A Physics-Informed Neural Network modelling methodology to analyse integrated thermofluid systems

Prepared by:

Kristina Karli Laugksch

LGKKRI001

Department of Mechanical Engineering

University of Cape Town

Supervisors:

Professor Pieter Rousseau

Professor Ryno Laubscher

June 2024

Submitted to the Department of Mechanical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Master of Science in Engineering specialising in Mechanical Engineering.

Keywords: machine learning; physics-informed neural networks; 1-D integrated thermofluid network modelling

Abstract

Physics-informed neural networks (PINNs) were developed to overcome the limitations of acquiring large training datasets that are commonly encountered when using purely data-driven machine learning methods. This study explores a PINN modelling methodology to analyse steady-state integrated thermofluid systems based on the mass, energy, and momentum balance equations, combined with the relevant component characteristics and fluid property relationships. The PINN methodology is applied to three thermofluid systems that encapsulate important phenomena typically encountered in integrated thermofluid systems modelling, namely: (i) a heat exchanger network with two different fluid streams and components linked in series and parallel; (ii) a recuperated closed Brayton cycle containing various turbomachines and heat exchangers, and (iii) a simplified boiler consisting of a furnace and radiative-convective superheater. The predictions of the three PINN models were compared to benchmark solutions generated via conventional, physics-based thermofluid process models. The largest average relative error across all three models is only 0.93%, indicating that the PINN methodology can successfully be implemented to generate accurate solutions using the non-dimensionalised forms of the balance equations. Furthermore, it was shown that the trained PINN models provided a significant increase in inference speed compared to the conventional process models. The PINN modelling methodology was then extended to develop a surrogate model for the heat exchanger network. An additional surrogate model was developed for comparison using a data-driven multilayer perceptron (MLP) neural network. The MLP surrogate model was able to interpolate accurately. However, its predictive performance declined when making predictions for samples that fell outside of the range of training data. Despite various refinements, the PINN surrogate model could only be trained successfully for datasets that contained up to five unique samples. This limitation could not be resolved within the scope of the present study and should be investigated further. The accuracy of the PINN surrogate model degraded significantly when used to extrapolate beyond the training envelope. Due to the constraint on the number of training samples, it is impossible to draw a general conclusion regarding the extrapolation ability of the PINN concept. In spite of its current limitations, the significant increase in computational speed offered by the PINN modelling methodology when used to analyse integrated thermofluid systems suggests that this is a promising modelling technique that should be explored further.

Declaration

I, Kristina Laugksch, hereby declare the work contained in this dissertation to be my own. All information which has been gained from various journal articles, textbooks or other sources has been referenced accordingly. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed: Signed by candidate _____

Date: 07 June 2024

Acknowledgements

I wish to express my sincere thanks to my supervisors, Professors Pieter Rousseau and Ryno Laubscher, for their technical advice as well as their continuous guidance, encouragement, and patience over the last few years.

I would like to thank the National Research Foundation of South Africa (Grant Numbers 138618 and 148757), the University of Cape Town, the Applied Thermofluid Process Modelling (ATProM) Research Unit, and the European Commission (under the Erasmus+ Programme) for contributing to funding this research.

I would like to extend my thanks to Warren Hulley and Richard Raymond from Data Sciences Corporation for their assistance in obtaining and setting up the GPU hardware required for this research.

Additionally, I would like to thank my friends and colleagues at the ATProM Research unit for their support over the last two years. The collaborative problem-solving, lunchtime discussions, and board game sessions have taught me a great deal. In particular, I would like to thank Priyesh Gosai for his mentorship in the academic sphere and beyond.

Finally, I would like to acknowledge the support provided by my family. To my parents, thank you for giving me the opportunity to follow this path. I am most grateful for the love and encouragement you have always given me. To my brother, thank you for always showing an interest in my work. And to Mike, thank you for always believing in me and for all your love, support, and motivation.

Research outputs from current work

Journal article:

Laugksch K, Rousseau P, Laubscher R. A PINN Surrogate Modeling Methodology for Steady-State Integrated Thermofluid Systems Modeling. Mathematical and Computational Applications. 2023; 28(2):52. https://doi.org/10.3390/mca28020052.

Conference presentations:

"A PINN surrogate modelling methodology for heat exchanger thermofluid networks," 5th African Conference on Computational Mechanics, Cape Town, South Africa, November 2022.

"A PINN modelling methodology for steady-state integrated thermofluid systems," SAIMechE WC Postgraduate Conference on Mechanical, Manufacturing, Materials & Biomedical Engineering 2023, Stellenbosch, South Africa, November 2023.

Table of Contents

List of Figures

List of Tables

List of Nomenclature

Acronyms and Abbreviations

AAC	Automotive air conditioning
Adam	Adaptive moment estimation
ANN	Artificial neural network
ATProM	Applied Thermofluid Process Modelling
C	Compressor
CFD	Computational fluid dynamics
DOE	Design of experiments
ELU	Exponential linear unit
H	Heater
HX	Heat exchanger
LHS	Latin hypercube sampling
MCR	Maximum Continuous Rating
MLP	Multilayer perceptron
MSE	Mean squared error
NFL	No Free Lunch
PC	Pre-cooler
PDE	Partial differential equation
POD	Proper orthogonal decomposition
PINN	Physics-informed neural network
ReLU	Rectified linear unit
ROM	Reduced order model
RX	Recuperator heat exchanger
SH	Superheater
sCO2	Supercritical carbon dioxide
SGD	Stochastic gradient descent
T	Turbine
tanh	Hyperbolic tangent
TNC	Truncated Newton method

General symbols

Symbol	Description	Unit
A	Empirical constant for flame modification factor based on type of fuel	
A_{rad}	Total surrounding projected area of the furnace	[m^2]
a_j^l	Activated output of the j^{th} neuron in the l^{th} layer	
B	Empirical constant for flame modification factor based on type of fuel	
Bo	Boltzmann number	
b_j^l	Bias parameter for the j^{th} neuron in the l^{th} layer	
CM	Corrected mass flow rate	
c_p	Specific heat capacity at constant pressure	[J/kgK]
$\boldsymbol{D}$	Matrix of coefficients	
$\boldsymbol{e}$	Vector of unknown quantities	
$\boldsymbol{f}$	Source term vector	
fan_{in}	Input dimension of a neural network layer	
fan_{out}	Output dimension of a neural network layer	
f_{mass}	Mass balance residual	
f_{energy}	Energy balance residual	
$f_{momentum}$	Momentum balance residual	
h	Specific static enthalpy	[J/kg]
h_0	Specific total enthalpy	[J/kg]
h_∞	Reference specific enthalpy	[J/kg]
h^*	Non-dimensional specific enthalpy	
J_{mass}	Combined mass balance residual	
J_{energy}	Combined energy balance residual	
$J_{momentum}$	Combined momentum balance residual	
K	Heat exchanger lumped loss coefficient	
l	Current layer in a neural network	
$\dot{m}$	Mass flow rate	[kg/s]
$\dot{m}_\infty$	Reference mass flow rate	[kg/s]
$\dot{m}^*$	Non-dimensional mass flow rate	
M	Flame modification factor	
n	Number of observations in the training dataset	
N_α	Number of instances of balance equation α	
p	Static pressure	[Pa]
p_0	Total pressure	[Pa]
p_∞	Reference pressure	[Pa]
p^*	Non-dimensional pressure	

PR	Total pressure ratio over the machine	
Δp_{0M}	Total pressure rise due to work done on the fluid	[Pa]
Δp_{0L}	Total pressure loss	[Pa]
$\dot{Q}$	Rate of heat transfer to the fluid	[W]
T	Temperature	[K]
$\overline{vc}$	Mean overall heat capacity based on the fuel flow rate	[J/kg$_f$K]
$\dot{W}$	Rate of work done by the fluid	[W]
w_{kj}^{l}	Weight parameter linking the neuron in the k^{th} row of the preceding layer $(l-1)$ with the neuron in the j^{th} row of the current layer (l)	
wt %	Percentage by weight	
$\boldsymbol{x}$	Vector of input parameters from the training dataset	
X_r	Normalised burner height	
x_∞	Reference value for parameter x	
ΔX	Correction factor for the actual flame position in the burner	
$\boldsymbol{y}$	Vector of desired output parameters from the training dataset	
$\hat{\boldsymbol{y}}$	Vector of predicted values from MLP network	
z_j^l	Final weighted input into the j^{th} neuron in layer l	

Greek symbols

Symbol	**Description**	**Unit**
ε_{HX}	Heat exchanger effectiveness	
ε_f	Furnace effective emissivity	
η	Isentropic efficiency of turbomachines	
ρ	Density	[kg/m^3]
σ	Stefan-Boltzmann number	[W/m^{2}K^4]
σ_x	Standard deviation	
$\sigma_l()$	Activation function for layer l	
$\sigma_{Linear}()$	Linear activation function	
$\sigma_{Logistic}()$	Logistic activation function	
$\sigma_{ReLU}()$	Rectified linear unit activation function	
$\sigma_{tanh}()$	Hyperbolic tangent activation function	
φ	Heat preservation coefficient	
ψ	Area-weighted furnace efficiency factor	

Chapter 1. Introduction

1.1 Background and motivation

Traditionally, integrated thermofluid systems have been analysed using conventional physics-based process models. While these models provide a high level of accuracy, the accuracy comes at the cost of an increase in computational expense due to their complex nature. Therefore, these conventional physics-based process models are not ideal for use in all applications. Examples of applications for which physics-based models are well-suited include surrogate modelling for design optimisation, performing parametric design studies, and instances where parameter discovery is required to find unknown properties, such as unknown boundary conditions. Recently, machine learning methods have been used to generate models for integrated thermofluid systems as computationally efficient alternatives to conventional physics-based models.

Machine learning methods are typically used to develop computationally efficient regression models that can accurately fit high-dimensional, highly non-linear relationships. Neural networks, a category of machine learning methods, are emerging as valuable engineering tools for component and system design and optimisation, as well as for real-time simulation for anomaly detection, diagnosis, and forecasting [1]. Artificial neural network (ANN) models, such as multilayer perceptron (MLP) models (also referred to as feedforward neural network models), are data-driven models that are trained to fit non-linear relationships between input and output variables through exposure to examples of desired output values for the given input values. The trained models make predictions very quickly by approximating the input-output behaviour of complex processes. MLP surrogate models are capable of modelling processes quickly and with high levels of accuracy [2]. The data used to train such models are obtained from either experimental measurements or high-fidelity simulation results. However, the prohibitive cost of procurement, the low fidelity of experimental data in industry-scale thermofluid systems, and the computational resources required in conventional simulation limit the usefulness of purely data-driven neural networks for analysing integrated thermofluid systems [1, 3]. Furthermore, because data-driven neural networks are trained using relationships present in the training data, the predictive performance of these models declines when they are required to extrapolate outside the envelope of the training data.

Physics-informed neural networks (PINNs) were developed to overcome some of the limitations of conventional neural networks by embedding the physics equations into the neural network loss function, thus circumventing the need for large databases of training data due to the physics-based regularisation [1]. The integration of prior knowledge, such as the physics-based governing equations, into the training process of the neural network, changes the process from supervised learning using examples of desired output values to unsupervised learning using governing physics equations (e.g., the conservation of momentum or energy). The concept of PINNs was first introduced by Raissi et al. [3] as a deep learning framework for modelling complex

physical systems involving partial differential equations (PDEs). Several other authors have since suggested that the limitations of conventional data-driven neural network models can be addressed by integrating the relevant physics-based principles directly into the learning process. Thus far, PINNs have largely been applied to problems in the field of multidimensional computational fluid mechanics. The application of PINNs to integrated thermofluid systems, which is the subject of the current work, has yet to be explored.

1.2 Problem statement and project scope

ANNs have been used to develop data-driven MLP surrogate models of thermofluid processes as computationally efficient alternatives to conventional physics-based thermofluid process models. These MLP surrogate models are capable of modelling processes quickly and with high levels of accuracy [2]. Furthermore, given a new set of input parameters within the envelope of training data, neural networks can use the learned correlation to interpolate and solve for the corresponding output values. In contrast, conventional physics-based thermofluid process models require the entire computationally expensive solution process to be repeated in order to arrive at the solution for a new set of input parameters. However, significant challenges are associated with obtaining the experimental or simulation data required to train the conventional neural network models [1, 3]. Furthermore, MLP surrogate models are unable to generalise to out-of-sample scenarios (i.e., extrapolate), thus limiting their usefulness for integrated thermofluid network modelling [2].

Unlike the purely data-driven approach, the PINN modelling methodology incorporates the relevant physics equations into the training process of the neural network model. The physics-based regularisation introduced by including the governing equations may provide a potential solution to the limitations of MLP surrogate models for application to integrated thermofluid systems.

The current work explores a PINN modelling methodology for integrated thermofluid systems modelling based on the mass, energy, and momentum balance equations, combined with the relevant component characteristics and fluid property relationships. The scope of this work is limited to steady-state thermofluid system operation, deep learning techniques, and fully connected neural networks. The PINN methodology was applied to three thermofluid systems that encapsulate important phenomena typically encountered in integrated thermofluid systems modelling, namely: (i) a heat exchanger network with two different fluid streams and components linked in series and parallel; (ii) a recuperated closed Brayton cycle containing various turbomachines and heat exchangers, and (iii) a simplified boiler consisting of a furnace and radiative-convective superheater.

1.3 Research objectives

This study will attempt to answer the following research questions:

1. To what extent is it possible to use a PINN modelling methodology to solve the mass, momentum, and energy balance equations for integrated thermofluid systems?
2. How do the accuracy and computational requirements of the PINN modelling methodology compare with those of a conventional physics-based thermofluid process model?
3. What, if any, limitations exist when applying the PINN modelling methodology to develop surrogate models for integrated thermofluid systems?

