-Dimensional Viscous Burgers Equation

6.4.1 Full-Order Model

The first problem is given by the one-dimensional viscous Burgers equation. This equation can generate discontinuous solutions representative of a moving shock profile even if initial conditions are smooth provided that the underlying Reynolds number is large enough. Hence, this equation is generally considered a benchmark problem for preliminary evaluation of numerical methods for analysis of fluid flows as it possesses characteristics of general nonlinear multi-dimensional advection-diffusion problems [17, 188]. The governing equation in a spatial domain $\Omega = [0, L]$, where $L = 1$, over the time interval $[0, t_{\text{end}}]$, $t_{\text{end}} = 2$, is

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + \frac{1}{Re}\frac{\partial^2 u}{\partial x^2}, \tag{6.5}$$

where $Re = \nu^{-1}$ is the Reynolds number, with ν the viscosity.

The parametric solution can be written as

$$u : \Omega \times [0, t_{\text{end}}] \times \mathbb{D} \mapsto \mathbb{R}, \quad (x, t; \boldsymbol{\mu}) \mapsto u, \tag{6.6}$$

where $\boldsymbol{\mu} \in \mathbb{D} \subset \mathbb{R}^{n_\mu}$ is the vector of input parameters, $\mathbb{D}$ the parameter domain, and $n_{\boldsymbol{\mu}}$ the number of input parameters. Here, $\boldsymbol{\mu} \equiv Re$ and $n_{\boldsymbol{\mu}} = 1$.

The domain boundary is divided into two points, with periodic boundary conditions at each point, i.e.,

$$u(0, t) = u(L, t) = 0. \tag{6.7}$$

At $t = 0$, the following condition is assumed

$$u(x, 0) = \frac{x}{1 + \sqrt{\frac{1}{t^*}}\exp(Re\frac{x^2}{4})}, \tag{6.8}$$

where $t^* = \exp(Re/8)$ is a characteristic time. An analytical solution exists and is given by

$$u(x, t) = \frac{\frac{x}{t+1}}{1 + \sqrt{\frac{t+1}{t_0}}\exp\left(Re\frac{x^2}{4t+4}\right)}. \tag{6.9}$$

The domain is discretized using $n_x = 128$ equidistant grid points and $n_t = 101$ time snapshots per case, and the analytical solution is computed at each of these points to generate the FOM data. The FOM computations take on average 0.004 s per DoE case.

6.4.2 Design of Experiments

As mentioned in Sec. 6.4.1, one input parameter is considered in this problem, namely, the Reynolds number Re. The parameter domain is taken as

$$\mathbb{D} = [50, 2250]. \tag{6.10}$$

When the Reynolds number is low, the system has a diffusion-dominated behavior, however, when the Reynolds number is large, it has a convection-dominated behavior.

Within the defined design space, 12 representative samples are identified. These samples are described in Table 6.1. Out of the 12 samples, 3 cases (Cases 3, 6 and 10) were randomly selected and set aside for testing to ensure that the emulator generalize well to unseen data. The remaining data (i.e., 9 cases) were used for training and validation of the neural networks, with a random split of 80%-20%. The datasets are shown in Table 6.2, where n_{set} indicates the total number of snapshots in each set.

Table 6.1: *1D Burgers equation.* Description of the 12 cases from the DoE study.

Case	Re
1	50
2	200
3	250
4	500
5	750
6	900
7	1000
8	1250
9	1500
10	1750
11	2000
12	2250

Table 6.2: *1D Burgers equation.* Dataset sizes used for formulating and testing the surrogates.

Set	Description	n_{set}
Training	80% of the snapshots from Cases 1,2,4,5,7-9,11-12	727
Validation	20% of the snapshots from Cases 1,2,4,5,7-9,11-12	182
Test	All of the snapshots from Cases 3,6,10	303

6.4.3 Compressed Representations

Autoencoder Results

Using Algorithm 5, an FCAE and a CAE are learned for the velocity field of the one-dimensional Burgers equation problem considered in Sec. 6.4.1. The data is in the range 0-0.5 and is kept unscaled (i.e., it is not normalized nor standardized). The network architectures and hyperparameters are tuned manually following the methodology explained in Sec. 2.3.5.3; for each autoencoder, we train multiple model instances and choose the one with the lowest validation error.

The selected FCAE from the manual search contains 10 dense layers, as shown in Table 6.3. This results in 39,140 trainable parameters. The autoencoder is trained by using a standard MSE loss with the following hyperparameters: a maximum number

of epochs of 1,000, a batch size of 10, a learning rate of 1×10^{-3} for the Adam optimizer, a L_2 regularization factor of 1×10^{-5}, and a latent vector of size 2 ($n_{\text{latent}} = 2$). ReLU activation function is applied to all the layers except the input layer, 5th and 10th dense layers. We note that although 1,000 epochs are chosen for training the FCAE, an early stopping criterion (cf. Sec. 2.3.3.2) is employed to prevent overfitting. MSE values of 1.021×10^{-4}, 1.092×10^{-4}, and 1.038×10^{-4} are achieved for the reconstructed fields on training, validation, and test data, respectively, as reported in Table 6.4.

Table 6.3: *1D Burgers equation.* Network structure of the FCAE.

Layer	Output shape	Activation
Input	128	–
1st Dense	100	ReLU
2nd Dense	50	ReLU
3rd Dense	25	ReLU
4th Dense	10	ReLU
5th Dense	2	–
6th Dense	10	ReLU
7th Dense	25	ReLU
8th Dense	50	ReLU
9th Dense	100	ReLU
10th Dense	128	–

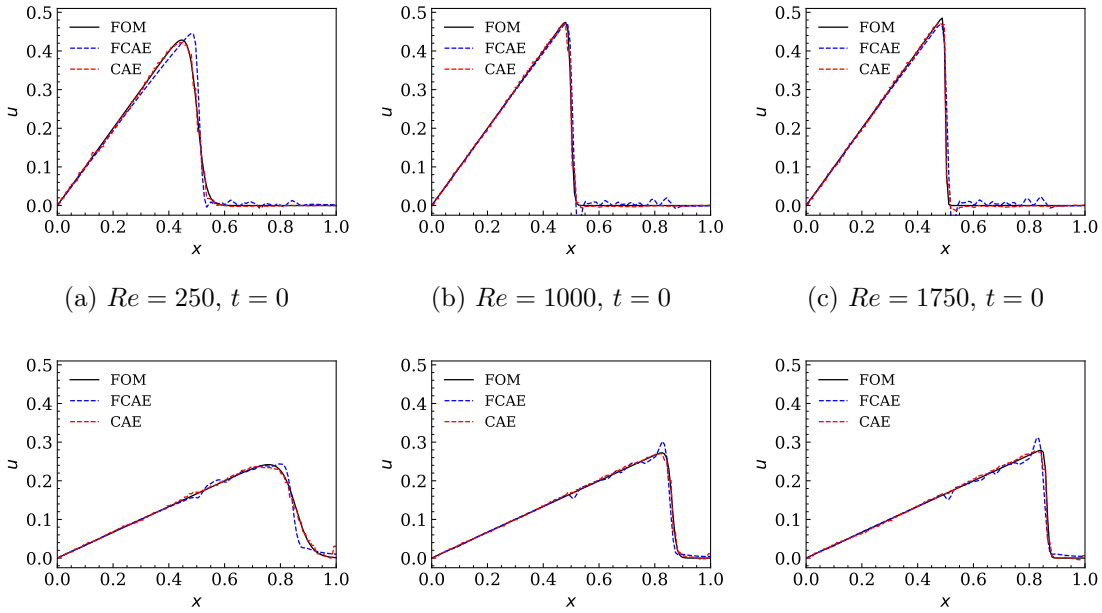
Table 6.4: *1D Burgers equation.* MSE values for the autoencoders. Results are shown over the snapshots from the training, validation, and test sets for the velocity field.

Data set	$\widetilde{u}^{\text{FCAE}}$	$\widetilde{u}^{\text{CAE}}$
Training set	1.021×10^{-4}	7.157×10^{-5}
Validation set	1.092×10^{-4}	7.945×10^{-5}
Test set	1.038×10^{-4}	6.945×10^{-5}

The architecture of the selected CAE is detailed in Table 6.5. As velocity is the only variable that is encoded in this problem, a one-dimensional CAE is used with only one channel ($n_{\text{c}} = 1$). The encoder contains six pairs of convolution and max pooling layers to reduce the dimensionality of the input field to a size of two degrees of freedom in the latent space ($n_{\text{latent}} = 2$). We note that the same level of compression is employed for the CAE and FCAE to provide a similar basis of comparison for accuracy. The decoder of the CAE contains six pairs of convolution and upsampling layers, and finally a single convolution layer to return to the dimensionality of the full-order field. Each convolution layer employs a 3×3 kernel filter, and utilizes a zero-padding at the edges of the domain to preserve the original input size. Each of the max pooling and upsampling layers uses a 2×2 window to achieve downsampling and upsampling, respectively. The stride length is set to 1 ($s = 1$). The CAE is trained by using a standard MSE loss with the following hyperparameters: a maximum number

of epochs of 1,000, a batch size of 10, a learning rate of 1×10^{-3} for the Adam optimizer, and the ReLU activation function. No L_2 regularization was used. Although the CAE contains more layers than the FCAE, it has only 5,014 trainable parameters, which helps in reducing model complexity and preventing overfitting. This is because all of the convolution, max pooling, and upsampling layers in the CAE perform local operations over nearby neurons instead of full operations over all the neurons. The MSE values for the CAE are reported in Table 6.4. Very low MSE values are observed on the validation and test datasets, which indicate that the CAE performs in the desired manner and doesn't overfit. Moreover, it is observed that the CAE achieves lower MSE values on the training, validation, and test sets compared to the FCAE; this is attributed to the capability of CAE to retain the spatial coherence of the flowfield data.

Figure 6.4 shows the reconstructed velocity field by FCAE and CAE for $Re = 250$, 1000 and 1750 at $t = 0$ and 2 s. It is observed that the reconstructed flowfields by both models are in good agreement with the original data for all the cases, although some oscillations are noticeable, particularly near and after the discontinuity zone in FCAE. It is also evident from this plot that the CAE reduces the oscillations considerably, and thus performs better than FCAE.



(a) $Re = 250$, $t = 0$ (b) $Re = 1000$, $t = 0$ (c) $Re = 1750$, $t = 0$

(d) $Re = 250$, $t = 2$ (e) $Re = 1000$, $t = 2$ (f) $Re = 1750$, $t = 2$

Figure 6.4: *1D Burgers equation.* Reconstruction of velocity field by FCAE and CAE. Results are shown at $t = 0$ and 2 for $Re = 250$, 1000 and 1750.

Comparison with POD

To further illustrate the performance of the autoencoders, in particular, their ability to perform nonlinear dimensionality reduction, we report hereafter results using the

Table 6.5: *1D Burgers equation.* Network structure of the CAE.

Layer	Output shape	Activation
Input	(128,1)	–
1st Conv1D	(128,25)	ReLU
1st MaxPooling1D	(64,25)	–
2nd Conv1D	(64,15)	ReLU
2nd MaxPooling1D	(32,15)	–
3rd Conv1D	(32,10)	ReLU
3rd MaxPooling1D	(16,10)	–
4th Conv1D	(16,5)	ReLU
4th MaxPooling1D	(8,5)	–
5th Conv1D	(8,2)	ReLU
5th MaxPooling1D	(4,2)	–
6th Conv1D	(4,1)	ReLU
6th MaxPooling1D	(2,1)	–
7th Conv1D	(2,1)	ReLU
1th UpSampling1D	(4,1)	–
8th Conv1D	(4,5)	ReLU
2nd UpSampling1D	(8,5)	–
9th Conv1D	(8,10)	ReLU
3rd UpSampling1D	(16,10)	–
10th Conv1D	(16,15)	ReLU
4th UpSampling1D	(32,15)	–
11th Conv1D	(32,20)	ReLU
5th UpSampling1D	(64,20)	–
12th Conv1D	(64,25)	ReLU
6th UpSampling1D	(128,25)	–
13th Conv1D	(128,1)	–

linear POD technique and compare them with those of the autoencoders.

POD is a classical linear model reduction technique that extracts dominant energetic structures from the FOM. The method extracts a set of basis functions from the data, and uses a subset of leading basis functions to construct approximations of the field variables of interest. In the following, we apply the EVD-based POD to each FOM case separately. More details about this approach can be found in Appendix A.2.

Figure 6.5 shows the percentage of the captured energy from POD, calculated using Eq. (A.13), for Cases 3, 6 and 10 and reveals the amount of information omitted by the POD representation for any particular number of modes. From this figure, it is observed that the growth rates of the energy distributions are different among the cases. To quantify these differences, the numbers of POD modes required to recover 90, 95 and 99% of the modal energy are identified; these variables are denoted by $n_{r,90\%}$, $n_{r,95\%}$ and $n_{r,99\%}$, respectively. Table 6.6 shows that the values of these variables are different among the cases. It is also observed that cases with higher Reynolds numbers require a higher number of modes for accurate reconstruction. For example, Case 10, which corresponds to $Re = 1750$, requires 16 modes to recover 99% of the

energy, while Case 3, which corresponds to $Re = 250$, requires only 5 modes to recover the same amount of energy. This is because a discontinuity exists at higher Reynolds numbers, making the flow nonlinear.

The ability of POD to reproduce the original FOM solution is visualized in Fig. 6.6, where the POD-based reconstructions of velocity field for different number of modes are shown at $t = 0$ and 2 for $Re = 250$, 1000 and 1750. In each case, results using between 2-101 modes are reported. From this figure, we see that the POD method exhibits strong oscillations, particularly when the number of retained modes is small, such as 2 and 5 modes (and even 10 modes, for $Re = 1000$ and 1750), and that the accuracy of the reconstructions is improved as the number of retained modes is increased. It is also evident from this plot that the magnitude of the oscillations for POD with 2 modes are much larger than that of the FCAE and CAE with a latent space of size 2. (cf. Fig. 6.4).

Finally, a quantitative comparison of the reconstruction accuracy between POD, FCAE and CAE is reported in Table 6.7. Overall, this comparison highlights the limitation of the POD technique when applied to nonlinear problems and demonstrates the superior performance of the AE-based methods to efficiently compress the flowfields at the same level of data compression.

Figure 6.5: *1D Burgers equation.* Variation of the captured modal energy, in percentage, as a function of the number of retained POD modes. Results are shown for the test cases.

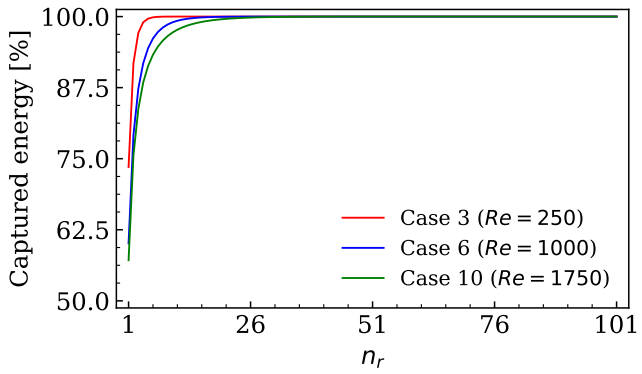
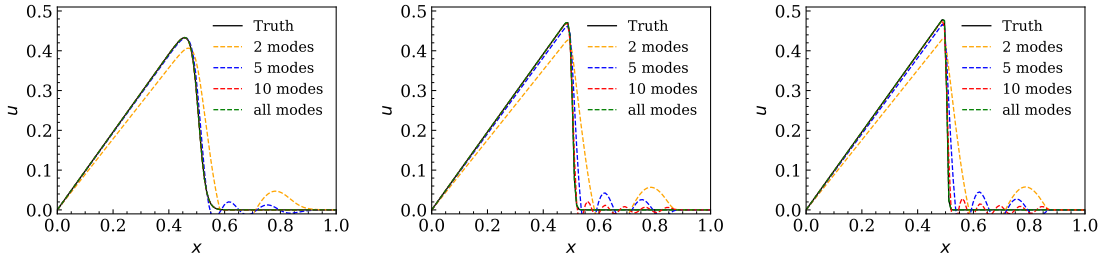
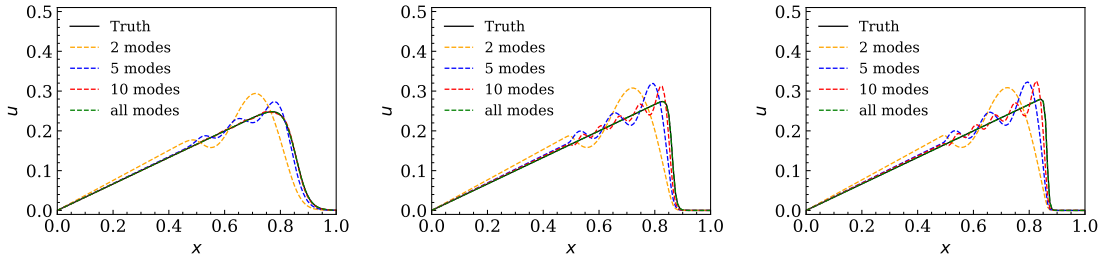


Table 6.6: *1D Burgers equation.* Number of POD modes required to capture 90%, 95% and 99% of the modal energy.

Case	$n_{r,90\%}$	$n_{r,95\%}$	$n_{r,99\%}$
$Re = 250$	2	3	5
$Re = 1000$	4	6	11
$Re = 1750$	5	8	16



(a) $Re = 250$, $t = 0$ (b) $Re = 1000$, $t = 0$ (c) $Re = 1750$, $t = 0$



(d) $Re = 250$, $t = 2$ (e) $Re = 1000$, $t = 2$ (f) $Re = 1750$, $t = 2$

Figure 6.6: *1D Burgers equation.* POD-reconstructed flowfield. Results are shown using $2, 5, 10$ and 101 POD modes at $t = 0$ and 2 for $Re = 250, 1000$ and 1750.

Table 6.7: *1D Burgers equation.* Comparison of MSE values between POD, FCAE and CAE for the reconstructed velocity field. Results are shown over the snapshots of each case from the test set.

	$Re = 250$	$Re = 1000$	$Re = 1750$
$\widetilde{u}^{\text{POD}}$ (2 modes)	3.511×10^{-4}	1.264×10^{-3}	1.579×10^{-3}
$\widetilde{u}^{\text{POD}}$ ($n_{r,90\%}$ modes)	3.511×10^{-4}	4.985×10^{-4}	5.632×10^{-4}
$\widetilde{u}^{\text{POD}}$ ($n_{r,95\%}$ modes)	1.229×10^{-4}	2.340×10^{-4}	2.743×10^{-4}
$\widetilde{u}^{\text{POD}}$ ($n_{r,99\%}$ modes)	1.565×10^{-5}	4.459×10^{-5}	6.066×10^{-5}
$\widetilde{u}^{\text{FCAE}}$ ($n_{\text{latent}} = 2$)	1.119×10^{-4}	4.341×10^{-5}	1.560×10^{-4}
$\widetilde{u}^{\text{CAE}}$ ($n_{\text{latent}} = 2$)	3.145×10^{-5}	6.412×10^{-5}	1.128×10^{-4}

6.4.4 Emulated Flowfields

The latent spaces obtained by the FCAE and CAE on the training and validation data are used to train the regressors. For each autoencoder, a regressor with five dense layers is designed to learn the mapping from the input parameter Re and time instant t to the corresponding sequence of latent variables $\boldsymbol{z}(t; Re)$, following Algorithm 6. The input and output data are standardized to stabilize the learning process. The detailed architecture of the selected regressors from the manual search is given in Table 6.8. The number of epochs is set to 1,500, the batch size to 25, the learning rate for the Adam optimizer to 1×10^{-3}, and the L_2 regularization factor

to 8×10^{-4}. This results in 1,048 trainable parameters. A comparison of the MSE losses on training, validation, and test sets is reported in Table 6.9 for each regressor. Evaluation of the MSE losses indicates satisfactory training performance without evidence of overfitting.

Table 6.8: *1D Burgers equation.* Network structure of each regressor.

Layer	Output shape	Activation
Input	2	–
1st Dense	9	ReLU
2nd Dense	32	ReLU
3rd Dense	16	ReLU
4th Dense	9	ReLU
5th Dense	2	–

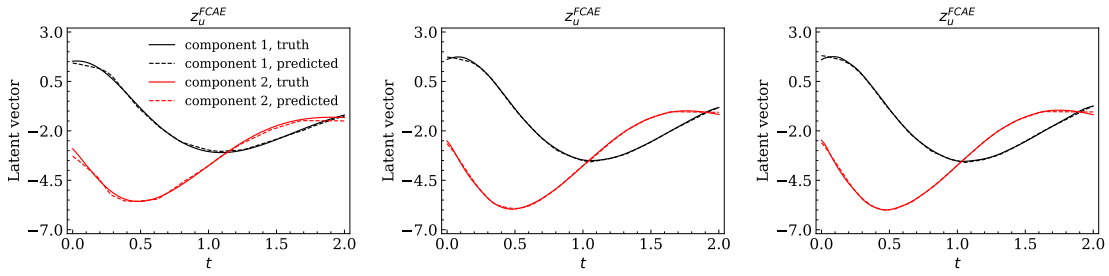
Table 6.9: *1D Burgers equation.* MSE values for each regressor. Results are shown over the latent vectors of velocity from the training, validation, and test sets.

Data set	$\boldsymbol{z}_u^{\text{FCAE}}$	$\boldsymbol{z}_u^{\text{CAE}}$
Training set	1.120×10^{-3}	2.328×10^{-3}
Validation set	1.684×10^{-3}	2.574×10^{-3}
Test set	1.454×10^{-3}	2.702×10^{-3}

For a new input parameter Re', the trained regressors are used to predict the corresponding time sequence of latent vectors, i.e., $\boldsymbol{z}_{\text{pred}}(t; Re') = \mathcal{R}(Re', t)$. This sequence is then passed to the trained decoders to obtain the emulated flowfield in physical space (cf. Algorithm 7). For the FCAE-$\mathcal{R}$ emulator, the decoding function $\mathcal{F}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}^{\text{FCAE}}(t; Re'))$ gives a one-dimensional vector $\boldsymbol{q}_{\text{pred}}(t; Re') \in \mathbb{R}^{n_{\text{in}}}$. Similarly, for the CAE-$\mathcal{R}$ emulator, the decoding function $\mathcal{G}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}^{\text{CAE}}(t; Re'))$ also gives a one-dimensional vector since the problem under consideration is one-dimensional.

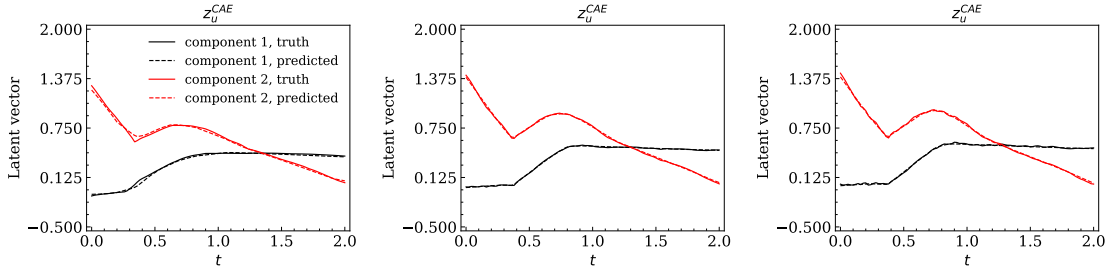
The temporal evolution of the components of the latent vectors from the FCAE and CAE is shown in solid lines in Figs. 6.7 and 6.8, respectively, for $Re = 250$, 1000 and 1750, which belong to the test set and were not seen during the training of the neural networks. The latent space representations obtained from the FCAE and CAE exhibit different behavior. The time-series predictions for the trained regressors are shown in dashed lines in these figures. The regressors are observed to predict the order and evolution of the latent space components accurately, indicating that the sequential behavior has been learned.

Figure 6.9 shows snapshots of the emulated velocity field at various time instances for $Re = 250$, 1000 and 1750 from the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$ frameworks. Qualitatively, the emulated flowfields from both frameworks show very good agreement with the FOM, although minor differences are observed, particularly around the discontinuity.



(a) $Re = 250$ (b) $Re = 1000$ (c) $Re = 1750$

Figure 6.7: *1D Burgers equation.* Temporal evolution of the components of the latent variables from FCAE for $Re = 250$, 1000 and 1750. Latent variables computed by the autoencoder (solid lines) and those predicted by the regressor (dashed lines) are compared.



(a) $Re = 250$ (b) $Re = 1000$ (c) $Re = 1750$

Figure 6.8: *1D Burgers equation.* Temporal evolution of the components of the latent variables from CAE for $Re = 250$, 1000 and 1750. Latent variables computed by the autoencoder (solid lines) and those predicted by the regressor (dashed lines) are compared.

For the sake of completeness, the MSE values over the snapshots of each case from the test set are shown in Table 6.10. Overall, the emulation errors for both surrogate models are very small (below 1.7×10^{-4}), with the CAE-$\mathcal{R}$ providing slight improvement in the global MSEs. This demonstrates the accuracy of both frameworks in capturing the salient features of the spatiotemporal flowfields.

Table 6.10: *1D Burgers equation.* MSE values for the emulated velocity field for test cases.

Test case	$u_{\text{pred}}^{\text{FCAE-}\mathcal{R}}$	$u_{\text{pred}}^{\text{CAE-}\mathcal{R}}$
$Re = 250$	1.222×10^{-4}	7.057×10^{-5}
$Re = 1000$	4.877×10^{-5}	7.165×10^{-5}
$Re = 1750$	1.641×10^{-4}	1.267×10^{-4}

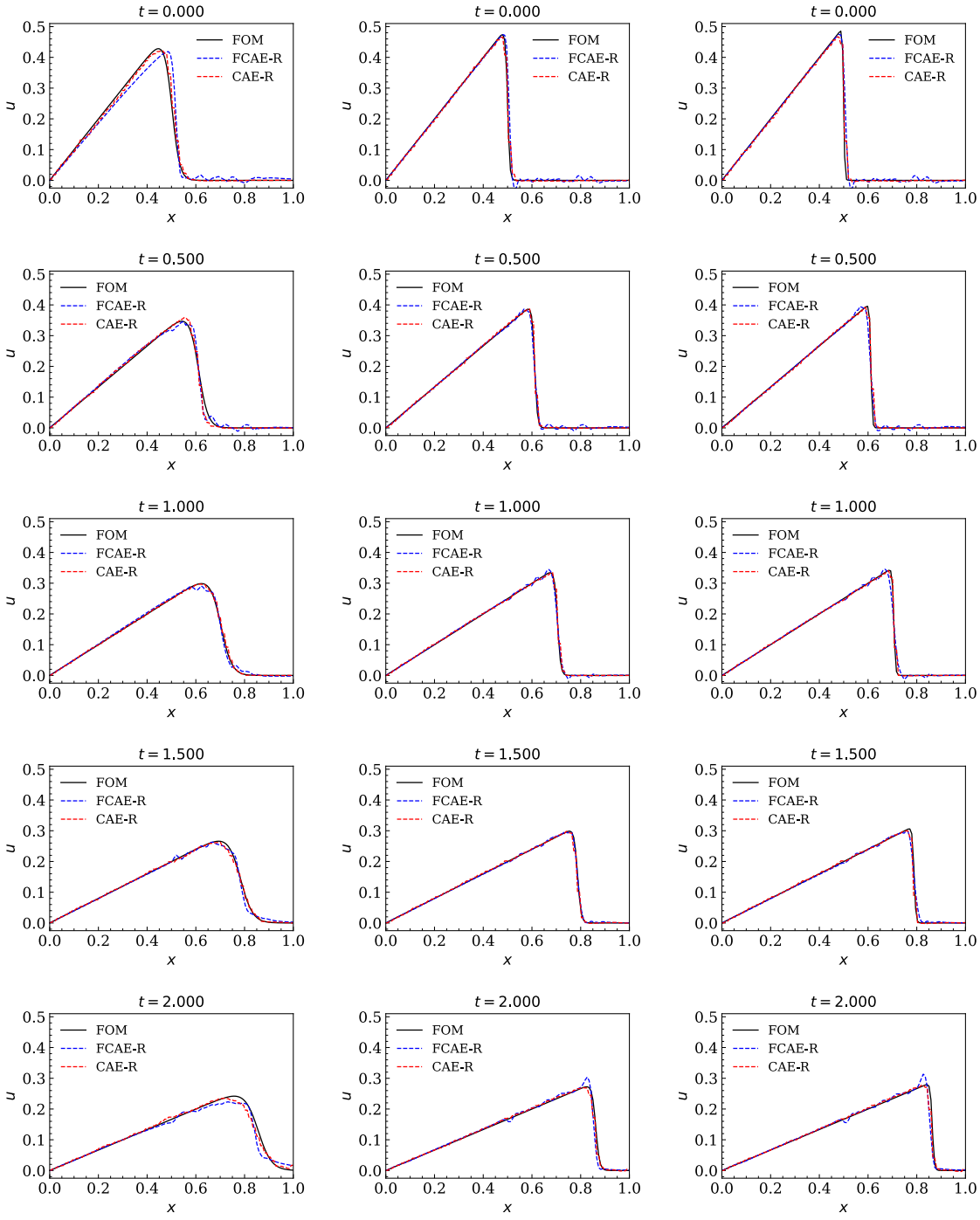


Figure 6.9: *1D Burgers equation.* Instantaneous snapshots of velocity field for $Re = 250$ (first column), $Re = 1000$ (second column), and $Re = 1750$ (third column). Flowfields computed by the FOM and those predicted by the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$ emulators are compared.

6.4.5 Computational Cost

Table 6.11 summarizes the computational time required for each step of the emulation process for both frameworks, with timing performed on a Linux desktop that has the following specifications: Intel(R) Core(TM) i7-10700T CPU @ 2.00 GHz. The preprocessing step, which consists of extracting the snapshots from the FOM in preparation for the training phase, has negligible time for the one-dimensional Burgers equation problem and thus is not considered here. The training step, which consists of training one autoencoder and one regressor, requires in total about 78 and 283 s for the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively. After the training is completed, the inference time is approximately 0.85 and 0.87 s for the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively, to predict the velocity field for a new design point.

In comparison, the FOM computations take on average 0.004 s per DoE case. We note that, exclusively in this problem, the computation time of the FOM is negligible just because an analytical solution exists. This will not be the case, however, for problems involving more complex physics and/or geometries and for which an analytical solution is not available, such as the ADR equation (cf. Sec. 6.5) and A-M1 injector flow (cf. Chapter 7). In such problems, numerical discretization and time integration are required, and computations can take minutes, days, or even weeks, to complete, depending on the involved complexity and available computing resources. It is in these scenarios that emulators are most useful.

Although this problem is relatively simple, it was used here as a use-case for evaluating the accuracy and efficacy of the emulator in representing nonlinear flowfields, such as the advection of a shock profile, before moving to more computationally expensive problems in the next section.

Table 6.11: *1D Burgers equation.* Computational time for each step of the emulation process.

Step	Time (FCAE-$\mathcal{R}$)	Time (CAE-$\mathcal{R}$)
Preprocessing	negligible	negligible
Autoencoder training	78 CPU-secs	236 CPU-secs
Regressor training	46 CPU-secs	47 CPU-secs
Flow prediction	0.85 CPU-secs	0.87 CPU-secs

6.5 Two-Dimensional Advection-Diffusion-Reaction Equation

6.5.1 Full-Order Model

The second problem is given by an advective-diffusive-reactive system modeled on the premixed combustion of a hydrogen-air mixture at constant and uniform pressure,

in a constant, divergence-free velocity field, and with constant, equal and uniform molecular diffusivities for all temperature and species [189]. This problem has been used considerably in literature as a use-case to support development and verification of various surrogate and reduced-order modeling approaches [189, 190, 191]. The one-step reaction mechanism is: $2H_2 + O_2 \rightarrow 2H_2O$. The nonlinear governing equation system in a spatial domain $\Omega = [0, L_x] \times [0, L_y]$, where $L_x = 18$ mm and $L_y = 9$ mm, over the time interval $[0, t_{\text{end}}]$, $t_{\text{end}} = 0.06$ s, is

$$\underbrace{\frac{\partial \boldsymbol{w}}{\partial t}}_{\text{unsteady}} = -\underbrace{\boldsymbol{v} \cdot \nabla_{\boldsymbol{x}} \boldsymbol{w}}_{\text{advection}} + \underbrace{\kappa \Delta_{\boldsymbol{x}} \boldsymbol{w}}_{\text{diffusion}} + \underbrace{\boldsymbol{q}}_{\text{reaction}}, \tag{6.11}$$

where t denotes the time coordinate, $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T$ the spatial coordinates, $\nabla_{\boldsymbol{x}}$ the gradient operator with respect to $\boldsymbol{x}$, $\Delta_{\boldsymbol{x}}$ the Laplacian operator with respect to $\boldsymbol{x}$, $\boldsymbol{v}$ the velocity vector, $\boldsymbol{q}$ the reaction source term, and κ the diffusivities. The parametric solution

$$\boldsymbol{w} : \Omega \times [0, t_{\text{end}}] \times \mathbb{D} \mapsto \mathbb{R}^4, \quad (\boldsymbol{x}, t; \boldsymbol{\mu}) \mapsto \begin{bmatrix} T \\ Y_F \\ Y_O \\ Y_P \end{bmatrix} \tag{6.12}$$

represents the thermo-chemical composition vector consisting of the temperature $T(\boldsymbol{x}, t; \boldsymbol{\mu})$ and the mass fractions of the hydrogen fuel, oxygen, and water product, i.e., $Y_F(\boldsymbol{x}, t; \boldsymbol{\mu})$, $Y_O(\boldsymbol{x}, t; \boldsymbol{\mu})$, and $Y_P(\boldsymbol{x}, t; \boldsymbol{\mu})$, respectively, where $\boldsymbol{\mu} \in \mathbb{D} \subset \mathbb{R}^{n_{\boldsymbol{\mu}}}$ is the vector of input parameters, $\mathbb{D}$ the parameter domain, and $n_{\boldsymbol{\mu}}$ the number of input parameters.

The reaction source term $\boldsymbol{q}(\boldsymbol{w}(\boldsymbol{x}, t; \boldsymbol{\mu}); \boldsymbol{\mu}) \in \mathbb{R}^4$ is of Arrhenius type and modeled as in Cuenot and Poinsot [192] as

$$\begin{aligned} q_T &= Q q_P \\ q_i &= -\nu_i \left(\frac{W_i}{\rho}\right) \left(\frac{\rho Y_F}{W_F}\right)^{\nu_F} \left(\frac{\rho Y_O}{W_O}\right)^{\nu_O} A \exp\left(-\frac{E_{\text{a}}}{RT}\right), \end{aligned} \tag{6.13}$$

for $i = F, O, P$. Here, $(\nu_F, \nu_O, \nu_P) = (2, 1, -2)$ denotes the stoichiometric coefficients, $(W_F, W_O, W_P) = (2.016, 31.9, 18)$ the molecular weights with units g·mol^{-1}, $\rho = 1.39 \times 10^{-3}$ g·cm^{-3} the density mixture, $R = 8.314$ J·mol^{-1}·K^{-1} the universal gas constant, and $Q = 9800$ K the heat of the reaction. The design parameters correspond to $\boldsymbol{\mu} = (A, E_{\text{a}})$, where A is the pre-exponential factor, and E_{a} the activation energy. These two parameters affect the reaction rate and the flame temperature of the system [193]. The velocity field is set to $\boldsymbol{v} = (50, 0)^T$ with units cm·s^{-1}, and the molecular diffusivities to $\kappa = 2$ cm^2·s^{-1}. Although quantitative information regarding the flame cannot be obtained using this simplified model problem, it contains the essential physics (advection, diffusion, and reaction phenomena) associated with complicated reacting flow problems, and thus serves as a benchmark problem for surrogate and reduced-order model development [189].

The domain boundary is divided into six segments, as shown schematically in Fig. 6.10. On the inflow boundary Γ_2, we impose Dirichlet boundary conditions

$T = 950$ K for temperature and $(Y_F, Y_O, Y_P) = (0.0282, 0.2259, 0)$ for the chemical composition. On the boundaries Γ_1 and Γ_3, we impose Dirichlet boundary conditions $T = 300$ K for temperature and $(Y_F, Y_O, Y_P) = (0, 0, 0)$ for the chemical composition. On the boundaries Γ_4, Γ_5 and Γ_6, we impose homogeneous Neumann conditions on the temperature and mass fractions, i.e., $\nabla T = 0$ and $\nabla Y_i = 0$ for $i = F, O, P$. At $t = 0$, the domain is considered empty and the temperature is set to 300 K, i.e., $\boldsymbol{w}_0 = (300, 0, 0, 0)^T$.

Equation (6.11) is solved using the finite-difference method [194] on a $n_x \times n_y$ uniform grid, with $n_x = 64$ and $n_y = 32$. Spatial and temporal discretization are achieved with second-order and first-order accuracy, respectively. The time step is set to $\Delta t = 5 \times 10^{-5}$ s, which implies 1,200 iterations to solve the problem. A total of $n_t = 121$ snapshots is acquired between $t = 0$–0.06 s. The corresponding computational cost is 52 CPU-secs per simulation.

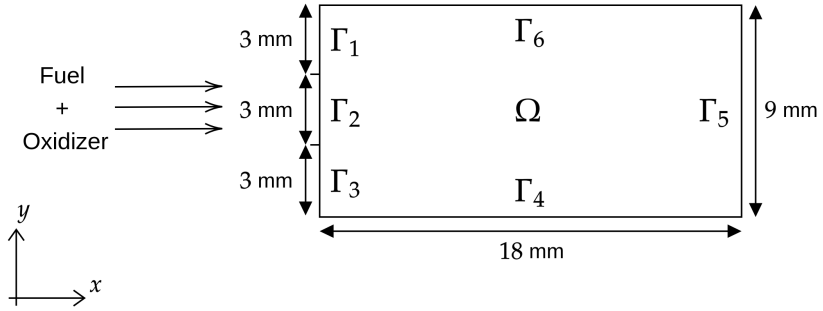


Figure 6.10: *ADR equation.* Schematic setup.

6.5.2 Design of Experiments

As mentioned in Sec. 6.5.1, two input parameters are considered in this work, namely, the pre-exponential factor A and the activation energy E_a, thus $n_{\boldsymbol{\mu}} = 2$. The parameter domain is taken as

$$\mathbb{D} = [2.3375 \times 10^{12}, 6.2 \times 10^{12}] \times [5.625 \times 10^3, 9 \times 10^3]. \tag{6.14}$$

Within the defined design space, 25 representative samples are identified using a uniform sampling of $\mathbb{D}$ on a 5×5 grid, as visualized in Fig. 6.11. These samples (i.e., Cases 1-25) are used for training and validation of the surrogates, with a random split of 80%-20%. Three cases (i.e., Cases 26–28) are randomly generated and set aside for testing to ensure that the surrogates generalize well to unseen data. These cases correspond to $\boldsymbol{\mu}^{26} = (2.5 \times 10^{12}, 6.75 \times 10^3)$, $\boldsymbol{\mu}^{27} = (6.0 \times 10^{12}, 8.0 \times 10^3)$ and $\boldsymbol{\mu}^{28} = (4.1 \times 10^{12}, 7.51 \times 10^3)$, respectively. Hence, 28 cases are considered in total (i.e., $n_c = 28$). The datasets are shown in Table 6.12, where n_{set} indicates the total number of snapshots in each set. Figure 6.12 reports the FOM solutions for Cases 1, 15 and 25 at $t = 0.06$ s.

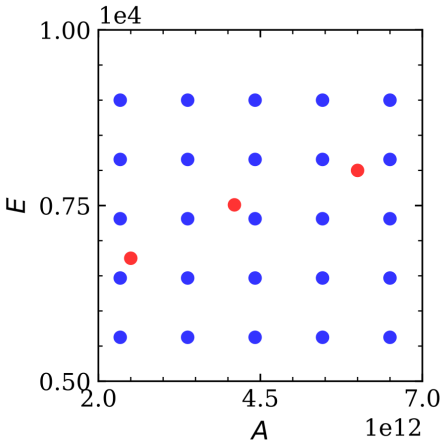
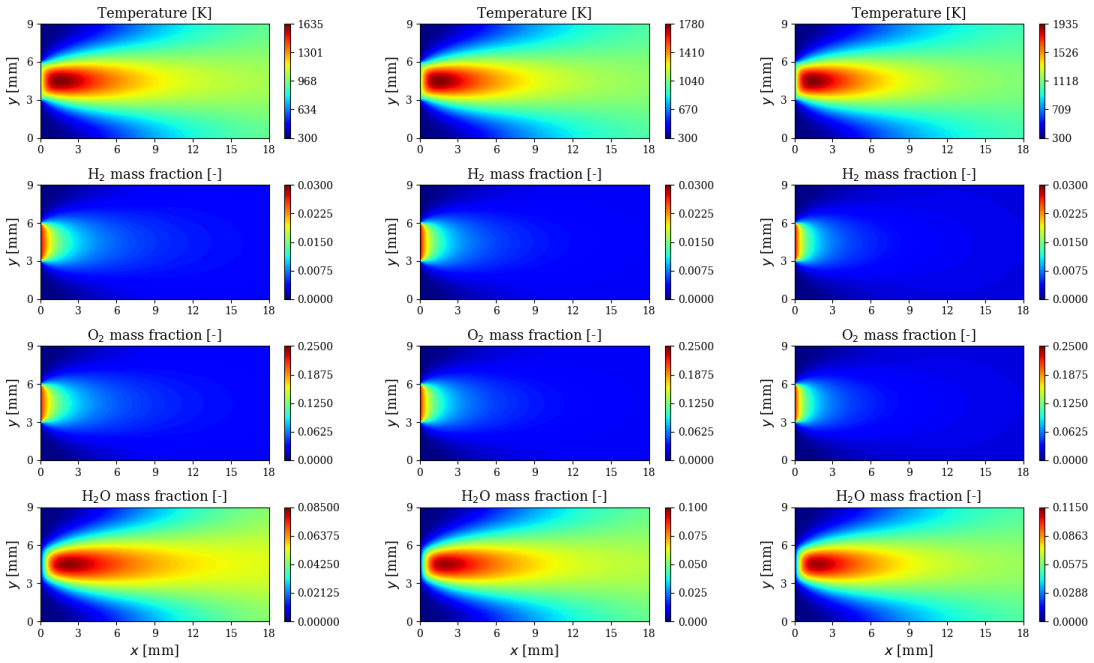


Figure 6.11: *ADR equation.* Visualization of the sample points in the parameter domain. Training and validation cases are represented in blue circle symbols, whereas testing cases are shown using red circle symbols.

Table 6.12: *ADR equation.* Dataset sizes used for formulating and testing the surrogates.

Set	Description	n_{set}
Training	80% of the snapshots from Cases 1-25	2,420
Validation	20% of the snapshots from Cases 1-25	605
Test	All of the snapshots from Cases 26–28	363



(a) Case 1 (b) Case 15 (c) Case 25

Figure 6.12: *ADR equation.* Instantaneous snapshots of field variables for Case 1 ($\boldsymbol{\mu}^1 = (2.3375 \times 10^{12}, 5.625 \times 10^3)$), Case 15 ($\boldsymbol{\mu}^{15} = (4.41875 \times 10^{12}, 9.0 \times 10^3)$), and Case 25 ($\boldsymbol{\mu}^{25} = (6.5 \times 10^{12}, 9.0 \times 10^3)$) at $t = 0.06$ s from FOM.

Autoencoder Results

Using Algorithm 5, an FCAE and a CAE are learned for each field variable of interest, namely the temperature, T, and mass fractions of fuel and product, Y_F and Y_P. The mass fraction of the oxidizer, Y_O, can be obtained by mass conservation (i.e., $\sum_i Y_i = 1$), and thus is not processed by the ROM and emulation frameworks. The data is normalized in the range 0-1. The network architectures and hyperparameters are tuned manually.

For each field variable, the selected FCAE contains 12 dense layers, as shown in Table 6.13. This results in 1,854,702 trainable parameters. The autoencoders are trained by using a standard MSE loss with the following hyperparameters: a maximum number of epochs of 1,000, a batch size of 24, a learning rate of 3×10^{-4} for the Adam optimizer, a L_2 regularization factor of 5×10^{-6}, and a latent vector of size 4 ($n_{\text{latent}} = 4$). ReLU activation function is applied to all the layers except the input layer, 6th and 12th dense layers. We note that, although 1,000 epochs are chosen for training the FCAEs, an early stopping criterion is employed to prevent overfitting. Table 6.14 reports the MSE values for each autoencoder. Low MSE values are observed on the validation and test datasets for each field variable. For example, MSE values of 1.340×10^{-5} and 7.867×10^{-6} are achieved for the reconstructed temperature field on validation and test data, respectively. This indicates that the autoencoders perform in the desired manner and do not overfit.

Table 6.13: *ADR equation.* Network structure of the FCAEs.

Layer	Output shape	Activation
Input	2048	–
1st Dense	400	ReLU
2nd Dense	200	ReLU
3rd Dense	100	ReLU
4th Dense	50	ReLU
5th Dense	25	ReLU
6th Dense	4	–
7th Dense	25	ReLU
8th Dense	50	ReLU
9th Dense	100	ReLU
10th Dense	200	ReLU
11th Dense	400	ReLU
12th Dense	2048	–

The architecture of the selected CAEs is detailed in Table 6.15. Similar to the FCAE, a separate CAE is designed for each field variable of interest, and hence, each CAE is built with only one channel ($n_c = 1$). The encoder contains four pairs of

Table 6.14: *ADR equation.* MSE values for the autoencoders. Results are shown over the snapshots from the training, validation, and test sets for the temperature and species mass fractions fields.

FCAE			
	Training set	Validation set	Test set
$\widetilde{T}$	1.268×10^{-5}	1.340×10^{-5}	7.867×10^{-6}
$\widetilde{Y}_F$	2.263×10^{-5}	2.407×10^{-5}	2.454×10^{-5}
$\widetilde{Y}_P$	1.998×10^{-5}	2.154×10^{-5}	8.563×10^{-6}
CAE			
	Training set	Validation set	Test set
$\widetilde{T}$	4.472×10^{-6}	4.675×10^{-6}	3.077×10^{-6}
$\widetilde{Y}_F$	6.279×10^{-6}	6.139×10^{-6}	1.721×10^{-5}
$\widetilde{Y}_P$	6.419×10^{-6}	6.765×10^{-6}	3.676×10^{-6}

convolution and max pooling layers followed by three dense layers to reduce the dimensionality of the input field to a size of four degrees of freedom in the latent space ($n_{\text{latent}} = 4$). It is noted that the same level of compression is employed for the CAE and FCAE to provide a similar basis of comparison for accuracy. The decoder of the CAE contains three dense layers, followed by four pairs of convolution and upsampling layers, and finally a single convolution layer to return to the dimensionality of the full-order field. Each convolution layer employs a 3×3 kernel filter, and utilizes a zero-padding at the edges of the domain to preserve the original input size. Each of the max pooling and upsampling layers uses a 2×2 window to achieve downsampling and upsampling, respectively. The stride length is set to 1 ($s = 1$). The CAEs are trained by using a standard MSE loss with the following hyperparameters: a maximum number of epochs of 1,000, a batch size of 24, a learning rate of 3×10^{-4} for the Adam optimizer, and the ReLU activation function. No L_2 regularization was used. Although the CAE contains more layers than the FCAE, it has only 28,855 trainable parameters, which helps in reducing model complexity and preventing overfitting. This is because all of the convolution, max pooling, and upsampling layers in the CAE perform local operations over nearby neurons instead of full operations over all the neurons. The MSE values for the CAEs are reported in Table 6.14. Very low MSE values are observed on the validation and test datasets for each field variable, which indicate that the CAEs perform in the desired manner and do not overfit. Moreover, it is observed that the CAEs achieve lower MSE values on the training, validation, and test sets for all of the field variables compared to the FCAEs.

Comparison with POD

To further illustrate the performance of the autoencoders, the reconstructed fields are compared with those from EVD-based POD (cf. Appendix A.2). The performance of POD, FCAE, and CAE is evaluated by using equal levels of data compression. The reconstructed temperature and fuel mass fraction fields for POD using four modes are

Table 6.15: *ADR equation.* Network structure of the CAEs.

Layer	Output shape	Activation
Input	(64,32,1)	–
1st Conv2D	(64,32,30)	ReLU
1st MaxPooling2D	(32,16,30)	–
2nd Conv2D	(32,16,20)	ReLU
2nd MaxPooling2D	(16,8,20)	–
3rd Conv2D	(16,8,15)	ReLU
3rd MaxPooling2D	(8,4,15)	–
4th Conv2D	(8,4,10)	ReLU
4th MaxPooling2D	(4,2,10)	–
Flatten	80	–
1st Dense	40	ReLU
2nd Dense	20	ReLU
3rd Dense	4	–
4th Dense	20	ReLU
5th Dense	40	ReLU
6th Dense	80	ReLU
Reshape	(4,2,10)	–
5th Conv2D	(4,2,10)	ReLU
1st UpSampling2D	(8,4,10)	–
6th Conv2D	(8,4,15)	ReLU
2nd UpSampling2D	(16,8,15)	–
7th Conv2D	(16,8,20)	ReLU
3nd UpSampling2D	(32,16,20)	–
8th Conv2D	(32,16,30)	ReLU
4th UpSampling2D	(64,32,30)	–
9th Conv2D	(64,32,1)	–

compared against those from FCAE and CAE with latent space dimensions of four, as shown in Figs. 6.13 and 6.14 for Case 27 at $t = 0.02$ s. The L_2 error distribution is also reported for each ROM technique to visualize the local errors between the reconstructed and FOM-computed fields. Overall, the flowfield reconstructed by the CAE yielded the lowest L_2 errors. POD exhibits discernible oscillations throughout the domain, whereas both the CAE and FCAE show excellent agreement with the FOM. These observations also hold for the other test cases, as indicated quantitatively in Table 6.16. This comparison highlights the merit of using deep autoencoders over traditional techniques, such as POD, for nonlinear dimensionality reduction. It also demonstrates that using an autoencoder with a convolutional neural network architecture instead of a fully-connected one can help further improve the accuracy of the reconstructions.

Finally, the temporal evolution of the temperature field computed by the FOM and that reconstructed by the CAE for Case 27 is shown in Fig. 6.15. It is clearly observed that the reconstructed flowfield is in good agreement with the original data at all the snapshots.

Table 6.16: *ADR equation.* Comparison of MSE values between POD, FCAE and CAE for the reconstructed temperature and species mass fractions fields. Results are shown over the snapshots of each case from the test set.

POD			
	Case 26	Case 27	Case 28
$\widetilde{T}$	4.476×10^{-4}	4.827×10^{-4}	4.658×10^{-4}
$\widetilde{Y}_F$	1.224×10^{-3}	1.604×10^{-3}	1.426×10^{-3}
$\widetilde{Y}_P$	4.755×10^{-4}	5.081×10^{-4}	4.931×10^{-4}
FCAE			
	Case 26	Case 27	Case 28
$\widetilde{T}$	7.694×10^{-6}	9.648×10^{-6}	6.258×10^{-6}
$\widetilde{Y}_F$	2.315×10^{-5}	2.856×10^{-5}	2.190×10^{-5}
$\widetilde{Y}_P$	7.748×10^{-6}	1.072×10^{-5}	7.216×10^{-6}
CAE			
	Case 26	Case 27	Case 28
$\widetilde{T}$	3.810×10^{-6}	3.141×10^{-6}	2.279×10^{-6}
$\widetilde{Y}_F$	1.868×10^{-5}	1.665×10^{-5}	1.630×10^{-5}
$\widetilde{Y}_P$	3.635×10^{-6}	4.579×10^{-6}	2.813×10^{-6}

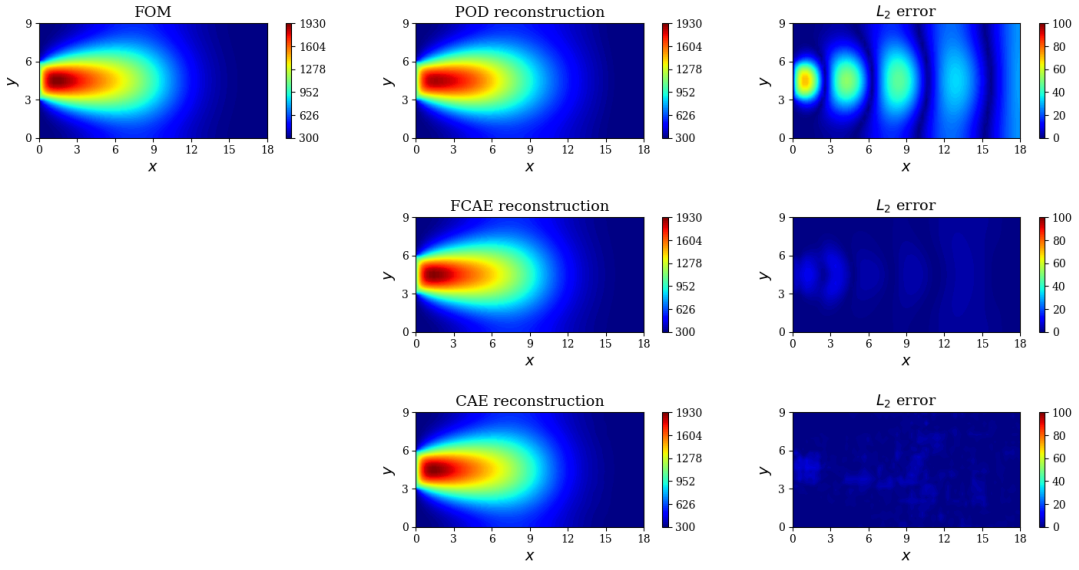


Figure 6.13: Reconstruction of temperature field (T, units K) and distribution of absolute error based on L_2-norm in POD, FCAE and CAE at $t = 0.02$ s for Case 27. The MSEs on normalized data at this time instant for POD, FCAE and CAE are 7.406×10^{-4}, 8.613×10^{-6} and 4.309×10^{-6}, respectively.

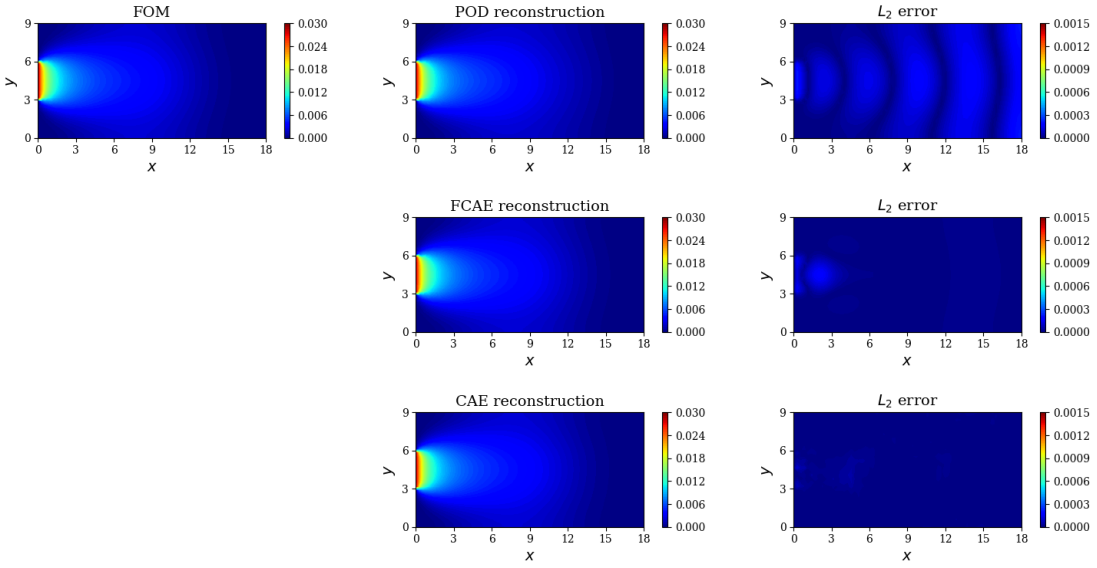


Figure 6.14: Reconstruction of fuel mass fraction field (Y_F) and distribution of absolute error based on L_2-norm in POD, FCAE and CAE at $t = 0.02$ s for Case 27. The MSEs on normalized data at this time instant for POD, FCAE and CAE are 9.922×10^{-4}, 2.484×10^{-5} and 1.626×10^{-6}, respectively.

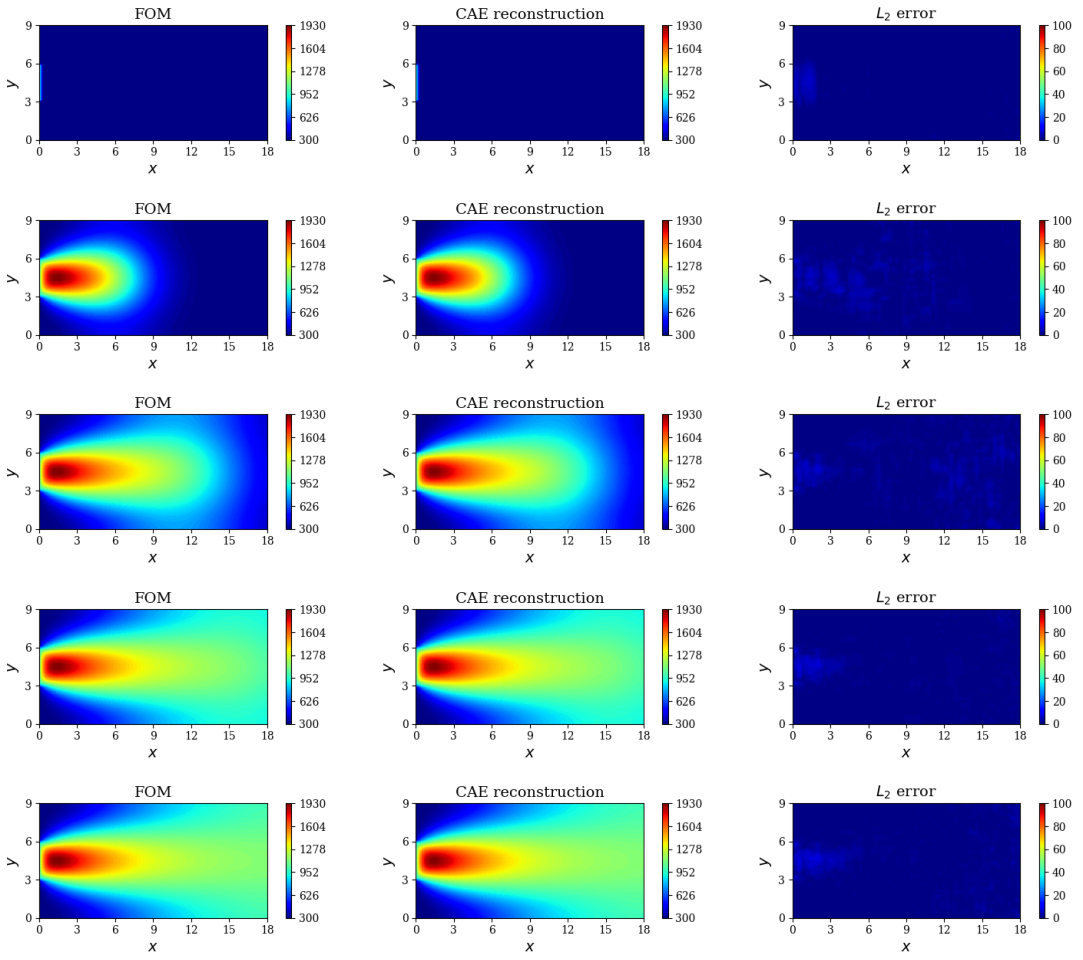


Figure 6.15: *ADR equation.* CAE-reconstructed temperature field (T, units K) for Case 27 at $t = 0$, 0.015, 0.03, 0.045 and 0.06 s (first – fifth rows). The MSEs on normalized data at these time instances are 1.442×10^{-6}, 3.838×10^{-6}, 3.686×10^{-6}, 1.737×10^{-6} and 2.426×10^{-6}, respectively.

6.5.4 Emulated Flowfields

The latent spaces obtained by the FCAE and CAE on the training and validation data are used to train the regressors. For each autoencoder, a regressor with five dense layers is designed to learn the mapping from the design parameters $\boldsymbol{\mu}$ and time instant t to the corresponding sequence of latent variables $\boldsymbol{z}(t;\boldsymbol{\mu})$, following Algorithm 6. The input and output data are standardized to stabilize the learning process. The detailed architecture of the regressors is given in Table 6.17. The number of epochs is set to 1,500, the batch size to 25, the learning rate for the Adam optimizer to 1×10^{-3}, and the L_2 regularization factor to 8×10^{-4}. This results in 1,077 trainable parameters. A comparison of the MSE losses on training, validation, and test sets is reported in Table 6.18 for each regressor. Evaluation of the MSE losses indicates satisfactory training performance without evidence of overfitting.

Table 6.17: *ADR equation.* Network structure of each regressor.

Layer	Output shape	Activation
Input	3	–
1st Dense	9	ReLU
2nd Dense	32	ReLU
3rd Dense	16	ReLU
4th Dense	9	ReLU
5th Dense	4	–

Table 6.18: *ADR equation.* MSE values for each regressor. Results are shown over the latent vectors of temperature and species mass fractions from the training, validation, and test sets.

FCAE			
	Training set	Validation set	Test set
$\boldsymbol{z}_T$	6.711×10^{-4}	8.271×10^{-4}	4.302×10^{-4}
$\boldsymbol{z}_{Y_F}$	1.069×10^{-3}	1.464×10^{-3}	6.955×10^{-4}
$\boldsymbol{z}_{Y_P}$	1.033×10^{-3}	1.014×10^{-3}	4.670×10^{-4}
CAE			
	Training set	Validation set	Test set
$\boldsymbol{z}_T$	2.377×10^{-3}	2.097×10^{-3}	1.926×10^{-3}
$\boldsymbol{z}_{Y_F}$	2.565×10^{-3}	2.299×10^{-3}	2.203×10^{-3}
$\boldsymbol{z}_{Y_P}$	2.245×10^{-3}	2.431×10^{-3}	1.309×10^{-3}

For a new design parameter $\boldsymbol{\mu}'$, the trained regressors are used to predict the corresponding time sequence of latent vectors, i.e., $\boldsymbol{z}_{\text{pred}}(t;\boldsymbol{\mu}') = \mathcal{R}(\boldsymbol{\mu}', t)$. This sequence is then passed to the trained decoders to obtain the emulated flowfield in physical space (cf. Algorithm 7). For the FCAE-$\mathcal{R}$ emulator, the decoding function $\mathcal{F}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}^{\text{FCAE}}(t;\boldsymbol{\mu}'))$ gives a one-dimensional vector $\boldsymbol{q}_{\text{pred}}(t;\boldsymbol{\mu}') \in \mathbb{R}^{n_{\text{in}}}$, whereas

for the CAE-$\mathcal{R}$ emulator, the decoding function $\mathcal{G}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}^{\text{CAE}}(t;\boldsymbol{\mu}'))$ gives a tensor $\boldsymbol{Q}_{\text{pred}}(t;\boldsymbol{\mu}') \in \mathbb{R}^{n_x \times n_y \times n_c}$.

The temporal evolutions of the components of the latent vectors for the variables T and Y_F are shown in solid lines in Fig. 6.16 for Case 27, which belongs to the test set and was not seen during the training of the neural networks. The latent space representations obtained from the FCAE and CAE exhibit different behavior. The time-series predictions for the trained regressors are shown in dashed lines in the same figure. The regressors are observed to predict the order and evolution of the latent space components accurately, indicating that the sequential behavior has been learned.

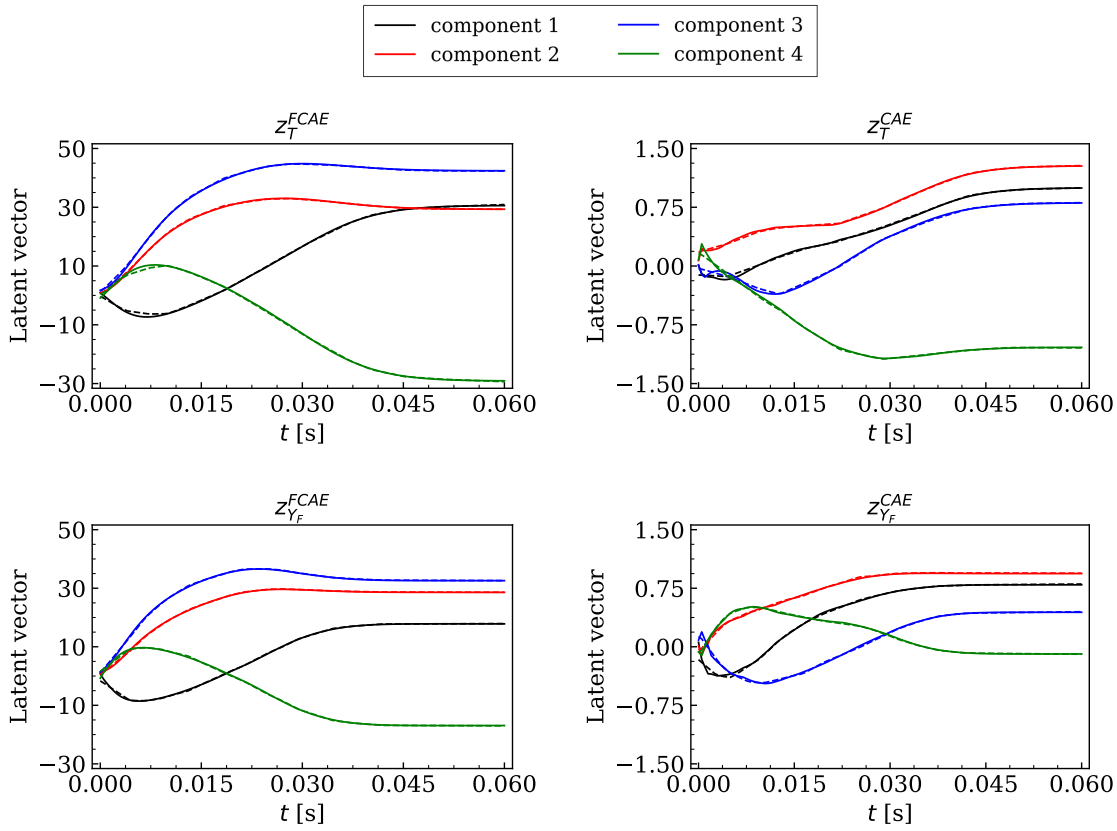


Figure 6.16: *ADR equation.* Temporal evolution of the components of the latent variables from FCAE (left) and CAE (right) for Case 27 for temperature and fuel mass fraction fields. Latent variables computed by the autoencoders (solid lines) and those predicted by the regressors (dashed lines) are compared.

Figures 6.17 and 6.18 show snapshots of the emulated temperature fields at various time instances for Case 27 from FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively. The absolute error distribution based on L_2-norm is also indicated for each snapshot. Qualitatively, the emulated flowfields from both frameworks show very good agreement with the FOM, although some differences are observed, particularly at $t = 0$. One possible explanation is that the training database does not contain as many snapshots of the

flame in early times as when the flame is more developed. This could be further improved by adding more snapshots at the desired time indices in the training database, or by incorporating the initial conditions into the models in the form of physics-based constraints, however, this is left as future work.

Axial and radial profiles of instantaneous emulated temperature and fuel mass fraction fields at various time instances for Case 27 are shown in Figs 6.19 and 6.20, respectively. The same observations can be made from these figures as those from Figs. 6.17 and 6.18.

For the sake of completeness, the MSE and L_2 relative error values over the snapshots of each case from the test set are shown in Table 6.19. Overall, the emulation errors for both surrogate models are very small (MSEs below 1.4×10^{-4}, and relative errors below 0.76%), with the CAE-$\mathcal{R}$ providing slight improvement in the global errors. This demonstrates the accuracy of both frameworks in capturing the salient features of the flowfields over a wide range of parameters in the design space.

Table 6.19: *ADR equation.* MSE values for the emulated temperature and species mass fractions fields for test cases. Relative errors, in percentage, are also indicated inside the parentheses.

FCAE-$\mathcal{R}$			
	Case 26	Case 27	Case 28
T_{pred}	9.020×10^{-5} (0.335%)	1.332×10^{-4} (0.402%)	8.789×10^{-5} (0.250%)
$Y_{F,\text{pred}}$	2.044×10^{-4} (0.600%)	1.089×10^{-4} (0.667%)	1.052×10^{-4} (0.365%)
$Y_{P,\text{pred}}$	9.452×10^{-5} (0.753%)	1.918×10^{-4} (0.633%)	8.140×10^{-5} (0.272%)
CAE-$\mathcal{R}$			
	Case 26	Case 27	Case 28
T_{pred}	5.262×10^{-5} (0.403%)	8.077×10^{-5} (0.224%)	8.160×10^{-5} (0.194%)
$Y_{F,\text{pred}}$	1.735×10^{-4} (0.336%)	1.005×10^{-4} (0.428%)	9.449×10^{-5} (0.228%)
$Y_{P,\text{pred}}$	6.734×10^{-5} (0.317%)	5.125×10^{-5} (0.408%)	5.106×10^{-5} (0.237%)

6.5.5 Computational Cost

Table 6.20 summarizes the computational time required for each step of the emulation process for both frameworks, with timing performed on a Linux desktop that has the following specifications: Intel(R) Core(TM) i7-10700T CPU @ 2.00GHz. The preprocessing step, which consists of extracting the snapshots from the FOM, requires about 0.1 s (0.004 s for each of the 25 cases used for training/validation). For each emulation framework and for each of the three field variables (i.e., T, Y_F and Y_P), one autoencoder and one regressor are trained. Thus, in each framework, the training step consists of designing three autoencoders and three regressors. This gives a total training time of 3,675 and 15,279 s for the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively. After training is completed, the total inference time to predict the flowfield for a new design point is approximately 3.6 and 4.5 s for the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively. In comparison, the FOM simulations take on average 52 s per case. Thus, if only the

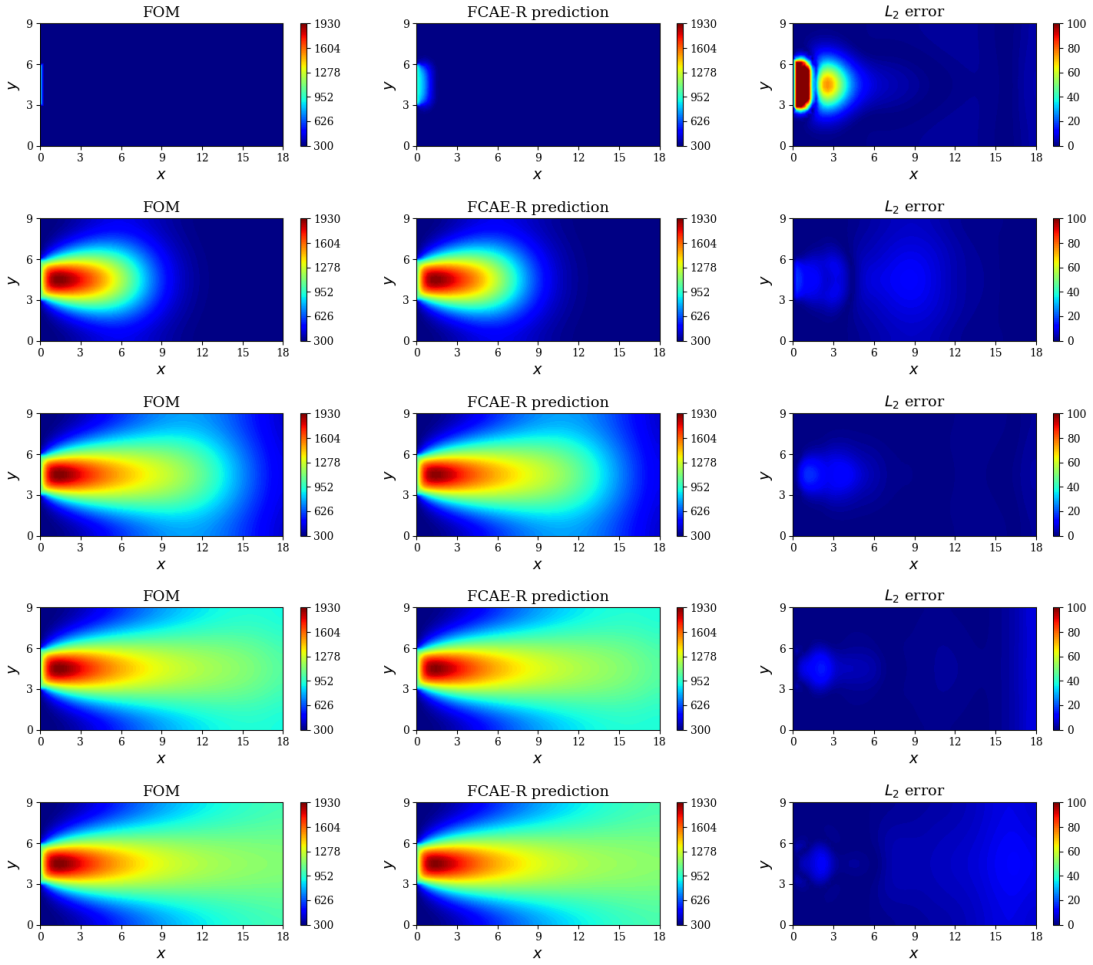


Figure 6.17: *ADR equation.* Instantaneous snapshots of temperature field (T, units K) for Case 27 at $t = 0$, 0.015, 0.03, 0.045 and 0.06 s (first – fifth rows). Flowfields computed by FOM and those predicted by the FCAE-$\mathcal{R}$ emulator are compared. The absolute error distribution based on L_2-norm is also indicated. The MSEs on normalized data at these time instances are 2.197×10^{-3}, 4.374×10^{-5}, 1.702×10^{-5}, 1.205×10^{-5} and 4.837×10^{-5}, respectively.

inference time is considered, an overall speedup of 14 and 12 times is achieved by the the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$, respectively. The speedup factor can be expected to increase further with the complexity of the FOM (see, for example, the emulation speedup in the A-M1 injector problem in Sec. 7.3.3.6). This highlights the potential of data-driven methods to lower the computational cost of numerical simulations for design purposes.

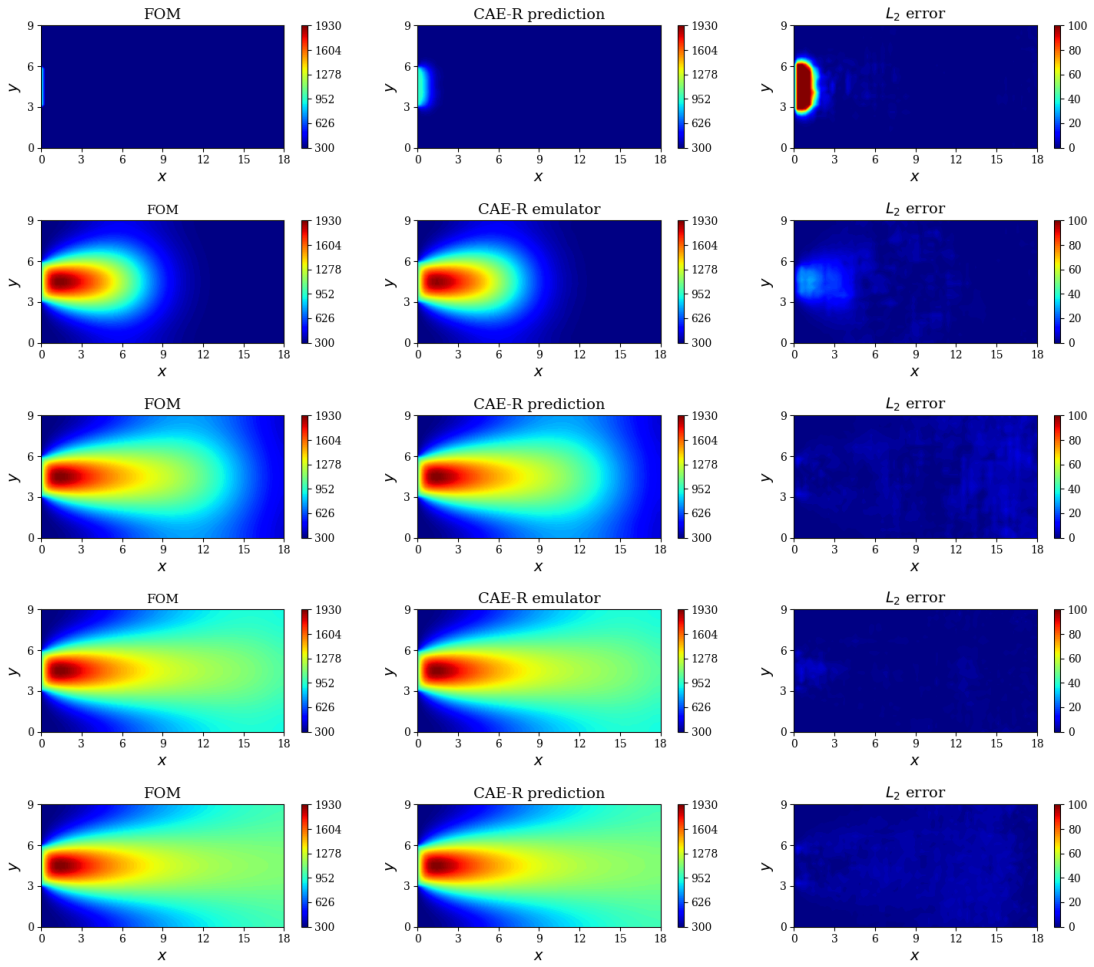


Figure 6.18: *ADR equation.* Instantaneous snapshots of temperature field (T, units K) for Case 27 at $t = 0$, 0.015, 0.03, 0.045 and 0.06 s (first – fifth rows). Flowfields computed by FOM and those predicted by the CAE-$\mathcal{R}$ emulator are compared. The absolute error distribution based on L_2-norm is also indicated. The MSEs on normalized data at these time instances are 2.707×10^{-3}, 3.542×10^{-5}, 1.643×10^{-5}, 2.196×10^{-6} and 1.498×10^{-5}, respectively.