The application of a PINN modelling methodology to analyse integrated thermofluid systems will be explored by addressing the following objectives:

a) Using a 1-D integrated network modelling approach, develop a conventional physics-based thermofluid network model of a simple heat exchanger network.
b) Apply the PINN modelling methodology to model the simple heat exchanger network and a recuperated closed Brayton cycle. The databases of input parameters used during the training of these two PINN models are generated using the 1-D process model generated in (a), and an existing 1-D process model of the recuperated closed Brayton cycle.
c) Investigate the numerical intricacies of the PINN modelling methodology using the two PINN models generated in (b).
d) Develop a conventional physics-based thermofluid network model of a simplified boiler using a 1-D integrated network modelling approach.
e) Apply the PINN modelling methodology to model the simplified boiler. The database of input parameters used during the training of this PINN model is generated using the conventional physics-based thermofluid model generated in (d).
f) Apply a conventional neural network approach to develop a data-driven surrogate model of the simple heat exchanger network using training data obtained from the conventional physics-based model generated in (a).
g) Investigate the possible application of the PINN modelling methodology to surrogate models for integrated thermofluid systems by extending the PINN surrogate model of the simple heat exchanger network developed in (b) to be trained using multiple samples simultaneously. This enables the evaluation of the current limitations of the PINN modelling methodology applied to surrogate modelling for integrated thermofluid systems.

1.4 Document structure

This document contains seven chapters. The content of the remaining six chapters is as follows:

- Chapter 2: *Literature review*
  In this chapter, a discussion of literature related to traditional physics-based thermofluid process modelling methods, machine learning techniques applied to thermofluid simulation, and scientific machine learning in the context of PINNs is presented.

- Chapter 3: *Theoretical background and methodology*
  An outline of the theoretical background of the three modelling methodologies that are applied in this study is presented in this chapter. These include physics-based models for integrated thermofluid systems, multilayer perceptron neural networks, and physics-informed neural networks.

- Chapter 4: *PINNs applied as numerical solvers*
  In this study, the PINN modelling methodology is first applied to a simple heat exchanger network and a recuperated closed Brayton cycle. A description of these case study problems as well as the conventional thermofluid process model and PINN model setups for these case studies is provided in this chapter. A discussion of the preliminary findings related to the PINN modelling methodology is also included. Finally, insights into the performance of PINN models for integrated thermofluid systems are discussed.

- Chapter 5: *PINNs applied to a thermofluid network with complex component characteristics*
  The PINN methodology is applied to a simplified boiler system in order to investigate the PINN performance for a case study that requires the consideration of complex component characteristic relationships. In this chapter, a description of the simplified boiler system as well as an overview of the PINN model setup for this case study is provided. A discussion of the PINN model performance for the simplified boiler system is also included.

- Chapter 6: *PINNs applied to surrogate modelling*
  In this chapter, the extension of the PINN modelling methodology to the development of surrogate models is investigated. An outline of the development of a conventional MLP surrogate model for the heat exchanger network, as well as a discussion of the model performance, is provided. The chapter also includes an overview of the extension of the PINN model of the heat exchanger network, described in Chapter 4, to allow the consideration of multiple simulation cases simultaneously. Finally, an evaluation of the application of the PINN modelling methodology to develop surrogate models for integrated thermofluid systems is included.

- Chapter 7: *Conclusions and recommendations*
  A summary of the findings and conclusions of this study, as well as recommendations for future work, are provided in this chapter.

Chapter 2. Literature review

In this chapter, the relevant literature is synthesised to provide an overview of the current state of knowledge in the following areas: (i) traditional thermofluid process modelling methods, (ii) the application of machine learning techniques to thermofluid simulation, and (iii) the use of scientific machine learning in the context of PINNs.

2.1 Traditional thermofluid process modelling

Traditionally, integrated thermofluid systems are analysed using process models that consist of physics-based methods that apply relevant fundamental balance equations to develop numerical solvers. There are a variety of widely available industrial tools that apply such methods to thermofluid process models. For example, Ortega et al. [4] developed a thermofluid process model to evaluate the performance of a directly heated tubular solar receiver for a supercritical carbon dioxide (sCO2) Brayton cycle. Predictions for the thermal performance of the receiver were made by evaluating the relevant radiation and convection heat loss mechanisms. The required computational fluid dynamics (CFD) modelling was performed using the Ansys Fluent® [5] fluid simulation software. This software applies numerical finite volume methods to solve the applicable PDEs. Wei et al. [6] employed the AspenPlus® [7] steady-state process modelling and simulation software package to build a process model for a biomass-fuelled oxy-fuel combustion power plant with a recuperated sCO2 Brayton cycle. This model was used to evaluate the thermodynamic performance of the power plant. Rauch et al. [8] employed a process model for a combined Brayton-Rankine cycle to determine the maximum thermal efficiency of the combined cycle. The process model consisted of a complete mathematical model and was developed using the Matlab® [9] programming platform, which is typically used for iterative analysis and design processes.

Other physics-based approaches use reduced order models (ROMs) in which the dimensionality of the models, and consequently their complexity, is reduced. The work of Baillie et al. [10] provides one example of how ROMs can be applied to thermofluid process modelling. The authors developed a heuristic-direct elimination model to approximate the system behaviour of a counter-flow air-to-water heat exchanger. This model was based on an original full system model; however, some of the state variables, parameters, and equations were eliminated from the original model to produce a model of lower dimensionality. A statistical sensitivity analysis was performed to identify which elements could be eliminated. The final heuristic-direct elimination model contained only 986 equations compared to the original 1127. This model was extremely accurate for a large subset of the input space and demonstrated a reduction in computing time compared to the original full system model.

Proper orthogonal decomposition (POD) is another approach used to generate ROMs from complete physics-based models. Ding et al. [11] applied a POD technique to develop ROMs for a variety of fluid flow and heat transfer problems. The POD technique was applied to generate models that predicted the fluid and temperature fields for natural convection in a cavity, lid-driven cavity flow, and heat conduction with a time-dependent heat source. In all three instances, the POD technique produced a model that had a significantly reduced computational time in comparison to a traditional finite volume method numerical solver.

Traditional physics-based numerical solvers, while accurate, can be computationally expensive and time-consuming due to their complexity. Conversely, the decreased complexity of reduced order models usually results in a loss of accuracy, which can be undesirable for certain problems. Furthermore, traditional physics-based methods can be inadequate for modelling poorly understood processes where the complete set of governing equations might not be available [2].

2.2 Machine learning applied to thermofluid simulation

Recently, machine learning and deep learning techniques have been increasingly applied to develop computationally efficient models of complex processes. The popularity of these techniques can be attributed to their capacity to fit highly non-linear relationships with a great deal of accuracy. One commonly used technique is that of ANNs in which many processing nodes, known as neurons, are organised into a network of interconnected layers. Conventionally, these ANNs are trained using large training datasets, and can thus be categorised as an example of a data-driven machine learning technique. Because only the given inputs and outputs are presented to the model during training, no information about the underlying process being modelled is embedded into the ANN.

ANNs have been applied to a variety of different applications, including thermofluid simulation. Hosoz and Ertunc [12] developed an artificial neural network to predict the performance of an automotive air conditioning (AAC) system. The neural network was trained using data obtained from an experimental AAC setup made up of original components from the air conditioning system of a small car. The trained ANN made accurate predictions for the various performance parameters with relative errors in the range of only 1.52–2.51%. The authors therefore demonstrated that ANNs can be used to model thermofluid systems, such as AACs, with high accuracy. Haffejee and Laubscher [13] used an MLP neural network to develop a data-driven surrogate model for an air-cooled condenser system at a power plant. The surrogate model was trained using simulation data obtained from a 1-D thermofluid network simulation model. Once trained, the MLP surrogate model could produce accurate predictions of various system performance parameters within a few seconds. This starkly contrasts the conventional thermofluid network model for the same system, which took a minimum of 20 minutes to solve for a single set of inputs. This work therefore demonstrates the significant increase in computational speed that can be achieved by implementing neural network surrogate models.

Further examples of data-driven ANNs applied to thermofluid simulation include the work of Fast and Palmé [14], Waxenegger-Wilfing et al. [15], Shi et al. [16], and Singh and Abbassi [17]. Fast and Palmé [14] employed artificial neural networks to develop a model for the online condition monitoring and diagnosis of a combined heat and power plant. The overall system was broken down into its basic components, each of which was modelled separately with its own ANN. The individual ANNs were combined to generate a single process model for the entire plant. By modelling all significant components separately, the authors decreased the complexity of the ANNs to be trained. This work thus highlights the modular nature of neural network models.

Waxenegger-Wilfing et al. [15] employed ANNs to develop a surrogate model to predict the maximum wall temperature of a rocket combustion chamber wall given a regenerative cooling design using supercritical methane. The surrogate model made predictions at least 1000 times faster than comparable 3-D CFD simulations, again highlighting the numerical efficiency of ANN surrogate models. Shi et al. [16] used ANN models to predict the thermal efficiency and NOx emissions of an ultra-supercritical coal-fired power plant. The models were trained using historical operating data supplemented with CFD simulation data. Including the CFD simulation data improved the accuracy of the model by improving its generalisation performance as it provided data, and thus enabled predictions, for different operating conditions. Singh and Abbassi [17] employed an ANN model in conjunction with a 3-D CFD model to investigate the thermal modelling of an off-highway machinery cabin. The ANN model was utilised in favour of a conventional 1-D process modelling approach due to the resulting reduction in the complexity of the overall thermofluid system. The data used to train the ANN was generated via a 1-D process model of the refrigeration cycle. The trained ANN model was used to predict the thermal state of the cabin air, which was then fed into the detailed CFD model to calculate the temperature and humidity fields within the cabin. By integrating the two modelling approaches, the authors were able to model the transient response of the evaporator given the instantaneous thermal state of the cabin air. This work demonstrated that ANN models can be used successfully in conjunction with CFD models.

Despite the aforementioned benefits of employing purely data-driven ANNs to generate surrogate models for thermofluid simulation, two significant limitations of this methodology have been identified in the literature. Pacheco-Vega et al. [18] developed feedforward neural networks to predict the performance of fin-tube heat exchangers used for refrigeration applications. The authors demonstrated that the predictive performance of the trained networks was directly linked to the size and distribution of the training data, with models built on undersized data performing poorly. The prohibitive cost of performing the large number of experiments required to generate large datasets often results in limited training data and ultimately affects the performance of the models. This work therefore showed that the usefulness of data-driven surrogate modelling techniques is limited in applications where experimental or simulation data are not readily available, such as industry-scale thermofluid systems. Additionally, Willard et al. [2] note that ANNs can only provide reliable predictions for instances that are within the bounds of the training space. ANNs are thus not effective for scenarios where

extrapolation beyond the training data is required. This can be particularly challenging for instances where the training data does not cover the full range of operation for the system being modelled.

2.3 Scientific machine learning: PINNs

Current data-driven machine learning techniques do not utilise the vast amount of prior knowledge of physical systems that already exists. Several authors have thus suggested that directly integrating scientific knowledge of the underlying process into machine learning frameworks can address some of the limitations associated with purely data-driven techniques [1, 2, 19, 20]. Willard et al. [2] identify three approaches through which scientific knowledge can be integrated into machine learning models, namely using: (i) a physics-guided loss function, (ii) physics-guided initialisation of the trainable model parameters, and (iii) a physics-guided approach to design the model architecture. PINNs employ a physics-guided loss function approach, with relevant governing physics equations being integrated into the loss function of the neural network. Prior knowledge of the underlying physical process is thus leveraged to train PINNs, instead of relationships extracted from large databases of training data. The governing physics equations constrain the solution space of the neural network to physically realistic bounds [1, 2, 21]

Raissi et al. [3] first proposed the concept of PINNs as a deep learning framework for solving complex physical systems. The authors used PINNs to generate solutions to complex partial differential equations across a variety of fields, such as the Schrödinger equation in quantum mechanics and the Navier-Stokes equations in fluid mechanics. Using the solutions to these equations, the authors were able to construct accurate and computationally efficient surrogate models. This work also demonstrated the capability of PINNs to discover hidden physics.

Since the introduction of PINNs by Raissi et al. [3], this methodology has largely been applied to problems in the fields of 1-D, 2-D, and 3-D fluid mechanics. PINNs are emerging as useful tools to simulate fluid flow for forward problems, where the solutions to the governing equations are approximated, and inverse problems, where parameters characterising the governing equation are extracted from the training data. Solving ill-posed inverse problems is beyond the reach of both traditional computational methods and purely data-driven machine learning methodologies [21].

Sun et al. [19] used PINNs to approximate the solutions to the Navier-Stokes equations in order to develop surrogate models of incompressible fluid flows for cardiovascular applications. The authors demonstrated that the application of a PINN methodology enabled the development of accurate surrogate models without using any labelled data (i.e., CFD simulation data). This work therefore shows that PINNs circumvent the need for large training datasets. Zhu et al. [20] employed a physics-constrained learning approach to develop surrogate models for PDEs describing steady-state Darcy flow without labelled training data. The authors demonstrated that given out-of-distribution test inputs, the generalisation performance of the physics-constrained surrogate

model was consistently better than that of the data-driven alternative. Ang and Ng [22] applied PINNs to develop surrogate models for fluid flows around aerofoils for different angles of attack. The surrogate models were used to make accurate predictions for the pressure and velocity fields. Once trained, the PINN surrogate models were able to generate results up to 4.5 times faster than conventional CFD solvers, demonstrating the significant reduction in computational cost that can be achieved with PINNs. Mao et al. [23] successfully employed PINNs to solve both forward and inverse problems for the 1-D and 2-D Euler equations that model high-speed aerodynamic flows. Cai et al. [21] reviewed the application of PINNs to problems in the field of fluid mechanics with reference to case studies that covered 3-D incompressible flow, compressible flow, and biomedical flow. The authors demonstrated that PINNs are emerging as useful tools to simulate fluid flow for both forward and inverse problems.