Table 6.20: *ADR equation.* Computational time in CPU-secs for each step of the emulation process For the autoencoder training step, results are also provided in GPU-secs.

Step	Time (FCAE-$\mathcal{R}$)	Time (CAE-$\mathcal{R}$)
Preprocessing	0.1 CPU-secs	0.1 CPU-secs
Autoencoder training	$3 \times 1,130$ CPU-secs	$3 \times 4,986$ CPU-secs
	(3×626 GPU-secs)	(3×990 GPU-secs)
Regressor training	3×95 CPU-secs	3×107 CPU-secs
Flow prediction	3×1.2 CPU-secs	3×1.5 CPU-secs

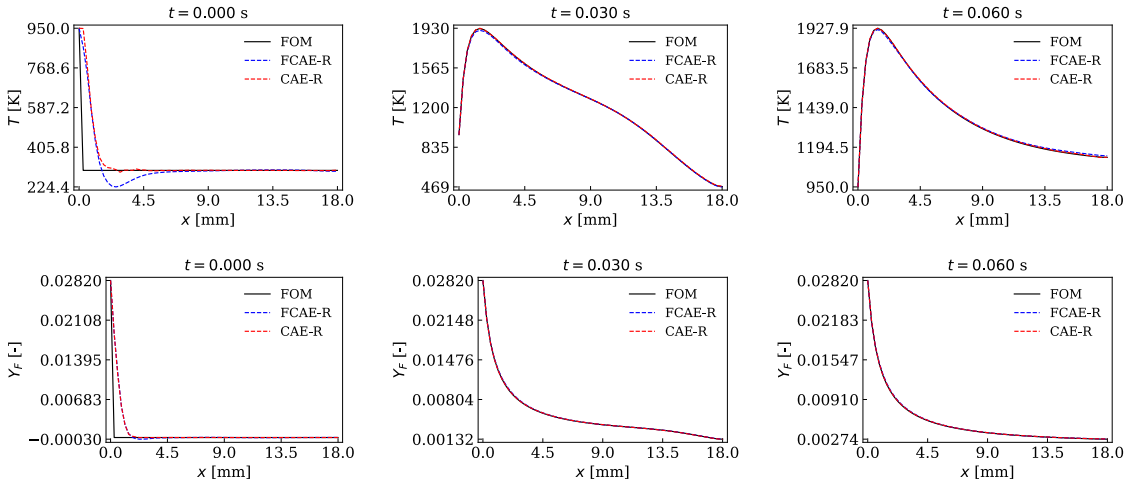


Figure 6.19: *ADR equation.* Axial profiles of the instantaneous temperature (top) and fuel mass fraction (bottom) fields at centerline at $t = 0$, 0.03 and 0.06 s for Case 27.

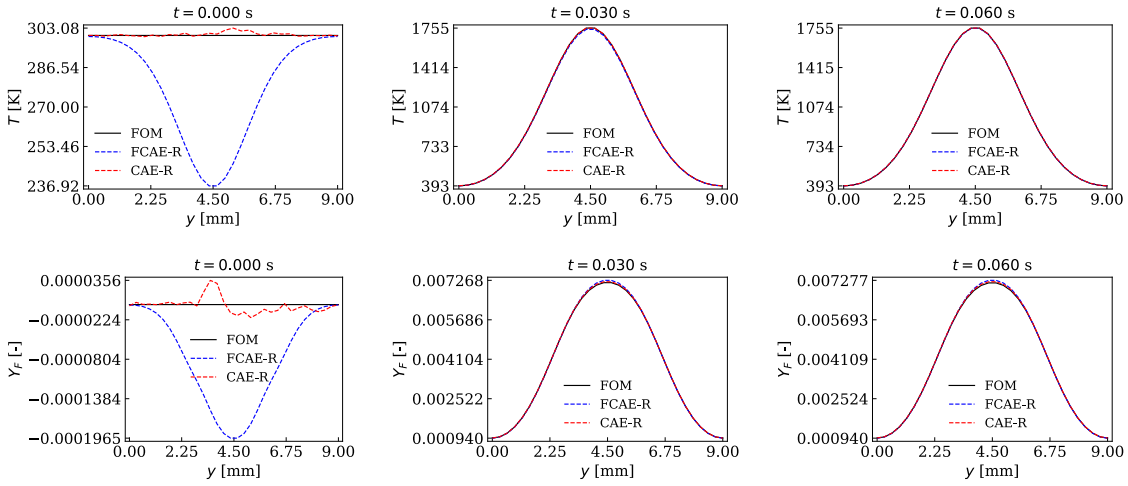


Figure 6.20: *ADR equation.* Radial profiles of the instantaneous temperature (top) and fuel mass fraction (bottom) fields at $x \approx 3$ mm at $t = 0$, 0.03 and 0.06 s for Case 27.

6.5.6 Sensitivity Analysis

In the following, a sensitivity analysis is performed to evaluate the influence of certain hyperparameters and design choices, such as the type of activation function, number of snapshots, and number of cases in the DoE, on the performance of the emulators.

Dependence on the Type of Activation Function

The dependence on the type of activation function is investigated and reported in Table 6.21. Results are demonstrated for the reconstructed temperature field from

the CAE. Here, we examine the sigmoid, tanh, ReLU (baseline), ELU, and Swish activation functions. From this table, it is observed that the errors are largest with the sigmoid function (1.226×10^{-3}), while they are relatively lower with the tanh, ReLU, ELU, and Swish functions (below 3.4×10^{-6}), with ELU and Swish providing slightly more accurate predictions than the others. As an aside, we note that during the hyperparameter tuning phase, which was carried in Secs. 6.5.3 and 6.5.4, only sigmoid, tanh, and ReLU were initially evaluated, and thus ReLU was selected in the final emulator models since it provided the best accuracy among these three functions, as reported in Table 6.21. The assessment of the ELU and Swish functions, however, was done at a much later stage in the project, mainly for the purpose of sensitivity analysis. Thus, for this reason, they were not selected in the final emulator models despite that they provided slightly better results than the ReLU function.

Table 6.21: *ADR equation*. Dependence on the choice of activation function: MSE values for the reconstructed temperature field from the CAE.

	sigmoid	tanh	ReLU (baseline)	ELU	Swish
Test error on $\widetilde{T}$	1.226×10^{-3}	3.304×10^{-6}	3.077×10^{-6}	2.263×10^{-6}	1.600×10^{-6}

Dependence on the Number of Snapshots

The dependence on the number of snapshots extracted from each FOM case, n_t, in the autoencoders, regressors and emulators is investigated and reported in Tables 6.22, 6.23 and 6.24, respectively. Results are demonstrated for the temperature and product mass fraction fields. Here, we examine $n_t = 121$ (baseline), 241 and 601. From these tables, we can see that, overall, the accuracy in the reconstructions and predictions is increased with increasing n_t. Note that this outcome is expected since deep learning models are data-hungry models.

Table 6.22: *ADR equation*. Dependence on the number of snapshots: MSE values for the reconstructed temperature and product mass fraction fields from the FCAE and CAE. Results are shown over the snapshots from the test set.

FCAE			
	$n_t = 121$ (baseline)	$n_t = 241$	$n_t = 601$
Test error on $\widetilde{T}$	7.867×10^{-6}	6.508×10^{-6}	5.585×10^{-6}
Test error on $\widetilde{Y}_P$	8.563×10^{-6}	5.935×10^{-6}	5.183×10^{-6}
CAE			
	$n_t = 121$ (baseline)	$n_t = 241$	$n_t = 601$
Test error on $\widetilde{T}$	3.077×10^{-6}	1.739×10^{-6}	6.412×10^{-7}
Test error on $\widetilde{Y}_P$	3.676×10^{-6}	1.269×10^{-6}	6.643×10^{-7}

Table 6.23: *ADR equation.* Dependence on the number of snapshots: MSE values for each regressor. Results are shown over the latent vectors of temperature and product mass fraction from the test set.

FCAE			
	$n_t = 121$ (baseline)	$n_t = 241$	$n_t = 601$
Test error on $\boldsymbol{z}_T$	4.302×10^{-4}	3.515×10^{-4}	4.044×10^{-5}
Test error on $\boldsymbol{z}_{Y_P}$	4.670×10^{-4}	3.432×10^{-4}	8.362×10^{-5}
CAE			
	$n_t = 121$ (baseline)	$n_t = 241$	$n_t = 601$
Test error on $\boldsymbol{z}_T$	1.926×10^{-3}	6.631×10^{-4}	4.975×10^{-4}
Test error on $\boldsymbol{z}_{Y_P}$	1.309×10^{-3}	2.386×10^{-4}	3.119×10^{-4}

Table 6.24: *ADR equation.* Dependence on the number of snapshots: MSE values for the emulated temperature and product mass fraction fields from the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$ on each case from the test set.

FCAE-$\mathcal{R}$			
	$n_t = 121$ (baseline)	$n_t = 241$	$n_t = 601$
T_{pred} (Case 26)	9.020×10^{-5}	8.563×10^{-5}	1.156×10^{-5}
T_{pred} (Case 27)	1.332×10^{-4}	7.920×10^{-5}	1.470×10^{-5}
T_{pred} (Case 28)	8.789×10^{-5}	6.764×10^{-5}	1.025×10^{-5}
$Y_{P,\text{pred}}$ (Case 26)	9.452×10^{-5}	9.060×10^{-5}	2.228×10^{-5}
$Y_{P,\text{pred}}$ (Case 27)	1.918×10^{-4}	1.092×10^{-4}	1.401×10^{-5}
$Y_{P,\text{pred}}$ (Case 28)	8.140×10^{-5}	5.692×10^{-5}	1.511×10^{-5}
CAE-$\mathcal{R}$			
	$n_t = 121$ (baseline)	$n_t = 241$	$n_t = 601$
T_{pred} (Case 26)	5.262×10^{-5}	1.298×10^{-5}	1.039×10^{-5}
T_{pred} (Case 27)	8.077×10^{-5}	1.661×10^{-5}	9.240×10^{-6}
T_{pred} (Case 28)	8.160×10^{-5}	1.150×10^{-5}	9.077×10^{-6}
$Y_{P,\text{pred}}$ (Case 26)	6.734×10^{-5}	1.575×10^{-5}	9.726×10^{-6}
$Y_{P,\text{pred}}$ (Case 27)	5.125×10^{-5}	1.110×10^{-5}	6.979×10^{-6}
$Y_{P,\text{pred}}$ (Case 28)	5.106×10^{-5}	9.937×10^{-6}	5.982×10^{-6}

Dependence on the Number of Cases in the DoE

The dependence on the number of cases in the DoE, n_c, is investigated and reported in Table 6.25. Results are demonstrated for the reconstructed temperature field from the CAE. Here, we examine $n_c = 12$, 28 (baseline), and 67. From these tables, it is observed that, overall, the accuracy in the reconstructions is increased with increasing n_c. Similar to the above, this outcome is also expected since deep learning models are data-hungry models.

Table 6.25: *ADR equation.* Dependence on the number of cases in the DoE: MSE values for the reconstructed temperature field from the CAE.

	$n_c = 12$	$n_c = 28$ (basline)	$n_c = 67$
Test error on $\widetilde{T}$	5.999×10^{-6}	3.077×10^{-6}	1.736×10^{-6}

6.6 Summary

The aim of this study was to develop data-driven DL-based surrogate modeling approaches to enable efficient prediction and parametric estimation of spatiotemporal flow dynamics. Two surrogate frameworks were constructed for this purpose. The first, referred to as FCAE-$\mathcal{R}$, featured an FCAE, for nonlinear dimensionality reduction of the flowfields, and a regression-based neural network ($\mathcal{R}$) for supervised learning of the latent space representations This model is suitable for any type of data (structured/unstructured, regular/irregular). The second, referred to as CAE-$\mathcal{R}$, was obtained by replacing the FCAE with a CAE to retain the spatial coherence of the input data while maintaining the same deep-learning-based regression method. This model is generally suitable for structured image data. In both cases, the surrogate model was constructed from a full-order model of spatiotemporal distribution of field variables of interest, for which a design of experiments on the input space was explored by evaluating changes in selected design and physical parameters.

The proposed frameworks were evaluated with two nonlinear model problems, the one-dimensional viscous Burgers equation, parametrized using the Reynolds number, and the two-dimensional ADR equation, parametrized using the pre-exponential factor and activation energy.

In each problem, a standalone assessment of the ROM strategies revealed that for the same level of data compression, the FCAE and CAE provided better reconstruction accuracy than the classical POD, with the CAE achieving the smallest errors. Furthermore, it was demonstrated that the spatiotemporal flowfields at new design points were well predicted by both surrogate frameworks. The two emulators had comparable prediction accuracy, with the CAE-$\mathcal{R}$ providing slight improvement in the global MSEs. In comparison to using the FOM to simulate the next design point of interest, the FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$ showed a speedup of up to 14 and 12 times, respectively (in the ADR problem).

List of Main Symbols

Latin Symbols

A	pre-exponential factor
$\boldsymbol{b}$	bias vector
E_a	activation energy

J	loss function
L, L_x, L_y	domain length
n_{in}	size of input layer of FCAE
n_{epoch}	number of epochs
n_{batch}	batch size
n_{latent}	size of latent space
n_{set}	number of elements in the set
$n_{\boldsymbol{\mu}}$	number of design/physical parameters
n_{c}	number of channels in CAE, number of cases in DoE
n_x	number of points along x direction
n_y	number points along y direction
n_t	number of snapshots extracted from each FOM case
n_{var}	number of field variables
n_{grid}	number of grid points in the subdomain of interest
n_r	number of retained POD modes
$n_{r,x\%}$	number of POD modes required to recover $x\%$ of modal energy
$\boldsymbol{q}$	FCAE input, reaction source term vector
$\widetilde{\boldsymbol{q}}$	FCAE output
Q	heat of reaction
$\boldsymbol{Q}$	CAE input
$\widetilde{\boldsymbol{Q}}$	CAE output
R	universal gas constant
Re	Reynolds number
s	stride length in CAE
s_{N}	normalization operator
s_{S}	standardization operator
t	physical time
t_{end}	final time
T	temperature
u	axial velocity
$\boldsymbol{v}$	velocity vector
$\boldsymbol{w}$	vector field, thermo-chemical composition vector
$\boldsymbol{W}$	weight matrix
W_i	molecular weight of the ith species
$\boldsymbol{x}$	spatial coordinate vector
x, y	spatial coordinates
Y_i	mass fraction of the ith species
$\boldsymbol{z}$	latent vector

Calligraphic Symbols

$\mathcal{F}_{\text{enc}}$, $\mathcal{F}_{\text{dec}}$	encoder function of FCAE, decoder function of FCAE
$\mathcal{G}_{\text{enc}}$, $\mathcal{G}_{\text{dec}}$	encoder function of CAE, decoder function of CAE
$\mathcal{R}$	regressor

$\mathcal{S}$	simulator

Greek and Blackboard Bold Symbols

Ω	spatial domain
Γ_1-Γ_6	boundary conditions
$\boldsymbol{\mu}$	vector of design/physical parameters
$\boldsymbol{\phi}$	parameters of regressor
λ_{reg}, ζ_{reg}	control parameter for L_2 regularization
λ_{lr}	initial learning rate
ν	viscosity
ν_i	stioichiometric coefficient of the ith species
κ	diffusivities
ρ	density
$\mathbb{D}$	parameter domain

Subscripts

dec	decoder
enc	encoder
F	fuel
O	oxidizer
P	product
pred	predicted variable
true	true variable
valid	validation

Superscripts

$\widetilde{\square}$	reconstructed variable
$\hat{\square}$	normalized or standardized variable
$\square'$	new value of variable

CHAPTER 7
LEARNING SPATIOTEMPORAL INJECTION MAPS USING A DATA-DRIVEN EMULATOR FOR RAPID DIESEL ENGINE DESIGN

In the previous chapter, a deep learning spatiotemporal emulation framework was introduced and investigated on canonical fluid flow problems. In this chapter, the emulator is applied to a representative engineering problem relevant to automotive propulsion, in order to address Objectives 1(b) and 3(b) (cf. Sec. 1.2). The selected example is specific to a diesel engine problem, but the general emulation approach can be taken as a model, and other industry-relevant problems could be addressed following a similar paradigm. Please note that this chapter depends on Chapters 2 and 3 as well as Secs. 4.1, 4.2, and 4.3.

7.1 Abstract

Fuel injector design has a substantial influence on the spray characteristics and fuel-air mixing in the combustion chamber, thus affecting the performance and emissions of direct-injection engines. To date, physics-based numerical approaches that link injector and spray dynamics, such as the Eulerian-Lagrangian Spray Atomization (ELSA) method and the two-stage one-way coupling (OWC) method, have shown great success in evaluating the complex interplay among injector geometry, fuel properties, and operating conditions in the determination of injector efficacy and spray development. However, such simulation approaches are too computationally expensive to be used routinely by industry for injector design, mainly due to the fine temporal and spatial resolution required to resolve wall-bounded flow within the injector. This chapter proposes a data-driven emulation framework that can learn and quickly predict spatiotemporal injection maps at the nozzle orifice exit, thus providing a computationally tractable link between internal nozzle flow and external spray and combustion development, and enabling rapid exploration of the desired design space. The emulation framework encompasses autoencoders for nonlinear dimensionality reduction and deep neural networks for regression and predictive modeling. As a demonstration case, the turbulent multiphase flow development in the side-oriented single-hole A-M1 diesel injector is considered. The accuracy of the prediction of flow features at the orifice exit of the injector is assessed, and the efficiency of the proposed emulator is evaluated. The emulator is also tested by evaluating the resulting spray and combustion characteristics from OWC spray simulations that employ the emulator-predicted injection maps as inflow boundary conditions, instead of the CFD-generated injection maps. The findings from this work will provide an efficient pathway to accelerate the design and development of next-generation advanced combustion engines.

7.2 Introduction and Literature Review

Critical to the quest for next-generation propulsion and power generation systems that will meet ever-tighter emissions and environmental regulations, high-fidelity CFD simulations of multiphase and reacting flows are in general computationally expensive, due to the broad spectrum of length and time scales that must be resolved. The cost of these calculations can easily become prohibitive in the case of design-oriented studies, for which a large number of parametric variations is needed to survey the full design space. A novel design methodology is thus needed.

Within the context of direct-injection diesel engine simulations, which are the focus of the present work, several approaches have been developed to represent the high-pressure fuel injection process and allow for coupling with established spray and combustion models, as shown in Table 7.1. These approaches need to account, in one way or another, for a wide range of physics phenomena, including in-nozzle cavitation, liquid fuel atomization, dispersion, evaporation, fuel-air mixing, ignition, and reaction (see Secs. 4.3.4 and 4.3.5 for a basic review of these physio-chemical processes). The presence of such a wide range of physics and the highly nonlinear interaction of turbulence with other processes makes numerical predictions of the flow dynamics within such systems extremely challenging.

In the *Eulerian-Lagrangian Spray Atomization* (ELSA) approach [195], the internal nozzle flow and near-nozzle region are dynamically coupled and modeled using an Eulerian framework. Based on a specified criterion, such as interfacial surface area, the representation of the spray is then transitioned to Lagrangian parcels. Examples of studies employing this approach can be found in Refs. [196, 197]. Although this framework allows for accurate coupling between the injector dynamics and the ensuing spray, the computational expense of this approach limits its application to engine simulations. In particular, the modeling of the internal flow development generally results in timesteps that are two orders of magnitude smaller than those commonly used in direct-injection engine simulations ($\mathcal{O}(1$ ns) vs. $\mathcal{O}(0.1$ μs)).

An alternative to the ELSA approach for representing the fuel injection process is the *two-stage static coupling approach*, also referred to as *one-way coupling* (OWC). In this approach, internal flow simulations are first performed to calculate key quantities at the nozzle orifice exit(s), namely, injected mass, velocity, turbulence levels, and liquid volume fraction. These spatiotemporal distributions are then used to initialize LE simulation of the ensuing spray. The initial size and velocity distribution of the spray droplets (or parcels) in the LE simulation can be set directly from information based on the flowfields at the nozzle orifice exit, as in Refs. [198, 199, 200, 201, 202, 203, 204]; this initialization technique is referred to as the *blob-injection method*. Alternatively, the droplets can be released on the liquid core surface instead of in the orifice cross section, for a more detailed treatment of the atomization and drop break-up processes within the dense spray near the nozzle. In this case, the initial size of the parcel droplets is not set equal to the orifice diameter, as is usual in the blob-injection method. Instead, the droplets begin with smaller diameters as determined from a primary breakup model [205]. The model defines an auxiliary grid onto which nozzle flow data obtained by the internal flow simulation are projected to compute

the initial droplet size and velocity distributions at the liquid core surface, as well as other quantities such as the initial spray angle needed for the initialization of the droplet parcels within the LE simulation; this initialization technique is referred to as the *core injection method* (CIM), or *Lagrangian parcel method*. Examples of studies employing the OWC approach in conjunction with CIM can be found in Refs. [206, 207]. Overall, although OWC enables a framework to couple the injector dynamics with the ensuing spray using timesteps that are compatible with engine simulations ($\mathcal{O}(0.1\ \mu\text{s})$), the internal flow simulations remain the computational bottleneck in this framework.

Another methodology for representing the fuel injection event is the *rate-of-injection* (ROI) approach, in which the LE simulation is initialized with a fuel injection model based on a one-dimensional representation of the injection conditions at the nozzle orifice exit(s). With this approach, it is assumed that pure liquid fuel is injected with a spatially constant profile of velocity, temperature, and mass across the orifice exit, thus neglecting in-nozzle cavitation as well as the spatial variation of flow properties across the orifice exit. The injection model can be obtained either from experiments, as in Refs. [208, 200], or from numerical simulations, as in Ref. [202]. Although this approach is well suited for representing highly turbulent and pure liquid injection, its applicability for injection conditions featuring cavitation and flash boiling is limited.

The above-mentioned currently available simulation tools are either too computationally expensive or overly simplistic. There remains a critical need for an efficient end-to-end simulation framework that can couple injector dynamics with the resultant spray and combustion characteristics. In particular, to facilitate engine design studies, the time-to-solution will need to be accelerated by orders of magnitude, and one approach is the use of data-driven emulation. The present work presents a novel data-driven approach in which deep learning surrogates are leveraged to predict the flowfields with high accuracy at a fraction of the cost of full-scale CFD simulations. The predicted flowfields at the injector exit are then used to initialize the injection conditions for the LE spray simulations, providing a computationally tractable link between the internal flow and external spray and combustion development. To the author's knowledge, this work is *a first of its kind* within the contexts of OWC simulations and data-enabled engine design.

The proposed emulation framework consists of three steps, as schematically illustrated in Fig. 7.1. In the first step, a spatiotemporal training database of the internal nozzle flow is generated from high-fidelity CFD simulations spanning a design space of interest. For highly resolved CFD simulations, the resulting database would be too large to be used directly for regression; reduced-order modeling is a critical step to enable emulation. As a first step, therefore, autoencoders are applied to compress the data and extract dominant features at the injector exit for physics-based data assimilation. The main appeal of autoencoders lies in their ability to model complex nonlinear relationships at the same level of data compression than those of classical ROM techniques such as POD. In the second step, a regression model based on neural networks is employed to learn the relationship between the selected design parameters and the reduced-space representations. In the third and last step, these tools

are integrated into a unified emulator model along with a reconstruction algorithm to predict the flowfields for new design settings. The predicted flowfields are then used to initialize the LE spray simulations using the OWC approach.

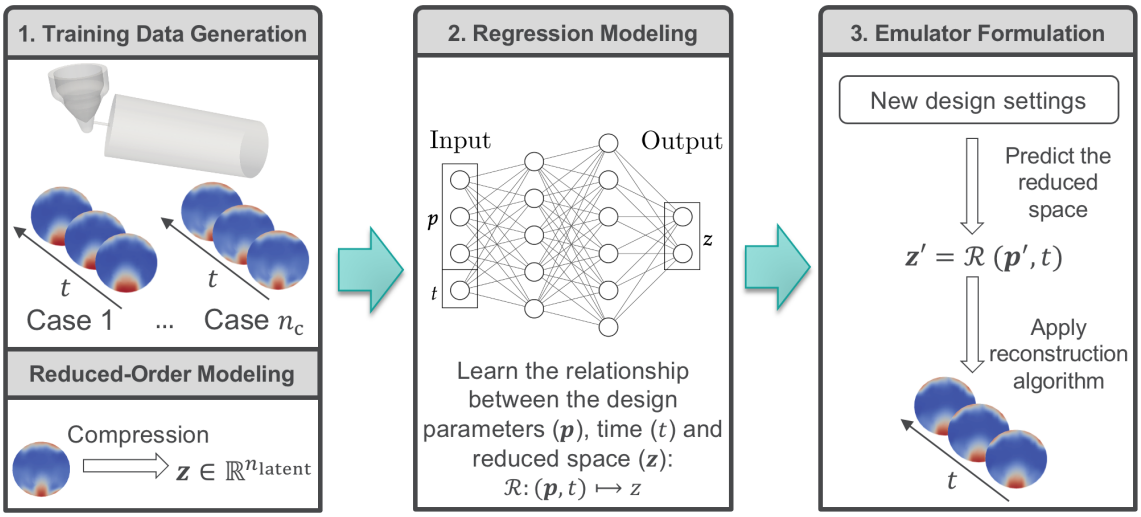


Figure 7.1: Overview of the proposed emulation framework and its application to the A-M1 injector flow.

This rest of this chapter is divided into two parts. The first part, Sec. 7.3, is concerned with internal nozzle flow simulation and emulation, and is structured as follows. Section 7.3.1 describes the simulation framework, in which the injector specifications, operating conditions, CFD setup, and sampled points obtained by the design of experiments are detailed. Section 7.3.2 introduces the proposed emulation framework. Section 7.3.3 presents the simulation and emulation results. Sec. 7.4 is concerned with one-way coupled spray simulation, and is structured as follows. Section 7.4.1 presents the spray modeling approach, including the spray initialization strategy and OWC method. Section 7.4.2 presents the results from the OWC spray simulation predictions. Finally, key findings and concluding remarks are summarized in Sec. 7.5.

Table 7.1: Overview of numerical studies focusing on coupled approaches for the simulation of in-nozzle flow and exterior spray.

One-stage fully-coupled approach					
Numerical model for nozzle and spray regions			Injector nozzle type	Reference	Year
TFM (with RPE model for cavitation)			1-hole Chalmers diesel	Berg et al. [206]	2005
LES-VOF + Lagrangian particle model (without cavitation model)			6-orifice swirl gas-turbine	Kim et al. [209]	2014
RANS-HMM (with HRM for cavitation)			1-hole ECN Spray-A	Xue et al. [208]	2015
RANS-HMM (with HRM for cavitation)			1-hole ECN Spray-A	Bravo et al. [210]	2015
ELSA approach					
Numerical model for nozzle and spray regions			Injector nozzle type	Reference	Year
URANS-ELSA (without cavitation model)			8-hole ECN Spray-G GDI	Saha et al. [196]	2018
RANS/LES-ELSA (without cavitation model)			1-hole ECN Spray-A	Anez et al. [197]	2019
LES-ICM-ELSA (without cavitation model)			1-hole ECN Spray-A	Anez et al. [197]	2019
Two-stage OWC approach					
In-nozzle flow model	Cavitation model	Spray model	Injector nozzle type	Reference	Year
TFM	RPE	LE (with CIM)	1-hole Chalmers diesel	Berg et al. [206]	2005
RANS-HMM	–	LE	6-hole mini-sac GDI	Som et al. [198]	2010
RANS-MFM	RPE	LE	Mini-sac diesel	Battistoni et al. [199]	2012
TFM	RPE	LE (with CIM)	5-hole diesel	Wang et al. [207]	2014
URANS/RANS-VOF	–	LE	1-hole ECN Spray-A	Quan et al. [201]	2016
URANS/RANS-VOF	HRM	LE	1-hole ECN Spray-H	Quan et al. [201]	2016
URANS-HMM	HRM	LE	8-hole ECN Spray-G GDI	Saha et al. [200]	2017
RANS-HMM	HRM	LE	8-hole XPI diesel	Travers et al. [202]	2019
HMM	Schnerr-Sauer	LE	1-hole HJ-SIP lubricator	Ravendran et al. [203]	2019
URANS-HMM	HRM	LE	8-hole ECN Spray-G GDI	Nocivelli et al. [204]	2020
ROI approach					
Source for boundary data at nozzle exit		Spray model	Injector nozzle type	Reference	Year
Experiment		LE	1-hole ECN Spray-A	Xue et al. [208]	2015
Experiment		LE	8-hole ECN Spray-G GDI	Saha et al. [200]	2017
CFD simulation		LE	8-hole XPI diesel	Travers et al. [202]	2019

7.3 Internal Flow Simulation and Emulation

This section is concerned with internal flow simulation and emulation. The material presented here is adapted from:

- [211] P. J. Milan, R. Torelli, B. Lusch, and G. M. Magnotti, "Data-driven model reduction of multiphase flow in a single-hole automotive injector," *Atomization and Sprays*, Vol. 30, pp. 401-429, 2020.
- [212] P. J. Milan, S. Mondal, R. Torelli, B. Lusch, R. Maulik, and G. M. Magnotti, "Data-driven modeling of large-eddy simulations for fuel injector design," *AIAA 2021-1016*, pp.1-15, 2021.

7.3.1 Simulation Framework

To generate data for training, evaluation and testing of the emulator framework, the commercially available CFD code `CONVERGE v3.0` [98] was employed to perform internal flow simulations to generate spatiotemporal distributions of key flowfield variables at the orifice exit of the injector. Details regarding the simulated conditions and modeling approaches are provided in the following sections.

7.3.1.1 Injector Configuration and Operating Conditions

Turbulent multiphase flow simulations were performed for pressurized liquid n-dodecane fuel in the A-M1 injector, which is a side-oriented single-hole injector geometry, in order to link fuel properties, operating conditions, and geometrical parameters with the injection conditions at the orifice exit. An illustration of the injector is provided in Fig. 7.2, which is comprised of an injector body (shown in gray), needle (blue), sac (green), and orifice (red) geometry comparable to those of the Engine Combustion Network (ECN) Spray C injector [213, 214] and the Spray Combustion Consortium (SCC) M1 injector [215, 216] (see Sec. 4.3.3 for a description of these components and their functions). The computational domain consists of the lower end of the injector internal volume as well as an outer cylindrical chamber. The dimensions of the outer chamber (not shown in Fig. 7.2), i.e., diameter of 4 mm and length of 10 mm, were selected to prevent wall effects from affecting the initial spray development. According to the nominal specifications, the injector orifice is oriented at an angle of 73° with respect to the needle axis and is characterized by a sharp inlet radius of curvature to promote cavitation inception. The orifice is 1.0 mm long and its diameter is 170 μm. A plane located at about 46 μm upstream from the orifice exit's cross section is selected as the subdomain of interest for emulation to characterize the thermodynamic and fluid mechanic conditions at the outlet of the injector, as shown in Fig. 7.2. The subdomain sampling at this location results in a slice containing 4,214 grid points. For the baseline case, a fixed needle lift of 15 μm is used.

The injection condition and fuel properties studied in this work are summarized in Table 7.2 and are similar to the baseline cold condition from the ECN. Liquid n-dodecane at a temperature of 323 K is injected with a fuel pressure of 1,500 bar into a nitrogen-filled chamber whose pressure is of 20 bar; this corresponds to a cavitation number of 1.014 (see Sec. 4.3.5 for the definition of the cavitation number). Dissolved gas in the fuel is represented using a trace amount of non-condensable gas (N_2), based on recommendations from Battistoni et al. [217]. For the baseline configuration, the mass fraction of non-condensable gas is set to 2×10^{-5}, which is consistent with previous numerical studies [218, 219]. All cases studied in this work assume that the injector sac and orifice are completely filled with fuel at the start of the simulation independent of the needle location. Because the flow profiles in the injector are evaluated at quasi-steady state, it is reasonable to assume that the initial conditions do not affect the velocity and species predictions at the location of interest for this work.

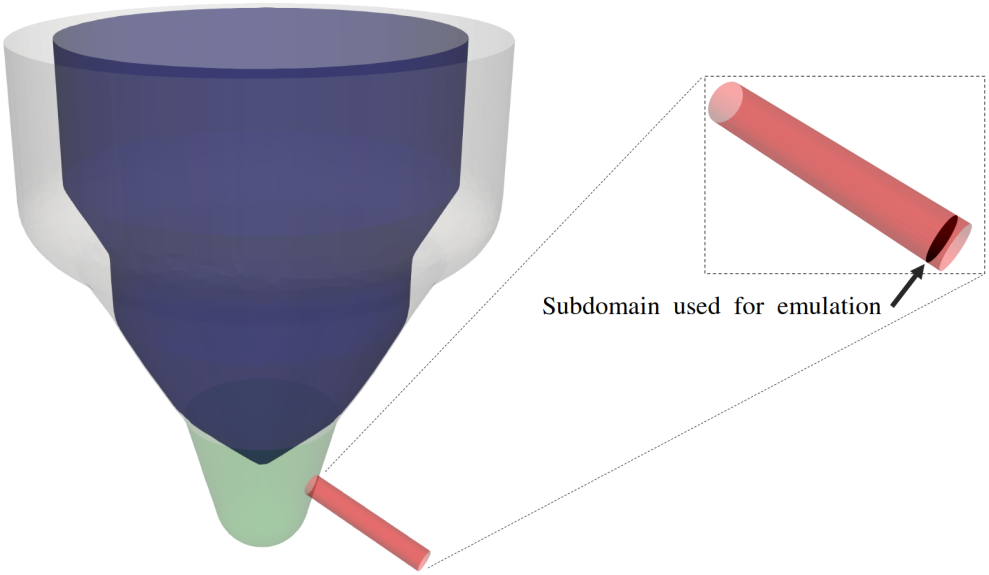


Figure 7.2: 3D view of the baseline A-M1 injector. Also shown are a zoomed-in view of the orifice and the subdomain that is used for emulation.

7.3.1.2 Design of Experiments

A DoE was defined using `JMP` [220], a commercially available statistical analysis software package. A DoE approach allows for the efficient selection of numerical simulations required to explore a range of design parameters that are known to affect the injector exit conditions. In this work, three design parameters were selected, namely needle lift, dynamic viscosity of the liquid-phase fuel, and level of non-condensable gas in the fuel. The baseline condition (referred to as "Case 0") and range of design parameters considered in this study are detailed in Table 7.3. The minimum needle lift is defined on the basis of computational constraints required to adequately resolve

Table 7.2: Summary of the operating condition for the A-M1 injector.

Parameter	Values
Fuel	n-dodecane
Injection pressure [bar]	1500
Ambient pressure [bar]	20
Fuel temperature [K]	323
Saturation Pressure [Pa]	133.3
Bulk Modulus [GPa]	1.475
Reference fuel density [kg/m^3]	727.4
Reference pressure [Pa]	100,000

the flow, while the maximum needle lift is informed from typical needle motion profiles from similar heavy-duty injectors [221]. The range of dynamic viscosity values is informed by fuel property space evaluated in the DoE from Ref. [222], which considered 13 different fuels including alcohols, alkanes, and methyl esters. The range of non-condensable gas concentrations was based on consideration of de-gassed fuel (Y_{N_2} = 1e-07) [217] and solubility limits of the same 13 different hydrocarbon fuels at a standard temperature and pressure condition [223]. Within the defined design space, 60 representative samples were identified using a variant of the Latin hypercube sampling (LHS) [224]. These samples are highlighted in Fig. E.1 and described in Table E.1 in Appendix E.1. The 60 samples were generated in two phases.[1] In the first phase, 36 CFD simulations were completed; we refer to this as DoE study "S36." In the second phase, the remaining 24 CFD simulations were completed, bringing the total number of available samples to 60; we refer to this as DoE study "S60." Thus, in Sec. 7.3.3, the first part of the results was obtained using DoE Study S36, and the second part of the results was obtained using DoE Study S60. Table 7.4 links the two studies to the relevant descriptions, results, and publications.

Table 7.3: Design space considered in this study. For each variable, the baseline, minimum and maximum values are indicated. Also, the fuel viscosity, μ_F, is specified at 323 K.

Design Variable	Baseline	Min.	Max.
Needle lift [μm]	15	15	400
μ_F [N-s/m^2]	9.418e-04	2.88e-04	1.51e-03
Y_{N_2} [-]	2e-05	1e-07	1e-03

[1]The data were generated in two phases strictly for time management reasons.

Table 7.4: Description of DoE studies and corresponding results subsections.

	Nb. of cases	Description	Results subsections	Publication
DoE study S36	36	Appendix E.2	Secs. 7.3.3.1, 7.3.3.2 and 7.3.3.3	[211]
DoE study S60	60	Appendix E.1	Secs. 7.3.3.4, 7.3.3.5 and 7.3.3.6	[212]

7.3.1.3 Computational Model Setup

A complete model description and model validation exercises can be found in Refs. [221, 225, 226, 222], but the salient details are summarized here. The cavitating flow within the injector is treated as a compressible, homogeneous, multiphase mixture comprised of four-components and two-phases, specifically liquid and vapor fuel, non-condensable gas, and ambient gas. In the single-fluid HMM (cf. Sec. 4.3.6), it is assumed the phases are strongly coupled and in local equilibrium, whereby all components are assumed to have the same pressure, temperature, and velocity within a given computational cell. The homogeneous mixture assumption is a valid one provided sufficient spatial resolution is employed.

The transient simulation methodology for the multiphase mixture is based on the solution of the filtered form of the Navier-Stokes equations for the LES technique, which allows the resolution of large-scale turbulent eddies while modeling the small-scale contributions to the turbulent flow (cf. Sec. 3.3). The governing equations solved in this problem include the continuity and momentum equations (written using index notation):

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0, \tag{7.1}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial \tau_{ij}^{\text{sgs}}}{\partial x_j} + f_{\text{tens},i}, \tag{7.2}$$

where ρ is the mixture density, p the pressure, u_i the mixture velocity, τ_{ij} the mixture viscous stress tensor, τ_{ij}^{sgs} the subgrid turbulent stress tensor, and $f_{\text{tens},i}$ the surface tensor force. Due to the adiabatic assumption used to model the flow development, solution of the mixture energy equation is omitted. Turbulence closure is achieved using the one-equation dynamic structure model [91] to characterize the effects of subgrid-scale motion. It is noted that the operators denoting the low-pass filtered and Favre-filtered quantities, commonly represented by the symbols $\overline{\square}$ and $\widetilde{\square}$, respectively (cf. Sec. 3.3), are not used here for the simplification of mathematical notation.

Within the mixture modeling approach, α is used to represent the total void fraction field. It can assume different values, as follows:

- $\alpha = 0$: the cell is filled with pure liquid
- $\alpha = 1$: the cell is filled with pure gas (i.e., fuel vapor, non-condensable gas, ambient gas, or a mixture of the three)

- $0 < \alpha < 0$: the cell is filled with both liquid and gas species

Using a pseudo-density concept, ρ is calculated using volume-weighted averaging

$$\rho = \alpha\rho_{\mathrm{g}} + (1-\alpha)\rho_{\mathrm{l}}, \tag{7.3}$$

where ρ_{g} and ρ_{l} are the total gas- and liquid-phase densities in the cell, respectively. The gas phase is described by the Redlich-Kwong (RK) EOS [227], while the liquid phase is treated as a compressible barotropic fluid. Using the vapor transport method, α is determined indirectly using a transport equation for each species mass fraction, Y_n,

$$\frac{\partial \rho Y_n}{\partial t} + \frac{\partial \rho Y_n u_j}{\partial x_j} = -\frac{\partial \mathscr{V}_{n,j}}{\partial x_j} - \frac{\partial \mathscr{V}^{\mathrm{sgs}}_{n,j}}{\partial x_j} + S_n, \tag{7.4}$$

where $\boldsymbol{\mathscr{V}}_n$ is the laminar diffusion velocity modeled using Fick's law, $\boldsymbol{\mathscr{V}}^{\mathrm{sgs}}_n$ is the subgrid scale turbulent diffusion velocity, and S_n is the source term for the nth species due to phase-change, modeled using the homogeneous relaxation model (HRM) [228] described below. α is then calculated using the mass fractions for the gas- and liquid-phase components

$$\alpha = \frac{m_{\mathrm{g}}/\rho_{\mathrm{g}}}{m_{\mathrm{g}}/\rho_{\mathrm{g}} + m_{\mathrm{l}}/\rho_{\mathrm{l}}}, \tag{7.5}$$

where m_{g} is the total gas mass fraction (i.e., fuel vapor, non-condensable gas, and ambient gas) and m_{l} is the total liquid mass fraction.

Although the presence of non-condensable gas is included in this model, no additional models are included to represent the adsorption and absorption of N_2. As a result, the mass transfer source term in the N_2 species transport equation is set to zero. The mass exchange between the liquid and vapor phases of the fuel species due to cavitation and condensation is modeled using HRM [228], where a first-order rate equation is assumed for the evolution of the instantaneous non-equilibrium mass fraction of fuel vapor, Y_{v}, towards its equilibrium value, $\overline{Y}_{\mathrm{v}}$, over a given time scale. The model is given by

$$\frac{dY_{\mathrm{v}}}{dt} = \frac{\overline{Y}_{\mathrm{v}} - Y_{\mathrm{v}}}{\theta}, \tag{7.6}$$

$$\theta = \theta_0 \alpha^{-0.54} \psi^{-1.76}, \tag{7.7}$$

where θ is the relaxation time scale, θ_0 is a coefficient set to 3.84×10^{-7}, and ψ is the nondimensional pressure ratio. The latter is given by

$$\psi = \frac{p_{\mathrm{sat}} - p}{p_{\mathrm{c}} - p_{\mathrm{sat}}}, \tag{7.8}$$

where p_{sat} and p_{c} are the saturation and critical pressures, respectively. The equilibrium vapor quantity $\overline{Y}_{\mathrm{v}}$ is determined by relating the mixture specific enthalpy, h, to the saturated liquid enthalpy, h_{f}, and heat of vaporization, L, at the mixture temperature, T

$$\overline{Y}_{\mathrm{v}} = \frac{h - h_{\mathrm{f}}(T)}{L(T)}. \tag{7.9}$$

The source term for the fuel vapor in Eq. (7.4) can be calculated as

$$S_{\mathrm{v}} = \frac{dY_{\mathrm{v}}}{dt}\rho(Y_{\mathrm{v}} + Y_{\mathrm{l}}). \tag{7.10}$$

Although the liquid-gas interface is not tracked in the mixture modeling approach, surface tension effects on the flow can be retained using the continuum surface force model developed by Brackbill et al. [229]. In this approach, surface tension is interpreted as a continuous, three-dimensional force in regions where strong gradients and curvature in α-field exist. To compute the volume force $f_{\mathrm{tens},i}$, curvature κ is calculated from local gradients in the pseudo-surface normal n_i

$$n_i = \frac{\frac{\partial \alpha}{\partial x_i}}{\left|\frac{\partial \alpha}{\partial x_i}\right|}, \tag{7.11}$$

where κ is defined in terms of the divergence of n_i

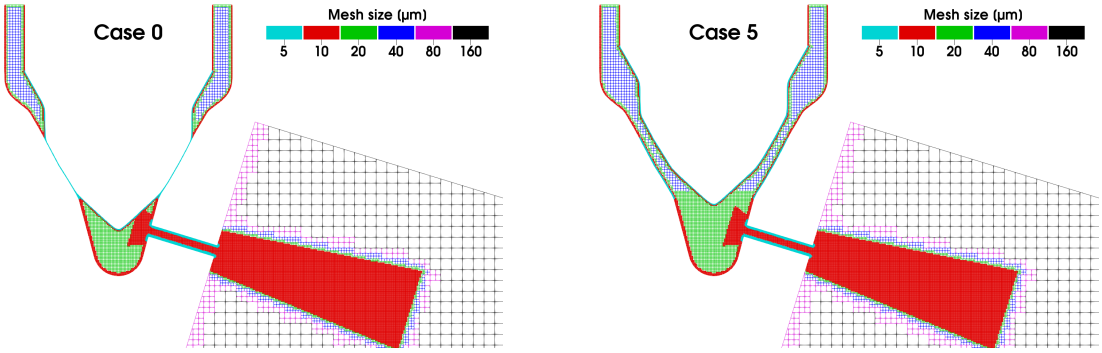
$$\kappa = -\frac{\partial n_i}{\partial x_i}. \tag{7.12}$$

$f_{\mathrm{tens},i}$ can then be obtained as follows

$$f_{\mathrm{tens},i} = \sigma\kappa\frac{\partial \widetilde{\alpha}}{\partial x_i}, \tag{7.13}$$

where $\widetilde{\alpha}$ is a modified void fraction field that approaches α in the limit of infinitesimally small interface thickness [229], and σ is the surface tension coefficient.

Spatial and temporal discretizations are achieved with second-order and first-order accuracy, respectively. A variable time-step algorithm is employed with a maximum Courant-Friedrichs-Lewy (CFL) number of 0.25, which results in time steps on the order of 1 ns. Unstructured meshes are generated using the fixed embedding strategy [98]. The mesh size distribution in the domain for the baseline configuration (i.e., Case 0) and Case 5 from the DoE study (see Table E.1) is shown in Fig. 7.3. A minimum mesh size of 5 μm is applied at the boundaries of the orifice body and in the needle seat area at very low needle lifts to improve the accuracy of the solution in such small gaps. For larger needle lifts such as Case 5, the mesh size criteria are relaxed in other regions of the domain (but kept unchanged in the orifice, sac and outer chamber) to reduce the overall mesh count. The total cell count is about 5.4 and 6.6 million for Cases 0 and 5, respectively. A grid-convergence study is performed to minimize numerical uncertainty (see Sec. 7.3.3.1). Each case takes between 1,400 and 2,000 CPU-hours per 10 μs of simulated injection time, depending on the cell count. A total of $n_t = 81$ snapshots is acquired starting from $t = 20$ μs, after the flow reaches its quasi-steady state, and spanning a period of 20 μs. The predicted output data require about 120 GB of storage per simulation.



(a) Case 0 (b) Case 5

Figure 7.3: Visualization of the fixed embedding strategy, shown at the centerplane of the domain for Cases 0 and 5. The corresponding needle vertical lift is 15 μm and 386.95 μm, respectively.

7.3.2 Emulator Framework

The emulator framework comprises two levels of neural networks to approximate the spatiotemporal exit conditions of the A-M1 injector. In the first level, an autoencoder network is used to perform a spatial compression of the sequence of high-dimensional snapshots into a sequence of latent vectors. Following the autoencoder, a regressor is used in the second level to learn the relationship between the design parameters and latent space. The emulation framework is implemented in `Python 3.7` using `TensorFlow` (cf. Sec. 2.3.9).

7.3.2.1 Autoencoder

An autoencoder is used to compress the representation of the flowfield data generated from the internal flow simulations. It consists of two parts: an encoder $\mathcal{F}_{enc}$ and a decoder $\mathcal{F}_{dec}$, as shown in Fig. 7.4. The encoder is used to map the high-dimensional flowfield into a low-dimensional latent space. The decoder is then used to expand the dimensionality of the latent space back to that of the original data, thereby reconstructing the dataset. Since the mesh in the injector geometry is unstructured (cf. Sec. 7.3.1.3), a fully-connected autoencoder architecture[2] (cf. Sec. 2.2.5.1) is employed in the current chapter rather than a convolutional autoencoder.

As a reminder, the nonlinearity of the activation functions has been noted as a key attribute of autoencoders in extracting nonlinear features with low reconstruction errors (cf. Secs. 2.2.5 and 2.2.7). In this study, ReLU is applied as the activation function to create the nonlinear mappings between the inputs and outputs. Mathematically, the forward propagation procedure in the autoencoder is given by Eq. (2.9).

The autoencoder is trained using the Adam optimizer (cf. Algorithm 1) in order to minimize J_{AE}, the loss function. J_{AE} is defined as the regularized MSE between

[2]Note that in this chapter we will mostly refer to FCAE as AE.

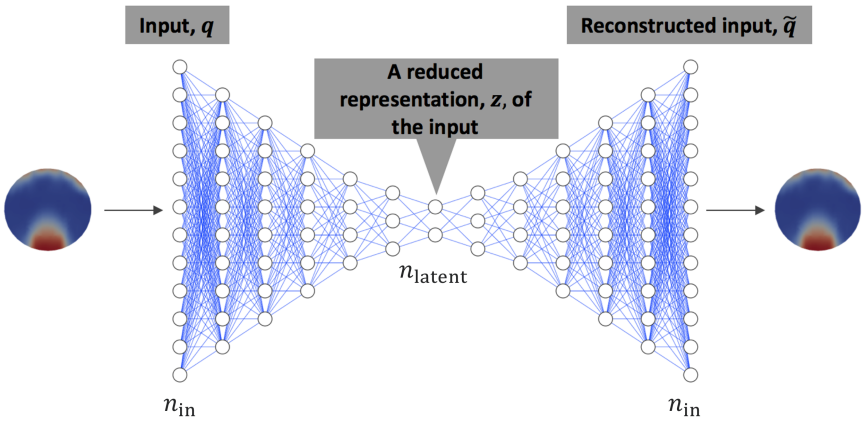


Figure 7.4: Schematic representation of the autoencoder.

the input and the output:

$$J_{\text{AE}}(\boldsymbol{\theta}) = ||\boldsymbol{q} - \widetilde{\boldsymbol{q}}||_{\text{MSE}} + \lambda_{\text{reg}} \sum_k ||\boldsymbol{W}_{\text{AE}}^{(k)}||_F^2, \tag{7.14}$$

where $\boldsymbol{q} \in \mathbb{R}^{n_{\text{in}}}$ and $\widetilde{\boldsymbol{q}} \in \mathbb{R}^{n_{\text{in}}}$ denote the encoder input and decoder output, respectively, with n_{in} being the dimension of the input layer. Here, $n_{\text{in}} = 4,214$ corresponds to the number of grid points in the subdomain of interest, as shown in Fig. 7.2. Also, $\boldsymbol{\theta} = \{\boldsymbol{W}_{\text{AE}}^{(1)}, \boldsymbol{b}_{\text{AE}}^{(1)}, \boldsymbol{W}_{\text{AE}}^{(2)}, \boldsymbol{b}_{\text{AE}}^{(2)}, \dots\}$ contains all of the parameters of the autoencoder (weight matrices and bias vectors). λ_{reg} is the L_2-regularization hyperparameter, which is used to prevent overfitting.

7.3.2.2 Regression Model

A regressor, $\mathcal{R}$, is constructed to learn the relationship between the set of selected design parameters, $\boldsymbol{p}$, and the temporal evolution of low-dimensional flowfields generated by the autoencoder, $\boldsymbol{z}(t)$, as schematically represented in Fig. 7.5. In the current study, $\boldsymbol{p}$ consists of three components (i.e., $n_{\boldsymbol{p}} = 3$), namely needle lift, liquid fuel dynamic viscosity, and amount of non-condensable gas in the fuel. Formally, the regressor $\mathcal{R}$ can be written as:

$$\mathcal{R} : (\boldsymbol{p}, t) \in \mathbb{R}^{n_{\boldsymbol{p}}} \times \mathbb{R} \mapsto \boldsymbol{z} \in \mathbb{R}^{n_{\text{latent}}}. \tag{7.15}$$

In this study, $\mathcal{R}$ is modeled as a DFNN (cf. Sec. 2.2.3). The regularized MSE is used as the loss function, $J_{\mathcal{R}}$, to train the model, i.e.,

$$J_{\mathcal{R}}(\boldsymbol{\phi}) = ||\boldsymbol{z}_{\text{true}} - \boldsymbol{z}_{\text{pred}}||_{\text{MSE}} + \zeta_{\text{reg}} \sum_k ||\boldsymbol{W}_{\mathcal{R}}^{(k)}||_F^2, \tag{7.16}$$

where $\boldsymbol{z}_{\text{true}}$ is the true latent vector, $\boldsymbol{z}_{\text{pred}}$ is the latent vector predicted by $\mathcal{R}$, and $\boldsymbol{\phi} = \{\boldsymbol{W}_{\mathcal{R}}^{(1)}, \boldsymbol{b}_{\mathcal{R}}^{(1)}, \boldsymbol{W}_{\mathcal{R}}^{(2)}, \boldsymbol{b}_{\mathcal{R}}^{(2)}, \dots\}$ contains the parameters of the regression model. ζ_{reg} is the L_2-regularization hyperparameter.

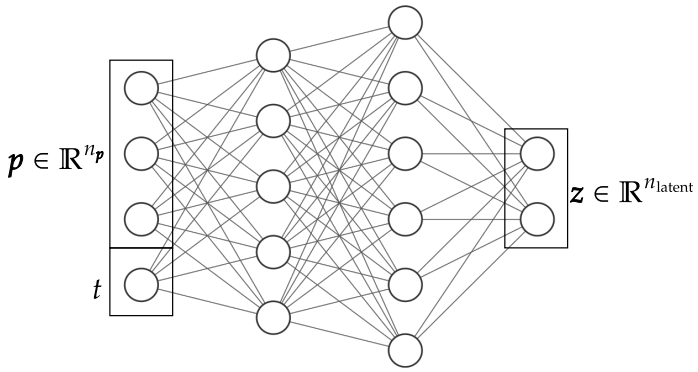


Figure 7.5: Schematic representation of the regression model.

7.3.3 Results and Discussion

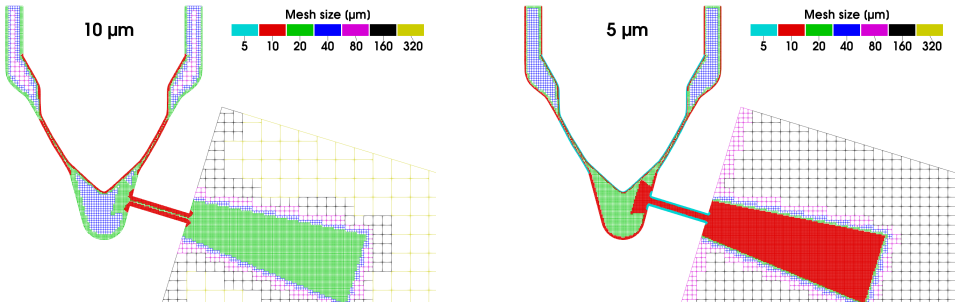
7.3.3.1 Mesh Analysis

Four minimum mesh sizes inside the orifice (20, 10, 5, and 3 μm) were evaluated on one of the CFD cases, which is characterized by a needle lift approximately half the way between maximum and minimum lifts. To this purpose, Case 13 was selected to evaluate the influence of minimum grid size on the CFD predictions. Several grid resolutions were evaluated by imposing different levels of refinement[3] on base mesh sizes of 640, 320, 160, and 96 μm, respectively. The total cell count is about 0.3, 1.3, 6.5 and 22.4 million, respectively. Figure 7.6 shows an example of the meshes obtained with two strategies, using 10 μm (left) and 5 μm (right) as the minimum mesh size inside the orifice. Figure 7.7 reports the predicted mass flow rates. The results indicate that a minimum mesh size of 5 μm was sufficient to ensure reliable predictions of mass flow rate. Similar results also hold for other simulated design points that are not shown here for the sake of brevity. In consideration of computational accuracy and efficiency, the intermediate grid with a minimum mesh size of 5 μm was thus selected for the remainder of this study.

7.3.3.2 Internal Flow Dynamics and Sensitivity to the Input Parameters

In this section, the flow structures and cavitation characteristics of the A-M1 injector are presented. Also, the effects of selected design parameters, namely needle

[3]The CFD code `CONVERGE` [98] uses the input parameter grid_scale (which denotes the level of refinement) to change the base grid according to the following formula: scaled_grid = base_grid/$2^{\text{grid\_scale}}$. For example, for a base grid of 320 μm and a refinement level of 5 at the boundaries of the wall of the orifice body, the corresponding scaled grid is 10 μm, and so forth.



(a) (b)

Figure 7.6: An example of mesh generated with two strategies on Case 13.

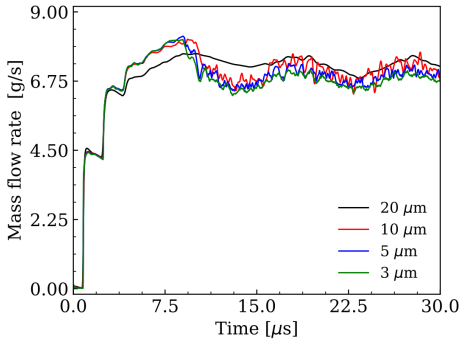


Figure 7.7: Predicted mass flow rates for different mesh strategies on Case 13.

lift, liquid fuel dynamic viscosity, and level of non-condensable gas, on the predicted flowfield at the injector exit is assessed. This will be done using results from the baseline simulation and selected cases from the DoE study S36.

Figure 7.8 shows the predicted mass flow rates from the baseline simulation and selected cases from the DoE study S36; results from Cases 0, 5, 10, 15, 20, 25 and 30 are reported here. A sharp increase in the predicted mass flow rate is observed in all of the cases during the initial stage. This is likely due to the initialization of the sac and orifice volumes, which results in a higher prediction of fuel mass delivery during the early injection transient compared to the quasi-steady-state value, which is reached after approximately 15 μs. This behavior is consistent with previous work on a similar injector [221]. For the baseline simulation, the mass flow rate is about 5.76 g/s whereas for the other cases with higher needle lifts, the mass flow rate is larger at approximately 6.5-7.0 g/s.

The flow profile at the injector exit during the quasi-steady state period ($20-40$ μs) is then characterized. Figure 7.9(a) shows the instantaneous flow streamlines and the total gas volume fraction field inside the orifice, as well as the composition of the gas phase at the exit of the orifice for Case 0 at $t = 25$ μs. The streamlines, colored by velocity magnitude, highlight the acceleration of the flow from the sac volume into the orifice. Due to losses associated with the area constriction at the orifice inlet, the local pressure decreases, as can be seen from the axial distribution along the bottom

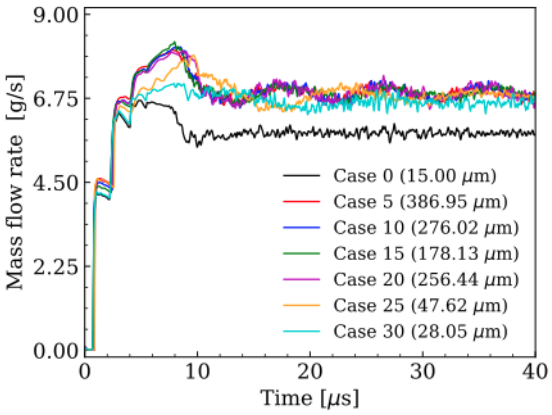
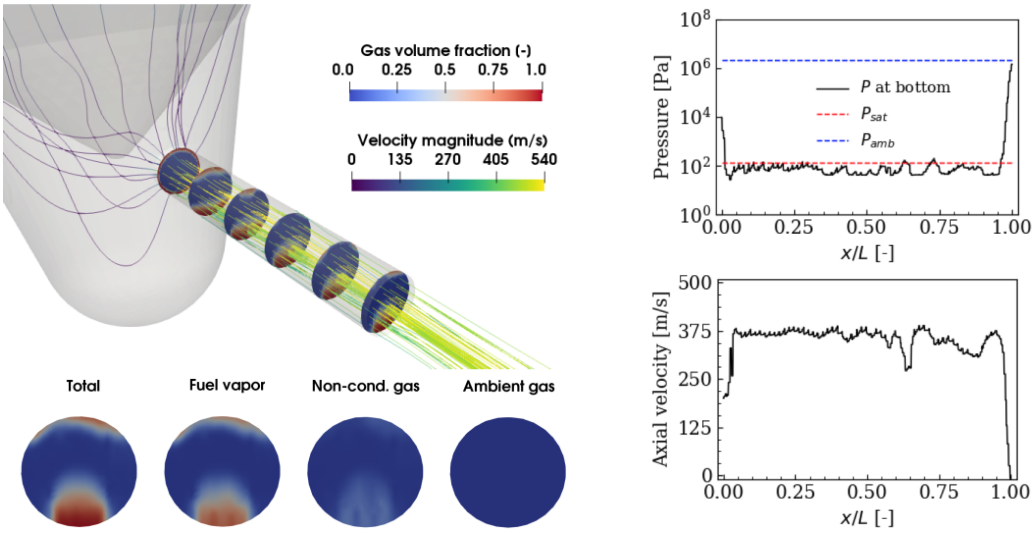


Figure 7.8: Predicted mass flow rates from baseline simulation (i.e., Case 0) and selected cases from the DoE study S36. The needle lift for each case is also indicated.

of the orifice in Fig. 7.9(b). When local pressure values lower than the saturation pressure of the fuel ($p_{\text{sat}} = 133.3$ Pa) are established, fuel vapor generation due to cavitation is predicted. Pressure recovery is then observed toward the exit of the orifice where the local pressure matches the ambient one ($p_{\text{amb}} = 20$ bar). For this case, the gas-phase composition is predicted to be composed of both fuel vapor and non-condensable gas (N_2) at the orifice exit.

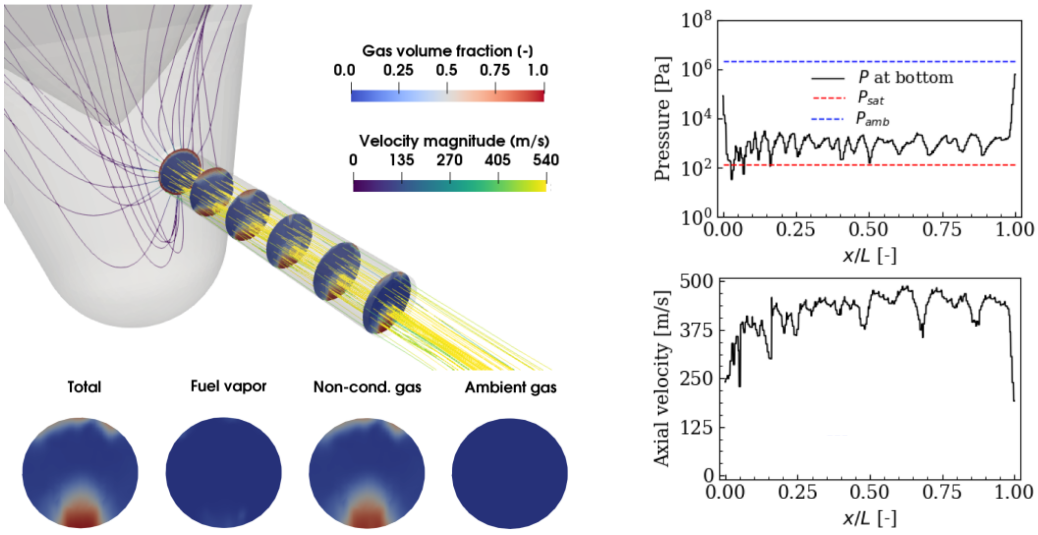
Figure 7.10 shows the same analysis as Fig. 7.9 but for Case 13, which is characterized by a larger needle lift (152.034 μm), higher fuel viscosity ($\mu_F = 1.324 \times 10^{-3}$ N-s/m^2) and higher concentration of non-condensable gas ($Y_{\text{N}_2} = 3.729 \times 10^{-4}$). Compared to the Case 0 predictions, the flow velocities within the orifice from Case 13 are found to be larger because of the higher needle lift. Additionally, the gas phase structure at the orifice exit in Case 13 is mostly composed of non-condensable gas. This phenomenon, commonly referred to as *pseudo-cavitation* [230, 231], is due to the local expansion of non-condensable gas. This is supported by Fig. 7.10(b) which shows that the local pressure at the bottom of the orifice exceeds the fuel saturation pressure along most of the orifice length. The relative decrease in cavitation formation with increased fuel viscosity and non-condensable gas concentration are consistent with previous findings from the literature [222, 232, 230]. In general, increased liquid viscosity causes a reduction of the magnitude of the velocity gradients near the wall, which results in lower local viscous losses and more contained pressure drops, therefore resulting in reduced cavitation intensity. Additionally, although increased levels of non-condensable gas did not sensibly change the predicted total void fraction distribution (i.e., the sum of fuel vapor and N_2), Battistoni et al. [230] found that increased non-condensable gas concentrations tend to suppress cavitation and formation of fuel vapor due to the competing effect associated with the expansion of these two species.

Figure 7.11 shows the composition of the gas phase at the exit of the orifice at $t = 25$ μs for selected cases from the DoE study S36. In particular, the total gas volume fraction distribution is decomposed into the contributions from fuel vapor,



(a) (b)

Figure 7.9: (a) 3D visualization of total gas volume fraction field and streamlines of velocity magnitude inside the orifice (top). Also shown is the composition of the gas phase at the exit of the orifice (bottom); (b) Pressure and axial velocity distributions along the axial direction at the bottom of the orifice. All results shown here are obtained at $t = 25$ μs from Case 0.



(a) (b)

Figure 7.10: (a) 3D visualization of total gas volume fraction field and streamlines of velocity magnitude inside the orifice (top). Also shown is the composition of the gas phase at the exit of the orifice (bottom); (b) Pressure and axial velocity distributions along the axial direction at the bottom of the orifice. All results shown here are obtained at $t = 25$ μs from Case 13.

non-condensable gas, and ambient gas to differentiate cavitation from the expansion of non-condensable gas. For all of the cases, no ambient gas is observed at the orifice exit plane at this time instant, indicating that gas ingestion is not predicted. Asymmetric flow features caused by the sharp inlet radius of curvature [221, 233] as well as differing intensities of cavitation and non-condensable gas expansion are observed across all of the cases. For example, Case 2, which represents a de-gassed condition, features the highest fuel vapor volume fraction. However, because all design parameters are simultaneously varied across all of the cases shown in Fig. 7.11, it is challenging to draw definitive conclusions about the relative influence of each design parameter on the orifice exit flowfield.

To characterize the impact of the selected design parameters on the flowfield characteristics shown in Fig. 7.11, variance-based sensitivity analysis [234] was performed. In this analysis, the instantaneous total gas-phase volume fraction and fuel-vapor volume fraction values were calculated at the orifice exit plane for each CFD case and selected as the response metrics. The total effect indices were then calculated, which quantify the effect of a given design parameter, and its interactions with other design parameters, on the variance observed in the response. A comparison of the total effect indices is illustrated in Fig. 7.12. The sensitivity analysis reveals that the fuel vapor volume fraction at the orifice exit is most highly influenced by the level of non-condensable gas in the fuel, where higher levels of non-condensable gas result in decreased levels of fuel vapor volume. In contrast, the needle lift position was found to be the dominant parameter for the total vapor volume fraction, where lower needle lifts resulted in higher levels of vapor generation that persisted until the orifice exit. For both fuel vapor and total gas volume fractions, the liquid fuel dynamic viscosity was found to be the least influential parameter. Although previous work in the literature has noted a strong negative correlation between liquid fuel dynamic viscosity and cavitation [222, 232, 235], it is important to note that these studies focused on cavitation inception. In contrast, the findings from the sensitivity analysis highlight that the persistence of gas-phase structures in the orifice is most strongly related to level of non-condensable gas in the fuel and needle lift position.

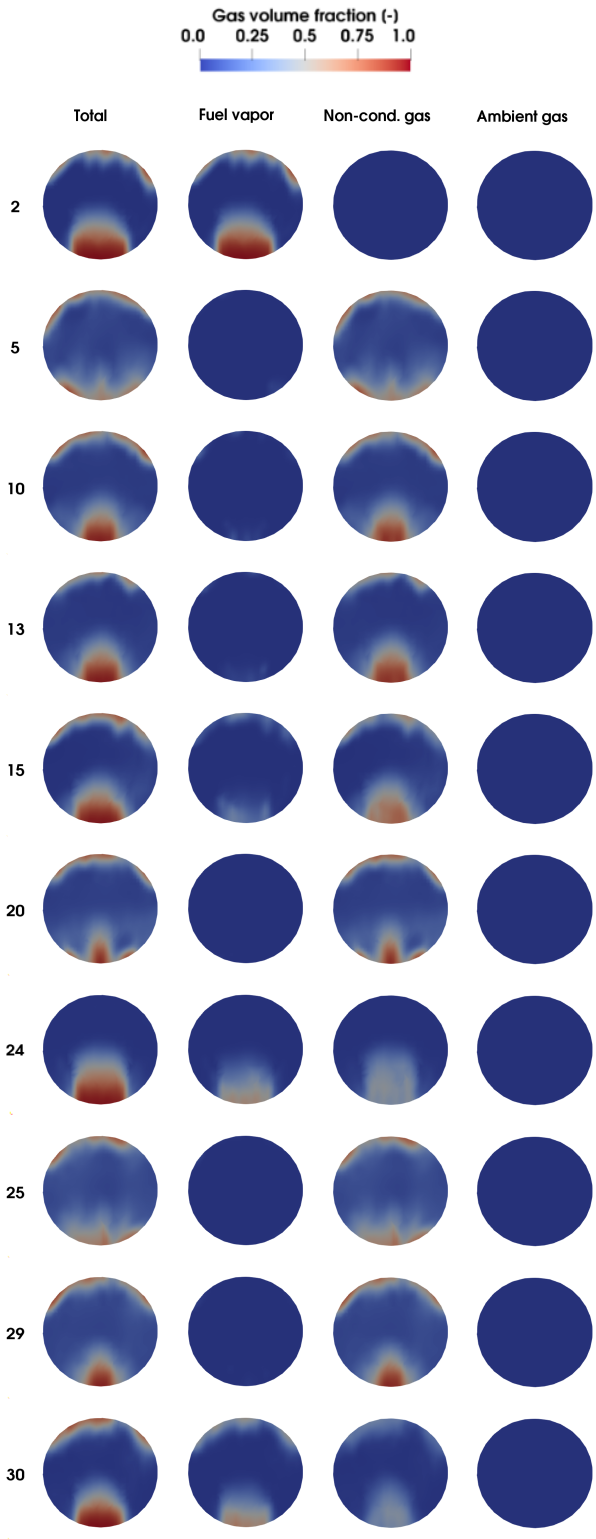


Figure 7.11: Instantaneous contours of the composition of the gas phase at the exit of the orifice at $t = 25$ μs for Cases $2, 5, 10, 13, 15, 20, 24, 25, 29, 30$.

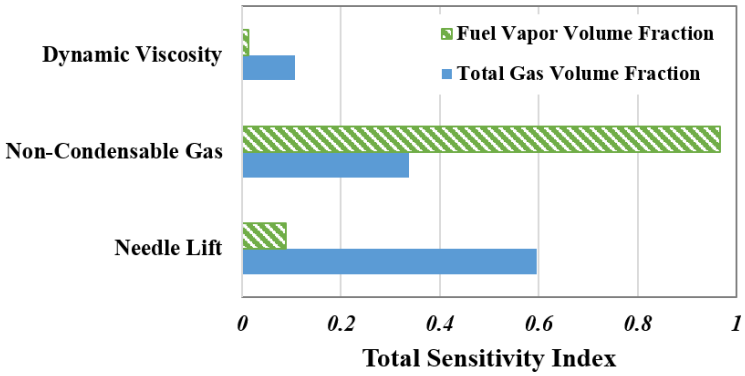


Figure 7.12: The sensitivities of total gas volume fraction and fuel vapor volume fraction at the orifice exit plane to changes in the three design parameters are quantified using the total sensitivity index.

7.3.3.3 Comparison between POD and Autoencoders

As previously mentioned, ROM is a critical step in developing accurate and efficient emulators for datasets featuring high dimensionality. Using the predicted flowfields across the evaluated design space, two ROM techniques, namely POD and autoencoders, are evaluated and compared in their ability to accurately and efficiently represent the flowfield in a reduced dimensional space. These findings are discussed in this section. We note that, here, the analysis is performed using the cases from the DoE study S36. We also note that, exclusively in this section, the autoencoders are implemented using `TensorFlow 1.14` (cf. Sec. 2.3.9).

POD Results

The POD characteristics are first investigated to assess the ability of the POD modes to represent the original dataset, as well as to identify possible limitations of this technique. As previously mentioned, the subdomain of interest for ROM is located near the exit of the orifice (see Fig. 7.2). Figure 7.13 shows the percentage of the captured energy obtained from POD for Cases 5, 10, 15, 20, 25 and 30. As previously noted, the captured modal energy from POD was calculated using Eq. (A.13). While, by definition, the whole energy is captured when the complete set of modes (81 here) is included, the growth rates of the energy distributions are found to be different among the cases. To quantify these differences, the number of POD modes required to recover 99% of the modal energy is identified; this variable is denoted by $n_{r,99\%}$. Table 7.5 shows that $n_{r,99\%}$ is different among all of the cases. For instance, Case 5 requires 47 modes to recover 99% of the energy while Case 30 requires 31 modes. This suggests the occurrence of different flow dynamics among all of the cases. In general, it is observed that cases with more "energetic" flows and finer scale features require a higher number of modes for accurate reconstruction. For example, Case 5 exhibits finer flow structures compared to the other cases (see Fig. 7.11) and requires the

largest number of modes to capture 99% of the modal energy, as noted in Table 7.5. Additionally, it is observed that conditions characterized by similar predicted flow structures at the injector exit (e.g., Cases 10 and 15) require similar $n_{r,99\%}$ values.

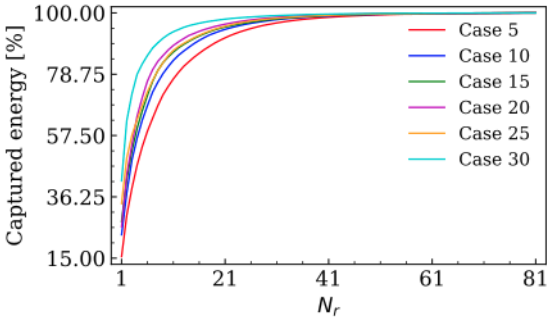


Figure 7.13: Variation of the captured modal energy, in percentage, as a function of the number of POD modes for the total gas volume fraction field. Results are shown for selected cases from the DoE study S36.

The ability of POD to reproduce the original CFD solution is visualized in Fig. 7.14, where the POD-based reconstructions of total gas volume fraction fields for different numbers of modes are shown at $t = 25$ μs for Cases 10 and 15. In each case, results using between 1-8 modes and $n_{r,99\%}$ modes are reported. It is important to note that $n_{r,99\%}$ is equal to 42 in Case 10 and to 40 in Case 15. In general, the accuracy of the POD reconstruction is observed to increase with the number of retained modes. In addition, the relatively large number of modes needed to capture the flow structures in detail is an indication of the slow convergence of POD representations toward the actual solution. This result highlights the challenge of applying POD as an efficient ROM technique for nonlinear problems.

Table 7.5: Number of POD modes, $n_{r,99\%}$, required to capture 99% of the modal energy for selected cases from the DoE study S36.

Case	$n_{r,99\%}$
5	47
10	42
15	40
20	37
25	39
30	31

Custom-AE Results

Next, the ability of autoencoders to perform efficient model reduction is evaluated and compared against that of POD. The training dataset was obtained by extracting

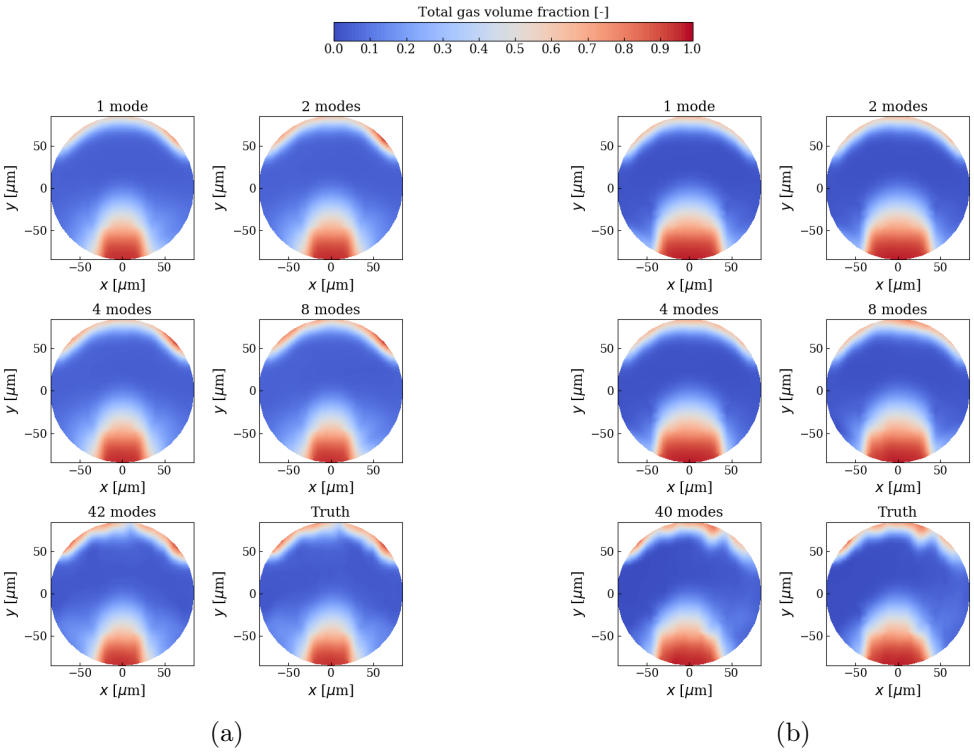


Figure 7.14: POD-based reconstruction of total gas volume fraction field for different numbers of modes. Results are shown at $t = 25$ μs from: (a) Case 10, (b) Case 15.

the total gas volume fraction fields at the exit of the orifice (see Fig. 7.2). Similar to the methodology for POD, a total of 81 snapshots was extracted from each CFD simulation, sampled at regular time intervals between $t = 20$ and 40 μs of simulated injection time. Each snapshot was flattened to a one-dimensional vector of size n_{in} (here, $n_{\text{in}} = 4,214$ corresponds to the number of grid points at the orifice exit) and provided as input to the autoencoder. It is worth mentioning that the design parameters were not explicitly provided as input to the model. Each weight matrix was randomly initialized using the He normal initializer (cf. Sec. 2.3.5.2). Each bias vector was initialized to 0. The autoencoder was trained on an NVIDIA Tesla K80 GPU (cf. Sec. 2.3.7). The optimized network architecture and hyperparameters were obtained by means of manual tuning, and the autoencoder with the minimum error was selected as the final model. In this part of the study, two autoencoder-based ROMs were developed. A customized autoencoder (shortened to "Custom-AE" for the sake of brevity) was formulated for each CFD case by training on the 81 snapshots from the simulated injection period. The training of each Custom-AE results in specialized set of autoencoder weights and biases. The results from the Custom-AE allow for direct comparison with POD for each case. For the second ROM, a universal autoencoder (shortened to "Univ-AE") was formulated by training the model with snapshots from various CFD cases throughout the three-dimensional design space. The merits of the two autoencoder-based ROMs will be discussed hereafter.