Following the successful application of PINNs to problems in the field of multidimensional fluid mechanics, research into this modelling methodology has gained significant momentum, resulting in numerous advances in the methodology. Jagtap et al. [24] investigated using adaptive activation functions in PINNs to improve their performance. By introducing a scalable hyperparameter into the activation function, the authors were able to adjust the gradient of the activation function and ultimately increase the rate of convergence of the neural network. Additionally, the use of an adaptive activation function improved the accuracy of the solution to the Burgers' equation. Wang et al. [25] applied the PINN methodology to a variety of case studies, including the Helmholtz equation and flow in a lid-driven cavity. The authors proposed using a learning rate annealing algorithm to apply dynamic weights to the various terms constituting the given PINN loss functions. These weights were used to balance the interplay between the different loss terms. This approach improved both the trainability and predictive accuracy of the PINNs. Laubscher [26] investigated the effect of different PINN architectures by applying two PINN modelling methodologies to predict the momentum, species, and temperature distributions of a dry air humidification problem in a 2-D rectangular domain. The first approach used a single PINN to solve all required PDEs and boundary conditions at once. In contrast, the second approach used three individual PINNs to solve the set of mass and momentum PDEs, species PDEs, and energy PDEs and their respective boundary conditions separately. The author showed that the PINN losses for the segregated-network PINN were on average 62% lower than those for the single-network PINN. Furthermore, while the single-network PINN struggled to enforce the species transport equation in all areas of the domain, the segregated-network PINN solved all the PDEs successfully.

Laubscher and Rousseau [1] applied a PINN methodology to develop steady-state and transient models for incompressible laminar flow with heat transfer. The authors solved the relevant PDEs in their non-dimensionalised forms to ensure that the loss values for the different equations had similar orders of magnitude. Applying this formulation of the equations ensured that all components of the loss function were considered equally during the loss function minimisation training process. Consequently, both the accuracy and the training time of the resulting PINN surrogate model were improved. Markidis [27] evaluated the

potential of using PINNs as linear solvers, using the 2-D Poisson equation as a case study. In this work, a transfer learning technique was applied to train the PINN. Using this technique, two different networks were trained to solve the Poisson equation for different source terms. The trained parameters from the first network were then leveraged and used to initialise the network parameters for the subsequent PINN solver, thereby transferring knowledge from the first network to the second. This technique reduced both the number of epochs used to train the final PINN solver as well as the computational cost of the training process.

2.4 Summary

The literature review has revealed that:

- Traditional thermofluid process models consist of numerical solvers developed using relevant fundamental balance equations. While accurate, these models can be computationally expensive and time-consuming because of the inherent iterative nature of the equation solvers that are generally used.
- Reduced order models have been developed to address the complexity and computational expense of traditional equation-based thermofluid process models. However, the reduction in complexity comes at a cost to accuracy.
- Data-driven machine learning techniques, such as ANNs, have been applied to develop computationally efficient models. These techniques require large training datasets containing many examples of corresponding inputs and outputs. ANN models have been used to produce a significant increase in computational speed at a high degree of accuracy when compared to traditional thermofluid process models.
- The usefulness of purely data-driven artificial neural networks is limited by the fact that the predictive accuracy of purely data-driven artificial neural networks is directly linked to the size and distribution of the training data. This is particularly significant for thermofluid process modelling applications where the prohibitive cost and low fidelity of experimental data in industry-scale thermofluid systems make large datasets difficult to obtain.
- PINNs have been developed to generate accurate surrogate models using governing physics equations to construct the loss function, instead of using labelled training data. This method thus circumvents the need for large training datasets.
- Recent momentum in research into physics-informed neural networks has resulted in numerous advances in the methodology including adaptive activation functions, dynamic weighting of the loss function constituents, the use of non-dimensionalised forms of the governing equations, and transfer learning approaches.

Despite the recent momentum of research into the PINN modelling methodology, it has yet to be applied to thermofluid process modelling applications (i.e., integrated 1-D network modelling). Thus, no examples of

PINNs applied to integrated thermofluid systems have been observed in the literature. In the context of this study, integrated 1-D network modelling refers to a technique that does not include spatial considerations when modelling the fluid flow path. Details of this modelling technique are provided in Chapter 3.1 (pg. 13).

An overview of the theoretical background and methodologies relating to conventional thermofluid process modelling, conventional multilayer perceptron neural networks, and physics-informed neural networks is provided in the following chapter.

Chapter 3. Theoretical background and methodology

In this chapter, the theoretical background and methodologies that were applied in this study are outlined. These include the existing methodology for physics-based models of integrated thermofluid systems, the existing methodology for multilayer perceptron neural networks, and, finally, the new methodology developed during this study for applying physics-informed neural networks to 1-D thermofluid process modelling. A description of the data generation approach implemented in this study is also provided here.

3.1 Conventional physics-based thermofluid network modelling

In conventional physics-based modelling of 1-D integrated thermofluid systems, the layout of the system being analysed is described in terms of nodes and elements, as shown schematically in Figure 1. Elements are control volumes that represent physical components such as pipes, valves, heat exchangers, boilers, or turbines. An element may also represent a single subdivision or increment of a physical component, such as a heat exchanger, which is discretised into several control volumes. Each element has one inlet and one outlet. The properties of the fluid within the element are assumed to be represented by a single weighted average value calculated between the inlet and the outlet of the element. Nodes represent any point where the inlets or outlets of two or more elements are joined and may also represent a physical reservoir or tank. Nodes may therefore have multiple inlets and outlets. The fluid properties within a node are assumed to be homogeneous and represented by a single averaged value.

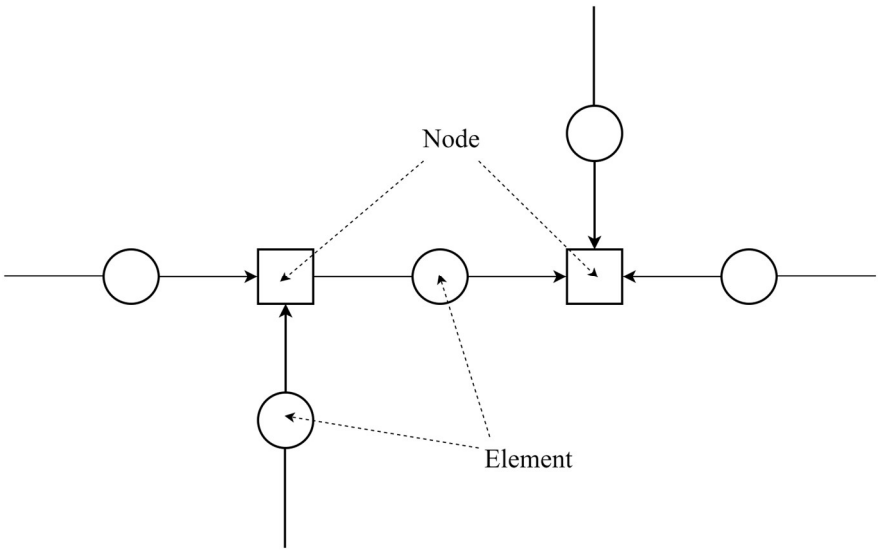


Figure 1. Thermofluid network represented by nodes and elements.

In 1-D integrated thermofluid network modelling applications, the governing physics equations are the mass, energy, and momentum balance equations. The mass flow rates, total enthalpies, and total pressures are the

fundamentally conserved quantities in the respective balance equations. The mass and energy balance equations are written for each node in the system, while the momentum balance equation is written between the inlet and outlet of each element in the system. By applying the governing equations to every node and element in the system in this way, a set of equations is generated that can be solved simultaneously to provide an accurate model of the system. The generic forms of the governing equations for 1-D integrated thermofluid network modelling are derived under the assumption of one-dimensional, steady-state flow through the network elements and are given by [28]:

Mass balance (written for each node):

$$\sum \dot{m}_e = \sum \dot{m}_i \tag{1}$$

Energy balance (written for each node):

$$\sum \dot{m}_e h_{0e} = \sum \dot{m}_i h_{0i} + \dot{Q} - \dot{W} \tag{2}$$

Momentum balance (written for each element):

$$p_{0e} = p_{0i} + \Delta p_{0M} - \Delta p_{0L} \tag{3}$$

In Equations (1), (2), and (3), the subscript i denotes any quantity measured at the inlet of a control volume, while the subscript e denotes any quantity measured at the exit of a control volume. For high-level integrated system analyses, which are the focus of this study, it can be assumed that the differences in the kinetic and potential energy terms between the inlets and outlets that form part of the total enthalpy, h_0, and total pressure, p_0, are negligible. This means that the total property values may be approximated with the static property values, and therefore $h_0 \approx h$ and $p_0 \approx p$ in Equations (2) and (3), respectively.

In Equation (2), $\dot{Q}$ [W] is the rate of heat transfer to the fluid and $\dot{W}$ [W] is the rate of work done by the fluid. In Equation (3), Δp_{0M} [Pa] is the total pressure rise due to work done on the fluid and Δp_{0L} [Pa] is the total pressure loss. These terms constitute the component characteristic parameters that must be calculated when implementing the balance equations. The methods and equations used to calculate the component characteristics vary depending on the configuration of the integrated thermofluid system being analysed.

In addition to the component characteristics, the fluid properties at each node and through each element are important parameters that must be calculated when implementing the balance equations. In conventional physics-based 1-D integrated thermofluid network modelling, the fluid properties can be calculated using the open-source *CoolProp* fluid property library and its *PropsSI* function [29]. The *PropsSI* function calculates the fluid properties implicitly by solving the equations of state using the Newton-Raphson root-finding method.

Two approaches are typically implemented to solve the set of governing equations for the 1-D integrated thermofluid system. The first approach involves writing the set of governing equations for the system (i.e., the steady-state mass, momentum, and energy balance equations) in a linearised form. The linear system of equations may be solved either directly by inverting the matrix of coefficients or by using an iterative solution approach, such as the Gauss-Seidel method. For large, complex thermofluid networks, using an iterative approach may be faster than the direct matrix inversion approach. However, the thermofluid networks analysed in this study are relatively small with simple network configurations. The direct matrix inversion approach was therefore used in this study.

To implement direct matrix inversion, the linearised equations are used to construct two matrix equations, namely, one that couples the mass and momentum balance equations and another for the energy balance equations. The general form of the matrix equations is given as follows:

$$[\boldsymbol{D}][\boldsymbol{e}] = [\boldsymbol{f}] \tag{4}$$

In Equation (4), $\boldsymbol{D}$ represents an $N \times N$ matrix of coefficients, where N is the number of unknown quantities, $\boldsymbol{e}$ represents an $N \times 1$ vector of unknown quantities (e.g., unknown enthalpies or pressures), and $\boldsymbol{f}$ is an $N \times 1$ vector containing all source terms. The set of linearised equations is solved by inverting the matrix of coefficients and making the vector of unknown quantities the subject of the formula. This process is given as:

$$[\boldsymbol{e}] = [\boldsymbol{D}]^{-1}[\boldsymbol{f}] \tag{5}$$

Following this, the coefficient matrix and source term vector are updated, and the process is repeated until convergence.

Alternatively, the set of governing equations for the system can be left in a non-linear form. The non-linear equations are then solved simultaneously by iteratively applying a root-finding function, such as the *fsolve* function in *Python*, until sufficient accuracy is obtained and the solution is converged. While the *fsolve* root-finding function requires a Jacobian to find a solution, the required Jacobian need not be explicitly provided via the *fprime* argument. Instead, the required Jacobian may simply be estimated numerically within the fsolve function using the finite difference method by default, as was the case in this study.

3.2 Multilayer perceptron neural networks

In this section, a brief overview of the theoretical background related to MLP neural networks is presented.

Multilayer perceptron (MLP) networks are the most used category of artificial neural networks [30]. These networks can be trained to fit non-linear relationships between input and output variables accurately through a supervised learning process in which the network is exposed to examples of desired output values (i.e., labels)

for the given input values. MLP networks consist of systems of interconnected signal processing nodes, known as neurons, which are divided into three sections: the input layer, hidden layers, and output layer, as shown in Figure 2.

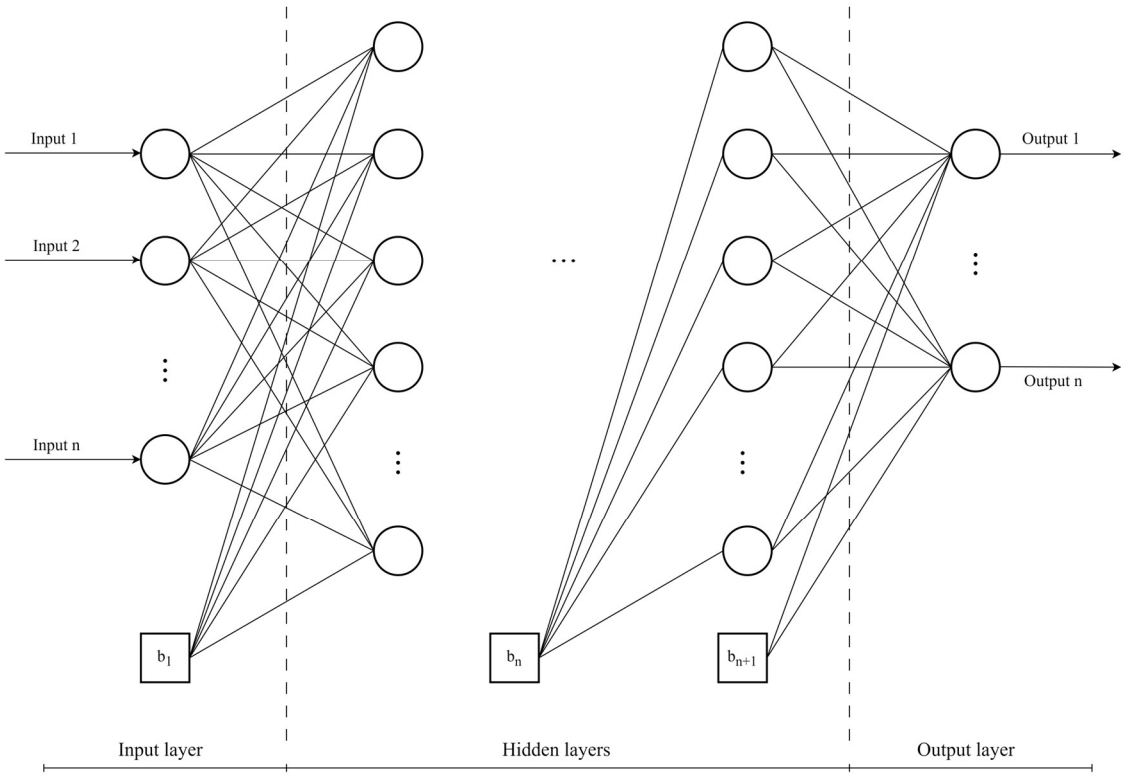


Figure 2. Typical architecture of a multilayer perceptron neural network.