For each Custom-AE, 81 snapshots from the simulated injection period were used

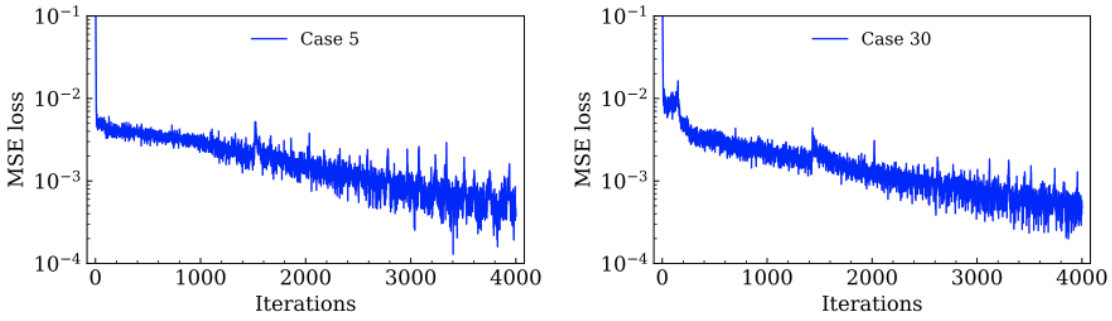
as training data, and the neural network that yielded the lowest training loss while balancing the size of the latent space vector was selected as the final model. Through manual exploration, the selected autoencoder configuration contains 8 hidden layers (4 encoder layers and 4 decoder layers) with the following hyperparameters: a batch size of 10, a learning rate of 1×10^{-3} for the Adam optimizer, and a latent vector of size 2, yielding a compression ratio (i.e., ratio of the size of the input layer to that of the code) of 2,106.5. L_2 regularization was not used in this case. The number of neurons per hidden layer in the encoder and decoder is $\boldsymbol{\mathcal{H}}_{\text{enc}} = \begin{bmatrix} 200 & 50 & 25 & 10 \end{bmatrix}^T$ and $\boldsymbol{\mathcal{H}}_{\text{dec}} = \begin{bmatrix} 10 & 25 & 50 & 200 \end{bmatrix}^T$, respectively, where $\mathcal{H}_j$ is the number of neurons in the jth hidden layer. This results in a total of 1,713,025 trainable parameters. The hyperparameters used for this configuration are summarized in Table 7.6. The variation of MSE loss with the number of iterations[4] is shown in Fig. 7.15, where representative results are reported using the Custom-AEs for Cases 5 and 30. MSE values of 6.830×10^{-4} and 4.805×10^{-4} are achieved for the reconstructed fields for Cases 5 and 30, respectively. The low MSE values indicate that the Custom-AEs accurately represent the predicted flowfield at the exit of the injector orifice. The training time of each of the Custom-AEs on the GPU is about 3 min.

Table 7.6: Hyperparameters used for the Custom-AEs.

Parameter	Value
Number of epochs	4,000
Batch size	10
Number of hidden layers	8
$\boldsymbol{\mathcal{H}}_{\text{enc}}$	$\begin{bmatrix} 200 & 50 & 25 & 10 \end{bmatrix}^T$
$\boldsymbol{\mathcal{H}}_{\text{dec}}$	$\begin{bmatrix} 10 & 25 & 50 & 200 \end{bmatrix}^T$
n_{latent}	2
Optimizer for network	Adam
Learning rate	1×10^{-3}
L_2 regularization	0
Activation function	ReLU

To further illustrate the performance of the Custom-AEs, the reconstructed total gas volume fraction fields are compared with those from POD. For POD, the reconstructed field is shown by using 2 modes and 8 modes. As shown in Fig. 7.16 for Case 5 at $t = 25$ μs, the reconstructed fields from both ROM methods show reasonable agreement with the CFD solution. The L_2 error is also reported for each ROM technique to visualize the local errors between the reconstructed and CFD-predicted flowfields. Overall, the flowfield reconstructed by the Custom-AE yielded the lowest L_2 errors and provided a more accurate reconstruction than POD. Similar results were found for the other snapshots as well. To assess the overall behavior, Table 7.7

[4]Note the difference between the number of epochs and iterations. An epoch is one forward pass and one backward pass of all of the training examples whereas each iteration is on one batch. As an example, if 1,000 training examples are considered, and the batch size is 50, then it will take 20 iterations to complete 1 epoch.



(a) (b)

Figure 7.15: Variation of the training MSE loss with the number of iterations for Custom-AE. Results are shown for: (a) Case 5, and (b) Case 30.

reports the MSE values for selected cases, calculated across the 81 snapshots for each CFD case. It is clear from this table that the Custom-AE approach provides better reconstruction accuracy than POD with 2 modes. Furthermore, despite a latent space of size 2, the Custom-AE outperforms the POD with 8 modes as well. This comparison highlights the merit of using well-trained autoencoders as a ROM technique. However, we note that to choose between ROMs to apply on future data, we should collect more snapshots and compare the accuracy of the ROMs on this held out data that was not used in the modeling process. Therefore, next, we split the data into training, validation, and test datasets using a variant of the hold-out method (cf. Sec. 2.3.4.1).

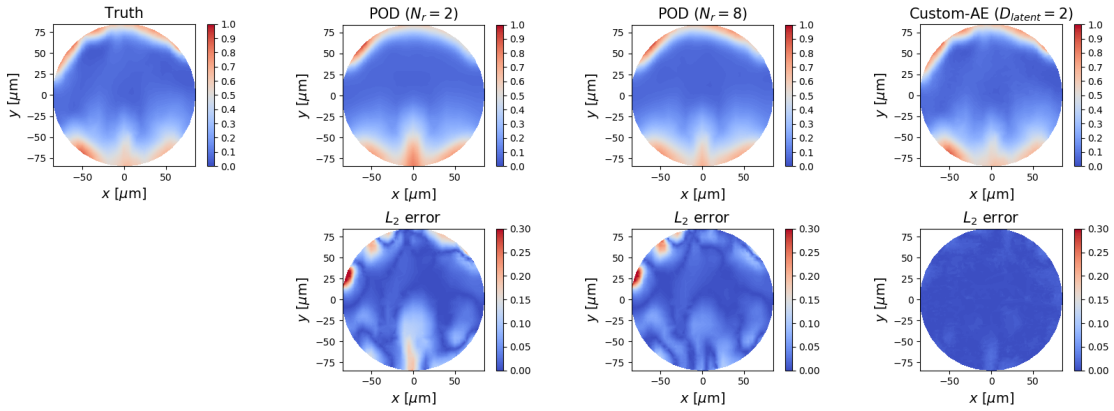


Figure 7.16: Reconstruction of total gas volume fraction field (top) and distribution of L_2-norm error (bottom) in POD and Custom-AE at $t = 25$ μs for Case 5. The MSEs at this time instant for 2-mode POD, 8-mode POD and Custom-AE are 4.890×10^{-3}, 3.092×10^{-3} and 5.688×10^{-5}, respectively.

Table 7.7: MSE values for POD, Custom-AE and Univ-AE. Results are shown over the 81 snapshots from Cases 5, 15 and 30. For POD, results are shown using 2 modes and 8 modes.

Case	POD ($n_r = 2$)	POD ($n_r = 8$)	Custom-AE	Univ-AE
5	3.719×10^{-3}	1.705×10^{-3}	6.830×10^{-4}	1.996×10^{-3}
15	3.706×10^{-3}	1.280×10^{-3}	4.603×10^{-4}	1.689×10^{-3}
30	3.357×10^{-3}	9.324×10^{-4}	4.805×10^{-4}	1.266×10^{-3}

Univ-AE Results

The previous paragraph demonstrated the ability of Custom-AEs to more accurately compress the instantaneous and ensemble-averaged flowfields than POD. However, similar to POD, the Custom-AE was applied to each CFD case separately, which resulted in different reconstruction functions (i.e., decoder weightings) across the selected design parameters. In the context of emulation, where changes in design parameters are related to a predicted flowfield, the optimal ROM method is one that provides a consistent or universal reconstruction function throughout the design space. Therefore, the Univ-AE was developed to extend the applicability of autoencoders as an efficient ROM within an emulation framework.

For the Univ-AE, all of the snapshots from Cases 1-30 and 31a–36a were used to formulate and test the autoencoder. The snapshots were divided into 4 sets, as shown in Table E.4. First, the snapshots from Cases 1-30 were collected and randomly split into three separate groups, where 70% of the snapshots was used for training, 20% for validation, and 10% for testing. These groups are referred to as training set, validation set, and test set $\mathbb{T}_1$, respectively. Then, a fourth group was considered, referred to as test set $\mathbb{T}_2$, using all of the snapshots from Cases 31a-36a.

L_2 regularization (cf. Sec. 2.3.3.1) and an early stopping criterion (cf. Sec. 2.3.3.2) were employed to prevent overfitting. The early stopping criterion was set at 5,000 iterations while the maximum number of epochs was set to 2,200. This means that if training exceeded 5,000 iterations without an improvement in the accuracy, the session would be terminated, and a different set of hyperparameters would be explored. The best model which provided the lowest validation loss in the training process was selected and used for model evaluation.

The selected Univ-AE from the manual search contains 10 hidden layers (5 encoder layers and 5 decoder layers) with the following hyperparameters: a batch size of 25, a learning rate of 2×10^{-5} for the Adam optimizer, a L_2 regularizer of 7×10^{-6}, and a latent vector of size 8, yielding a compression ratio of 526.625. The number of neurons per hidden layer in the encoder and decoder is $\boldsymbol{\mathcal{H}}_{\text{enc}} = \begin{bmatrix}400 & 200 & 100 & 50 & 25\end{bmatrix}^T$ and $\boldsymbol{\mathcal{H}}_{\text{dec}} = \begin{bmatrix}25 & 50 & 100 & 200 & 400\end{bmatrix}^T$, respectively, which results in 3,589,071 trainable parameters. The dataset sizes and hyperparameters used for the Univ-AE are summarized in Tables E.4 and 7.8, respectively. The variation of MSE loss with the number of iterations is shown in Fig. 7.17. The curve shows good convergence and

no evidence of overfitting is observed. MSE values of 1.3×10^{-3} and 1.528×10^{-3} are achieved for the reconstructed fields on training data and validation data, respectively. The training time of the Univ-AE on the GPU is about 20 min, i.e., roughly 7 times slower than each of the Custom-AEs. This is expected since the Univ-AE consists of a larger neural network and contains more trainable parameters than the customized autoencoders. However, this representative time is almost two orders of magnitude smaller than the time required to generate the training data using the CFD simulations.

Table 7.8: Hyperparameters used for the Univ-AE.

Parameter	Value
Number of epochs	2,200
Batch size	25
Number of hidden layers	10
$\mathscr{H}_{\text{enc}}$	$[400 \quad 200 \quad 100 \quad 50 \quad 25]^T$
$\mathscr{H}_{\text{dec}}$	$[25 \quad 50 \quad 100 \quad 200 \quad 400]^T$
n_{latent}	8
Optimizer for network	Adam
Learning rate	2×10^{-5}
L_2 regularization	7×10^{-6}
Activation function	ReLU

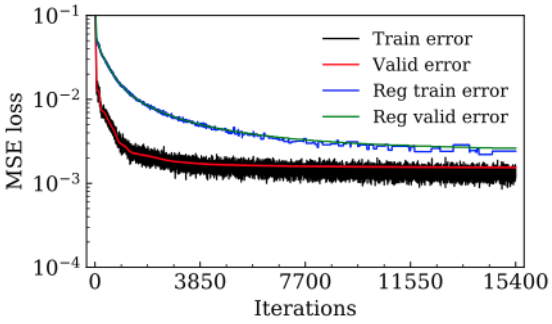


Figure 7.17: Variation of the MSE loss with the number of iterations for Univ-AE. The non-regularized training error (Train error), non-regularized validation error (Valid error), regularized training error (Reg train error) and regularized validation error (Reg valid error) are indicated.

A first validation of the Univ-AE is performed through quantitative assessment on the testing set $\mathbb{T}_1$, which contains snapshots that the autoencoder has not seen previously. A low MSE value of 1.579×10^{-3} is observed on this dataset, indicating that the autoencoder performs in the desired manner and doesn't overfit. Additionally, this result indicates that the Univ-AE performs well in terms of interpolation and generalizability on new snapshots. To better address the generalizability of the model and show that flowfields at unknown operating points can still be reproduced

by Univ-AE, the autoencoder is then assessed on the testing set $\mathbb{T}_2$, containing snapshots from new values of the design parameters not explored during training. Low MSE values (i.e., smaller than 2.6×10^{-3}) are observed on the cases from the second test set, as shown in Table 7.9, indicating that the model can also generalize well to new operating conditions.

Table 7.9: MSE values for Univ-AE on the testing set $\mathbb{T}_2$. Results are shown over the 81 snapshots from Cases 31a-36a.

Case	Univ-AE
31a	9.020×10^{-4}
32a	2.506×10^{-3}
33a	8.840×10^{-4}
34a	1.364×10^{-3}
35a	1.537×10^{-3}
36a	1.357×10^{-3}

Figures 7.18 and 7.19 show the temporal evolution of the flowfields computed by CFD and those reconstructed by the Univ-AE for Cases 31a and 32a, respectively. In these figures, instantaneous distributions of the total void fraction are shown between $t = 25$ and 40 μs. It is clearly observed that the reconstructed flowfields are in good agreement with the original data at all of the time steps.

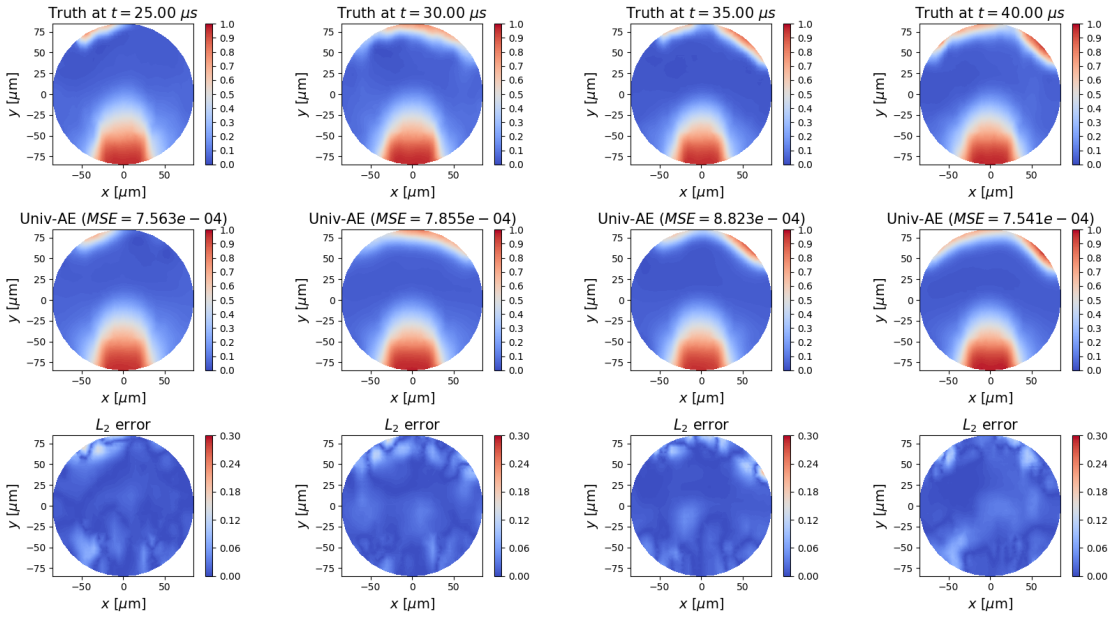


Figure 7.18: Temporal evolution of total gas volume fraction field for Case 31a from test set $\mathbb{T}_2$. Flowfields computed by CFD (referred to as "Truth") and those reconstructed by Univ-AE are compared. The MSE value and distribution of L_2-norm error are also indicated for each snapshot.

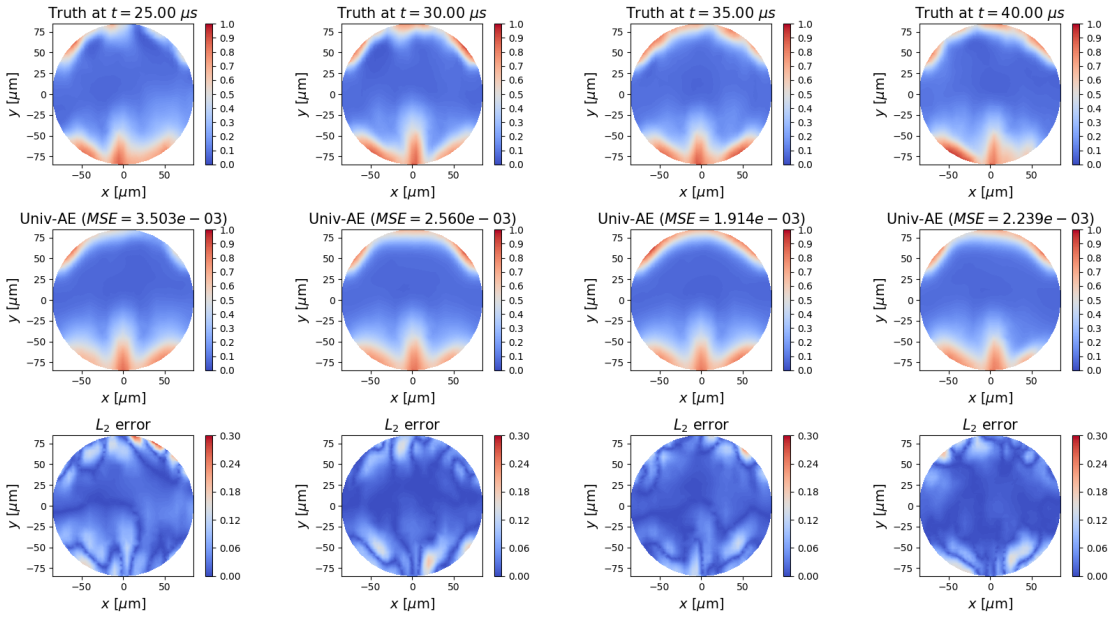


Figure 7.19: Temporal evolution of total gas volume fraction field for Case 32a from test set $\mathbb{T}_2$. Flowfields computed by CFD (referred to as "Truth") and those reconstructed by Univ-AE are compared. The MSE value and distribution of L_2-norm error are also indicated for each snapshot.

For completeness, Table 7.7 reports a comparison of the accuracy of the Univ-AE method relative to POD and Custom-AE. Overall, the reconstructed flowfields by the Univ-AE are on par with those from 8-mode POD in terms of accuracy. Although the Univ-AE framework yields slightly higher MSE values than POD, our experience to date has shown that training with more data will improve the accuracy of the Univ-AE. If it is determined that a higher level of accuracy is needed from the Univ-AE to produce adequate emulation results, then future work will focus on this additional training campaign. Additionally, although the manual search process yielded a high-performing ROM, as indicated by the low MSE values, it is possible that the optimal architecture and set of hyperparameters has not yet been found. Through the use of automated tools, such as `DeepHyper` [70], a more comprehensive search can be conducted to further optimize the Univ-AE.

In summary, the use of an Univ-AE enables accurate nonlinear data reduction for the CFD-predicted multiphase flow development within a diesel-relevant fuel injector, while simultaneously providing a universal reconstruction method across the entire design space of interest. The compressed representations obtained by the Univ-AE affords the opportunity to systematically build an efficient emulator model to relate changes in design parameters to the resultant injector exit flow conditions, and thus the Univ-AE is selected hereafter as the model for dimensionality reduction.

7.3.3.4 Compressed Representations

In the previous section, a comparison between POD and autoencoders on the void fraction field revealed that the Univ-AE model is an accurate and efficient ROM technique that is well suited for emulation. In this section, the Univ-AE model is applied to all the field variables of interest, namely the three velocity components, u, v and w, in addition to the total void fraction α. We note that in this section and remaining sections of this chapter, the process of model building and evaluation is performed using the DoE study S60. Also, the neural networks are implemented using the `Keras` API of `TensorFlow 2.4`[5] (cf. Sec.2.3.9), and the Univ-AE model will be referred to as AE for short.

The snapshots from the 60 cases in the DoE study S60 were divided into three sets using the hold-out method (cf. Sec. 2.3.4.1), as shown in Table E.2. First, out of the 60 cases, 5 cases (Cases $32, 44, 51, 52, 59$) were randomly selected and set aside for testing to ensure that the emulator, including its constituents (i.e. the autoencoder and regression model), generalize well to unseen data. The remaining data (i.e., 55 cases) were used for training and validation of the neural networks, with a random split of 80%-20%.

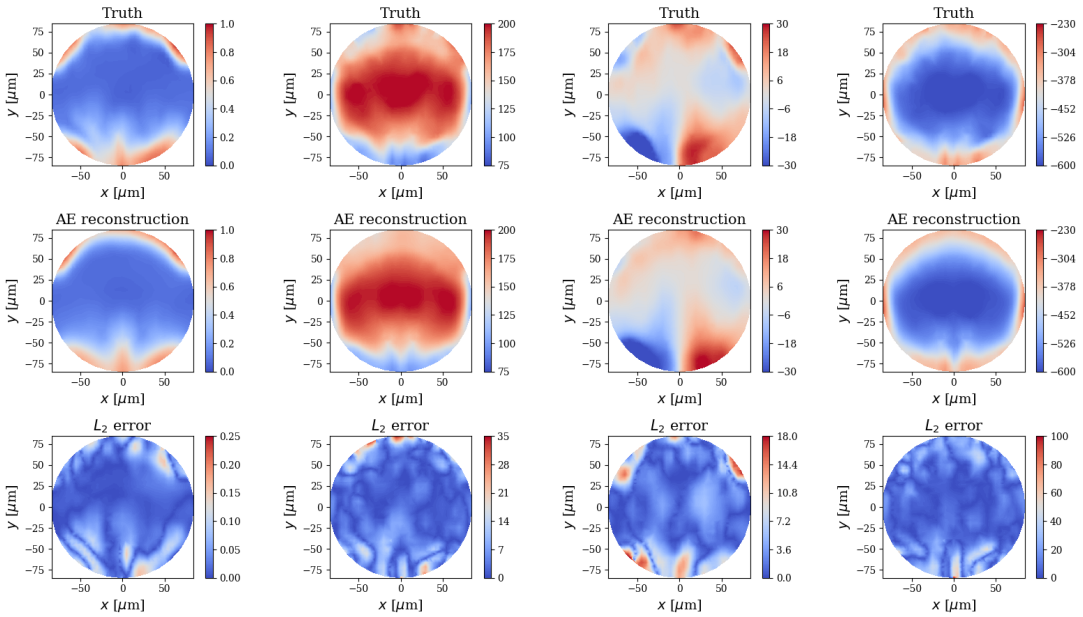
A separate autoencoder is designed for each field variable using Algorithm 5. The variables, except for α, are normalized in the range 0-1 (cf. Sec. 2.3.5.1). The network architecture and hyperparameters[6] are tuned manually following the methodology in Sec. 2.3.5.3, and the selected hyperparameters are summarized in Table 7.10. Each autoencoder contains 10 hidden layers (5 encoder layers and 5 decoder layers) with the following hyperparameters: a batch size of 25, a learning rate of 1×10^{-4} for the Adam optimizer, an L_2 regularization factor of 7×10^{-6}, and a latent vector of size 8. The number of neurons per hidden layer in the encoder and decoder is $\boldsymbol{\mathcal{H}}_{\text{enc}} = \begin{bmatrix}400 & 200 & 100 & 50 & 25\end{bmatrix}^T$ and $\boldsymbol{\mathcal{H}}_{\text{dec}} = \begin{bmatrix}25 & 50 & 100 & 200 & 400\end{bmatrix}^T$, respectively. Table 7.11 reports the MSE values for each autoencoder. Low MSE values are observed on the validation and test datasets, which indicate that the autoencoders perform in the desired manner and do not overfit.

The ability of the autoencoders to reconstruct the original CFD solution is visualized in Fig. 7.20, where the AE-based reconstructions of instantaneous α, u, v, and w fields are shown for Case 52, which belongs to the test set. Similar results are also obtained for the other test cases (not shown here). Overall, the reconstructed fields of all variables are in good agreement with the original data, although some noticeable differences are observed for the v-velocity component field. One possible explanation is that this flow variable exhibits higher spatial variation in comparison

[5] We note that by the time this phase of the project started, ALCF had upgraded `Tensorflow` to version 2.x on their clusters and the previous versions were no longer supported; that is why we switched to this version of `TensorFlow` (and `Keras`) so that we can continue training our neural neural networks on the GPUs.

[6] The architecture of the autoencoders in this section was kept the same as that of the Univ-AE model in Sec. 7.3.3.3, however, some of the hyparameters such as the learning rate and L_2 regularization were modified to better learn the data from the DoE table S60.

to the other quantities studied in this work, which would present a challenge for the same architecture to be used for each field variable.



(a) α [-] (b) u [m/s] (c) v [m/s] (d) w [m/s]

Figure 7.20: Instantaneous snapshots of α, u, v, and w for Case 52 at $t = 20.25$ μs. Flowfields computed by CFD (referred to as "Truth") and those reconstructed by AE are compared. The distribution of L_2-norm error is also indicated for each snapshot.

7.3.3.5 Emulated Flowfields

The latent space obtained by the AEs on the training and validation data are used

Table 7.10: Hyperparameters used for the autoencoders.

Parameter	Value
Number of epochs	2,200
Batch size	25
Number of hidden layers	10
$\mathscr{H}_{\text{enc}}$	$\begin{bmatrix}400 & 200 & 100 & 50 & 25\end{bmatrix}^T$
$\mathscr{H}_{\text{dec}}$	$\begin{bmatrix}25 & 50 & 100 & 200 & 400\end{bmatrix}^T$
n_{latent}	8
Optimizer for network	Adam
Learning rate	1×10^{-4}
L_2 regularization	7×10^{-6}
Activation function	ReLU

Table 7.11: MSE values for each autoencoder. Results are shown over the snapshots from the training, validation and test sets.

Data set	α	u	v	w
Training set	1.342e-03	4.125e-03	3.989e-03	3.319e-03
Validation set	1.475e-03	4.716e-03	4.593e-03	3.752e-03
Test set	1.498e-03	4.682e-03	4.703e-03	3.713e-03

to train the regressors. For each field variable, a regressor with four hidden layers is designed using Algorithm 6 to learn the mapping between the design parameters, $\boldsymbol{p}$, and the sequence of latent variables, $\boldsymbol{z}(t)$. The input and output data are standardized to stabilize the learning process (cf. Sec. 2.3.5.1). The detailed architecture of the four regressors is given in Table 7.12. The number of epochs is set to 2,000, the batch size to 25, the learning rate for the Adam optimizer to 1×10^{-3}, and the L_2 regularization factor to 8×10^{-4}. A comparison of the MSE losses on training, validation, and test sets is reported in Table 7.13 for each regressor. Evaluation of the MSE losses indicates satisfactory training performance without evidence of overfitting.

Table 7.12: Hyperparameters used for the regressors.

Parameter	Value
Number of epochs	2,000
Batch size	25
Number of hidden layers	4
$\mathscr{H}$	$[9, 32, 16, 9]$
Optimizer for network	Adam
Learning rate	1×10^{-3}
L_2 regularization	8×10^{-4}
Activation function	ReLU

Table 7.13: MSE values for each regressor. Results are shown over the time series from the training, validation, and test sets.

Data set	α	u	v	w
Training set	3.483e-01	2.139e-01	4.828e-01	1.507e-01
Validation set	3.748e-01	2.518e-01	4.934e-01	1.581e-01
Test set	3.471e-01	2.500e-01	4.981e-01	1.791e-01

For a new design parameter $\boldsymbol{p}'$, the trained regressors are used to predict the corresponding time sequence of latent vectors, i.e., $\boldsymbol{z}_{\text{pred}}(t; \boldsymbol{p}') = \mathcal{R}(\boldsymbol{p}', t)$. This sequence is then passed to the trained decoders to obtain the emulated flowfield $\boldsymbol{q}_{\text{pred}}(t; \boldsymbol{p}') = \mathcal{F}_{\text{dec}}(\boldsymbol{z}_{\text{pred}}(t; \boldsymbol{p}'))$ (cf. Algorithm 7). The temporal evolution of the components of the latent vectors is shown in solid lines in Figs. 7.21 and 7.22 for Cases 52

and 59, respectively, which belong to the test set and were not seen during the training of the neural networks. For the latent space representing the total void fraction and the v-velocity component, temporal fluctuations in the components are observed about a fixed average, whereas the latent space in the u- and w-velocity components exhibit temporal fluctuations about a moving average. The time-series predictions for the trained regressors are shown in dashed lines in Figs. 7.21 and 7.22. The regressors are observed to predict the trend and order of the latent space components accurately, indicating that the mean behavior has been learned. However, the magnitude of the temporal fluctuations is not well captured, suggesting that further effort is needed to improve these predictions. Some of the ideas under consideration for future work are the use of more data for training, and the use of more advanced architectures, like LSTM (cf. Sec. 2.2.6), which have shown good success in capturing temporal dependencies for modeling sequential data.

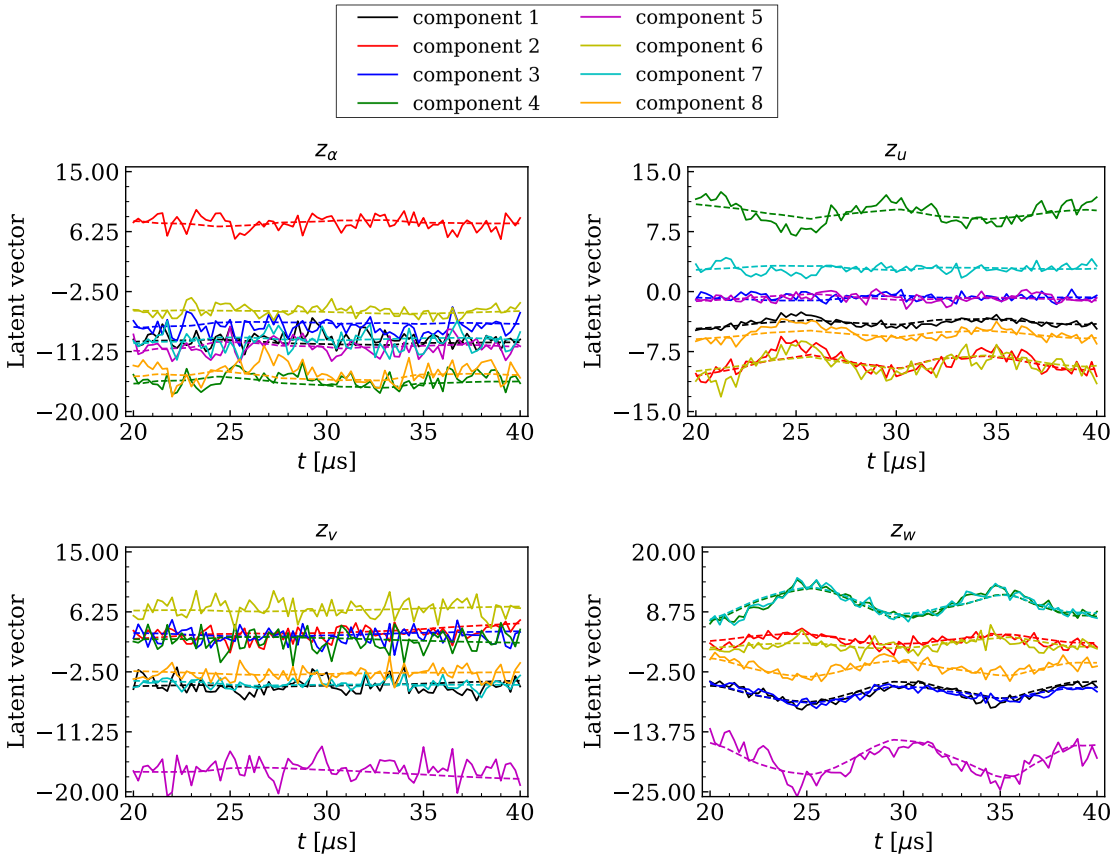


Figure 7.21: Temporal evolution of latent variables for Case 52. Latent variables computed by the autoencoder (solid lines) and those predicted by the regressor (dashed lines) are compared.

Figures 7.23 and 7.24 show time-averaged snapshots of the simulated and emulated flowfields for Cases 52 and 59, respectively. In Case 52, a three-pronged structure in α is predicted along the bottom of the orifice exit, which has been found to be caused by the local expansion of non-condensable gas and its interaction with the flow

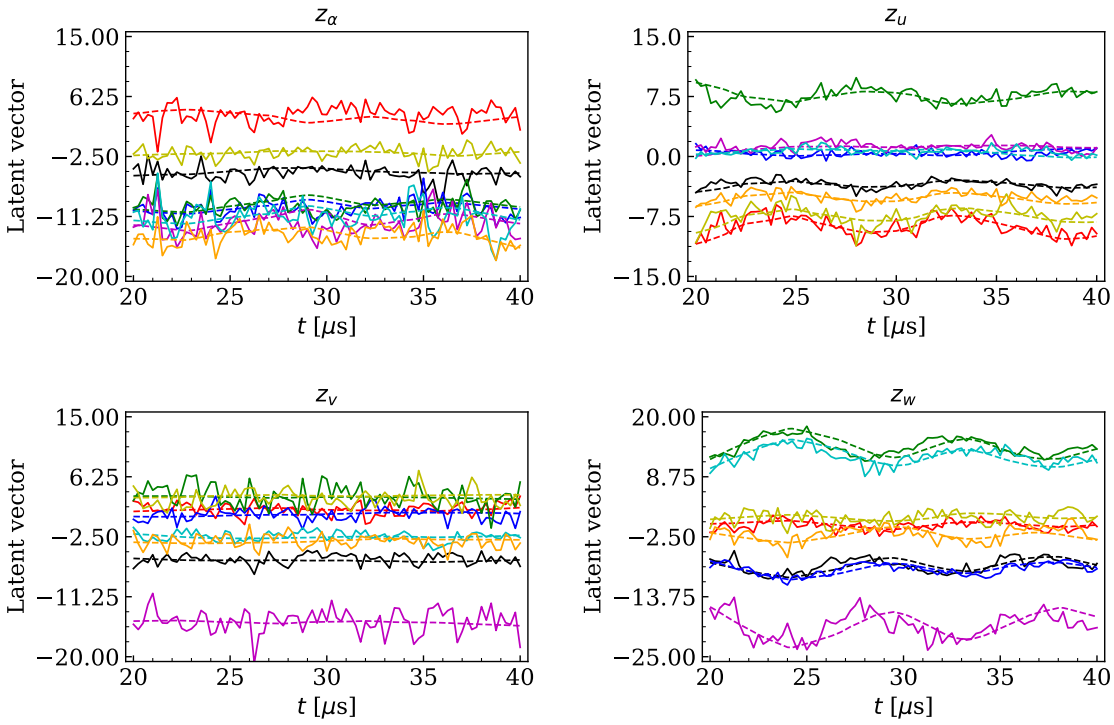


Figure 7.22: Temporal evolution of latent variables for Case 59. Latent variables computed by the autoencoder (solid lines) and those predicted by the regressor (dashed lines) are compared.

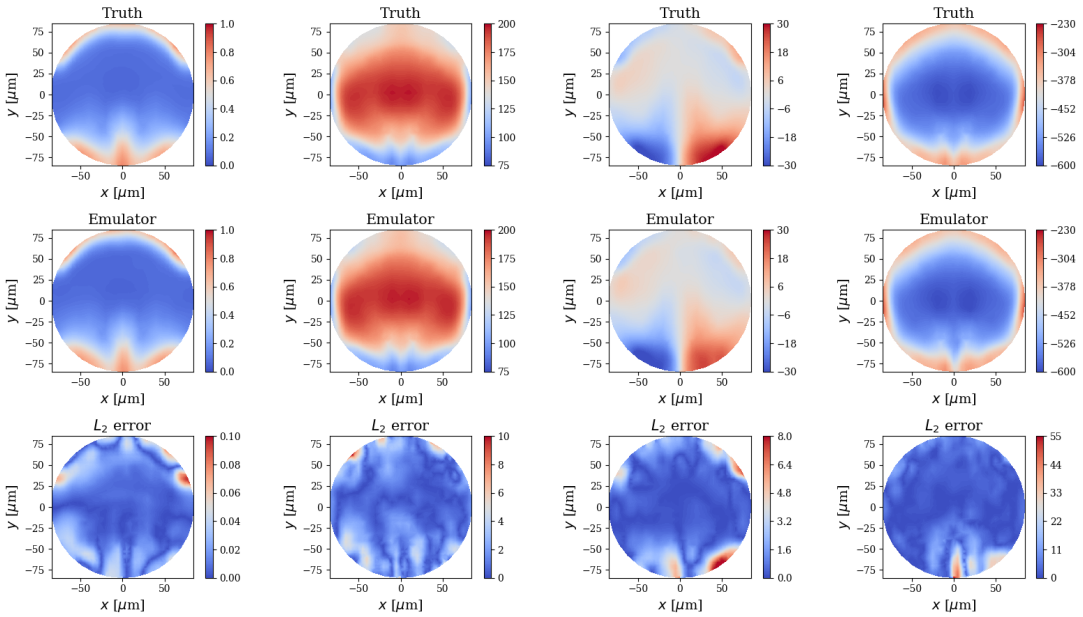
vortices [211]. On the other hand, in Case 59, a dome-like structure in α is predicted along the bottom of the orifice exit. These characteristics are well captured by the emulator. Good agreement is also observed between the simulated and emulated predictions for the other field variables.