The number of neurons in the input and output layers is dictated by the number of independent variables that are being fed into the network, and the number of dependent variables that are predicted by the network, respectively. These values are dependent on the individual system being modelled. The number of neurons per hidden layer, as well as the number of hidden layers, are hyperparameters that define the complexity of the model. These hyperparameters are specified by the designer and are tuned during the training and validation process to find the best-performing network configuration. When a network contains multiple hidden layers, as well as a large number of neurons per hidden layer, it is considered to be a deep neural network [31]. In a fully connected, or dense, MLP network, each neuron in each layer is connected to every other neuron in both the previous and subsequent layers. Each connection has an associated weight value which is applied to the signal being passed between the neurons.

MLP networks use the forward propagation algorithm to make predictions. This process involves passing input parameters from the neurons in the input layer to every neuron in each downstream layer of the network, ending with the output layer. The output signals from the final output layer are then the predictions made by the MLP network. During the forward propagation process, the output signal of each neuron in the input and hidden layers is multiplied by a weight, w, and summed with the other incoming weighted signals. A bias

value b is added to the sum of weighted incoming signals. The sums of the weighted incoming signals, including the bias values, are passed into activation functions to calculate the activated output signals of given neurons. The forward-propagation equation is given for the activated output of the neuron in the j^{th} row of the l^{th} layer of the MLP network, a_j^l, as [32]:

$$a_j^l = \sigma_l(\sum_{k=1}^{n} a_k^{l-1} w_{kj}^l + b_j^l) \tag{6}$$

In Equation (6), w_{kj}^l is the weight parameter linking the neuron in the k^{th} row of the preceding layer $l-1$, with the neuron in the j^{th} row of the current layer l, b_j^l is the bias parameter for the neuron in the j^{th} row of the current layer l, and σ_l represents the activation function for the current layer l. In this case, the preceding layer $l-1$ contains n neurons. The final weighted input into the j^{th} neuron in layer l can also be denoted by z_j^l and is given by [32]:

$$z_j^l = \sum_{k=1}^{n} a_k^{l-1} w_{kj}^l + b_j^l \tag{7}$$

Activation functions, $\sigma_l()$, are used to model non-linearities present in the data. Activation functions are specified for an entire layer in a neural network. These functions transform the final weighted input signal into a given neuron, z_j^l, to an output value that will be fed into downstream neurons. Furthermore, activation functions determine whether a specific neuron is activated or not, based on the intensity or magnitude of the final weighted input signal. Many activation functions can be applied to MLP networks. The linear, logistic, hyperbolic tangent (tanh), and rectified linear unit (ReLU) functions are commonly used activation functions. The profiles of these activation functions are shown in Figure 3, and they are mathematically defined as [31]:

$$\sigma_{Linear}(z_j^l) = z_j^l \tag{8}$$

$$\sigma_{Logistic}(z_j^l) = \frac{1}{1+e^{-z_j^l}} \tag{9}$$

$$\sigma_{\tanh}(z_j^l) = \tanh(z_j^l) = \frac{2}{1+e^{-2z_j^l}} - 1 \tag{10}$$

$$\sigma_{\mathrm{Re}LU}(z_j^l) = \begin{cases} z_j^l & if \quad z_j^l > 0 \\ 0 & otherwise \end{cases} \tag{11}$$

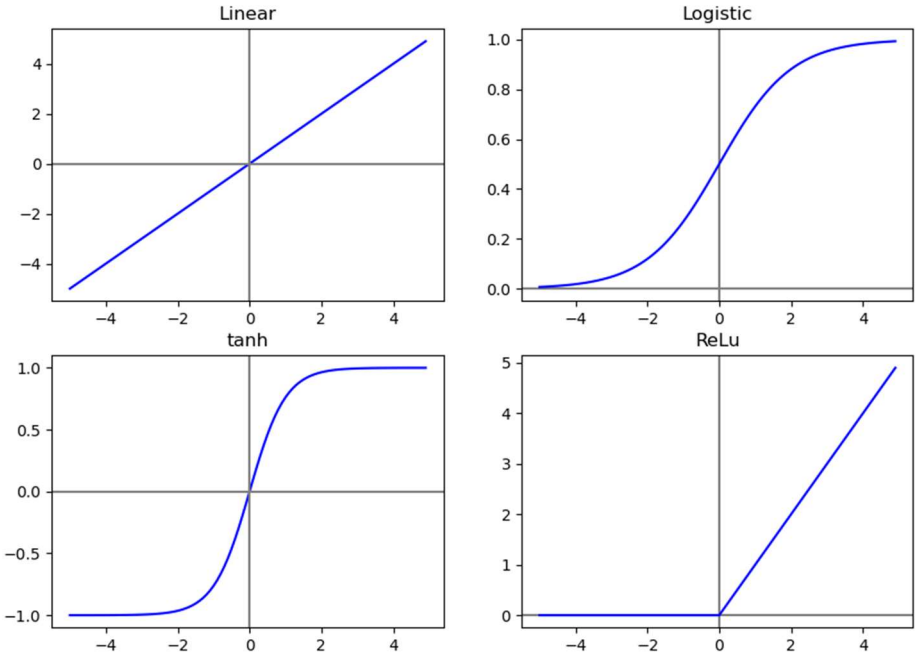


Figure 3. Linear, logistic, tanh, and ReLU activation functions.

The choice of activation function for an MLP network is largely problem specific; however, for networks applied to regression problems, the linear activation function is used for the output layer and the ReLU or tanh activation functions are typically recommended for the input and hidden layers [26].

The network weight and bias values constitute the trainable network parameters. The network weights for the different layers are randomly initialised and set to arbitrary values. A common approach for weight initialisation is the normal Glorot initialisation scheme in which weights are initialised with a mean of 0 and a standard deviation σ_x calculated as [31]:

$$\sigma_x = \sqrt{\frac{2}{fan_{in} + fan_{out}}} \tag{12}$$

where fan_{in} and fan_{out} refer to the input and output dimensions of a given network layer, respectively. The bias values are commonly initialised to the value of 0.

Once initialised, the neural network is trained, and the trainable network parameters (i.e., the weight and bias values) are optimised to minimise a selected network loss function. The loss function provides a measure of how well the network represents the non-linear relationship between the input and output variables. For MLP networks applied to regression tasks, the mean squared error (MSE) is typically used as the loss function. The MSE loss considers the differences between the predicted values and the desired output values from the training dataset. These differences are known as the residuals. The use of the desired output values implies that

MLP neural networks rely on supervised learning, as opposed to unsupervised learning. The MSE loss is calculated as [31]:

$$MSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{13}$$

In Equation (13), n is the number of observations in the training dataset, $\boldsymbol{y}$ is an $n \times 1$ vector of the desired output values from the training dataset, and $\hat{\boldsymbol{y}}$ is an $n \times 1$ vector of the predicted values from the network.

The network loss function is minimised using a gradient-based optimisation process. During this process, an automatic differentiation technique such as, for example, the backpropagation algorithm, is used to calculate the gradient of the network loss function (i.e., the network's error) with respect to each trainable network parameter, starting from the output layer and propagating backwards to reach the input layer. The error gradients are used to quantify the contribution of each weight and bias value to the overall network loss. Using the calculated error gradients, a gradient descent-based optimisation algorithm is then applied to adjust the network parameters. There are several different gradient descent-based optimisation algorithms that can be used such as, for example, stochastic gradient descent (SGD), momentum optimisation, AdaGrad, RMSProp, and adaptive moment estimation (Adam) [30]. The network parameters are trained to minimise the loss function using an iterative implementation of the forward propagation process, loss function evaluation, backpropagation process, and gradient descent-based parameter optimisation process.

During training, the network parameters are optimised to produce the best predictions for the training dataset. For MLP neural networks, the training process is terminated once a specified number of epochs has been completed [30]. One epoch refers to a single complete pass through the training dataset. A common problem encountered during the training of MLP networks is that of overfitting, which occurs when the model learns the training data too closely, rather than learning the underlying relationship. A cross-validation process must therefore be implemented in conjunction with the training process to select a combination of hyperparameters that mitigates the likelihood of overfitting. To complete the cross-validation process, a subset of the training data, known as the validation dataset, is passed through the network to estimate the generalisation error and inform the hyperparameter selection. Once trained, the performance of the MLP model is then evaluated on a hold-out test set, which contains new, unseen samples from the same distribution as the training dataset. Typically, approximately 70% of the overall dataset is used for training, 10% is used for validation, and 20% is reserved for testing the trained model [30].

Hyperparameters are model configuration settings that are not learned from the data during training. They are instead user-defined inputs that are used to control various aspects of the network's architecture, optimisation process, and training behaviour. In addition to the number of hidden layers and the number of neurons per hidden layer, tuneable network hyperparameters include the learning rate, batch size, and number of epochs.

The hyperparameters controlling the network architecture (i.e., the width and depth of the hidden layers) have the most pronounced effect on the model performance and should be tuned first. The model learning rate controls the step size taken to update the network parameters during the optimisation process and therefore affects how quickly or slowly the model converges during training. The batch size is a hyperparameter used in minibatch learning where the network parameters are updated after a subset of the training data (i.e., a minibatch) is passed through the network, rather than after each sample. An epoch is complete when all minibatches have been passed through the network. This minibatch-learning approach can improve the efficiency of the model [31]. The No Free Lunch (NFL) theorem proposed by Wolpert [33] suggests that no machine learning algorithm is universally better than another across all problems. In the context of neural networks, the NFL theorem suggests that the neural network architecture and hyperparameters must be selected based on the characteristics of the specific problem being addressed. Experimentation, empirical evaluation, and iterative selections are often required to identify the most effective hyperparameter configuration for the given problem. Neural network hyperparameters should therefore be selected using an iterative, empirical tuning process.

ANNs, like many other machine learning methods, do not perform well when the input parameters have very different scales [31]. This is because the MSE losses for different output parameters will have varying orders of magnitude, resulting in a biased optimisation process during which the optimiser favours the disproportionately large losses while neglecting smaller losses. A biased optimisation process ultimately leads to prolonged training times and inaccurate results [26]. Therefore, a feature scaling transformation must be applied to the training data prior to use. Normalisation, or min-max scaling, is generally applied to training data for MLP networks and involves scaling all input and target output values to a range between 0 and 1. Normalisation is applied to each input and output parameter individually according to [31]:

$$\boldsymbol{x}_{scaled} = \frac{\boldsymbol{x} - x_{\min}}{x_{\max} - x_{\min}} \tag{14}$$

where $\boldsymbol{x}$ is an $n \times 1$ vector of all samples for a given input parameter, where n is the total number of observations, $x_{\min}$ is the minimum sample value for the given parameter, $x_{\max}$ is the maximum sample value for the given parameter, and $\boldsymbol{x}_{scaled}$ is an $n \times 1$ vector of the scaled samples for the given input parameter. Similarly, min-max scaling must be applied to the vector of target output parameters, $\boldsymbol{y}$, before training the MLP network. A post-processing step must be applied to extract the unscaled values for the network predictions by rearranging Equation (14) to make $\boldsymbol{x}$ — or $\boldsymbol{y}$, in the case of outputs — the subject of the equation and using the saved parameters from the initial scaling process.

3.3 PINN methodology for thermofluid network modelling

A description of the new methodology developed during this study for applying physics-informed neural networks to 1-D thermofluid process modelling is provided in this section.

The network structure of a PINN is the same as that of an MLP network. However, the loss function for a PINN is constructed differently from the loss function for an MLP. Instead of considering the mean squared error between the predicted output values and the known target values, the governing physics equations for the system being modelled are embedded directly into the PINN loss function. As mentioned in Chapter 3.1 (pg. 13), the governing physics equations for the analysis of integrated thermofluid systems are the mass, energy, and momentum balance equations given by Equations (1), (2), and (3). The corresponding conserved parameters (i.e., the mass flow rate through each element, and enthalpy and pressure at each node) are the variables for which the PINN models generate predictions.

The values of the terms in the balance equations are typically of different orders of magnitude that depend on the specific thermofluid network being analysed. For instance, one could find that $\dot{m} \approx 10^2$ kg/s, $\dot{m}h \approx 10^7$ W, and $p \approx 10^6$ Pa. These large differences will result in the energy balance equations providing larger contributions to the overall PINN loss function than the mass or momentum balance equations. The optimisation processes will therefore be biased towards minimising the disproportionately large energy balance losses, which negatively impacts the PINN training process. To prevent the biasing of the optimisation process, the PINN loss functions must apply the balance equations in their non-dimensional (i.e., normalised) form. To do this, the conserved variables must be non-dimensionalised using some reference quantities as follows:

$$\dot{m}^* = \frac{\dot{m}}{\dot{m}_\infty}, p_0{}^* = \frac{p_0}{p_\infty}, h_0{}^* = \frac{h_0}{h_\infty} \tag{15}$$

In Equation (15), * denotes a non-dimensional variable and $_\infty$ denotes a reference quantity. The magnitudes of the reference quantities (i.e., $\dot{m}_\infty, h_\infty,$ and p_∞) are selected such that they represent the maximum physically realistic values of the conserved variables for the thermofluid system being analysed.

To further avoid biasing the optimisation process, the vector of input parameters into the PINN must also be normalised. This involves the scaling of all input values to a range between 0 and 1 and is achieved by dividing each input parameter by a corresponding reference value, x_∞. The normalisation of the vector of input parameters therefore requires the definition of reference values for all categories of input parameters (e.g., a single reference value to be used for all temperatures or all pressure loss coefficients) in addition to the reference values for the conserved parameters.

The normalised values of the input parameters for a PINN model are calculated as follows:

$$\boldsymbol{x}_{scaled} = \frac{\boldsymbol{x}}{x_{\infty}} \tag{16}$$

In Equation (16), $\boldsymbol{x}$ is an $n \times 1$ vector containing the values of a given input parameter for all observations, where n is the number of observations, $\boldsymbol{x}_{\infty}$ is the corresponding reference value for the category of input parameter (e.g., the temperature reference value or the pressure reference value), and $\boldsymbol{x}_{scaled}$ is an $n \times 1$ vector containing scaled values of the given input parameter for all observations. Due to the normalisation of the input parameters, the predicted output values generated by the PINN model will also be normalised. A post-processing step must therefore be applied to extract the unscaled values for the PINN predictions. This is achieved by rearranging Equation (16) to make $\boldsymbol{x}$ — or $\boldsymbol{y}$, in the case of outputs — the subject of the equation and using the relevant reference value.

The non-dimensionalised balance equations are applied throughout the thermofluid system to generate a set of residual functions. This entails writing the normalised balance equations for each node and element in the system with all quantities on one side of the equal sign. Using residual loss functions enables one to evaluate how well the predicted variables satisfy the governing equations. The greater the accuracy of the predictions, the lower the residual values.

Using the normalised balance equations, the generic residual loss functions for integrated thermofluid systems are given as:

$$f_{mass} = \sum \dot{m}_e^* - \sum \dot{m}_i^* \tag{17}$$

$$f_{energy} = \sum \dot{m}_e^* h^*_{0e} - \sum \dot{m}_i^* h^*_{0i} - \frac{\dot{Q}}{\dot{m}_{\infty} h_{\infty}} + \frac{\dot{W}}{\dot{m}_{\infty} h_{\infty}} \tag{18}$$

$$f_{momentum} = p^*_{0e} - p^*_{0i} - \frac{\Delta p_{0M}}{p_{\infty}} + \frac{\Delta p_{0L}}{p_{\infty}} \tag{19}$$

The residual loss functions for the balance equations are then collected to form combined residual loss functions. The combined mass balance residual loss, J_{mass}, energy balance residual loss, J_{energy}, and momentum balance residual loss, J_{mom}, are calculated as mean squared losses and are given by:

$$J_{mass} = \frac{1}{N_{mass}} \sum_{i=1}^{N_{mass}} (f_{mass,i})^2 \tag{20}$$

$$J_{energy} = \frac{1}{N_{energy}} \sum_{i=1}^{N_{energy}} (f_{energy,i})^2 \tag{21}$$

$$J_{momentum} = \frac{1}{N_{momentum}} \sum_{i=1}^{N_{momentum}} (f_{momentum,i})^2 \tag{22}$$

In Equations (20), (21), and (22), the individual residuals for each instance of the mass, energy, and momentum balance equations (i.e., the mass and energy balance residuals for each node, and the momentum balance residual for each element) are squared and summed before being divided by the total number of instances of each balance equation (i.e., the number of nodes or elements). The number of instances of each balance equation is given by the quantity N_α where α refers to the different categories of balance equations.

The residual losses are then added to obtain the overall PINN loss function as follows:

$$J_{loss} = J_{mass} + J_{energy} + J_{momentum} \tag{23}$$

The validation of the loss function is an important process required to develop PINN models for integrated thermofluid systems. During the validation process, known true values for the predicted quantities — preferably obtained from a benchmark solution — are passed through the entire loss function calculation. The relevant fluid properties, component characteristics, residual values, and overall loss value are then calculated using the true values for the predicted quantities. As the input quantities are true values, they should satisfy the balance equations exactly, and the magnitude of the overall loss value should be negligibly small. However, if the overall loss value is not negligibly small, it indicates that the PINN loss function has not been set up to correctly represent the underlying physics for the integrated thermofluid system. The loss function validation approach can therefore be applied to verify that the PINN loss function accurately represents the integrated thermofluid system.

The overall PINN loss value J_{loss} is the value that is minimised during the PINN optimisation process. By minimising this value, the PINN model is trained to predict the mass flow rates, enthalpies, and pressures that satisfy the balance equations. Therefore, PINNs applied to integrated thermofluid systems modelling are used as non-linear solvers for the mass, energy, and momentum balance equations. To ensure that the PINN model predictions satisfy the balance equations with a suitable degree of accuracy, the training of these models was terminated based on a tolerance value for the total model loss, rather than using a set number of epochs. In this study, a value of 1E–6 was selected as the desired tolerance for the total PINN model loss. A second restriction of a maximum number of epochs was placed on the training process of the PINN models to prevent the process from running indefinitely, should the models not be able to reach the desired loss tolerance.

A schematic of the PINN layout for integrated thermofluid systems, with an overview of the predicted quantities for a given input vector $\boldsymbol{x}$, is shown in Figure 4. This figure also indicated that the fluid

properties (e.g., ρ, c_p, T) and component characteristics (e.g., $\dot{W}, \dot{Q}, \Delta p_{0M}, \Delta p_{0L}$), which form inputs to the residual loss functions for the balance equations, are calculated outside of the PINN loss function. These values are updated at the beginning of every training iteration and are calculated using the predicted values of the mass flow rates, enthalpies, and pressures for the current training iteration. The residual functions use the fluid property and component characteristic values to estimate the overall model loss value.

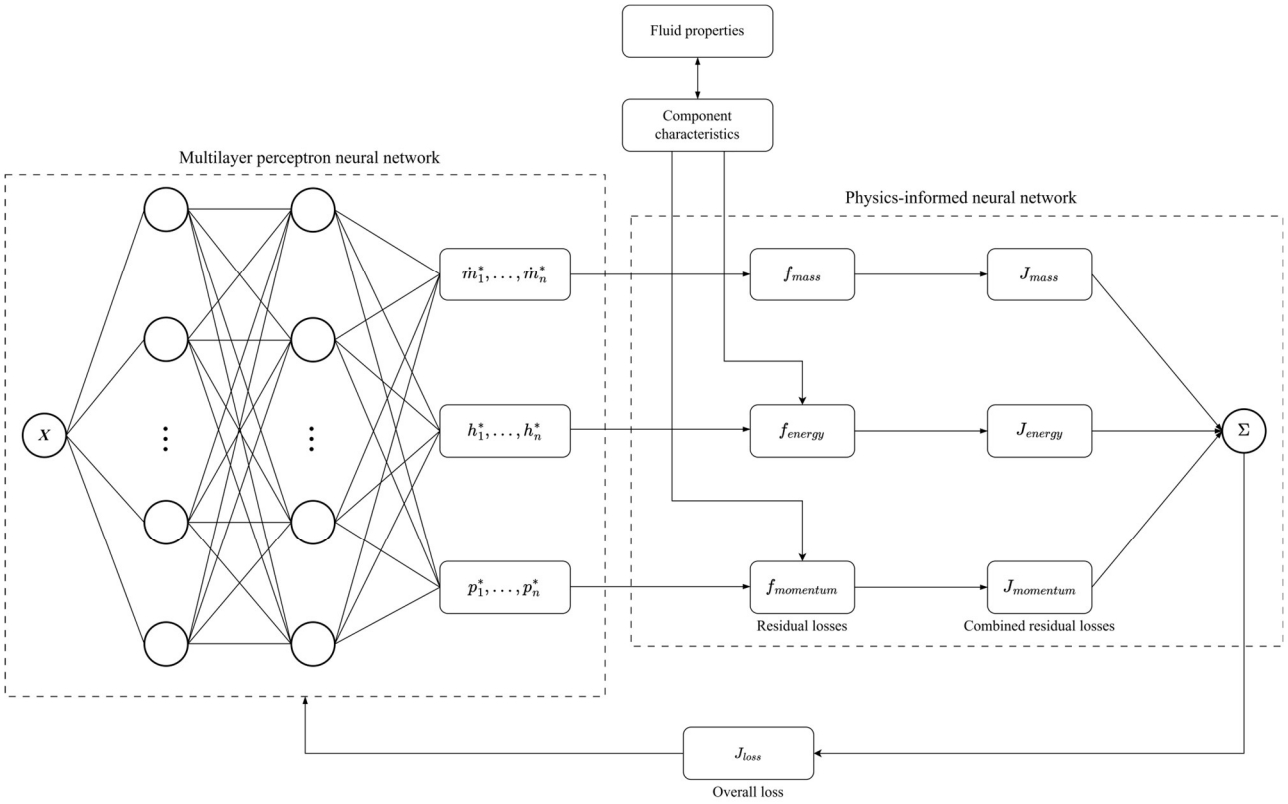


Figure 4. Physics-informed neural network layout for integrated thermofluid systems.

3.4 Data generation

In this study, the data used to train and evaluate the machine learning models were generated via simulations which used conventional physics-based thermofluid process models for the relevant integrated thermofluid systems. The input parameters for these conventional models were varied within a physically realistic range to simulate a range of operating conditions. A design of experiment (DOE) process was implemented to ensure that the selected sample points adequately represented the entire design space [34]. Generally, the number of samples selected is relatively small compared to the entire design space. A DOE process therefore ensures that the maximum amount of information about the design space is provided with the minimum number of samples.

The statistical method known as Latin-Hypercube sampling (LHS) developed by McKay et al. [35] was selected as the DOE process used to generate the data for this study. LHS generates a near-random subset of

parameters with a uniform distribution from a multi-dimensional design space by selecting M samples from each of the N input parameters ($x_1, x_2, \ldots, x_N$). To do this, the range of each input parameter is divided into M equally probable non-overlapping intervals. A single value is then selected from each interval, ensuring that no value is repeated. The number of samples selected, M, is a pre-defined value. The M random samples for a single input parameter are randomly combined with the M samples for all N input parameters, thereby generating a randomly sampled subset of $M \times N$ input parameters.

In this study, the LHS method was implemented using the pyDOE *Python* library [36]. The LHS method was used to generate a set of M samples for each of the N input parameters. The sample values ranged from 0 to 1 and followed a uniform distribution. These sample values were then transformed from having the uniform distribution produced by the LHS to having a normal distribution. This transformation involved specifying mean and standard deviation values for all input parameters and employing the *SciPy* stats function [37]. The randomly generated samples for all input parameters therefore followed a normal distribution.

A description of the application of the PINN modelling methodology to two case studies, namely, a simple heat exchanger network and a recuperated closed Brayton cycle, is provided in the following chapter. It outlines the two thermofluid systems and their associated complexities, as well as the PINN model setup for each case study. Finally, an overview of the performance of the PINN models for the two case studies is provided.

Chapter 4. PINNs applied as numerical solvers

In this study, the PINN modelling methodology was first applied to two case studies involving complex network structures but containing simple component characteristic calculations. The case studies were selected as they highlight important phenomena that occur when analysing integrated thermofluid network modelling. The PINN models were used to investigate whether the PINN modelling methodology could be applied successfully to integrated thermofluid systems. Therefore, the PINN models for the two initial case studies were trained to make predictions for single samples only. A description of the two case studies and an overview of the PINN model setup for these case studies is provided here. Finally, the results generated with these PINN models are presented along with a discussion of the PINN model performance.

4.1 Case study descriptions

The first case study modelled using the PINN methodology was a simple heat exchanger network. In this system, heat is transferred between two different fluid streams via various heat exchangers. The two fluids used in the heat exchanger network are dry air and sCO2. The heat exchanger network was selected as it encapsulates the important phenomena and complexities associated with thermofluid process modelling. This system requires the consideration of two different fluids that interact via heat transfer through heat exchangers, as well as the consideration of complexities associated with configuring components in parallel and series. The layout of the heat exchanger network is shown in Figure 5.

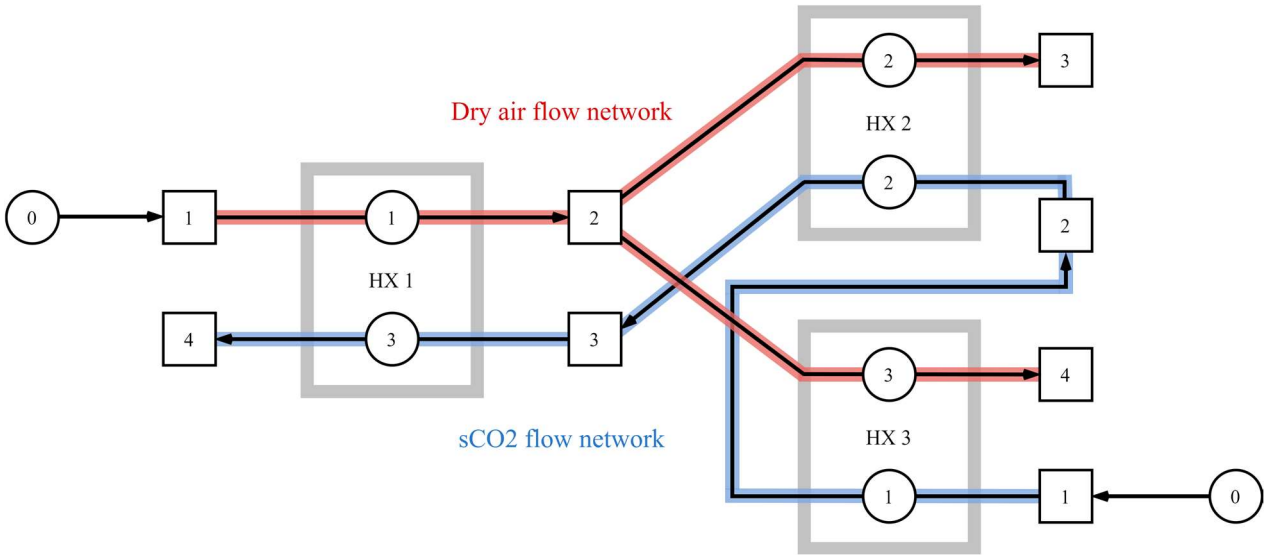


Figure 5. Thermofluid network model superimposed onto the physical layout of the heat exchanger network. Heat exchanger – HX.