For the sake of completeness, the relative errors ϵ_{rel} for the time-averaged emulated fields are shown in Table 7.14. Overall, the emulation errors are less than 2% for the u and w fields, less than 8% for the α field, and less than 27% for the v field. As we will see in Sec. 7.4.2, despite the relatively large errors obtained for the v field, its effect on the ensuing spray and combustion simulations seems to be minimal. If higher accuracy emulators are needed, a more exhaustive network architecture and hyperparameter search could be conducted in the future using automated optimization frameworks, such as `DeepHyper`.

7.3.3.6 Computational Cost

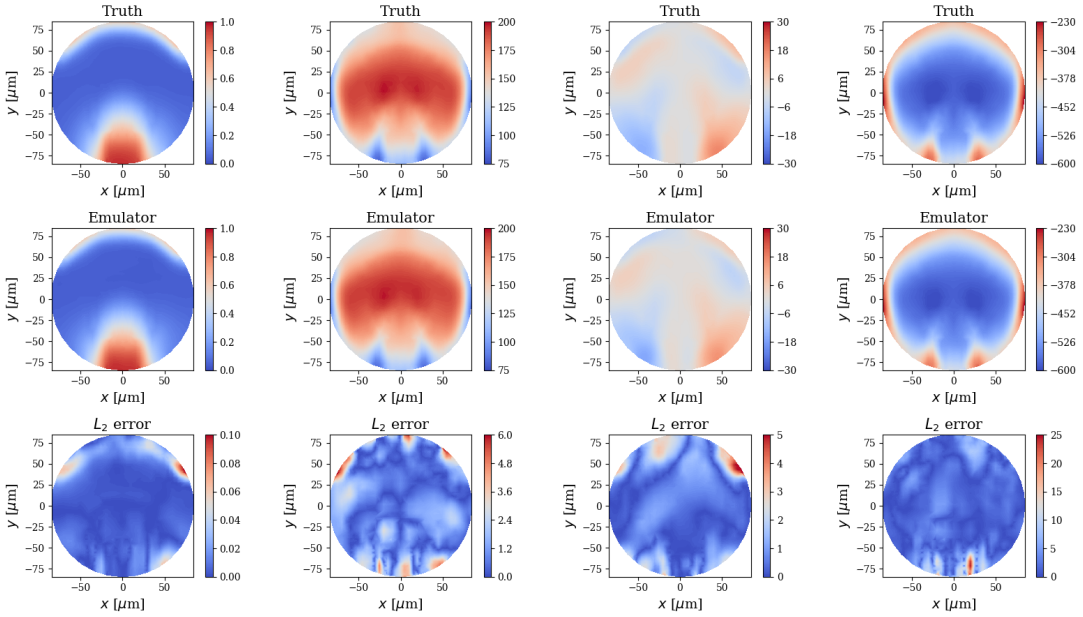
Table 7.15 summarizes the computational time required for each step of the emulation process, with timing performed on a Linux laptop that has the following specifications: Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz[7]. The preprocessing step, which

[7]Note that in Table 2.2 in Sec. 2.3.7 the CPU-time was assessed using an Intel Core i7-10700T



(a) α [-] (b) u [m/s] (c) v [m/s] (d) w [m/s]

Figure 7.23: Time-averaged contours of α, u, v and w from CFD and emulator for Case 52. The distribution of L_2-norm error is also indicated for each field variable.



(a) α [-] (b) u [m/s] (c) v [m/s] (d) w [m/s]

Figure 7.24: Time-averaged contours of α, u, v and w from CFD and emulator for Case 59. The distribution of L_2-norm error is also indicated for each field variable.

CPU @ 2.00 GHz, while here (i.e., in Table 7.15) the CPU-time was computed using an Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz; that is why there is a difference in the reported autoencoder training time for the A-M1 injector between the two tables (7,412 CPU-secs vs. 1 CPU-hour, or 3,600 CPU-secs.)

Table 7.14: Relative errors for the time-averaged emulated fields for test cases.

Test Case	α	u	v	w
Case 32	0.075	0.011	0.266	0.009
Case 44	0.039	0.011	0.210	0.009
Case 51	0.049	0.018	0.216	0.013
Case 52	0.069	0.013	0.177	0.019
Case 59	0.053	0.011	0.210	0.008

includes file conversion and extraction of the snapshots from CFD, requires about 165 CPU-hours (3 hours for each of the 55 cases used for training/validation). For each of the four field variables (i.e., α, u, v and w), one autoencoder and one regressor are trained, requiring 1 CPU-hour and 0.0375 CPU-hours, respectively. Hence, in total, the cost of the training step is 4.15 CPU-hours. After the training is completed, the inference time is approximately 8 seconds (2 seconds per variable, and we have four variables) for the emulator to predict the flowfield for a new design point. In comparison to CFD simulations, which on average take 6,000 CPU-hours per case, an overall speedup of 35 times is achieved in predicting the injection conditions using the emulation framework. We note that the speedup is estimated to be 2.7 million times if only the inference time is considered. In the current implementation, the data preprocessing step takes up a substantial portion of the total emulation cost. By streamlining this process, it is estimated that the preprocessing cost can be reduced to a total of one hour for all 55 cases. With this in mind, the expected speedup (including time for data preprocessing and model training) is estimated to be 1,165 times faster than running a new CFD case. This clearly shows the transformational potential of the emulation framework to enable efficient exploration of the design space.

Table 7.15: Computational time in CPU-time for each step of the emulation framework. Results are also indicated in GPU-time for the autoencoder training step.

Step	Time
Preprocessing	165 CPU-hours.
Autoencoder training	4×1 CPU-hours
	(4×35.6 GPU-mins)
Regressor training	4×0.0375 CPU-hours
Flow prediction	4×2 CPU-secs.

7.4 One-Way Coupled Spray Simulation

The previous section demonstrated that overall the emulator was able to successfully predict the injection conditions at the orifice exit of the injector for design points that were unseen during training, despite some flow prediction discrepancies, particularly in the v-velocity field. In this section, we evaluate the performance of the emulator and the significance of these discrepancies in the context of the OWC spray modeling approach. This section is organized as follows. Sec. 7.4.1 describes the LE framework, including the spray initialization strategy and OWC approach. The OWC spray simulation results are presented in Sec. 7.4.2.

The material presented here is excerpted and adapted from the following publication. Only work conducted as part of the present study is included here: [236] S. Mondal, R. Torelli, B. Lusch, P. J. Milan, and G. M. Magnotti, "Accelerating the generation of static coupling injection maps using a data-driven emulator," *SAE Int. J. Advances & Curr. Prac. in Mobility*, Vol. 3, pp.1408-1424, 2021.

7.4.1 Reacting Spray Modeling Approach

The pressure in the constant-volume cylinder chamber is 20 bar (subcritical condition), and thus the classical situation of jet atomization exists where a well defined interface separates the injected liquid from the ambient gas (cf. Sec. 4.3.4.1). To model the spray development process leading to combustion in the chamber, the LE framework is used (cf. Sec. 4.3.7). A schematic of the computational domain is shown in Fig. 7.25. The dispersed liquid phase is handled using a Lagrangian approach based on the blob injection method [237]. Primary and secondary atomization processes are modeled using the Kelvin-Helmholtz Rayleigh-Taylor spray breakup theory [238], and evaporation is accounted for using the Frossling correlation [239]. The continuous gas phase is handled using an Eulerian finite volume approach within the URANS framework. To achieve turbulence closure, the rapid distortion RNG k-ϵ turbulence model [238] is used.

The liquid spray is initialized using the OWC approach in `CONVERGE` [98]. In this approach, information is obtained from the internal flow simulation, or, alternatively, from the emulator framework, in the form of spatiotemporal injection maps at the orifice exit of the injector. The map files include position coordinates, liquid volume fraction,[8] velocity components, turbulence, and temperature information. At a given cell, parcels are injected only if the liquid volume fraction (LVF) is higher than 0.1 [98]. A constant TKE of 3,000 m^2/s^2 is assumed for all the OWC simulations following the work of Nocivelli et al. [204]. The temperature of the injected spray parcels is almost uniform due to the constant internal energy assumption for the internal flow simulation. More details on the parcel initialization strategy and OWC approach can be found in Refs. [98, 236, 204].

[8] The liquid volume fraction α_l is simply equal to $\alpha_l = 1 - \alpha$.

An unsteady flamelet progress variable (UFPV) approach [240] is employed to capture autoignition and combustion for a turbulent non-premixed flame. In this model, the transient flame evolution is represented by mixture fraction and its variance, reaction progress variable, and scalar dissipation rate, and the flame structure is obtained from the unsteady flamelet equations. To model the chemical kinetics, a detailed chemical mechanism for n-dodecane comprised of 2,755 species and 11,117 reactions [241] is used.

Fixed embedding and adaptive mesh refinement (AMR) based on local gradients in temperature, velocity, and fuel mass fraction is employed with a base grid size of 2 mm to achieve a minimum cell size of 250 μm. This resulted in a peak cell count of 940,000. An example of the grid refinement strategy for the OWC simulations is shown in Fig. 7.25. Each case takes about 20 CPU-hours per 10 μs of simulated time. As an aside, this cost is much lower than that of the internal flow simulation, which on average takes 1,500 CPU-hours for the same duration of simulated time, as previously reported in Secs. 7.3.1.3 and 7.3.3.6.

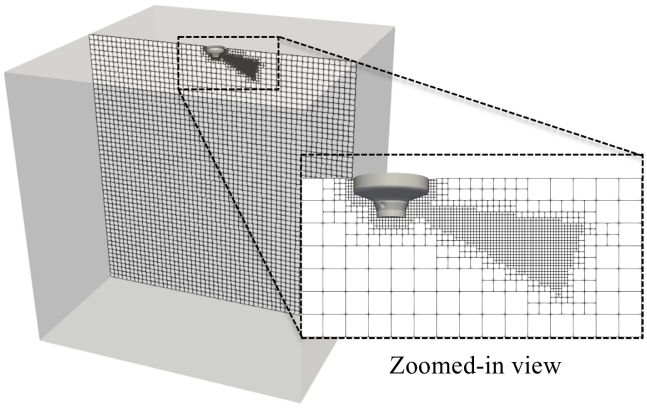


Figure 7.25: Computational domain and grid refinement strategy for OWC spray simulations (adapted from Refs. [236] and [242]).

7.4.2 Results and Brief Discussion

The ability of the emulator to link the internal flow development with the external spray is assessed by comparing the reacting spray predictions from simulations using the CFD-predicted and emulated injection maps. For each of the test cases, two OWC spray simulations are performed with the same computational setup except for the source of the injection map files: (1) map files obtained from the internal flow simulation, and (2) map files obtained from the emulator framework.

A qualitative comparison of the spray and combustion characteristics is shown in Fig. 7.26 for Case 59. The results show close agreement between the two OWC simulations; the droplet size distribution as well as the overall flame structure are very similar. This also shows that the errors in the emulated v-velocity field have minimal impact on the ensuing spray and combustion characteristics. More detailed

results including quantitative comparisons of liquid and vapor penetration, peak heat release rate, and peak temperature between the two simulations can be found in Mondal et al. [236]. These quantitative results confirm the observations made from the qualitative analysis, however, they were not included here for brevity.

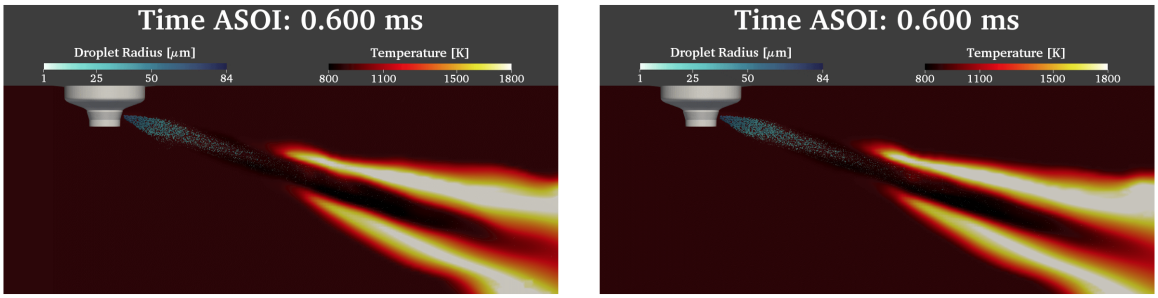


Figure 7.26: Comparison between the OWC simulations using the CFD generated field data (left) and emulated field data (right) for Case 59. Results are shown for the temperature field at time ASOI = 0.6 ms. Radii of the Lagrangian parcels are also indicated in each plot (Courtesy of Gina M. Magnotti).

7.5 Summary

The work presented in this chapter had three aims: (1) characterize the internal flow dynamics in the diesel-relevant A-M1 injector nozzle using high-fidelity simulations, (2) develop a data-driven emulation framework based on deep learning techniques to enable efficient, parametric, multiphase flow predictions in the injector, and (3) use the emulator-predicted flowfields at the injector exit to initialize external spray and combustion simulations using a static coupling approach. The main results and contributions can be summarized as follows:

- Large eddy simulations were employed to study the impact of selected design parameters, namely needle lift, liquid fuel dynamic viscosity, and level of non-condensable gas in the fuel, on the distribution and composition of gas-phase flow structures at the orifice exit of the A-M1 injector. Due to the sharp inlet radius of curvature, cavitation formation was predicted for all cases. However, various different gas-phase flow structures were predicted across the design space. Using variance-based sensitivity analysis, it was found that the fuel vapor volume fraction at the orifice exit was most strongly influenced by the level of non-condensable gas in the fuel, whereas the total vapor volume fraction was most highly influenced by needle lift. Although liquid fuel viscosity is known to influence cavitation inception, viscosity was found to have the least influence on the extent of the gas-phase flow structures within the orifice.

- The large amount of data generated via the LESs, which spanned the multi-dimensional design space of interest, was then used to define a spatiotemporal

training database characterizing the flowfield at the exit of the injector. ROMs were then formulated to allow for design parameters to be related to the CFD-predicted injector exit conditions. Results using POD were compared with those of two autoencoder-based methods, a customized autoencoder ("Custom-AE") that yielded a different reconstruction function for each CFD case, and a universal autoencoder ("Univ-AE") that yielded a single and consistent reconstruction function across the design space of interest. All ROMs were able to provide accurate reconstructions of the exit flow conditions, as indicated by MSE values below 4×10^{-3}. Across all of the cases studied, it was found that POD required a larger number of modes to accurately reconstruct the total gas-phase volume distribution relative to the autoencoder-based methods. Additionally, for the same level of data compression, the Custom-AEs provided better reconstruction accuracy than POD, while the Univ-AE provided comparable reconstruction accuracy. These findings indicate that the Univ-AE developed in this work is an accurate and efficient ROM technique that is well suited for use in emulation.

- An emulator framework was then designed to approximate the spatiotemporal exit conditions of the A-M1 diesel injector over the design space of interest. The emulator featured autoencoders, for dimensionality reduction, and deep neural networks for supervised regression of the latent space representations. The time-averaged flowfields at design points unseen during training were well predicted by the emulator for most variables (emulation errors less than 2% for u, w, and less than 8% for α) except for the v-component field (emulation errors less than 27%). In comparison to using CFD to simulate the next design point of interest, the emulator showed a speedup of 35 times (or 2.7 million times, if only inference time is considered) in predicting the injection conditions and allowing for rapid exploration of the design space.

- Finally, the predicted flowfields from the emulator were used to initialize external spray and combustion simulations using the OWC approach. The results showed very good agreement with baseline spray simulations that employ CFD-generated injection maps.

Overall, these results are promising and demonstrate that the proposed framework is an efficient means of improving the efficiency and accuracy of end-to-end simulation tools for use in engine design and analysis.

List of Main Symbols

Latin Symbols

$\boldsymbol{b}$	bias vector
Ca	cavitation number, area contraction coefficient
$\boldsymbol{f}_{\text{tens}}$	surface tension force
h	mass-specific enthalpy of the mixture
h_{f}	saturated liquid enthalpy

J	loss function
L	heat of vaporization
m_{g}	total gas mass fraction
m_{l}	total liquid mass fraction
$\boldsymbol{n}$	pseudo-surface normal vector
n_{in}	size of input layer of autoencoder, number of grid points at orifice exit
n_{latent}	size of latent space
n_t	number of temporal snapshots acquired from each CFD case
n_r	number of retained POD modes
$n_{r,99\%}$	number of POD modes required to recover 99% of modal energy
$n_{\boldsymbol{p}}$	number of design/physical parameters
p	pressure
$\boldsymbol{p}$	vector of design/physical parameters
p_{c}	critical pressure
p_{sat}	saturation pressure
p_{amb}	ambient pressure
$\boldsymbol{q}$	autoencoder input, field variable
$\widetilde{\boldsymbol{q}}$	autoencoder output, reconstructed field variable
S_n	source term for the nth species
S_{v}	source term for the fuel vapor
t	physical time
T	temperature
$\boldsymbol{u}$	mixture velocity vector
$\boldsymbol{W}$	weight matrix
$\boldsymbol{x}$	spatial coordinate vector
x, y	spatial coordinates
Y_{l}	mass fraction of the liquid phase
Y_n	mass fraction of the nth species
$Y_{\mathrm{N_2}}$	level of non-condensable gas in the fuel
Y_{v}	mass fraction of the fuel vapor
$\overline{Y}_{\mathrm{v}}$	equilibrium value for mass fraction of the fuel vapor
$\boldsymbol{z}$	latent vector

Calligraphic Symbols

$\mathcal{F}_{\mathrm{enc}}$, $\mathcal{F}_{\mathrm{dec}}$	encoder function, decoder function
$\boldsymbol{\mathcal{H}}$	vector of neurons per layer in all the hidden layers
$\mathcal{R}$	regressor
$\boldsymbol{\mathcal{V}}_n$	diffusion velocity vector for the nth species

Greek and Blackboard Bold Symbols

ψ	non-dimensional pressure ratio
α	total void fraction field
$\overline{\alpha}$	modified total void fraction field
μ_F	dynamic viscosity of liquid fuel
ρ	density
ρ_g	gas-phase density
ρ_l	liquid-phase density
$\boldsymbol{\tau}$	mixture viscous stress tensor
κ	curvature
σ	surface tension coefficient
λ_{reg}, ζ_{reg}	control parameter for L_2 regularization
θ	relaxation time scale
$\boldsymbol{\theta}, \boldsymbol{\phi}$	parameters of autoencoder and regressor
$\mathbb{T}_1$, $\mathbb{T}_2$	test sets

Subscripts

dec	decoder
enc	encoder
pred	predicted variable
true	true variable

Superscripts

sgs	subgrid-scale
$\widetilde{\square}$	reconstructed variable
$\square'$	new value of variable

Part III

Conclusions

CHAPTER 8
SUMMARY AND FUTURE WORK

8.1 Summary of Results

This research effort focused on developing deep-learning (DL)-based surrogate modeling approaches and software to accelerate the simulation and design of complex thermo-fluid systems. Particular emphasis was placed on high-pressure practical flows for automotive and aerospace chemical propulsion, as well as on high-dimensional problems involving multiple input parameters. A detailed list of objectives and tasks was presented in Sec. 1.2, and a summary of DL models/frameworks developed is presented in Table 8.1. Some of the highlights of the key results are presented below:

- *Accelerating real-fluid simulations with deep learning*: a DL methodology was proposed for fast calculation of real-fluid thermophysical properties in numerical simulation of supercritical flows. The method featured a deep feedforward neural network with appropriate boundary information, referred to as DFNN-BC, which can be coupled to a flow solver in a robust manner. The approach was demonstrated in primitive- and conservative-variable-based solvers and was evaluated on several problems of increasing complexity, involving up to seven (major) chemical species. The approach reduced the computation time of real-fluid properties by a factor of up to 3.7 and the overall simulation time by a factor of up to 2.3. Memory usage was reduced by up to five orders of magnitude in comparison with the table look-up method. The results and formulations are presented in Chapter 5.

- *Data-driven surrogate modeling for spatiotemporal emulation with deep learning*: a data-driven DL-based surrogate modeling framework was developed to enable efficient prediction and parametric estimation of spatiotemporal flow dynamics. The framework featured autoencoders, for nonlinear dimensionality reduction, and neural-network-based regressors ($\mathcal{R}$) for supervised learning of the latent space. Two versions of this framework were proposed, for the two different autoencoders: (1) FCAE-$\mathcal{R}$, with a fully-connected autoencoder that can be applied to all types of data (structured/unstructured, regular/irregular), and (2) CAE-$\mathcal{R}$, with a convolutional autoencoder that can retain spatial coherence of the input data to improve the prediction accuracy. With the current formulation, the latter is suited only for data on structured rectangular domains with uniform grids. The proposed emulators were evaluated on two canonical, parametric problems with up to two input parameters and compared with traditional surrogate modeling approaches. The emulators showed a speedup of up to 14 times over the full-order model in predicting the flowfields for new input parameter instances. The results and formulations are presented in Chapter 6.

- *Data-driven spatiotemporal emulation for rapid engine design with deep learning*: an emulation framework was developed to enable efficient multiphase flow predictions in fuel injectors with realistic geometries and operating conditions for rapid exploration of design spaces. In particular, the emulator was designed to approximate the spatiotemporal conditions at the exit of a diesel injector (on unstructured grids) for a wide range of design parameters, operating conditions, and fuel properties. These parameters were explored by evaluating changes in three input parameters, namely needle lift, fuel viscosity, and level of non-condensable gas in the fuel. The emulator featured fully-connected autoencoders for nonlinear dimensionality reduction and neural-network-based regressors for supervised regression of the latent space. The time-averaged flowfields at new design points were well predicted for most variables. In comparison to using high-fidelity simulations to simulate the next design point of interest, the emulator showed a speedup of up to 2.7 million times in predicting the injection conditions. The predicted flowfields from the emulator were also used to initialize external spray simulations using a static coupling approach, and showed very good agreement with the baseline simulations. The results and formulations are presented in Chapter 7.

8.2 Major Contributions

This thesis has made contributions to the fields of scientific deep learning (SciDL), computational fluid dynamics (CFD), and chemical propulsion. The main contributions can be grouped into two categories:

Contributions on the database and software side:

- Generation of a unique spatiotemporal database from CFD simulations comprising several canonical and engineering problems for DL and reduced-order model (ROM) development.
  Significance and potential impact: the generated database can be used to support future model upgrade and development. It can also serve as use-cases to compare between different surrogate modeling strategies in the future.

- Formulation of a coupled DL and physical-simulation framework for accelerating the execution of high-fidelity CFD simulations on supercomputers.
  Significance and potential impact: the proposed DL model can act as an add-on to allow scientists and engineers to reduce the time-to-results of complex and expensive simulations without compromising accuracy, and can be extended to many areas of computationally-intensive science and engineering, such as turbulent combustion (including flamelet modeling), climate science, and molecular dynamics, to name a few. While the model has been demonstrated herein on distributed, homogeneous architectures, it can also be implemented on heterogeneous CPU-GPU architectures and will offer options (GPUs and DL) to offload expensive subroutines and kernels to further reduce the simulation time.

- Formulation of a novel data-driven emulation framework using autoencoders and neural-network-based regressors for rapid exploration of multi-parametric design spaces.
  Significance and potential impact: the developed surrogate framework (with both of its variants) can predict high-dimensional flowfields with relatively high accuracy and at the fraction of the cost of full-scale CFD simulations for unseen parameter instances in the design space. The model can also be used to perform surrogate-based design optimization to support selection of optimal design parameters to satisfy the design requirements of interest. Further, the model can be applied to a broad of range of engineering systems and will support future engineering innovation and design.

Contributions on the physics and application side:

- First demonstration of deep neural networks (DNNs) as a supervised machine learning (ML) technique for accelerating the evaluation of real-fluid thermophysical properties in simulation of supercritical flows.
  Significance and potential impact: the development of efficient methods for the evaluation of real-fluid properties has attracted significant interest over the past three decades (see Table 5.1). Most (if not all) methods previously developed, however, still face the inevitable "curse of dimensionality." The DFNN-BC model presented herein, however, can overcome this limitation and can be applied to real-fluid mixtures with multiple species, thus enabling high-fidelity simulations of complex multiscale multiphysics flows (such as supercritical flows in a rocket-engine swirl injector) at reduced cost.

- Sensitivity analysis of the predicted injection conditions at the orifice exit of the newly proposed A-M1 injector from the Spray Combustion Consortium (SCC). From the present study, it was found that the fuel vapor volume fraction at the orifice exit was most strongly influenced by the level of non-condensable gas in the fuel, while the total vapor volume fraction was most highly influenced by the needle lift.
  Significance and potential impact: these results not only enhance the basic understanding of injector physics, but also establish a quantitative basis to identify and prioritize the key design parameters and flow variables that exert dominant influence on injector behavior at different conditions.

- First demonstration of data-driven DL emulators as an ML framework to help eliminate computational bottlenecks in automotive engine design.
  Significance and potential impact: by relating design/physical parameters to the spatiotemporal flowfields at the exit of the injector orifice, the proposed framework allows for efficient coupling between injector and spray simulations and significantly accelerates the time-to-design of advanced internal combustion engines. The emulation framework can also be applied to different engineering applications where efficient and accurate predictions of complex spatiotemporal fields are desired.

8.3 Recommendations for Future Work

Several areas of improvement to the current models are suggested for future research:

- *ROM-DNN-based tabulation*: the DFNN-BC model developed as part of this study for the evaluation of real-fluid properties consisted of a single neural network and was applied to mixtures involving up to seven chemical species. For some applications, such as turbulent combustion, however, the system can comprise tens, or sometimes even hundreds, of species. In this scenario, the training database of properties can consume tens (or hundreds) of GB and could overwhelm a single neural network. A very large neural network trained on a powerful AI accelerator could be envisioned – see the last bullet point below. Alternatively, multiple neural networks could be implemented, with one network used for each variable or group of correlated variables, to achieve the desired approximations. A ROM (such as an autoencoder) could be employed to reduce the dimensionality of the database, and then a DNN fit using the reduced data (hence, the name ROM-DNN), in a fashion similar to that in FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$.

- *Physics-informed surrogate modeling*: the emulators developed here were mostly data-driven and thus required considerable training data in order to make accurate predictions, especially for practical problems. An emerging approach that is gaining popularity in SciDL is physics-informed neural networks (PINNs) (cf. Sec. 2.1.2). In this approach, the governing equations are enforced through a loss function, which allows preservation of the desired physics and reduction of the amount of labelled data needed for training. Other promising approaches that could reduce the training database size, or the number of expensive simulation calls, include: (1) *transfer learning*, which is an ML technique where a pre-trained model developed for a problem is reused partly or wholly as the starting point for a model on a different but similar (often more complex) problem [55, 243], and (2) *active learning*, which is a supervised learning approach in which the algorithm actively chooses the input training points in regions with high gradients or in the vicinity of the global optimum [244]. These approaches are still in early stages of research and are worth investigating in the future.

- *Multi-block surrogate modeling*: rather than constructing a single surrogate for the entire emulation domain of interest (as in FCAE-$\mathcal{R}$ and CAE-$\mathcal{R}$), it is possible to construct separate surrogates for the different subdomains/components characterizing the original model, but this requires the development of appropriate methods to enforce compatibility at the interface of subdomains, which can be non-trivial. Such an approach can make surrogate models amenable to very large scale and decomposable systems [245, 246]. Another interesting direction is the development of *geometry-adaptive convolutional neural networks* and *geometry-adaptive convolutional autoencoders* [247] to overcome the limitations of classical CNNs and CAEs in processing data on irregular domains (i.e., domains not necessarily rectangular and with non-uniform grids). Such an

approach can permit leveraging of the parameter-sharing feature in CNNs and CAEs (which is enabled by the filter-based convolution operations) on complex geometries. Ultimately, multi-block and geometry-adaptive methodologies can be combined to create powerful surrogates with broad applicability.

- *Probabilistic DL emulation*: the emulator frameworks developed here were all non-probabilistic and provided only point estimates of the results. It would be beneficial to upgrade these frameworks with posterior inference in order to capture uncertainty and assign confidence to predictions. One way to achieve this would be to replace the autoencoders with variational autoencoders and the neural-network-based regressors with probabilistic neural networks (cf. Sec. 2.2.6).

- *Usage of purpose-built artificial intelligence (AI) accelerators for DL training and inference*: AI accelerators are specialized hardware, or processors, dedicated to accelerating ML/DL computations. They are often designed to improve throughput, reduce latency, and deliver more computational, memory, and communication bandwidth, and can thus support dataflow-oriented workloads, such as AI and high performance computing (HPC), at dramatically faster speeds and scalability, as compared with multi-core CPUs and GPUs and multithreading. In particular, with the availability of such technology, ever-larger neural networks and surrogate/approximation models with tens of billions[1] of parameters can now be trained efficiently to tackle some of the world's most challenging problems in science and engineering.

- *Mathematics of surrogate modeling*: despite the empirical success of data-driven (and physics-informed) DL surrogate modeling approaches, little is known about their theoretical convergence guarantees and, although not evident, especially for nonlinear problems, this is an area for future exploration.

- *Recommendations for method selection*: while DL offers promising approaches to many previously intractable problems, and to as yet unidentified challenges, other approaches to surrogate modeling will of course be required as the field advances. These will include traditional statistical and curve-fitting methods, and also hybrid models, like those of Refs. [248, 249]. To this end, a research effort is needed to provide clear, considered recommendations and guidelines for the selection and application of methods.

[1] This number of parameters will certainly increase in the next few years as the AI technology continues to evolve.

Table 8.1: Summary of DL models/frameworks developed in this study.

Problem	Accelerate the simulation and design of complex thermo-fluid systems with deep neural networks	
DL tasks/capabilities	Regression with scalar responses	Regression with spatiotemporal responses
Models/frameworks	• DFNN-BC (Sec. 5.4) • DFNN-BC/CFD coupling (Sec. 5.4)	• FCAE-$\mathcal{R}$ (Secs. 6.3 and 7.3.2) • CAE-$\mathcal{R}$ (Sec. 6.3)
Canonical cases	• 0D real-fluid thermodynamics (Secs. 5.4.4 and 5.4.5) • Quasi-1D counterflow diffusion flames (Sec. 5.6)	• 1D Burgers equation (Sec. 6.4) • 2D ADR equation (Sec. 6.5)
Engineering cases	• GCLSC rocket injector flow (Sec. 5.5)	• A-M1 diesel injector flow (Chapter 7)
Generic goals	• Function approximation • Solver acceleration	• Simulator approximation • Rapid exploration of design spaces • Enhanced optimization • Efficient one-way coupling and end-to-end simulations

Appendices

APPENDIX A
PROPER ORTHOGONAL DECOMPOSITION

Proper orthogonal decomposition (POD) (also known as *principal component analysis* (PCA), or *Karhunen-Loéve procedure*) is a *linear* model reduction technique that extracts dominant energetic structures from the calculated flowfield. Detailed explanations of the POD technique can be found in Refs. [15, 250, 251], but a brief summary is provided here.

Let $f(\boldsymbol{x}, t)$ be a flow variable (e.g., velocity components, temperature) at spatial coordinate $\boldsymbol{x} \in \Omega$ and time $t \geq 0$, and $f'(\boldsymbol{x}, t) = f(\boldsymbol{x}, t) - \overline{f}(\boldsymbol{x})$ the fluctuations, where $\overline{f}$ is the time-averaged mean of f. POD decomposes $f'(\boldsymbol{x}, t)$ into separable spatial and temporal components

$$f'(\boldsymbol{x}, t) = \sum_{i=1}^{\infty} \beta_i(t)\phi_i(\boldsymbol{x}), \tag{A.1}$$

where $\beta_i(t)$ and $\phi_i(\boldsymbol{x})$ represent the time-varying coefficient and spatial function (mode shape) of the ith mode, respectively. These quantities can be computed numerically using the method of snapshots owing to the advantages of its computational efficiency. In particular, two approaches can be employed: the first is based on a *singular value decomposition* (SVD) of the data matrix, while the second is based on an *eigenvalue decomposition* (EVD) of the correlation matrix.

A.1 Singular Value Decomposition

In this section, the SVD-based POD technique [15, 250, 251, 252] is explained. Let $\boldsymbol{f}^{*,i} \in \mathbb{R}^m$ be the i^{th} snapshot in time (for a total of n snapshots) of the discretized solution of f, where m is the number of spatial points in the subdomain of interest. The POD procedure starts with decomposing $\boldsymbol{f}^{*,i}$ into its mean, $\overline{\boldsymbol{f}^*} \in \mathbb{R}^m$, and fluctuating components, $\hat{\boldsymbol{f}}^i \in \mathbb{R}^m$, as defined below

$$\boldsymbol{f}^{*,i} = \overline{\boldsymbol{f}^*} + \hat{\boldsymbol{f}}^i, \tag{A.2a}$$

$$\overline{\boldsymbol{f}^*} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{f}^{*,i}. \tag{A.2b}$$

The *snapshot matrix*, also called *data matrix*, is then built from the collection of n snapshots

$$\boldsymbol{S} = \left[\hat{\boldsymbol{f}}^1 \mid \hat{\boldsymbol{f}}^2 \mid \ldots \mid \hat{\boldsymbol{f}}^n\right] \in \mathbb{R}^{m \times n}. \tag{A.3}$$

For clarity, we can also write $\boldsymbol{S}$ as

$$\boldsymbol{S} = \begin{bmatrix} \hat{f}(\boldsymbol{x}_1, t_1) & \hat{f}(\boldsymbol{x}_1, t_2) & \dots & \hat{f}(\boldsymbol{x}_1, t_n) \\ \hat{f}(\boldsymbol{x}_2, t_1) & \hat{f}(\boldsymbol{x}_2, t_2) & \dots & \hat{f}(\boldsymbol{x}_2, t_n) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{f}(\boldsymbol{x}_m, t_1) & \hat{f}(\boldsymbol{x}_m, t_2) & \dots & \hat{f}(\boldsymbol{x}_m, t_n) \end{bmatrix}. \tag{A.4}$$

We then compute the *reduced SVD* (also called *economy-sized SVD*) of $\boldsymbol{S}$, with $r_{\boldsymbol{S}}$ denoting the rank of $\boldsymbol{S}$

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{A.5}$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times r_{\boldsymbol{S}}}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times r_{\boldsymbol{S}}}$ are orthogonal matrices, i.e., $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_{r_{\boldsymbol{S}}}$ with $\boldsymbol{I}_{r_{\boldsymbol{S}}}$ the identity matrix, and $\boldsymbol{\Sigma} \in \mathbb{R}^{r_{\boldsymbol{S}} \times r_{\boldsymbol{S}}}$ is a diagonal matrix that contains the non-negative singular values, σ_i, arranged in descending order. The singular values can be interpreted as the weight of contribution of each energetic mode in the POD reconstruction. The change with time of the modes is represented by V whereas their spatial distribution is represented by U. To determine the dominant modes needed to represent the flow data, the first $r \leq r_{\boldsymbol{S}}$ singular values are selected such as they capture a certain percentage e of the total energy (for instance, e can be set to 0.99):

$$\frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{r_{\boldsymbol{S}}} \sigma_i^2} = e. \tag{A.6}$$

Finally, the flow can be reconstructed using the reduced number of modes, as follows

$$\boldsymbol{S} \approx \widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}^T, \tag{A.7}$$

where $\widetilde{\boldsymbol{U}} \in \mathbb{R}^{m \times r}$, $\widetilde{\boldsymbol{\Sigma}} \in \mathbb{R}^{r \times r}$, and $\widetilde{\boldsymbol{V}} \in \mathbb{R}^{n \times r}$ are subparts of $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$, respectively.

A.2 Eigenvalue Decomposition

In this section, the EVD-based POD technique [15, 250, 251, 188] is explained. We first build the snapshot matrix $\boldsymbol{S}$ (cf. Eq. (A.3)). The POD basis functions (representing the spatial distributions of the modes) can then be extracted by solving the eigenvalue problem for the *correlation matrix* $\boldsymbol{C} = \boldsymbol{S}^T\boldsymbol{S} \in \mathbb{R}^{n \times n}$. The eigenvalue problem can be written in the following form

$$\boldsymbol{C}\boldsymbol{\Xi} = \boldsymbol{\Lambda}\boldsymbol{\Xi}, \tag{A.8}$$

where $\boldsymbol{\Lambda} = \text{diag}\{\lambda_1, \lambda_2, ..., \lambda_n\} \in \mathbb{R}^{n \times n}$ is the diagonal matrix of eigenvalues in descending order ($\lambda_1 \geq \lambda_2 \geq \dots \lambda_n$), and $\boldsymbol{\Xi} \in \mathbb{R}^{n \times n}$ is the eigenvector matrix. It is noted that the number of non-zero eigenvalues of $\boldsymbol{C}$ is $r_{\boldsymbol{C}} \leq n$, where $r_{\boldsymbol{C}}$ is the rank of $\boldsymbol{C}$. The POD basis matrix can then be defined as

$$\boldsymbol{\Phi} = \boldsymbol{S}\boldsymbol{\Xi} \in \mathbb{R}^{m \times n}. \tag{A.9}$$

The POD bases must be *normalized* so that they are orthonormal. To this end, the orthogonal POD basis matrix, $\boldsymbol{\Phi}' \in \mathbb{R}^{m \times n}$, is defined as

$$\Phi'_{ij} = \frac{1}{\|\Phi_j\|}\Phi_{ij} = \frac{1}{\sqrt{\lambda_j}}\Phi_{ij}, \tag{A.10}$$

where the second equality is true if the eigendecomposition subroutine returns normalized eigenvectors. For the purposes of efficient dimensionality reduction, a reduced basis, $\boldsymbol{\Psi} \in \mathbb{R}^{m \times r}$, is built by choosing the first r columns of $\boldsymbol{\Phi}'$, where $r < n$. The coefficients of this reduced basis (representing the time-varying coefficients of the retained modes) can be extracted as

$$\boldsymbol{A} = \boldsymbol{\Psi}^T \boldsymbol{S} \in \mathbb{R}^{r \times n}. \tag{A.11}$$

The POD approximation, $\widetilde{\boldsymbol{S}} \approx \boldsymbol{S}$, is then obtained via

$$\widetilde{\boldsymbol{S}} = \left[\widetilde{\boldsymbol{f}}^1 \mid \widetilde{\boldsymbol{f}}^2 \mid \ldots \mid \widetilde{\boldsymbol{f}}^n\right] = \boldsymbol{\Psi}\boldsymbol{A} \in \mathbb{R}^{m \times n}, \tag{A.12}$$

where $\widetilde{\boldsymbol{f}}^i \in \mathbb{R}^m$ is the POD approximation to $\hat{\boldsymbol{f}}^i$. The captured modal energy, e, can be defined as

$$e = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{n} \lambda_i}, \tag{A.13}$$

which indicates that as r is increased, more energy is retained, and consequently, the reconstruction accuracy is increased. Generally, r is chosen as the number of modes needed to ensure that 99% of the modal energy is captured (i.e., $e \approx 0.99$).