The PINN model for the heat exchanger network predicts the enthalpies and pressures at each of the nodes, as well as the mass flow rates through each of the elements for both the dry air and sCO2 flow networks. The input parameters for the heat exchanger network are given in Table 1. These parameters were varied using the LHS method described in Chapter 3.4 (pg. 24) to generate different sample points, thus simulating a variety of operating conditions. The ranges within which these parameters were varied are also shown in Table 1.

Table 1. Input features for the heat exchanger network with the ranges in which they were varied.

Parameter	**Symbol**	**Unit**	**Min. value**	**Max. value**
Air outlet pressure	$p_{out,air}$	MPa	0.085	0.105
sCO2 outlet pressure	$p_{out,CO2}$	MPa	7.4	30
Air inlet mass flow rate	$\dot{m}_{in,air}$	kg/s	7	14
sCO2 inlet mass flow rate	$\dot{m}_{in,CO2}$	kg/s	3	9
Air inlet temperature	$T_{in,air}$	K	523	923
sCO2 inlet temperature	$T_{in,CO2}$	K	373	673
Heat exchanger total loss coefficients	K_1, K_2, K_3	-	500	1000
Heat exchanger effectiveness values	$\varepsilon_1, \varepsilon_2, \varepsilon_3$	-	0.6	0.9

The second case study modelled using the PINN methodology was a recuperated closed sCO2 Brayton cycle. Research has shown that power cycles for electricity generation that use sCO2 as the working fluid are increasingly considered to be preferable over conventional steam Rankine cycles [38]. This is because of the higher thermal efficiencies that can be achieved with sCO2 cycles compared to steam Rankine cycles, and the significant reduction in the overall size of the sCO2 systems as smaller, more compact turbomachinery can be used. The recuperated sCO2 Brayton cycle was selected as it aligns with the focus of the Applied Thermofluid Process Modelling (ATProM) Research Unit under which this study was conducted. Furthermore, this cycle requires the consideration of different turbomachine components with associated non-linear performance characteristics, in addition to heat transfer processes, thus encompassing additional phenomena typically encountered in integrated thermofluid systems modelling. The process flow diagram for the recuperated Brayton cycle is shown in Figure 6.

The PINN model for the recuperated Brayton cycle predicts the enthalpies and pressures at each node, as well as the total mass flow rate through the cycle. The input parameters for the recuperated Brayton cycle are given in Table 2. These parameters were varied using the LHS method described in Chapter 3.4 (pg. 24) to generate different sample points, thus simulating a variety of operating conditions. The ranges within which these parameters were varied are also given in Table 2.

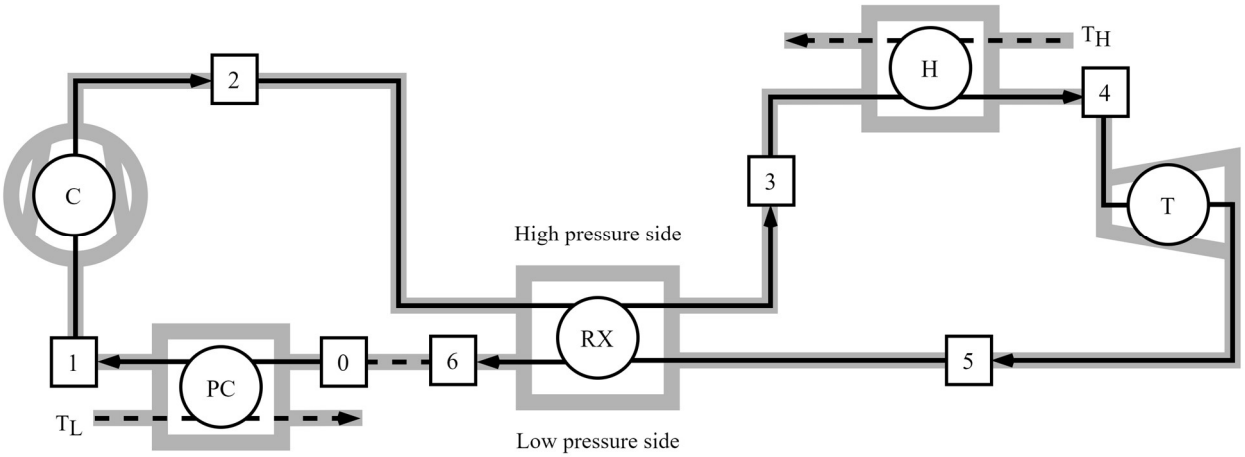


Figure 6. Thermofluid network model superimposed onto the process flow diagram for the recuperated Brayton cycle. Compressor - C, recuperator heat exchanger - RX, heater - H, turbine - T, pre-cooler - PC.

Table 2. Input features for the recuperated Brayton cycle with the ranges in which they were varied.

Parameter	**Symbol**	**Units**	**Min. value**	**Max. value**
Minimum cycle temperature	T_L	K	308	323
Maximum cycle temperature	T_H	K	773	923
Minimum cycle pressure	p_L	MPa	7.4	10
Heat exchanger total loss coefficient	K	-	10	60
Recuperator effectiveness	ε_{RX}	-	0.9	1.0
Heater effectiveness	ε_H	-	0.6	1.0
Pre-cooler effectiveness	ε_{PC}	-	0.6	1.0

The non-linear performance characteristics that must be considered when modelling the recuperated Brayton cycle include the pressure ratio and isentropic efficiency of the turbomachines present in the cycle. These performance characteristics are calculated as functions of the corrected mass flow rate through the components. The corrected mass flow rate is calculated as follows:

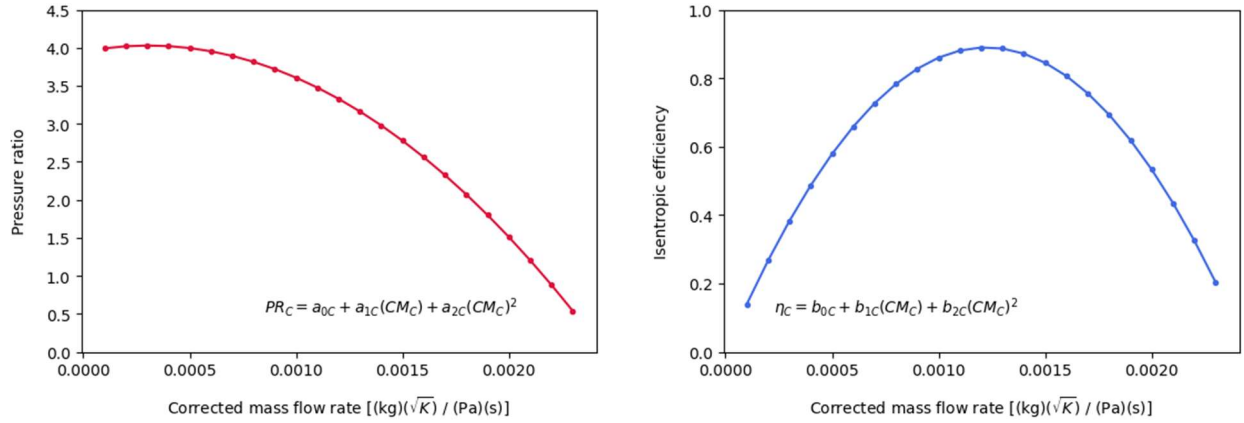
$$CM = \frac{\dot{m}\sqrt{T}}{p} \tag{24}$$

The relationships between the corrected mass flow rate and the pressure ratio or isentropic efficiency are represented by second order polynomial curves. For the purposes of the present study, these curves were not obtained from actual performance data, but simply mimic the typical trends observed in real-world turbomachines. The values of the coefficients were extracted from an existing model of the recuperated Brayton cycle system [39], which can be found in Appendix A (pg. 79). The coefficients for these polynomials are given in Table 3. The resulting component characteristic performance graphs for the compressor and

turbine are shown in Figure 7 and Figure 8, respectively. These figures illustrate the highly non-linear relationships introduced by the polynomial curve fits for which the PINN model needs to account.

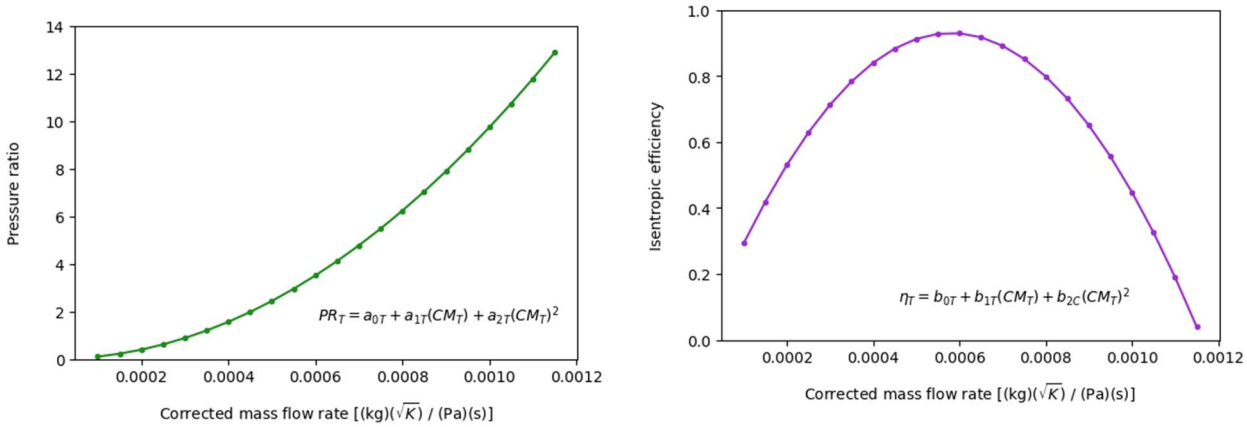
Table 3. Coefficients for the second order polynomial curves used to obtain the turbo machine performance characteristics.

Compressor		**Turbine**	
a_{0C}	3.947422E+00	a_{0T}	–8.437242E–23
a_{1C}	5.373216E+02	a_{1T}	3.313531E–09
a_{2C}	–8.776627E+05	a_{2T}	9.730980E+06
b_{0C}	–1.942890E–16	b_{0T}	1.054712E–15
b_{1C}	1.453743E+03	b_{1T}	3.199740E+03
b_{2C}	–5.936429E+05	b_{2T}	–2.752242E+06



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

*Figure 7. Polynomial curves for the compressor performance characteristics. (**a**) Pressure ratio vs. corrected mass flow rate; (**b**) Isentropic efficiency vs. corrected mass flow rate.*



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

*Figure 8. Polynomial curves for the turbine performance characteristics. (**a**) Pressure ratio vs. corrected mass flow rate; (**b**) Isentropic efficiency vs. corrected mass flow rate.*

The generic residual loss functions used for PINNs applied to integrated thermofluid systems, given by Equations (17)-(19), require component characteristics for the various thermofluid network components as inputs. The generic forms of the equations used to calculate these component characteristics for the simple heat exchanger network and the recuperated Brayton cycle are given as follows [28, 40]:

The total pressure rise due to work done on the fluid:

$$\Delta p_{0M} = p_{0i}\left(PR-1\right) \tag{25}$$

with $PR=\frac{p_{0e}}{p_{0i}}$ the total pressure ratio over the machine.

The total pressure loss:

$$\Delta p_{0L} = \frac{K}{\rho}\left|\dot{m}\right|\dot{m} \tag{26}$$

with ρ the average fluid density in the machine and K a non-dimensional total pressure loss factor.

The rate of work done by the fluid:

$$\dot{W} = \eta\dot{m}\left(h_{0i}-h_{0es}\right) \tag{27}$$

with η the isentropic efficiency of the machine, h_i the inlet enthalpy, and h_{es} the isentropic outlet enthalpy.

The rate of heat transfer to the fluid:

$$\dot{Q}_h = \frac{\dot{Q}_{\max,h}}{\left|\dot{Q}_{\max,h}\right|}\varepsilon_{HX}\dot{Q}_{\max,HX} \tag{28}$$

$$\dot{Q}_c = \frac{\dot{Q}_{\max,c}}{\left|\dot{Q}_{\max,c}\right|}\varepsilon_{HX}\dot{Q}_{\max,HX} \tag{29}$$

where ε_{HX} is the heat exchanger effectiveness, $\dot{Q}_{\max,HX}$ is the maximum possible rate of heat transfer, and the subscripts h and c refer to the hot and cold fluid streams, respectively.

In Equation (29), the maximum possible rate of heat transfer across the heat exchanger, $\dot{Q}_{\max,HX}$, is given by:

$$\dot{Q}_{\max,HX} = \max\left(\left|\dot{Q}_{\max,h}\right|,\left|\dot{Q}_{\max,c}\right|\right) \tag{30}$$

The maximum possible rate of heat transfer for the hot and cold fluid streams used in Equation (30) are defined as:

$$\dot{Q}_{\max,h} = \dot{m}\left(h_h\big|_{T_{ci}} - h_h\big|_{T_{hi}} \right) \tag{31}$$

$$\dot{Q}_{\max,c} = \dot{m}\left(h_c\big|_{T_{hi}} - h_c\big|_{T_{ci}} \right) \tag{32}$$

In Equation (31), $h_h\big|_{T_{ci}}$ and $h_h\big|_{T_{hi}}$ are the enthalpy of the hot fluid at the inlet temperature of the cold fluid and the hot fluid, respectively. In Equation (32), $h_c\big|_{T_{hi}}$ and $h_c\big|_{T_{ci}}$ are the enthalpy of the cold fluid at the inlet temperature of the hot fluid and the cold fluid, respectively.

For the heat exchanger network, the rate of work done by the fluid $\dot{W}$ and the total pressure rise due to work done on the fluid Δp_{0M} are set to zero for all heat exchangers. Similarly, for the recuperated Brayton cycle, the rate of work done by the fluid $\dot{W}$ and the total pressure rise due to work done on the fluid Δp_{0M} are set to zero for all heat exchangers. Additionally, the rate of heat transfer to the fluid $\dot{Q}$ is set to zero for all turbomachines.