A.3 Relationships Between the Two Modal Decompositions

Mathematically, the SVD- and EVD-based PODs are closely related. One can show that the rank of the correlation matrix $\boldsymbol{C}$ is the same as that of the corresponding snapshot matrix $\boldsymbol{S}$, i.e., $r_{\boldsymbol{C}} = r_{\boldsymbol{S}}$. Furthermore, the eigenvalues λ_i of $\boldsymbol{C}$ are equal to the square of the singular values σ_i of $\boldsymbol{S}$, i.e., $\lambda_i = \sigma_i^2$ [251]. On this note, that is why we use σ_i^2 in the definition of the captured modal energy e in Eq. (A.6), as opposed to only λ_i in Eq. (A.13).

APPENDIX B
PROOF OF THE FOUR FUNDAMENTAL EQUATIONS OF BACKPROPAGATION

Proof of Eq. (2.19). Starting from the definition (2.18) of the error $\delta_j^{(L)}$ in the output layer, we have

$$\begin{aligned}
\delta_j^{(L)} &= \frac{\partial J_{\boldsymbol{x}}}{\partial a_j^{(L)}} \\
&= \sum_k \frac{\partial J_{\boldsymbol{x}}}{\partial h_k^{(L)}} \frac{\partial h_k^{(L)}}{\partial a_j^{(L)}} \quad \text{(chain rule)} \\
&= \frac{\partial J_{\boldsymbol{x}}}{\partial h_j^{(L)}} \frac{\partial h_j^{(L)}}{\partial a_j^{(L)}} \quad \text{(because } a_j^{(L)} \text{ doesn't affect } h_k^{(L)} \text{ for } j \neq k\text{)} \\
&= \frac{\partial J_{\boldsymbol{x}}}{\partial h_j^{(L)}} \sigma'(a_j^{(L)}) \quad \text{(recall that } h_j^{(L)} = \sigma(a_j^{(L)}\text{),}
\end{aligned} \tag{B.1}$$

which is just Eq. (2.19), in component form. □

Proof of Eq. (2.20). Starting from the definition (2.18) of the error $\delta_j^{(l)}$ at a layer $l \in \{1, 2, \ldots, L-1\}$, we have

$$\begin{aligned}
\delta_j^{(l)} &= \frac{\partial J_{\boldsymbol{x}}}{\partial a_j^{(l)}} \\
&= \sum_k \frac{\partial J_{\boldsymbol{x}}}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} \quad \text{(chain rule)} \\
&= \sum_k \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} \delta_k^{(l+1)} \quad \text{(use the definition of } \delta_k^{(l+1)}\text{).}
\end{aligned} \tag{B.2}$$

To evaluate the first term on the last line, note that

$$\begin{aligned}
a_k^{(l+1)} &= \sum_j W_{kj}^{(l+1)} h_j^{(l)} + b_k^{(l+1)} \\
&= \sum_j W_{kj}^{(l+1)} \sigma(a_j^{(l)}) + b_k^{(l+1)}.
\end{aligned} \tag{B.3}$$

Taking the derivative, we obtain

$$\frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} = W_{kj}^{(l+1)} \sigma'(a_j^{(l)}). \tag{B.4}$$

Substituting back into Eq. (B.2), we obtain

$$\delta_j^{(l)} = \sum_k W_{kj}^{(l+1)} \delta_k^{(l+1)} \sigma'(a_j^{(l)}), \tag{B.5}$$

which is just Eq. (2.20), in component form. □

Proof of Eq. (2.21). Starting from the left-hand side of the equation, we have

$$\begin{aligned}
\frac{\partial J_{\boldsymbol{x}}}{\partial b_j^{(l)}} &= \sum_k \frac{\partial J_{\boldsymbol{x}}}{\partial a_k^{(l)}} \frac{\partial a_k^{(l)}}{\partial b_j^{(l)}} \quad \text{(chain rule)} \\
&= \frac{\partial J_{\boldsymbol{x}}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial b_j^{(l)}} \quad \text{(because } b_j^{(l)} \text{ doesn't affect } a_k^{(l)} \text{ for } j \neq k\text{)} \\
&= \delta_j^{(l)} \times 1 \\
&= \delta_j^{(l)},
\end{aligned} \tag{B.6}$$

which is just Eq. (2.21), in component form. □

Proof of Eq. (2.22). Starting from the left-hand side of the equation, we have

$$\begin{aligned}
\frac{\partial J_{\boldsymbol{x}}}{\partial w_{jk}^{(l)}} &= \sum_m \frac{\partial J_{\boldsymbol{x}}}{\partial a_m^{(l)}} \frac{\partial a_m^{(l)}}{\partial w_{jk}^{(l)}} \quad \text{(chain rule)} \\
&= \frac{\partial J_{\boldsymbol{x}}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial W_{jk}^{(l)}} \quad \text{(because } W_{jk}^{(l)} \text{ doesn't affect } a_m^{(l)} \text{ for } j \neq m\text{)} \\
&= \delta_j^{(l)} \frac{\partial a_j^{(l)}}{\partial W_{jk}^{(l)}} \quad \text{(use the definition of } \delta_j^{(l)}\text{)} \\
&= \delta_j^{(l)} h_k^{(l-1)} \quad \text{(because } a_j^{(l)} = \sum_{k'} W_{jk'}^{(l)} h_{k'}^{(l-1)} + b_j^{(l)}\text{)},
\end{aligned} \tag{B.7}$$

which is just Eq. (2.22). □

APPENDIX C
THERMODYNAMIC RELATIONS BASED ON THE SRK EOS

C.1 Mixing Rules

The parameters αa and b are derived from the extended corresponding-states principle [150], and are evaluated as follows

$$\alpha a = \sum_{i=1}^{N} \sum_{j=1}^{N} \chi_i \chi_j \alpha_{ij} a_{ij}, \tag{C.1}$$

$$b = \sum_{i=1}^{N} \chi_i b_i, \tag{C.2}$$

where χ_i is the mole fraction of the ith species. The cross parameter $\alpha_{ij} a_{ij}$ is given by

$$\alpha_{ij} a_{ij} = \sqrt{\alpha_i \alpha_j a_i a_j}(1 - \kappa_{ij}), \tag{C.3}$$

where κ_{ij} is the binary interaction coefficient that accounts for molecular energy and volumetric effects, and is set to zero here. The terms a_i, b_i and α_i can be determined from the following relationships [253, 152]

$$a_i = 0.42747 \frac{R_u^2 T_{\mathrm{c},i}^2}{p_{\mathrm{c},i}}, \tag{C.4}$$

$$b_i = 0.08664 \frac{R_u T_{\mathrm{c},i}}{p_{\mathrm{c},i}}, \tag{C.5}$$

$$\alpha_i = \left(1 + S_i(1 - \sqrt{T_{\mathrm{r},i}})\right)^2, \tag{C.6}$$

where $T_{\mathrm{r},i}$ is the reduced temperature, and S_i is a function of the acentric factor w_i. These terms are given by

$$T_{\mathrm{r},i} = \frac{T}{T_{\mathrm{c},i}}, \tag{C.7}$$

$$S_i = 0.48508 + 1.5517 w_i - 0.15613 w_i^2. \tag{C.8}$$

C.2 Internal Energy, Enthalpy, Specific Heats, and Speed of Sound

The mass-specific internal energy e is obtained from sum of the low-pressure ideal-gas value and a departure function that accounts for the deviation from the ideal-gas behavior. The departure function can be obtained using fundamental thermodynamic relationships like the Gibbs equation and Maxwell's relations [167], which gives

$$e(T, p, Y_i) = e_0 + \int_{\rho_0}^{\rho} \left(\frac{p}{\rho^2} - \frac{T}{\rho^2} A_T \right) d\rho, \tag{C.9}$$

where the subscript 0 indicates the ideal-gas state, and the integral term represents the departure function. Equation (C.9) can be integrated analytically using a specific EOS (here, SRK EOS) to give

$$e = e_0 + \frac{T^2}{bW}\left(\frac{\partial(a\alpha/T)}{\partial T}\right)_{Y_i} \ln\left(1 + \frac{b\rho}{W}\right), \tag{C.10}$$

where an expression for $(\partial(a\alpha)/\partial T)_{Y_i}$ is given in Appendix C.3. The mass-specific enthalpy h is obtained through the caloric equation of state

$$\rho h = \rho e + p. \tag{C.11}$$

The mass-specific heat capacity at constant volume c_v is obtained using

$$c_v = c_{v,0} + \frac{T}{bW}\left(\frac{\partial^2(a\alpha)}{\partial T^2}\right)_{Y_i} \ln\left(1 + \frac{b\rho}{W}\right), \tag{C.12}$$

where an expression for $(\partial^2(a\alpha)/\partial T^2)_{Y_i}$ is given in Appendix C.3. The mass-specific heat capacity at constant pressure c_p can be obtained from c_v as follows [167]

$$c_p = c_v + \frac{\frac{T}{\rho^2}A_T^2}{A_\rho}. \tag{C.13}$$

The speed of sound a is evaluated as

$$a^2 = \left(\frac{\partial p}{\partial \rho}\right)_{s,Y_i} = \gamma A_\rho, \tag{C.14}$$

where s is the entropy of the mixture and γ is the ratio of specific heats ($\gamma = c_p/c_v$). Expressions for A_T and A_ρ are given in Appendix C.3.

C.3 Partial Derivatives

The parameter $a\alpha$ in the mixing rules (see Eq. (C.1)) is function of temperature and chemical composition. Its first derivative with respect to temperature is given by

$$\left(\frac{\partial(a\alpha)}{\partial T}\right)_{Y_i} = \sum_{i=1}^{N}\sum_{j=1}^{N} \chi_i\chi_j\sqrt{a_i a_j}\frac{d(\sqrt{\alpha_i\alpha_j})}{dT}, \tag{C.15}$$

where

$$\frac{d(\sqrt{\alpha_i\alpha_j})}{dT} = \frac{1}{2}\left(\frac{\alpha_i}{\alpha_j}\right)^{0.5}\frac{d\alpha_j}{dT} + \frac{1}{2}\left(\frac{\alpha_j}{\alpha_i}\right)^{0.5}\frac{d\alpha_i}{dT}, \tag{C.16}$$

$$\frac{d\alpha_i}{dT} = -\frac{S_i}{\sqrt{TT_{c,i}}}\left[1 + S_i\left(1 - \sqrt{\frac{T}{T_{c,i}}}\right)\right]. \tag{C.17}$$

The second derivative of $a\alpha$ with respect to temperature is evaluated as

$$\left(\frac{\partial^2(a\alpha)}{\partial T^2}\right)_{Y_i} = \sum_{i=1}^{N}\sum_{j=1}^{N}\chi_i\chi_j\sqrt{a_i a_j}\frac{d^2(\sqrt{\alpha_i\alpha_j})}{dT^2}, \tag{C.18}$$

where

$$\begin{aligned}\frac{d^2(\sqrt{\alpha_i\alpha_j})}{dT^2} = &\frac{1}{2}\left(\frac{1}{\alpha_i\alpha_j}\right)^{0.5}\frac{d\alpha_i}{dT}\frac{d\alpha_j}{dT} - \frac{1}{4}\left(\frac{\alpha_i}{\alpha_j^3}\right)^{0.5}\left(\frac{d\alpha_j}{dT}\right)^2 \\ &- \frac{1}{4}\left(\frac{\alpha_j}{\alpha_i^3}\right)^{0.5}\left(\frac{d\alpha_i}{dT}\right)^2 + \frac{1}{2}\left(\frac{\alpha_i}{\alpha_j}\right)^{0.5}\frac{d^2\alpha_j}{dT^2} + \frac{1}{2}\left(\frac{\alpha_j}{\alpha_i}\right)^{0.5}\frac{d^2\alpha_i}{dT^2},\end{aligned} \tag{C.19}$$

$$\frac{d^2\alpha_i}{dT^2} = \frac{1}{2}\frac{S_i^2}{TT_{\mathrm{c},i}} + \frac{1}{2}\frac{S_i}{\sqrt{T^3T_{\mathrm{c},i}}}\left[1 + S_i\left(1 - \sqrt{\frac{T}{T_{\mathrm{c},i}}}\right)\right]. \tag{C.20}$$

The parameter $\alpha_{ij}a_{ij}$ depends on temperature only, and its first derivative is given by

$$\frac{d(\alpha_{ij}a_{ij})}{dT} = \sqrt{a_i a_j}\frac{d(\sqrt{\alpha_i\alpha_j})}{dT}. \tag{C.21}$$

The partial derivatives A_T and A_ρ are given by

$$A_T = \frac{\rho R_u}{W - b\rho} - \frac{1}{W}\left(\frac{\partial(a\alpha)}{\partial T}\right)_{Y_i}\frac{\rho^2}{(W + b\rho)}, \tag{C.22}$$

$$A_\rho = \frac{WR_uT}{(W - b\rho)^2} - \frac{a\alpha}{W}\frac{\rho(2W + b\rho)}{(W + b\rho)^2}. \tag{C.23}$$

C.4 Preconditioning Terms

The partial derivative $(\partial p/\partial\rho_i)_{T,\rho_{j\neq i}}$ that appears in the preconditioning terms A_{Y_k} and B_{Y_k} (see Eqs. (5.10) and (5.11)) can be expressed as

$$\begin{aligned}\left(\frac{\partial p}{\partial\rho_i}\right)_{T,\rho_{j\neq i}} = &\frac{WR_uT}{W_i(W - b\rho)^2}(W + \rho(b_i - b)) - \frac{2\rho\sum_j\chi_j a_{ij}\alpha_{ij}}{W_i(W + b\rho)} \\ &+ \frac{a\alpha\rho^2 b_i}{W_i(W + b\rho)^2}.\end{aligned} \tag{C.24}$$

The partial-density internal energy of the ith species, $\breve{e}_i$, which appears in the term B_{Y_k} (see Eq. (5.11)), can be evaluated by substituting Eq. (C.10) into Eq. (5.12), as follows

$$\begin{aligned}\breve{e}_i = e_{i,0} &+ \frac{2}{bW_i}\left[\sum_j\chi_j\left(T\frac{d(a_{ij}\alpha_{ij})}{dT} - a_{ij}\alpha_{ij}\right)\right]\ln\left(1 + \frac{b\rho}{W}\right) \\ &+ \frac{b_i}{bW_i}\left[T\left(\frac{\partial(a\alpha)}{\partial T}\right)_{Y_j} - a\alpha\right]\left[\frac{\rho}{W + b\rho} - \frac{1}{b}\ln\left(1 + \frac{b\rho}{W}\right)\right].\end{aligned} \tag{C.25}$$

APPENDIX D
PARAMETRIC ANALYSIS FOR NEURAL NETWORK DESIGN FOR 0D THERMODYNAMIC EXAMPLE

A parametric analysis was conducted to select the network topology and hyperparameters of the DFNN model in the canonical example in Sec. 5.4.5. The effect of the regularization parameter, λ_{reg}, on the MSE loss for training and validation data is quantified in Table D.1. Three values of λ_{reg} are reported: 0 – no regularization; 0.001 – small regularization; 0.005 – large regularization. For the case $\lambda_{\text{reg}} = 0$, a relatively small MSE is observed on the training data, but at the expense of a larger MSE on the validation data. This is indicative of overfitting. Adding regularization significantly improves the prediction accuracy on the validation data as seen for the case $\lambda_{\text{reg}} = 0.001$, while compromising little on the training prediction accuracy, as is evident from the small increase in MSE. However, for large values of λ_{reg} such as in the third case, a strong bias may be induced and worsen the prediction accuracy. Consequently, $\lambda_{\text{reg}} = 0.001$ was chosen in this study.

Table D.1: Effect of the regularization parameter on the MSE loss for training and validation data.

	$\lambda_{\text{reg}} = 0$	$\lambda_{\text{reg}} = 0.001$	$\lambda_{\text{reg}} = 0.005$
MSE training	0.0120	0.0125	0.0490
MSE validation	0.0140	0.0128	0.0491

The variation of MSE loss with the number of epochs for different activation functions is shown in Fig. D.1. The tanh and ReLU functions outperform the sigmoid function. The performance of the tanh and ReLU functions are comparable, with the ReLU function providing slightly more accurate predictions. For this reason, the ReLU function was chosen in this work.

Four different architectures with varying numbers of hidden layers and neurons per hidden layer were compared, as seen in Table D.2. Going from one hidden layer (DFNN1) to two (DFNN2) increases the R^2-score, and consequently, the accuracy of the model. Similarly, going from two hidden layers to four (DFNN3) also improves the R^2-score. In addition, increasing the number of neurons per hidden layer helps in general to achieve better accuracy, as evidenced by the higher R^2-score of DFNN4 compared to that of DFNN3. Some caution is suggested, however, in considering this trend, as a more complicated model could eventually lead to overfitting. While DFNN4 seems the best choice in terms of accuracy, a look into the running times of all the neural networks reveals that DFNN4 is very time-consuming (see Fig. D.2). It is 10 times slower than the brute force approach. On the other hand, DFNN3 can provide a speedup of 2.4 times in property evaluation, and therefore, is the best compromise between speed and accuracy.

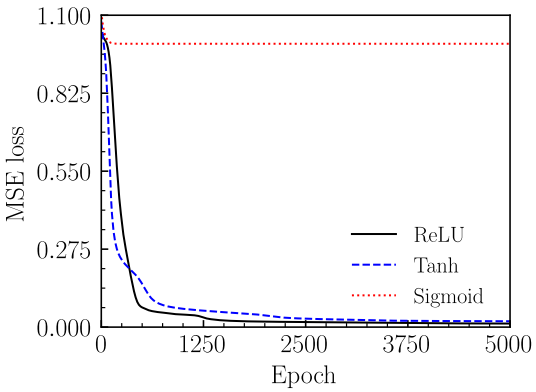


Figure D.1: Variation of the MSE loss with the number of epochs on training data for different activation functions.

Table D.2: Effect of the network architecture on the R^2-score for training and validation data.

	DFNN1	DFNN2	DFNN3	DFNN4
Architecture, $\mathscr{H}$	$[9]^T$	$[9 \quad 9]^T$	$[9 \quad 9 \quad 9 \quad 9]^T$	$[100 \quad 100 \quad 100 \quad 9]^T$
R^2-score training	0.980	0.985	0.988	0.993
R^2-score validation	0.978	0.984	0.987	0.992

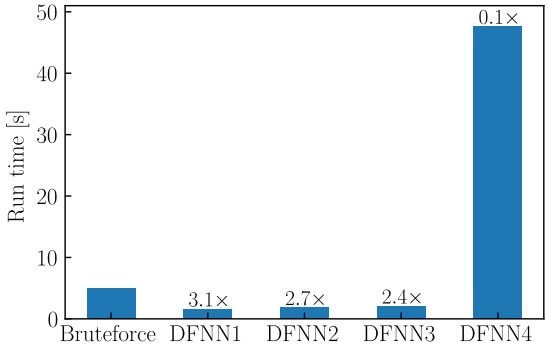


Figure D.2: Comparison of the serial running time for output generation between the brute force approach and various DFNN models. The time indicated corresponds to 1,000,000 executions of each model.

APPENDIX E
DOE STUDIES FOR A-M1 INJECTOR PROBLEM

E.1 Description of Cases from the DoE Study S60

In the DoE study "S60", 60 representative samples were identified using a variant of the LHS [224]. These samples are highlighted in Fig. E.1 and described in Table E.1. Out of the 60 samples (or CFD simulations), 5 cases (Cases $32, 44, 51, 52, 59$) were randomly selected and set aside for testing to ensure that the emulator, including its constituents (i.e. the autoencoder and regression model), generalize well to unseen data. The remaining data (i.e., 55 cases) were used for training and validation of the neural networks, with a random split of 80%-20%. The datasets are shown in Table E.2.

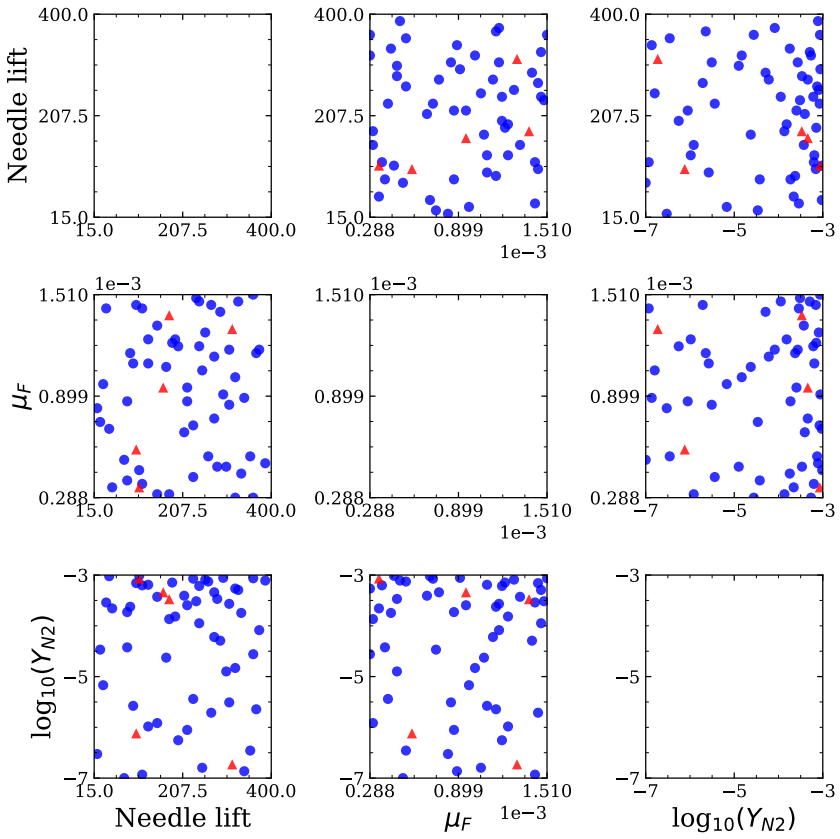


Figure E.1: 2D projections of the sample points from the DoE study S60. Training and validation cases are represented in blue circles, whereas testing cases (i.e., Cases) are shown using red triangle symbols.

Table E.1: Description of the 60 cases from the DoE study S60. The fuel viscosity, μ_F, is specified at 323 K.

Case	Needle lift [μm]	μ_F [N-s/m^2]	Y_{N_2} [-]
1	93.31	1.158e-03	2.374e-04
2	80.26	5.158e-04	1.000e-07
3	112.88	4.537e-04	9.661e-04
4	230.34	7.229e-04	8.475e-04
5	386.95	4.951e-04	7.797e-04
6	184.66	1.220e-03	7.119e-04
7	41.10	1.427e-03	2.882e-04
8	243.39	1.199e-03	6.102e-04
9	360.85	1.510e-03	8.644e-04
10	276.02	7.644e-04	4.577e-04
11	308.64	1.179e-03	2.713e-04
12	295.59	9.094e-04	8.814e-04
13	152.03	1.324e-03	3.729e-04
14	86.78	8.679e-04	1.865e-04
15	178.13	3.087e-04	1.357e-04
16	282.54	4.744e-04	3.390e-04
17	217.28	9.508e-04	2.543e-04
18	119.40	3.708e-04	6.272e-04
19	236.86	1.489e-03	3.052e-04
20	256.44	1.282e-03	8.136e-04
21	262.96	5.365e-04	7.458e-04
22	54.15	3.501e-04	2.204e-04
23	132.45	1.096e-03	6.441e-04
24	289.06	1.406e-03	5.094e-05
25	47.62	7.022e-04	9.492e-04
26	321.69	2.880e-04	5.424e-04
27	328.22	1.469e-03	5.085e-04
28	191.18	1.241e-03	1.526e-04
29	106.35	1.448e-03	6.949e-04
30	28.05	7.437e-04	3.399e-05

Case	Needle lift [μm]	μ_F [N-s/m^2]	Y_{N_2} [-]
31	249.92	1.054e-03	1.597e-07
32	178.14	1.386e-03	3.353e-04
33	302.12	4.744e-04	1.264e-05
34	99.83	1.096e-03	2.653e-06
35	217.29	8.679e-04	8.895e-07
36	373.90	1.179e-03	8.227e-05
37	171.61	1.075e-03	2.360e-05
38	334.75	4.330e-04	1.796e-04
39	276.02	1.137e-03	6.021e-05
40	341.27	8.886e-04	1.366e-07
41	34.58	9.715e-04	6.769e-06
42	21.53	8.265e-04	2.982e-07
43	132.46	1.241e-03	1.040e-06
44	315.17	1.303e-03	1.867e-07
45	321.69	1.013e-03	1.477e-05
46	152.03	3.087e-04	1.215e-06
47	119.41	1.427e-03	1.169e-07
48	269.49	1.448e-03	1.941e-06
49	86.78	3.916e-04	3.769e-05
50	360.85	2.880e-04	2.759e-05
51	106.36	5.780e-04	7.609e-07
52	112.88	3.501e-04	8.555e-04
53	354.32	5.365e-04	3.486e-07
54	210.76	6.815e-04	3.919e-04
55	308.64	8.472e-04	3.101e-06
56	230.34	4.123e-04	3.625e-06
57	243.39	1.469e-03	1.124e-04
58	197.71	1.199e-03	5.569e-07
59	165.08	9.508e-04	4.582e-04
60	367.37	1.158e-03	2.270e-06

Table E.2: Dataset sizes from the DoE study S60 used for formulating and testing the emulator and its constituents (i.e., the autoencoder and regression model).

Set	Description	Number of snapshots
Training set	80% of the snapshots from Cases 1-31, 33-43, 45-50, 53-58, 60	3,564
Validation set	20% of the snapshots from Cases 1-31, 33-43, 45-50, 53-58, 60	891
Test set	All of the snapshots from Cases 32, 44, 51, 52, 59	405

E.2 Description of Cases from the DoE Study S36

DoE Study S36 comprises a subset of the cases from DoE Study S60;[1] these are Cases 1-30, 32, 52, 36, 31, 54 and 59. In DoE Study S36, Cases 32, 52, 36, 31, 54 and 59 were renamed as Cases 31a, 32a, 33a, 34a, 35a and 36a, respectively, for convenience. The design parameters for these 36 cases are highlighted in Fig. E.2 and described in Table E.3.

The snapshots from the 36 cases in DoE Study S36 were divided into 4 sets, as shown in Table E.4. First, the snapshots from Cases 1-30 were collected and randomly split into three separate groups, where 70% of the snapshots were used for training, 20% for validation, and 10% for testing the Univ-AE model (cf. Sec. 7.3.3.3). These groups are referred to as the training set, validation set, and test set $\mathbb{T}_1$, respectively. Finally, a fourth group, using the snapshots from Cases 31a-36a and referred to as the test set $\mathbb{T}_2$, was considered.

[1] As noted above, the data for Study S36 were collected in the first phase of the project, with only 36 CFD simulations, while the second phase comprised 24 more simulations, bringing the total number of available samples to 60. The cases in DoE Study S36 were used to analyze the internal flow dynamics and to compare the POD to the autoencoder method, and were published as AAS publication [211]. The cases from DoE Study S60 were used to construct and evaluate the final emulator framework, and were published as AIAA and SAE publications [212, 236].

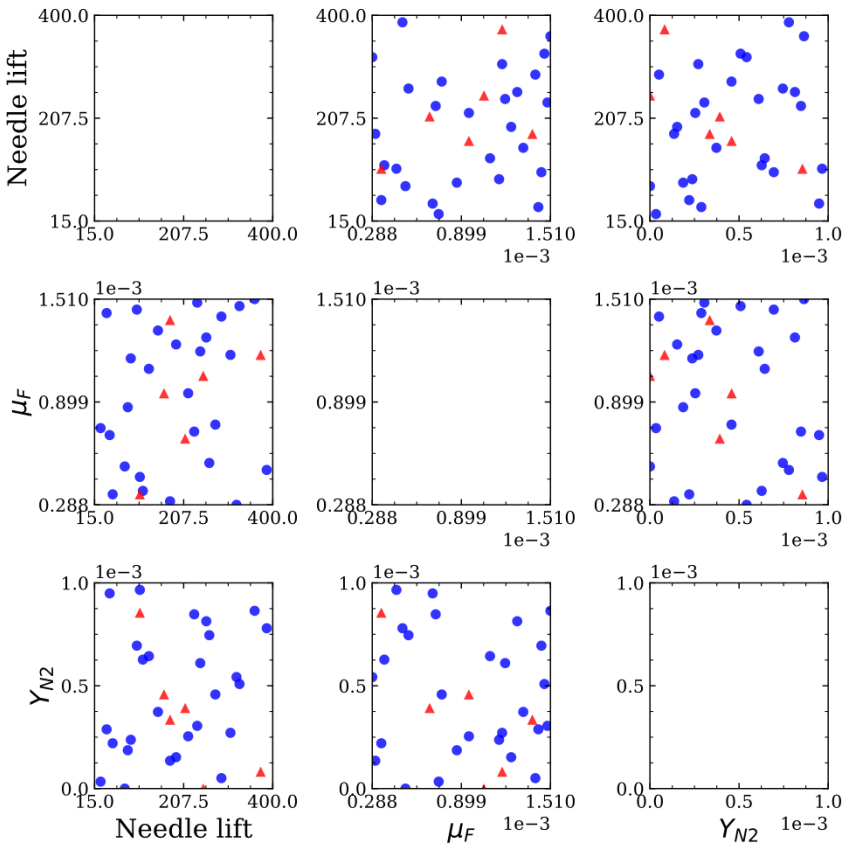


Figure E.2: 2D projections of the sample points from the DoE study S36. Training and validation cases are represented in blue circles, whereas testing cases (i.e., Cases) are shown using red triangle symbols.

Table E.3: Description of the 36 cases from the DoE study S36. The fuel viscosity, μ_F, is specified at 323 K.

Case	Needle lift [μm]	μ_F [N-s/m^2]	Y_{N_2} [-]
1	93.31	1.158e-03	2.374e-04
2	80.26	5.158e-04	1.000e-07
3	112.88	4.537e-04	9.661e-04
4	230.34	7.229e-04	8.475e-04
5	386.95	4.951e-04	7.797e-04
6	184.66	1.220e-03	7.119e-04
7	41.10	1.427e-03	2.882e-04
8	243.39	1.199e-03	6.102e-04
9	360.85	1.510e-03	8.644e-04
10	276.02	7.644e-04	4.577e-04
11	308.64	1.179e-03	2.713e-04
12	295.59	9.094e-04	8.814e-04
13	152.03	1.324e-03	3.729e-04
14	86.78	8.679e-04	1.865e-04
15	178.13	3.087e-04	1.357e-04
16	282.54	4.744e-04	3.390e-04
17	217.28	9.508e-04	2.543e-04
18	119.40	3.708e-04	6.272e-04
19	236.86	1.489e-03	3.052e-04
20	256.44	1.282e-03	8.136e-04
21	262.96	5.365e-04	7.458e-04
22	54.15	3.501e-04	2.204e-04
23	132.45	1.096e-03	6.441e-04
24	289.06	1.406e-03	5.094e-05
25	47.62	7.022e-04	9.492e-04
26	321.69	2.880e-04	5.424e-04
27	328.22	1.469e-03	5.085e-04
28	191.18	1.241e-03	1.526e-04
29	106.35	1.448e-03	6.949e-04
30	28.05	7.437e-04	3.399e-05
31a	178.14	1.386e-03	3.353e-04
32a	112.88	3.501e-04	8.555e-04
33a	373.90	1.179e-03	8.227e-05
34a	249.92	1.054e-03	1.597e-07
35a	210.76	6.815e-04	3.919e-04
36a	165.08	9.508e-04	4.582e-04

Table E.4: Dataset sizes from the DoE study S36 used for formulating and testing the Univ-AE.

Set	**Description**	**Number of snapshots**
Training set	70% of the snapshots from Cases 1-30	1,701
Validation set	20% of the snapshots from Cases 1-30	486
Test set $\mathbb{T}_1$	10% of the snapshots from Cases 1-30	243
Test set $\mathbb{T}_2$	All of the snapshots from Cases 31a-36a	486

REFERENCES

[1] A. Forrester, A. Sobester, and A. Keane, *Engineering Design via Surrogate Modeling*. John Wiley & Sons, 2008.

[2] S. Koziel and L. Leifsson (Eds.), *Surrogate-Based Modeling and Optimization*. Springer, 2013.

[3] I. Kalashnikova, S. Arunajatesan, M. F. Barone, B. G. van Bloemen Waanders, and J. A. Fike, "Reduced order modeling for prediction and control of large-scale systems," Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), Tech. Rep.

[4] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering*. Cambridge University Press, 2019.

[5] J. Oefelein, G. Lacaze, R. Dahms, A. Ruiz, and A. Misdariis, "Effects of real-fluid thermodynamics on high-pressure fuel injection processes," *SAE International Journal of Engines*, vol. 7, pp. 1125 –1136, 2014.

[6] T. Daniel, F. Casenave, N. Akkari, and D. Ryckelynck, "Model order reduction assisted by deep neural networks (ROM-Net)," *Advanced Modeling and Simulation in Engineering*, vol. 7, pp. 1–27, 2020.

[7] P. J. Milan, X. Wang, and V. Yang, "Three-dimensional investigation of fluid dynamics in a rocket engine injector at supercritical pressure," *ILASS-Americas 2021*, pp. 1–10, 2021.

[8] X. Wang, Y. Wang, and V. Yang, "Three-dimensional flow dynamics and mixing in a gas-centered liquid-swirl coaxial injector at supercritical pressure," *Physics of Fluids*, vol. 31, pp. 1–14, 2019.

[9] M. Frangos, Y. Marzouk, K. Willcox, and B. van Bloemen Waanders, "Surrogate and reduced-order modeling: A comparison of approaches for large-scale statistical inverse problems," in *Large-Scale Inverse Problems and Quantification of Uncertainty*. John Wiley Sons, Ltd, 2010, ch. 7, pp. 123–149.

[10] M. S. Eldred and D. M. Dunlavy, "Formulations for surrogate-based optimization with data fit, multifidelity, and reduced-order model," *AIAA 2006-7117*, pp. 1–20, 2006.

[11] C. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[12] N. Zhang, J. Xiong, J. Zhong, and K. Leatham, "Gaussian process regression method for classification for high-dimensional data with limited samples," in *2018 Eighth International Conference on Information Science and Technology (ICIST)*, 2018, pp. 358–363.

[13] G. Pang and G. E. Karniadakis, "Physics-informed learning machines for partial differential equations: Gaussian processes versus neural networks," in *Emerging Frontiers in Nonlinear Science*, P. G. Kevrekidis, J. Cuevas-Maraver, and A. Saxena, Eds. Cham: Springer International Publishing, 2020, pp. 323–343.

[14] P. Holmes, J. L. Lumley, and G. Berkooz, *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, ser. Cambridge Monographs on Mechanics. Cambridge University Press, 1996.

[15] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," *Annual Review of Fluid Mechanics*, vol. 25, pp. 539–575, 1993.

[16] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, 2010.

[17] R. Maulik, B. Lusch, and P. Balaprakash, "Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders," *Physics of Fluids*, vol. 33, no. 3, pp. 1–20, 2021.

[18] T. Baltrusaitis, P. Robinson, and L. Morency, "Constrained local neural fields for robust facial landmark detection in the wild," in *2013 IEEE International Conference on Computer Vision Workshops*, 2013, pp. 354–361.

[19] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, 2006.

[20] Y. Wu *et al.*, *Google's neural machine translation system: Bridging the gap between human and machine translation*, 2016. arXiv: 1609.08144.

[21] G. Cybenko, "Approximation by superpositions of a sigmoid function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[23] L. Hardesty, *Explained: Neural networks*, https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414, 2017.

[24] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017.

[25] Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, https://www.tensorflow.org/, Software available from tensorflow.org, 2015.

[26] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[27] J. N. Kutz, "Deep learning in fluid dynamics," *Journal of Fluid Mechanics*, vol. 814, pp. 1–4, 2017.

[28] R. Vinuesa and S. L. Brunton, "The potential of machine learning to enhance computational fluid dynamics," *arXiv preprint arXiv:2110.02085*, pp. 1–13, 2021.

[29] N. Omata and S. Shirayama, "A novel method of low-dimensional representation for temporal behavior of flow fields using deep autoencoder," *AIP Advances*, vol. 9, pp. 1–14, 2019.

[30] T. Murata, K. Fukami, and K. Fukagata, "Nonlinear mode decomposition with machine learning for fluid dynamics," *arXiv preprint arXiv:1906.04029*, pp. 1–15, 2019.

[31] Y. Seong, C. Park, J. Choi, and I. Jang, "Surrogate model with a deep neural network to evaluate gas–liquid flow in a horizontal pipe," *Energies*, vol. 13, pp. 1–12, 2020.

[32] J. Xu and K. Duraisamy, "Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 372, pp. 1–36, 2020.

[33] K. Urban and A. Patera, "A new error bound for reduced basis approximation of parabolic partial differential equations," *Comptes Rendus Mathematique*, vol. 350, pp. 203–207, 2012.

[34] M. Yano, "A space-time Petrov-Galerkn certified reduced basis method: Application to the Boussineq equations," *SIAM Journal on Scientific Computing*, vol. 36, A232–A266, 2013.

[35] K. Urban and A. Patera, "An improved error bound for reduced basis approximation of linear parabolic problems," *Mathematics of Computation*, vol. 83, pp. 1599–1615, 2014.

[36] M. Yano, A. Patera, and K. Urban, "A space-time hp-interpolation-based certified reduced basis method for Burgers' equation," *Math. Models Methods Appl. Sci.*, vol. 24, pp. 1903–1935, 2014.

[37] C. W. Rowley, T. Colonius, and R. M. Murray, "Model reduction for compressible flows using POD and Galerkin projection," *Physica D: Nonlinear Phenomena*, vol. 189, pp. 115–129, 2004.

[38] M. Maumann, P. Benner, and J. Heiland, "Space-time Galerkin POD with application in optimal control of semilinear partial differential equations," *SIAM Journal on Scientific Computing*, vol. 40, A1611–A1641, 2018.

[39] P. C. Constantine and Q. Wang, "Residual minimizing model interpolation for parametrized nonlinear dynamical systems," *SIAM Journal on Scientific Computing*, vol. 34, A2118–A2144, 2012.

[40] K. Carlberg, C. Bou-Mosleh, and C. Farhat, "Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations," *International Journal for Numerical Methods in Engineering*, vol. 86, pp. 155–181, 2011.

[41] Y. Choi and K. Carlberg, "Space-time least-squares Petrov-Galerking projection for nonlinear model reduction," *SIAM Journal on Scientific Computing*, vol. 41, A26–A58, 2019.

[42] R. Chakir and Y. Maday, "A two-grid finite-element/reduced basis scheme for the approximation of the solution of parameter dependent P.D.E," *9e Colloque national en calcul des structures*, pp. 1–6, 2009.

[43] R. Chakir, P. Joly, Y. Maday, and P. Parnaudeau, "A non intrusive reduced basis method: Application to computational fluid dynamics," *2nd ECCOMAS Young Investigators Conference*, pp. 1–4, 2013.