A conventional, physics-based model of the heat exchanger network was developed in *Python 3.10.13* using the direct matrix inversion technique described in Chapter 3.1 (pg. 13). This model was used to generate the benchmark solution against which the PINN model predictions were compared. Similarly, a physics-based model was implemented in *Python 3.10.13* for the recuperated Brayton cycle. The conventional physics-based model of the recuperated Brayton cycle leveraged an existing model of the same Brayton cycle system [39], which can be found in Appendix A (pg. 79). Unlike the conventional model of the heat exchanger network, the conventional model for the recuperated Brayton cycle made use of an iterative solution methodology in which the non-linear root-finding function *fsolve* [41] was employed to solve the set of balance equations.

4.2 PINN setup

In this study, the PINN models for both the heat exchanger network and recuperated Brayton cycle were trained and tested on 10 different samples individually. The samples were selected to cover a range of operation conditions for the two thermofluid systems to demonstrate the range of applicability of the PINN methodology. The distribution of the samples was generated using the LHS method outlined in Chapter 3.4 (pg. 24). The initial values for the network weights of the two PINN models were randomly generated using the Glorot initialisation procedure outlined in Chapter 3.2 (pg. 15), and the network biases were all set to zero.

Two different optimisation approaches were used for the unsupervised training of the PINN models of the heat exchanger network and the recuperated Brayton cycle. One set of PINN models was trained using only the first-order Adam optimisation algorithm to update the network parameters. The Adam optimisation algorithm was selected as it has been shown to outperform other optimisation algorithms for training PINNs [42]. A second set of PINN models was trained using a hybrid optimisation approach in which the Adam optimiser was implemented first to prime the parameters before the second-order truncated Newton method (TNC) was applied in conjunction with the Adam optimiser to train the network to convergence [27]. The TNC optimiser was implemented by coupling the *SciPy optimize.minimize* function with the neural network. The analytical gradients of the cost function with respect to the trainable network parameters were made available to the TNC optimiser using the *TensorFlow Gradient.Tape* function. These gradients were passed into the *SciPy optimize.minimize* function, along with the objective cost function, to adjust and optimise the trainable network parameters.

When applying both the first- and second-order optimisers, the PINN models were first trained using only the first-order optimiser for a set number of iterations. This approach prevented the models from converging to a local minimum too quickly, as would be the case if only the second-order optimiser was implemented [43]. During the training of the PINN models for both the heat exchanger network and the recuperated Brayton cycle, the first 400 training iterations were completed using the Adam optimiser. After these 400 iterations, the models were refined using a combination of the TNC and Adam optimisers applied in a continuous loop. In this loop, the TNC optimiser was applied until the change in the model loss values was negligible. At this point, the PINN models were trained using the Adam optimiser for an additional 50 iterations before the TNC optimiser was implemented again. This loop continued until the model loss tolerance was reached.

In this study, the PINN models were implemented in *Python 3.9.16* using the *TensorFlow* framework, version *2.9.3*. The *TensorFlow* framework was selected as it is compatible with the second-order optimisation approach.

4.3 Results and discussion

Two adjustments were made to the loss function calculation and training strategy during the initial training and testing of the PINN models for the single sample cases. These adjustments are discussed in the first section of results. The remaining results are presented and discussed in five additional sections, namely, (i) the results of the loss function validation procedures, (ii) the results from the hyperparameter search processes, (iii) the accuracy of the PINN models, (iv) the results of investigation of the different optimisation approaches, and (v) the computational expense of the PINN models .

4.3.1 PINN model adjustments

The PINN methodology was first applied to the heat exchanger network. During the initial stages of this process, the PINN model was trained from its randomly initialised state using the loss function derived by implementing the balance equations. Although this approach did result in convergence on some occasions, the PINN model failed to converge in most cases, as the predictions made during the training process were unrealistic. For example, some predictions included negative values for both pressure and enthalpy. The inconsistent performance of the PINN model was attributed to the fact that the random initialisation of the network parameters resulted in the model sometimes searching for a solution outside of the physically realistic domain.

To avoid this, the trainable network parameters were preconditioned using a supervised pre-training step. Desired output values for all the nodes and elements throughout the thermofluid system were set equal to prescribed boundary values obtained from the input parameters. For example, in the case of the heat exchanger network, the given values for the inlet mass flow rate and outlet pressure were swept through the network resulting in the target mass flow rate through each element and the target pressure at each node in the thermofluid network being set equal to these values. The PINN model was then trained to predict these boundary values using the MSE loss function outlined in Chapter 3.2 (pg. 15). Despite making use of a supervised learning approach, this process does not require the generation of a large training dataset as the target values are extracted from the given input boundary values. The PINN model was given a set of boundary conditions as inputs and was then trained to predict the same values as outputs. The PINN is thus pre-trained in a supervised mode to predict a trivial result in order to obtain realistic initial values for the trainable parameters and consequently obtain physically realistic predictions for the conserved parameters.

Following the supervised pre-training, the PINN model was refined using the PINN loss function derived by implementing the balance equations. As the PINN was already constrained to make physically realistic predictions, the size of the solution space in which the model searched for the optimal network parameters that provided accurate predictions for all parameters was significantly reduced. Using this two-step training approach, the PINN model consistently produced realistic results and the two-step training approach was therefore implemented for all PINN models developed in this study. A flow diagram of the PINN training strategy, including the supervised pre-training process, is shown in Figure 9.

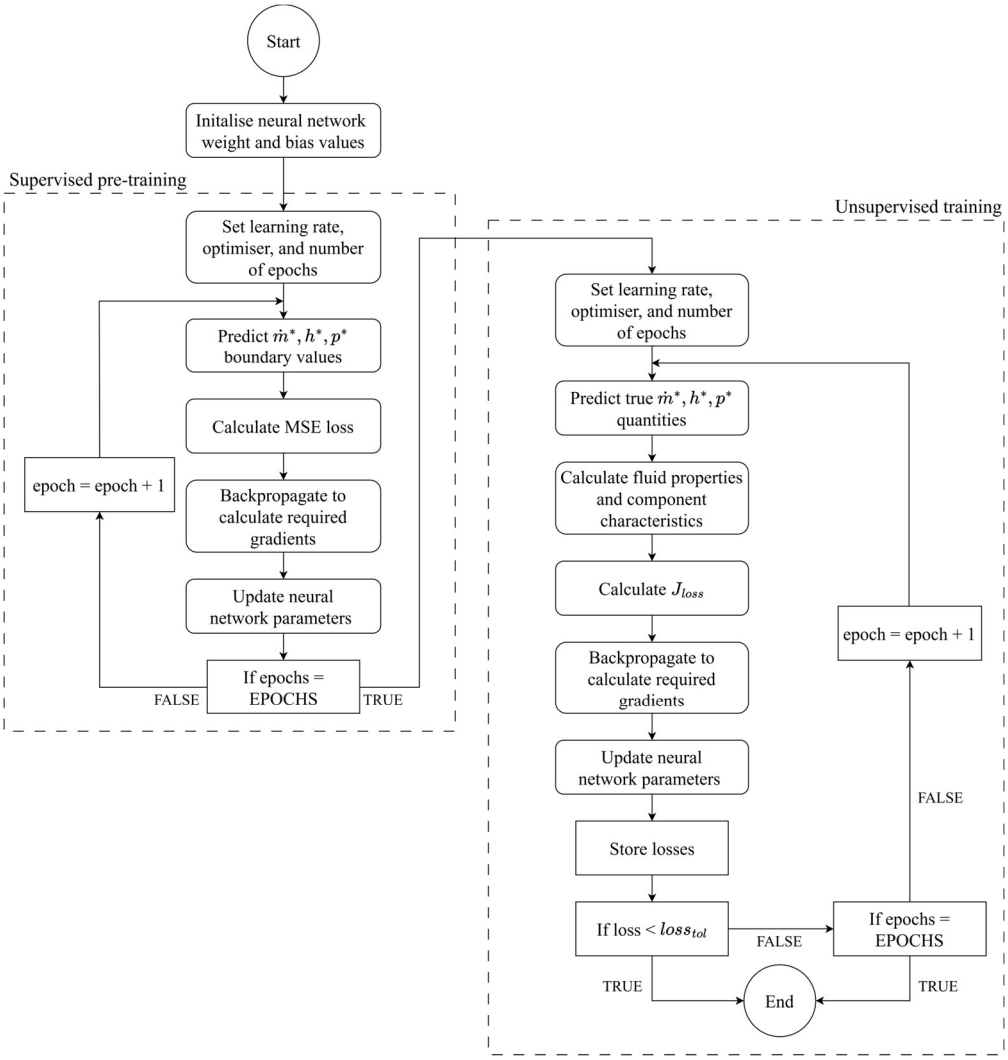


Figure 9. PINN training strategy.

Initially, the required fluid properties were calculated using the *CoolProp PropsSI* function outlined in Chapter 3.1 (pg. 13). However, the use of the *PropsSI* function significantly impacted the computational speed of the PINN training process, as the fluid property calculations alone accounted for almost 80% of the total PINN training time. In an attempt to reduce the computational expense of the fluid property calculations, *CoolProp*'s bicubic interpolation method [44] was applied as an alternative to the *PropsSI* function. The bicubic interpolation method requires the generation of pressure-enthalpy tables for each fluid property, and the required values at specified pressure-enthalpy combinations are then calculated using tabular interpolation. While the use of the bicubic interpolation method reduced the PINN training time by nearly half, the fluid property calculations still accounted for approximately 50% of the total PINN training time. Furthermore, *CoolProp*'s bicubic interpolation method is not capable of performing vectorised fluid property calculations. Consequently, each quantity (e.g., each nodal temperature) had to be calculated separately, adding to the computational expense of the fluid property calculations.

An alternative custom fluid property calculation method was therefore developed. This method was inspired by *CoolProp*'s bicubic interpolation method and used pressure-enthalpy and pressure-temperature tables that contained true reference values for the required fluid properties. The fluid property reference values were calculated using the *CoolProp* fluid property library. The ranges of temperature, pressure, and enthalpy values were determined according to the needs of the system. For example, in the case of the Brayton cycle, the carbon dioxide working fluid was always in the supercritical range, and the temperature, pressure, and enthalpy values — for which corresponding fluid property tables were generated — reflected this. The true fluid property relationships for sCO2 generated from the tabular data are shown in Figure 10.

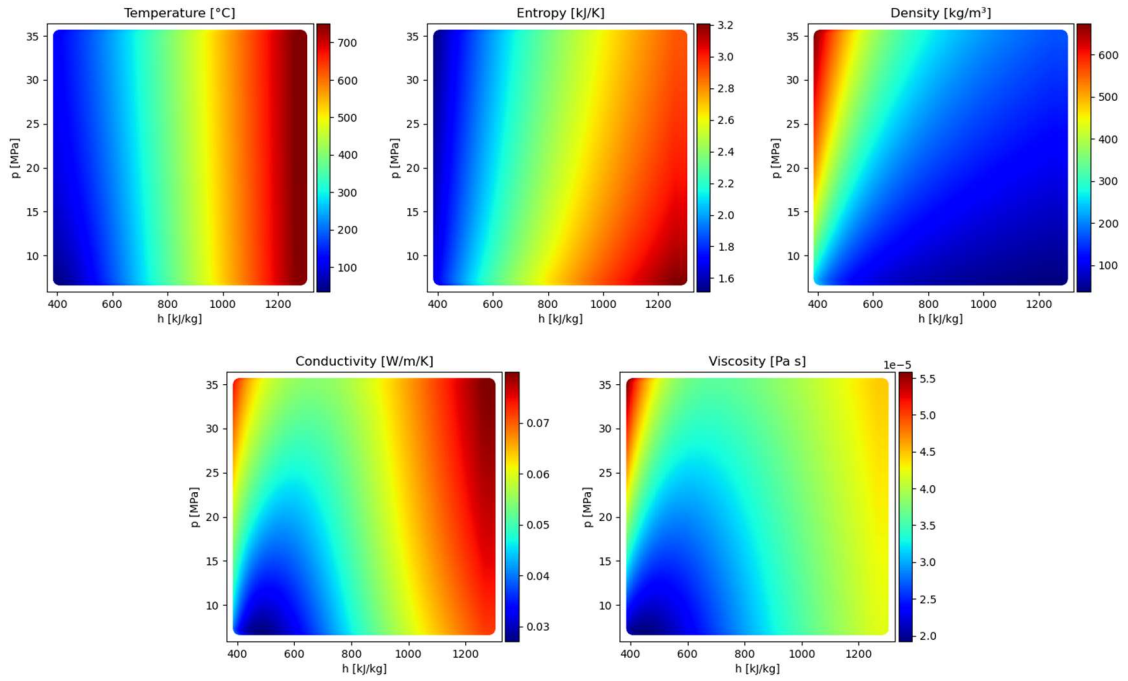


Figure 10. Fluid property relationships for sCO2 as functions of pressure and enthalpy.

Bivariate splines were then fitted to the tabular data to generate computationally efficient polynomials that could predict the fluid property values. This study applied the *RectBivariateSpline* function [45] from the *SciPy interpolate* library to generate fluid property polynomials of degree 3. These polynomials significantly increased the computational speed of the fluid property calculations. Additionally, this method is able to support vectorised calculations for the fluid properties. Another advantage of this method is that it can be used to predict fluid properties for cases in which the input pressure and temperature or pressure and enthalpy values are physically unrealistic, which might occur during the early stages of the PINN training. This is because the method returns the values at the boundary of the pressure-temperature tables (e.g., the property value at the minimum pressure), given unrealistic inputs such as negative pressures. In contrast, *CoolProp* cannot perform fluid property calculations given such inputs.

The entropy and thermal conductivity relationships for sCO2 are two particularly non-linear fluid property relationships with different profiles. These were therefore used to evaluate the accuracy of the approximations made by the fluid property polynomials. The true fluid properties were calculated using *CoolProp* and compared to the estimated fluid properties calculated using the bivariate spline polynomials. The actual fluid properties, the estimated fluid properties, and the absolute error for the sCO2 entropy and thermal conductivity are shown in Figure 11 and Figure 12, respectively. The absolute error magnitudes of 1E–11 and 1E–7 indicate that the fluid property polynomial approach can be used to make accurate estimates of the fluid property values. Interestingly, Figure 11 illustrates that the largest errors are found in the areas around the critical point of the fluid.