[44] E. Grosjean and Y. Maday, "Error estimate of the non intrusive reduced basis method with finite volume schemes," *arXiv preprint arXiv:2103.11720*, pp. 1–20, 2021.

[45] M. G. Fernández-Godino, C. Park, N.-H. Kim, and R. T. Haftka, "Review of multi-fidelity models," *arXiv preprint arXiv:1609.07196v3*, pp. 1–46, 2017.

[46] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating scientific knowledge with machine learning for engineering and environmental systems," *arXiv preprint arXiv:2003.04919v5*, pp. 1–35, 2021.

[47] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Review Physics*, vol. 3, pp. 422–440, 2021.

[48] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[49] Q. Hernández, A. Badías, D. González, F. Chinesta, and E. Cueto, "Structure-preserving neural networks," *Journal of Computational Physics*, vol. 426, pp. 1–16, 2021.

[50] J. Zhang and X. Zhao, "Spatiotemporal wind field prediction based on physics-informed deep learning and lidar measurements," *Applied Energy*, vol. 288, pp. 1–13, 2021.

[51] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *Journal of Machine Learning Research*, vol. 18, pp. 1–43, 2018.

[52] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, pp. 1–25, 2020.

[53] Z. Zhang, Y. Shin, and G. E. Karniadakis, "GFINNs: Generic formalism informed neural networks for determinisic and stochastic dynamical systems," *arXiv preprint arXiv:2109.00092*, pp. 1–20, 2021.

[54] F. Emmert-Streib, Z. Yang, H. Feng, S. Tripathi, and M. Dehmer, "An introductory review of deep learning for prediction models with big data," *Frontiers in Artificial Intelligence*, vol. 3, pp. 1–23, 2020.

[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[56] C. C. Aggarwall, *Neural Networks and Deep Learning*. Springer International Publising AG, 2018.

[57] M. Nielsen, *Neural Networks and Deep learning*. Determination Press, 2015.

[58] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[59] A. Krizhevsky, *The cifar-10 dataset*, https://www.cs.toronto.edu/~kriz/cifar.html, 2009.

[60] L. Weng, *From autoencoder to Beta-VAE*, https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html, 2018.

[61] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos," *Pattern Recognition Letters*, vol. 105, pp. 13–22, 2018.

[62] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, pp. 1–43, 2013.

[63] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal Uncertainty Fuzziness Knowledge Based Systems*, vol. 6, pp. 107–116, 1998.

[64] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, pp. 1–14, 2013.

[65] I. Goodfellow *et al.*, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661v1*, pp. 1–9, 2014.

[66] A. Vaswani *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762v5*, pp. 1–15, 2017.

[67] Wikipedia, *Backpropagation*, https://en.wikipedia.org/wiki/Backpropagation, 2021.

[68] L. Prechelt, "Automatic early stopping using cross validation: Quantifying the criteria," *Neural Networks*, vol. 11, pp. 761–767, 1998.

[69] J. Brownlee, *Weight initialization for deep learning neural networks*, https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/, 2021.

[70] P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild, "DeepHyper: Asynchronous hyperparameter search for deep neural networks," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, 2018, pp. 42–51.

[71] Google, *Model search*, https://github.com/google/model_search, 2021.

[72] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, pp. 145–151, 1999.

[73] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[74] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, pp. 1–15, 2014.

[75] NVIDIA, *CuDNN*, https://developer.nvidia.com/cudnn.

[76] ——, *TensorRT*, https://developer.nvidia.com/tensorrt.

[77] F. Chollet *et al.*, *Keras*, https://github.com/fchollet/keras, 2015.

[78] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[79] A. Ng and K. Katanforoosh, *CS230 Deep learning. Section 5 (week 5): TensorFlow and Pytorch*, https://cs230.stanford.edu/section/5/, 2021.

[80] T. E. Oliphant, *Guide to NumPy*. Continuum Press, 2015.

[81] F. M. White, *Viscous Fluid Flow*. McGraw-Hill, 1991.

[82] J. D. Anderson, *Computational Fluid Dynamics*. McGraw-Hill, 1995.

[83] F. A. Williams, *Combustion Theory*. Benjam Commings, 1985.

[84] U. Piomelli, "Large-eddy simulation: Achievements and challenges," *Progress in Aerospace Sciences*, vol. 35, pp. 335–362, 1999.

[85] S. Goldstein, "Fluid mechanics in the first half of this century," *Annual Review of Fluid Mechanics*, vol. 1, pp. 1–29, 1969.

[86] P. Moin and K. Mahesh, "Direct numerical simulation: A tool in turbulence research," *Annual Review of Fluid Mechanics*, vol. 30, pp. 539–578, 1998.

[87] S. B. Pope, *Turbulent Flows*. Cambridge University Press, 2000.

[88] P. Sagaut, *Large Eddy Simulation for Incompressible Flows*. Springer, 2006.

[89] E. Garnier, N. Adams, and P. Sagaut, *Large Eddy Simulation for Compressible Flows*. Springer, 2009.

[90] G. Erlebacher, M. Y. Hussaini, C. G. Speziale, and T. A. Zang, "Toward the large-eddy simulation of compressible turbulent flows," *Journal of Fluid Mechanics*, vol. 238, pp. 155–185, 1992.

[91] E. Pomraning and C. J. Rutland, "A dynamic one-equation non-viscosity les model," *AIAA Journal*, vol. 40, pp. 659–701, 2002.

[92] P. Moin and J. Kim, "Numerical investigation of turbulent channel flow," *Journal of Fluid Mechanics*, vol. 118, pp. 341–377, 1982.

[93] W. Calhoon and S. Menon, "Subgrid modeling for reacting large eddy simulations," *AIAA 96-0561*, pp. 1–24, 1996.

[94] J. H. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics*. Springer-Verlag, 2002.

[95] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education, 2007.

[96] N. Zong, "Modeling and simulation of cryogenic fluid injection and mixing dynamics under supercritical conditions," *PhD Thesis, The Pennsylvania State University*, 2005.

[97] H. Huo, "Large-eddy simulation of supercritical fluid flow and combustion," *PhD Thesis, The Pennsylvania State University*, 2011.

[98] K. J. Richards, P. K. Senecal, and E. Pomraning, *CONVERGE (v3.0) Manual*, Convergent Science, Inc., 2020.

[99] Britannica, *Internal-combustion engine*, https://www.britannica.com/technology/internal-combustion-engine, 2021.

[100] J. B. Heywood, *Internal Combustion Engine Fundamentals*. McGraw-Hill, 1988.

[101] Wikipedia, *Four-stroke engine*, https://en.wikipedia.org/wiki/Four-stroke_engine, 2021.

[102] A. Khajepour, M. S. Fallah, and A. Goodarzi, *Electric and Hybrid Vehicles: Technologies, Modeling and Control - A Mechatronic Approach*. Wiley, 2014.

[103] DieselNet, *Diesel fuel injector nozzles*, https://dieselnet.com/tech/engine_fi_nozzle.php, 2017.

[104] J. Oefelein, R. N. Dahms, and G. M. L. Lacaze, "Detailed modeling and simulation of high-pressure fuel injection processes in diesel engines," *SAE International Journal of Engines*, vol. 5, pp. 1410–1419, 2012.

[105] P. Jenny, D. Roekaerts, and N. Beishuizen, "Modeling of turbulent dilute spray combustion," *Progress in Energy and Combustion Science*, vol. 38, no. 6, pp. 846–887, 2012.

[106] Y. Sun, Z. Guan, and K. Hooman, "Cavitation in diesel fuel injector nozzles and its influence on atomization and spray," *Chemical Engineering & Technology*, vol. 42, no. 1, pp. 6–29, 2019.

[107] A. Ferrari, "Fluid dynamics of acoustic and hydrodynamic cavitation in hydraulic power systems," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, pp. 1–32, 2017.

[108] C. Hirt and B. Nichols, "Volume of fluid (VOF) method for the dynamics of free boundaries," *Journal of Computational Physics*, vol. 39, pp. 201–225, 1981.

[109] G. H. Yeoh and J. Tu, *Computational Techniques for Multiphase Flows*. Elsevier Ltd., 2010.

[110] C. E. Brennen, *Fundamentals of Multiphase Flows*. Cambridge University Press, 2005.

[111] S. Clerc, "Numerical simulation of the homogeneous equilibrium model for two-phase flows," *Journal of Computational Physics*, vol. 161, no. 1, pp. 354–375, 2000.

[112] C. E. Brennen, *Cavitation and Bubble Dynamics*. Oxford University Press, 1995.

[113] S. Subramaniam, "Lagrangian–Eulerian methods for multiphase flows," *Progress in Energy and Combustion Science*, vol. 39, pp. 215–245, 2013.

[114] A. Panchal and S. Menon, "A hybrid Eulerian-Eulerian/Eulerian-Lagrangian method for dense-to-dilute dispersed phase flows," *Journal of Computational Physics*, vol. 439, 2021.

[115] G. P. Sutton and O. Biblarz, *Rocket Propulsion Elements*. John Wiley & Sons, 2010.

[116] M. J. L. Turner, *Rocket and Spacecraft Propulsion*. Springer, 2009.

[117] C. Lioi, "Linear combustion stability analysis of oxidizer-rich staged combustion engines," *PhD Thesis, Georgia Institute of Technology*, 2019.

[118] Britannica, *Liquid-propellant rocket engines*, https://www.britannica.com/technology/rocket-jet-propulsion-device-and-vehicle/Liquid-propellant-rocket-engines, 2021.

[119] C. Lioi, D. Ku, and V. Yang, "Linear acoustic analysis of main combustion chamber of an oxidizer-rich staged combustion engine," *Journal of Propulsion and Power*, vol. 34, pp. 1505–1518, 2018.

[120] G. P. Sutton, *History of Liquid Propellant Rocket Engines*. American Institute of Aeronautics and Astronautics, 2006.

[121] X. Wang, Y. Wang, and V. Yang, "Geometric effects on liquid oxygen/kerosene bi-swirl injector flow dynamics at supercritical conditions," *AIAA Journal*, vol. 55, pp. 3467–3475, 2017.

[122] L. Zhang, X. Wang, Y. Li, S.-T. Yeh, and V. Yang, "Supercritical fluid flow dynamics and mixing in gas-centered liquid-swirl coaxial injector," *Physics of Fluids*, vol. 30, pp. 1–16, 2018.

[123] X. Wang, H. Huo, U. Unnikrishnan, and V. Yang, "A systematic approach to high-fidelity modeling and efficient simulation of supercritical fluid mixing and combustion," *Combustion and Flame*, vol. 195, pp. 203–215, 2018.

[124] X. Wang, L. Zhang, Y. Li, S.-T. Yeh, and V. Yang, "Supercritical combustion of gas-centered liquid-swirl coaxial injectors for staged-combustion engines," *Combustion and Flame*, vol. 197, pp. 204–214, 2018.

[125] P. Milan, J.-P. Hickey, X. Wang, and V. Yang, *Deep-learning accelerated calculation of real-fluid properties in numerical simulation of complex flowfields*, vol. 444, pp. 1–25, 2021.

[126] P. Milan, X. Wang, J.-P. Hickey, Y. Li, and V. Yang, "Accelerating numerical simulations of supercritical fluid flows using deep neural networks," *AIAA 2020-1157*, pp. 1–12, 2020.

[127] V. Yang, "Modeling of supercritical vaporization, mixing, and combustion processes in liquid-fueled propulsion systems," *Proceedings of the Combustion Institute*, vol. 28, pp. 925–942, 2000.

[128] D.-Y. Peng and D. B. Robinson, "A new two-constant equation of state," *Industrial & Engineering Chemistry Fundamentals*, vol. 15, pp. 59–64, 1976.

[129] J. C. Oefelein and R. Sankaran, "Large eddy simulation of reacting flow physics and combustion," in *Exascale Scientific Applications: Scalability and Performance Portability*, T. P. Straatsma, K. B. Antypas, and T. J. Williams, Eds., CRC Press, 2018, ch. 11, pp. 231–256.

[130] G. Soave, "Equilibrium constants from a modified redlich-kwong equation of state," *Chemical engineering science*, vol. 27, pp. 1197–1203, 1972.

[131] P. Milan, Y. Li, X. Wang, S. Yang, W. Sun, and V. Yang, "Time-efficient methods for real fluid property evaluation in numerical simulation of chemically reacting flows," *11th US National Combustion Meeting, 71TF-0396*, pp. 1–10, 2019.

[132] P. Boncinelli, F. Rubechini, A. Arnone, M. Cecconi, and C. Cortese, "Real gas effects in turbomachinery flows: A computational fluid dynamics model for fast computations," *Journal of Turbomachinery*, vol. 126, pp. 268–276, 2004.

[133] S. Kawai, H. Terashima, and H. Negishi, "A robust and accurate numerical method for transcritical turbulent flows at supercritical pressure with an arbitrary equation of state," *Journal of Computational Physics*, vol. 300, pp. 116–135, 2015.

[134] M. Pini, A. Spinelli, G. Persico, and S. Rebay, "Consistent look-up table interpolation method for real-gas flow simulations," *Computers & Fluids*, vol. 107, pp. 178–188, 2015.

[135] S. Bhalla, M. Yao, J.-P. Hickey, and M. Crowley, "Compact representation of a multi-dimensional combustion manifold using deep neural networks," in *Machine Learning and Knowledge Discovery in Databases*, U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, Eds., Cham: Springer International Publishing, 2020, pp. 602–617.

[136] G. Xia, D. Li, and C. L. Merkle, "Consistent properties reconstruction on adaptive cartesian meshes for complex fluids computations," *Journal of Computational Physics*, vol. 225, pp. 1175–1197, 2007.

[137] Z. Liu, J. Liang, and Y. Pan, "Construction of thermodynamic properties look-up table with block-structured adaptive mesh refinement method," *Journal of Thermophysics and Heat Transfer*, vol. 28, pp. 50–58, 2014.

[138] A. Rubino, M. Pini, M. Kosec, S. Vitale, and P. Colonna, "A look-up table method based on unstructured grids and its application to non-ideal compressible fluid dynamic simulations," *Journal of Computational Science*, vol. 28, pp. 70–77, 2018.

[139] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[140] C. J. Lapeyre, A. Misdariis, N. Cazard, D. Veynante, and T. Poinsot, "Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates," *Combustion and Flame*, vol. 203, pp. 255 –264, 2019.

[141] Z. M. Nikolaou, C. Chrysostomou, L. Vervisch, and S. Cant, "Progress variable variance and filtered rate modeling using convolutional neural networks and flamelet methods," *Flow, Turbulence and Combustion*, vol. 103, pp. 485–501, 2019.

[142] M. Ihme, "Construction of optimal artificial neural network architectures for application to chemical systems: Comparison of generalized pattern search method and evolutionary algorithm," in *Artificial Neural Networks*, C. L. P. Hui, Ed., Rijeka: IntechOpen, 2011, ch. 7, pp. 125–150.

[143] Z. Shadram, T. M. Nguyen, A. Sideris, and W. A. Sirignano, "Neural network flame closure for a turbulent combustor with unsteady pressure," *AIAA Journal*, vol. 59, pp. 621–635, 2021.

[144] O. Owoyele, P. Kundu, M. M. Ameen, T. Echekki, and S. Som, "Application of deep artificial neural networks to multi-dimensional flamelet libraries and spray flames," *International Journal of Engine Research*, pp. 11–18, 2019.

[145] J. Xing *et al.*, "Predicting kinetic parameters for coal devolatilization by means of artificial neural networks," *Proceedings of the Combustion Institute*, vol. 37, pp. 2943–2950, 2019.

[146] W. J. Coirier, "Efficient real gas Navier-Stokes computations of high speed flows using an LU scheme," *AIAA 90-0391*, pp. 1–18, 1990.

[147] S. Kawai, H. Terashima, and H. Negishi, "A robust and accurate numerical method for transcritical turbulent flows at supercritical pressure with an arbitrary equation of state," *Journal of Computational Physics*, vol. 300, pp. 116–135, 2015.

[148] P. C. Ma, Y. Lv, and M. Ihme, "An entropy-stable hybrid scheme for simulations of transcritical real-fluid flows," *Journal of Computational Physics*, vol. 340, pp. 330–357, 2017.

[149] D. T. Banuti, P. C. Ma, J.-P. Hickey, and M. Ihme, "Thermodynamic structure of supercritical LOX-GH2 diffusion flames," *Combustion and Flame*, vol. 196, pp. 364–376, 2018.

[150] T. W. Leland and P. S. Chappelear, "The corresponding states principles. A review of current theory and practice," *Industrial and Engineering Chemistry Fundamentals*, vol. 60, pp. 15–43, 1968.

[151] T. H. Chung, M. Ajlan, L. L. Lee, and K. E. Starling, "Generalized multiparameter correlation for nonpolar and polar fluid transport properties," *Industrial & Engineering Chemistry Research*, vol. 27, pp. 671–679, 1988.

[152] B. E. Poling, J. M. Prausnitz, and J. P. O'Connell, *The Properties of Gases and Liquids*. McGraw-Hill, 2001.

[153] E. Fuller, P. Schettler, and J. C. Giddings, "A new method for prediction of binary gas," *Industrial and Engineering Chemistry Research*, vol. 58, pp. 18–27, 1996.

[154] S. Takahashi and M. Hongo, "Diffusion coefficients of gases at high pressures in the CO_2-C_2H_4 system," *Journal of Chemical Engineering Japan*, vol. 15, pp. 57–59, 1982.

[155] J. C. Wilke, "Diffusional properties of multicomponent gases," *Chemical Engineering Progress*, vol. 46, pp. 95–104, 1950.

[156] H. Meng, G. C. Hsiao, V. Yang, and J. S. Shuen, "Transport and dynamics of liquid oxygen droplets in supercritical hydrogen streams," *Journal of Fluid Mechanics*, vol. 527, pp. 115–139, 2005.

[157] J. C. Oefelein, "Advances in modeling supercritical fluid dynamics and combustion in high-pressure propulsion systems," *AIAA 2019-0634*, pp. 1–32, 2019.

[158] H. Meng and V. Yang, "A unified treatment of general fluid thermodynamics and its application to a preconditioned scheme," *Journal of Computational Physics*, vol. 189, pp. 277–304, 2003.

[159] N. Zong and V. Yang, "An efficient preconditioning scheme for real-fluid mixtures using primitive pressure-temperature variables," *International Journal of Computational Fluid Dynamics*, vol. 21, pp. 217–230, 2007.

[160] S.-Y. Hsieh and V. Yang, "A preconditioned flux-differencing scheme for chemically reacting flows at all mach numbers," *International Journal of Computational Fluid Dynamics*, vol. 8, pp. 31–49, 1997.

[161] G. Wilczek-Vera and J. H. Vera, "Understanding cubic equations of state: A search for the hidden clues of their success," *American Institute of Chemical Engineers Journals*, vol. 61, pp. 2824–2831, 2015.

[162] R. Swanson and E. Turkel, "On central-difference and updwind schemes," *Journal of Computational Physics*, vol. 101, pp. 292–306, 1992.

[163] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the message-passing interface*. The MIT Press, 2014.

[164] M. X. Yao, "Thermoacoustic instabilities in counterflow diffusion flames," *MASc Thesis, University of Waterloo*, 2019.

[165] A. C. Zambon and H. K. Chelliah, "Self-sustained acoustic-wave interactions with counterflow flames," *Journal of Fluid Mechanics*, vol. 560, pp. 249–278, 2006.

[166] D. G. Goodwin, R. L. Speth, H. K. Moffat, and B. W. Weber, *Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes*, https://www.cantera.org, Version 2.4.0, 2018.

[167] K. Denbigh, *The Principles of Chemical Equilibrium*. Cambridge University Press, 1966.

[168] D. Liu and Y. Wang, "Multi-fidelity physics-constrained neural network and its application in materials modeling," *Journal of Mechanical Design*, vol. 141, pp. 686–707, 2019.

[169] H.-G. Li, N. Zong, X.-Y. Lu, and V. Yang, "A consistent characteristic boundary condition for general fluid mixture and its implementation in a preconditioning scheme," *Advances in Applied Mathematics and Mechanics*, vol. 4, pp. 72–92, 2012.

[170] A. Mani, "Analysis and optimization of numerical sponge layers as a nonreflective boundary treatment," *Journal of Computational Physics*, vol. 231, pp. 704–716, 2012.

[171] H. Tsuji, "Counterflow diffusion flames," *Progress in Energy and Combustion Science*, vol. 8, pp. 93–119, 1982.

[172] R. W. Bilger, S. H. Starner, and R. J. Kee, "On reduced mechanisms for methane air combustion in nonpremixed flames," *Combustion and Flame*, vol. 80, pp. 135–149, 1990.

[173] H. Huo, X. Wang, and V. Yang, "A general study of counterflow diffusion flames at subcritical and supercritical conditions: Oxygen/hydrogen mixtures," *Combustion and Flame*, vol. 161, pp. 3040–3050, 2014.

[174] M. X. Yao, J.-P. Hickey, P. C. Ma, and M. Ihme, "Molecular diffusion and phase stability in high-pressure combustion," *Combustion and Flame*, vol. 210, pp. 302–314, 2019.

[175] M. Burke, M. Chaos, Y. Ju, F. Dryer, and S. Klippenstein, "Comprehensive H2/O2 kinetic model for high-pressure combustion," *Internal Journal of Chemical Kinetics*, vol. 44, pp. 444–474, 2012.

[176] P. J. Milan, G. M. Magnotti, and V. Yang, "Data-driven deep learning surrogates for parametric prediction of reacting flows," *ILASS-Americas 2021*, pp. 1–13, 2021.

[177] S. Mak *et al.*, "An efficient surrogate model for emulation and physics extraction of large eddy simulations," *Journal of the American Statistical Association*, vol. 113, pp. 1443–1456, 2018.

[178] S.-T. Yeh *et al.*, "Common proper orthogonal decomposition-based spatiotemporal emulator for design exploration," *AIAA Journal*, vol. 56, pp. 2429–2442, 2018.

[179] Y.-H. Chang *et al.*, "Kernel-smoothed proper orthogonal decomposition–based emulation for spatiotemporally evolving flow dynamics prediction," *AIAA Journal*, vol. 57, pp. 5269–5280, 2019.

[180] Y. Li, X. Wang, Y.-H. Chang, P. J. Milan, and V. Yang, "A novel surrogate model for emulation of bi-fluid swirl injector flow dynamics," *AIAA 2020-1070*, pp. 1–41, 2020.

[181] P. Milan, X. Wang, Y. Li, and V. Yang, "Machine learning approaches for computational fluid dynamics of supercritical fluid flows," *ILASS-ASIA-2020-5-16*, pp. 1–9, 2020.

[182] Y.-H. Chang *et al.*, "Reduced-order modeling for complex flow emulation by common kernel-smoothed proper othogonal decomposition," *AIAA Journal*, vol. 59, pp. 3291–3303, 2021.

[183] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, pp. 1–10, 2018.

[184] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[185] J. Almotiri, K. Elleithy, and A. Elleithy, "Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition," in *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2017, pp. 1–5.

[186] L. Agostini, "Exploration and prediction of fluid dynamical systems using auto-encoder technology," *Physics of Fluids*, vol. 32, pp. 1–32, 2020.

[187] J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," *International Conference on Artificial Neural Networks*, pp. 52–59, 2011.

[188] O. San and T. Iliescu, "Proper orthogonal decomposition closure models for fluid flows: Burger equations," *arXiv preprint arXiv:1308.3276*, pp. 1–22, 2013.

[189] M. Buffoni and K. Wilcox, "Projection-based model reduction for reacting flows," *AIAA 2010-5008*, pp. 1–14, 2010.

[190] K. Lee and K. T. Carlberg, "Model reduction of dynamical systems on non-linear manifolds using deep convolutional autoencoders," *Journal of Computational Physics*, vol. 404, pp. 1–32, 2020.

[191] K. Lee and E. J. Parish, "Parameterized neural ordinary differential equations: Applications to computational physics problems," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 477, p. 20 210 162, 2021.

[192] B. Cuenot and T. Poinsot, "Asymptotic and numerical study of diffusion flames with variable lewis number and finite rate chemistry," *Combustion and Flame*, vol. 104, pp. 111–137, 1996.

[193] I. Glassman, R. A. Yetter, and N. G. Glumac, *Combustion*. Elsevier Inc, 2015.

[194] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007.

[195] A. Vallet, A. A. Burluka, and R. Borghi, "Development of a Eulerian model for the atomization of a liquid jet," *Atomization and Sprays*, vol. 11, pp. 619–642, 2001.

[196] K. Saha, P. Srivastava, S. Quan, P. K. Senecal, E. Pomraning, and S. Som, "Modeling the dynamic coupling of internal nozzle flow and spray formation for gasoline direct injection applications," *SAE Technical Paper 2018-01-0314*, pp. 1–15, 2018.

[197] J. Anez, A. Ahmed, N. Hecht, B. Duret, J. Reveillon, and F. Demoulin, "Eulerian–Lagrangian spray atomization model coupled with interface capturing method for diesel injectors," *International Journal of Multiphase Flow*, vol. 113, pp. 325–342, 2019.

[198] S. Som, D. Longman, A. Ramírez, and S. Aggarwal, "A comparison of injector flow and spray characteristics of biodiesel with petrodiesel," *Fuel*, vol. 89, pp. 4014–4024, 2010.

[199] M. Battistoni, C. Grimaldi, and F. Mariani, "Coupled simulation of nozzle flow and spray formation using diesel and biodiesel for CI engine applications," in *SAE 2012 World Congress & Exhibition*, SAE International, 2012.

[200] K. Saha, S. Som, M. Battistoni, S. Quan, P. K. Senecal, and E. Pomraning, "Coupled Eulerian internal nozzle flow and Lagrangian spray simulations for GDI systems," in *WCX™ 17: SAE World Congress Experience*, SAE International, 2017.

[201] S. Quan *et al.*, "A one-way coupled volume of fluid and Eulerian-Lagrangian method for simulating sprays," *ASME 2016 Internal Combustion Engine Division Fall Technical Conference*, pp. 1–9, 2016.

[202] M. Traver, Y. Pei, T. Tzanetakis, R. Torelli, C. Powell, and S. Som, "Investigation and simulation of gasoline in a diesel fuel injector for gasoline compression ignition applications," in *11. Tagung Einspritzung und Kraftstoffe 2018*, H. Tschöke and R. Marohn, Eds., Wiesbaden: Springer Fachmedien Wiesbaden, 2019, pp. 423–442.

[203] R. Ravendran, B. Endelt, J. d. C. Christiansen, P. Jensen, M. Theile, and I. Najar, "Coupling method for internal nozzle flow and the spray formation for viscous liquids," *International Journal of Computational Methods and Experimental Measurements*, vol. 7, pp. 130–141, 2019.

[204] L. Nocivelli *et al.*, "Analysis of the spray numerical injection modeling for gasoline applications," in *WCX SAE World Congress Experience*, SAE International, 2020.

[205] G. Bianchi and P. Pelloni, "Modeling the diesel fuel spray breakup by using a hybrid model," in *International Congress & Exposition*, SAE International, 1999.

[206] E. von Berg *et al.*, "Coupled simulations of nozzle flow, primary fuel jet breakup, and spray formation," *Journal of Engineering for Gas Turbines and Power*, vol. 127, pp. 897–908, 2004.

[207] F. Wang, Z. He, J. Liu, and Q. Wang, "Diesel nozzle geometries on spray characteristics with a spray model coupled with nozzle cavitating flow," *International Journal of Automotive Technology*, vol. 16, pp. 4014–4024, 2014.

[208] Q. Xue *et al.*, "An Eulerian CFD model and X-ray radiography for coupled nozzle flow and spray in internal combustion engines," *International Journal of Multiphase Flow*, vol. 70, pp. 77–88, 2015.

[209] D. Kim *et al.*, "High-fidelity simulation of atomization in a gas turbine injector high shear nozzle," *ILASS-Americas 2014*, pp. 1–15, 2014.

[210] L. Bravo, Q. Xue, S. Som, C. Powell, and C.-B. M. Kweon, "Fuel effects on nozzle flow and spray using fully coupled Eulerian simulations," *ASME 2015 Power Conference*, pp. 1–11, 2015.

[211] P. J. Milan, R. Torelli, B. Lusch, and G. Magnotti, "Data-driven model reduction of multiphase flow in a single-hole automotive injector," *Atomization and Sprays*, vol. 30, pp. 1–29, 2020.

[212] P. J. Milan, S. Mondal, R. Torelli, B. Lusch, R. Maulik, and G. M. Magnotti, "Data-driven modeling of large-eddy simulations for fuel injector design," *AIAA 2021-1016*, pp. 1–12, 2021.

[213] Engine Combustion Network, https://ecn.sandia.gov/, 2019.

[214] B. A. Sforzo *et al.*, "Fuel nozzle geometry effects on cavitation and spray behavior at diesel engine conditions," *Proceedings of the 10th International Symposium on Cavitation*, vol. CAV18-05098, pp. 474–480, 2018.

[215] Spray Combustion Consortium, https://scc.sandia.gov/, 2020.

[216] K. Yasutomi, J. Hwang, L. M. Pickett, B. Sforzo, K. Matusik, and C. F. Powell, "Transient internal nozzle flow in transparent multi-hole diesel injector," in *WCX SAE World Congress Experience*, SAE International, 2020.

[217] M. Battistoni, D. Duke, A. B. Swantek, F. Z. Tilocco, C. F. Powell, and S. Som, "Effects of noncondensable gas on cavitating nozzles," *Atomization and Spray*, vol. 2015, pp. 453–483, 2015.

[218] R. Torelli, S. Som, Y. Pei, Y. Zhang, and M. L. Traver, "Influence of fuel properties on internal nozzle flow development in a multi-hole diesel injector," *Fuel*, vol. 204, pp. 171–184, 2017.

[219] R. Torelli, G. M. Magnotti, S. Som, Y. Pei, and M. L. Traver, "Exploration of cavitation-suppressing orifice designs for a heavy-duty diesel injector operat-

ing with straight-run gasoline," *SAE Technical Paper 2019-24-0126*, pp. 1–15, 2019.

[220] SAS Institute Inc., *JMP® 14 fitting linear models*, Cary, NC: SAS Institute Inc., 2018.

[221] H. Guo *et al.*, "Internal nozzle flow simulations of the ECN Spray C injector under realistic operating conditions," in *WCX 20: SAE World Congress Experience*, SAE International, 2020, pp. 1–11.

[222] G. M. Magnotti and S. Som, "Assessing fuel property effects on cavitation and erosion propensity using a computational fuel screening tool," *ASME 2019 Internal Combustion Engine Division Fall Technical Conference*, pp. 1–13, 2019.

[223] R. Battino, T. R. Rettich, and T. Tominaga, "The solubility of nitrogen and air in liquids," *Journal of Physical and Chemical Reference Data*, vol. 13, pp. 563–600, 1984.

[224] M. D. McKay, "Latin hypercube sampling as a tool in uncertainty analysis of computer models," in *WSC '92*, 1992, pp. 557–564.

[225] R. Torelli *et al.*, "Comparison of in-nozzle flow characteristics of naphtha and n-dodecane fuels," in *WCX 17: SAE World Congress Experience*, SAE International, 2017, pp. 1–16.

[226] R. Torelli *et al.*, "Evaluation of shot-to-shot in-nozzle flow variations in a heavy-duty diesel injector using real nozzle geometry," *SAE Int. J. Fuels Lubr.*, vol. 11, pp. 379–395, 2018.

[227] O. Redlich and J. Kwong, "On the thermodynamics of solutions. V. An equation of state. Fugacities of gaseous solutions," *Chemical reviews*, vol. 44, pp. 233–244, 1949.

[228] Z. Bilicki and J. Kestin, "Physical aspects of the relaxation model in two-phase flow," *Proceedings of the Royal Society London A*, vol. 428, pp. 379–397, 1990.

[229] J. Brackbill, D. Kothe, and C. Zemach, "A continuum method for modeling surface tension," *Journal of Computational Physics*, vol. 100, pp. 335–354, 1992.

[230] M. Battistoni, Q. Xue, S. Som, and E. Pomraning, "Effects of off-axis needle motion on internal nozzle and near exit flow in a multi-hole fuel injector," *SAE International Journal of Fuels and Lubricants*, vol. 7, pp. 167–182, 2014.

[231] D. J. Duke, A. L. Kastengren, F. Z. Tilocco, A. B. Swantek, and C. F. Powell, "X-ray radiography measurements of cavitating nozzle flow," *Atomization and Sprays*, vol. 23, pp. 841–860, 2013.

[232] G. M. Magnotti, M. Battistoni, K. Saha, and S. Som, "Influence of turbulence and thermophysical fluid properties on cavitation erosion predictions in channel flow geometries," *SAE Int. J. Adv. & Curr. Prac. in Mobility*, vol. 1, pp. 691–705, 2019.

[233] A. Tekawade *et al.*, "A comparison between CFD and 3D X-ray diagnostics of internal flow in a cavitating diesel injector nozzle," *30th Annual Conference on Liquid Atomization and Spray Systems (ILASS-Americas)*, 2019.

[234] I. M. Sobol, "Sensitivity estimates for nonlinear mathematical models," *MMCE*, vol. 1.4, pp. 407–414, 1993.

[235] J. Shi and M. Arafin, "CFD investigation of fuel property effect on cavitating flow in generic nozzle geometries," *23rd Annual Conference on Liquid Atomization and Spray Systems (ILASS-Europe)*, 2010.

[236] S. Mondal, R. Torelli, B. Lusch, P. J. Milan, and G. M. Magnotti, "Accelerating the generation of static coupling injection maps using a data-driven emulator," in *SAE WCX Digital Summit*, SAE International, 2021.

[237] R. Reitz, "Modeling atomization processes in high-pressure vaporizing sprays," *Atomization Spray Technology*, vol. 3, pp. 309–337, 1988.

[238] J. C. Beale and R. D. Reitz, "Modeling spray atomization with the Kelvin-Helmholtz/Rayleigh-Taylor hybrid model," *Atomization and Sprays*, pp. 623–650, 1999.

[239] A. A. Amsden, P. J. O'Rourke, and T. D. Butler, "KIVA-II: A computer program for chemically reactive flows with sprays," 1989.

[240] A. Nunno, P. Kundu, and S. Som, *Extending the unsteady flamelet-progress variable model to split injection and compression ignition engine applications*, in preparation, 2021.

[241] C. K. Westbrook, W. J. Pitz, O. Herbinet, H. J. Curran, and E. J. Silke, "A comprehensive detailed chemical kinetic reaction mechanism for combustion of n-alkane hydrocarbons from n-octane to n-hexadecane," *Combustion and Flame*, vol. 156, pp. 181–199, 2009.

[242] G. M. Magnotti, P. Kundu, A. C. Nunno, and S. Som, "Linking cavitation erosion in a multi-hole injector with spray and combustion development," *ICLASS 2021*, pp. 1–8, 2021.

[243] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Artificial Neural Networks and Machine Learning – ICANN 2018*, V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, Eds., Cham: Springer International Publishing, 2018, pp. 270–279.

[244] O. Owoyele, P. Pal, and A. Vidal Torreira, "An Automated machine learning-genetic algorithm framework with active learning for design optimization," *Journal of Energy Resources Technology*, vol. 143, 2021.

[245] K. Shukla, A. D. Jagtap, and G. E. Karniadakis, "Parallel physics-informed neural networks via domain decomposition," *Journal of Computational Physics*, vol. 447, pp. 1–19, 2021.

[246] C. Hoang, Y. Choi, and K. Carlberg, "Domain-decomposition least-squares Petrov-Galerkin (DD-LSPG) nonlinear model reduction," *Computer Methods in Applied Mechanics and Engineering*, vol. 384, pp. 1–41, 2021.

[247] H. Gao, L. Sun, and J.-X. Wang, "PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain," *Journal of Computational Physics*, vol. 428, pp. 1–27, 2021.

[248] R. Maulik, T. Botsas, N. Ramachandra, L. R. Mason, and I. Pan, "Latent-space time evolution of non-intrusive reduced-order models using gaussian process emulation," *Physica D: Nonlinear Phenomena*, vol. 416, pp. 1–17, 2021.

[249] S. Mondal, G. M. Magnotti, B. Lusch, R. Maulik, and R. Torelli, "Machine Learning-Enabled Prediction of Transient Injection Map In Automotive Injectors With Uncertainty Quantification," *ASME 2021 Internal Combustion Engine Division Fall Technical Conference*, pp. 1–12, 2021.

[250] Y. C. Liang, H. P. Lee, S. P. Lim, W. Z. Lin, K. H. Lee, and C. G. Wu, "Proper orthogonal decomposition and its applications – part I: Theory," *Journal of Sound and Vibration*, vol. 252, pp. 527 –544, 2002.

[251] K. Taira *et al.*, "Modal analysis of fluid flows: An overview," *AIAA Journal*, vol. 55, pp. 4013–4041, 2017.

[252] T. Barber, M. Ahmed, and N. A. Shafi, "POD snapshot data reduction for periodic fluid flows," *AIAA 2005-287*, pp. 1–12, 2005.

[253] M. J. Morna and H. N. Shapiro, *Fundamentals of Engineering Thermodynamics*. John Wiley & Sons, 2000.

VITA

Petro Junior Milan obtained a BEng in Aerospace Engineering from Polytechnique Montreal (Canada) in 2016. During his time at Polytechnique (2012-2016), he carried out two summer research internships and three undergraduate research assistantships under the guidance of Prof. Huu Duc Vo and Prof. Dominique Pelletier in the fields of computational fluid dynamics (CFD), plasma aerodynamics, compressor/wing aerodynamics, flow control, and sensitivity/uncertainty analysis. He also interned in industry in the Installation & Turbine Aerodynamics group at Pratt & Whitney Canada in the summers of 2015 and 2016, working on the numerical modeling and aerothermodynamic analysis of mixed-flow exhaust turbofans.

Beginning in 2016 at the Georgia Institute of Technology, Milan was initially a graduate research assistant at the Computational Combustion Laboratory, working with Prof. Suresh Menon from 2016 to 2018. He moved to Prof. Vigor Yang's research group in August 2018 to pursue his interest in combining machine learning with CFD, and obtained an MS in Computational Science and Engineering and a PhD in Aerospace Engineering (along with a Minor in Mathematics), both in 2021. During his time in Yang's group, he worked on several projects in the fields of CFD, rocket propulsion, supercritical turbulent combustion, surrogate modeling, and deep learning. He also did two internships in the Multi-Physics Computational Research Section at the U.S. Department of Energy's Argonne National Laboratory in the spring and summer of 2020, where he worked on using deep learning to accelerate multiphase flow simulations for diesel engines. Milan has been the recipient of a Master's Research Scholarship from the *Fonds de recherche du Québec – Nature et technologies* (FRQNT) (2016-2018), a Postgraduate Scholarship D from the Natural Sciences and Engineering Research Council of Canada (NSERC) (2018-2021), and an AE Graduate Student Fellowship from the Georgia Tech's School of Aerospace Engineering (2020-2021). After his PhD, he expects to move to the San Francisco Bay Area to work at a machine learning startup on next-generation artificial intelligence and data-intensive scientific applications.