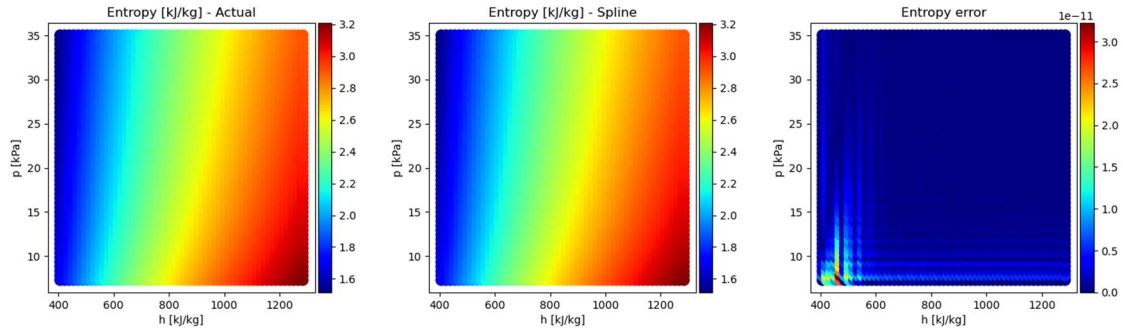


Figure 11. Comparison of entropy values for sCO2 generated via CoolProp and a bivariate spline.

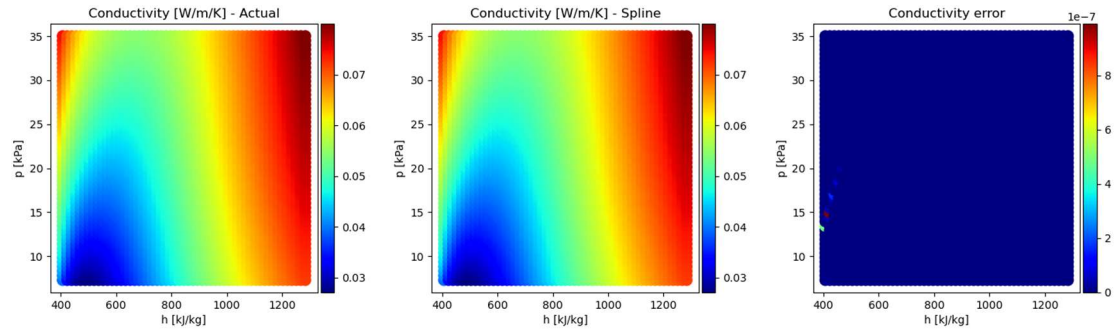


Figure 12. Comparison of thermal conductivity values for sCO2 generated via CoolProp and a bivariate spline.

Fluid property polynomials were generated for water over pressure-enthalpy ranges that include the 2-phase region to investigate the accuracy of this method for properties of fluids across the 2-phase region. The true fluid property relationships for water are shown in Figure 13. The temperature fluid property relationship contains several discontinuities. This relationship was therefore selected to evaluate the performance of the fluid property polynomials for water across the 2-phase region. The actual fluid properties, the estimated fluid properties, and the absolute error for the water temperature are shown in Figure 14. While the errors on the temperature estimations for water are larger than the errors on the estimations for the sCO2 properties, the magnitudes of the errors for the water properties are still acceptable. Furthermore, it is worth noting that the

largest errors on the estimations for the water fluid properties are found on the saturated liquid and saturated vapour lines.

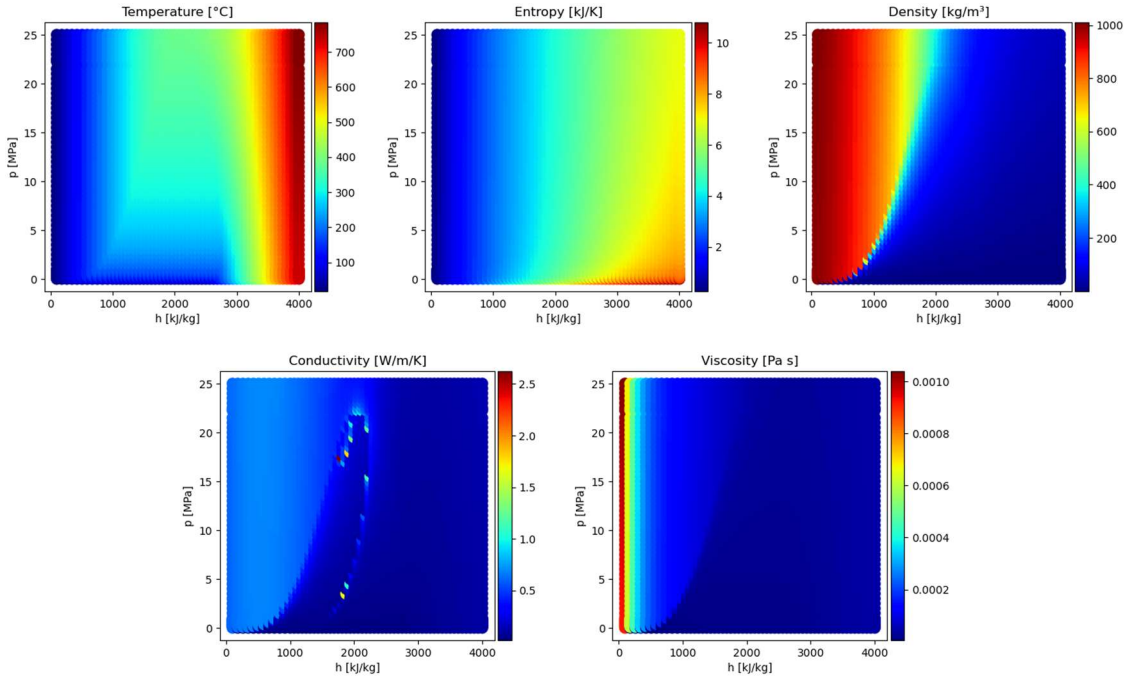


Figure 13. Fluid property relationships for water as functions of pressure and enthalpy.

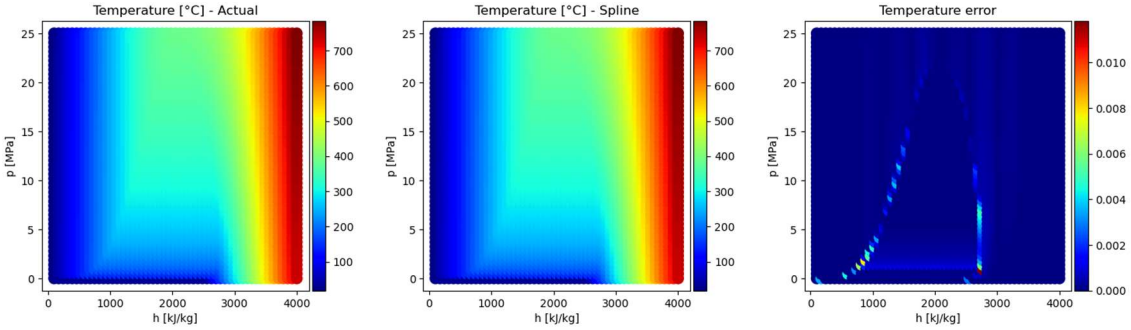


Figure 14. Comparison of temperature values for water generated via CoolProp and a bivariate spline.

From the results for supercritical CO2 and water in the 2-phase region, it can be concluded that the proposed method of calculating fluid properties with interpolated bivariate spline polynomials can be applied to estimate the fluid property values for pure fluids accurately. Therefore, this method of calculating fluid properties was implemented for the PINN models of the heat exchanger network and the recuperated Brayton cycle. The conventional physics-based thermofluid process models were also adjusted to use this method to ensure consistency between the PINN predictions and the benchmark solutions.

4.3.2 Loss function validation

As outlined in Chapter 3.3 (pg. 21), the validation of the PINN loss function is vital when implementing a PINN modelling methodology. Sets of 10 different samples were passed through the loss functions of the PINNs for the heat exchanger network and the recuperated Brayton cycle. Values for the average combined residuals and overall losses, J_{loss}, were calculated across all the samples for each thermofluid system. The results of this process are summarised in Table 4 and Table 5. The results show that the average overall loss values are below 1E–09, indicating that the loss functions accurately represent the underlying physics of both systems. Therefore, these loss functions can confidently be implemented in the PINN models.

Table 4. Average combined residuals and overall loss for the loss function generated for heat exchanger network.

Residual	$J_{mass,CO2}$	$J_{energy,CO2}$	$J_{mom,CO2}$	$J_{mass,air}$	$J_{energy,air}$	$J_{mom,air}$	J_{loss}
Value	0.0E+00	1.3E–11	3.7E–10	1.1E–15	2.1E–15	1.3E–10	5.1E–10

Table 5. Average combined residuals and overall loss for the loss function generated for the recuperated Brayton cycle.

Residual	J_{mass}	J_{energy}	J_{mom}	J_{loss}
Value	0.0E+00	7.6E–09	1.1E–14	7.6E–09

4.3.3 Hyperparameter search results

The hyperparameter tuning processes for the PINN models of the heat exchanger network, sCO2 Brayton cycle, and simplified boiler system were performed using a single randomly selected sample from each of the datasets containing 10 unique samples. First, a coarse grid search was implemented to find the best-performing PINN architectures for the two thermofluid systems. The coarse grid search was used to investigate the effect of the number of hidden layers and the number of neurons per hidden layer on the total PINN model losses. The hyperparameter search was implemented on the models which made use of the supervised pre-training step and used only the Adam optimiser during training. For the architecture hyperparameter search process, the learning rates for the Adam optimiser were fixed at 1E–04 for the heat exchanger network and 5E–05 for the recuperated Brayton cycle. The PINN model for the heat exchanger network was trained for 500 iterations and the PINN model for the recuperated Brayton cycle was trained for 800 iterations. The tanh activation function was used as an initial starting point for both PINN models. The choice of activation function was later investigated and refined.

Models with 1, 2, and 3 hidden layers were tested for the heat exchanger network. Hidden layers with widths of 1, 8, 16, and 32 neurons were tested for each model depth. Similarly, models with 1, 2, and 3 hidden layers were tested for the recuperated Brayton cycle. However, hidden layers with widths of 1, 8, and 16 neurons

were tested for each of the model depths. The results of the architecture hyperparameter search for the heat exchanger network and the recuperated Brayton cycle are shown in Table 6 and Table 7, respectively.

Table 6. Total model losses for different PINN architectures for the heat exchanger network.

	Number of hidden layers		
Number of neurons per layer	**1**	**2**	**3**
1	1.7E–01	1.2E–01	1.1E–01
8	2.6E–02	1.6E–02	1.2E–02
16	1.3E–03	1.0E–03	1.8E–04
32	6.5E–05	1.5E–05	5.7E–06

Table 7. Total model losses for different PINN architectures for the recuperated Brayton cycle.

	Number of hidden layers		
Number of neurons per layer	**1**	**2**	**3**
1	1.8E–01	1.1E–01	8.7E–02
8	3.2E–02	5.1E–03	2.3E–03
16	4.5E–05	4.7E–06	1.0E–06

The results presented in Table 6 and Table 7 show that the total model loss can be reduced by increasing either the depth (i.e., the number of hidden layers) or the width (i.e., the number of neurons per hidden layer) of the PINN. However, the width of the PINN had a more prominent effect on the total model loss than the depth of the PINN.

In this study, the PINN architectures which resulted in the smallest total model loss values were selected. Therefore, the final PINN model for the heat exchanger network consists of 3 hidden layers with 32 neurons each and the PINN for the recuperated Brayton cycle consists of 3 hidden layers with 16 neurons per layer.

Once the architectures of the PINN models were selected, another hyperparameter search process was implemented to find the best-performing activation functions. This involved training the PINN models for 600 iterations with a learning rate of 1E–04 in the case of the heat exchanger network and 800 iterations with a learning rate of 5E–05 for the recuperated Brayton cycle. The results of this process are presented in Table 8 and Table 9. The tanh activation function resulted in the smallest final model loss values in both cases. Therefore, the tanh activation function was selected for the PINN models of both systems.

Table 8. Final model losses for the PINN model of the heat exchanger network using different activation functions.

Activation function	**Final model loss**
ReLU	9.4E–05
ELU	1.4E–06
Tanh	7.8E–07
Sigmoid	1.7E–04

Table 9. Final model losses for the PINN model of the recuperated Brayton cycle using different activation functions.

Activation function	Final model loss
ReLU	4.3E–04
ELU	4.5E–04
Tanh	4.1E–06
Sigmoid	1.4E–02

4.3.4 PINN performance: Accuracy

In this study, two PINN models with different optimisation approaches were developed for both the heat exchanger network and the recuperated Brayton cycle, as outlined in Chapter 4.2 (pg. 31). The solutions generated by the trained PINN models for the different samples were compared to benchmark solutions that were generated using the conventional, physics-based thermofluid process models of the same systems. These conventional physics-based process models are described in Chapter 4.1 (pg. 26). The results of this comparison for the heat exchanger network are shown in Table 10 and Table 11, and the results for the recuperated Brayton cycle are given in Table 12 and Table 13.

Table 10. Performance (absolute errors and relative error percentages) per parameter for the PINN model of the heat exchanger network that used only the Adam optimisation algorithm.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	0.0057	2.1528	19.3834	0.0144	2.3408	0.1151
	Min.	1.81E–04	4.66E–02	9.08E–02	4.83E–04	1.21E–01	2.93E–02
	Avg.	0.0031	1.2780	2.9944	0.0055	1.0755	0.0647
Relative error (%)	Max.	0.1289	0.3153	0.1000	0.2793	0.2808	0.1084
	Min.	3.37E–03	5.59E–03	3.95E–04	8.92E–03	1.54E–02	2.84E–02
	Avg.	0.0635	0.1562	0.0154	0.1088	0.1319	0.0593

Table 11. Performance (absolute errors and relative error percentages) per parameter for the PINN model of the heat exchanger network that used the hybrid Adam-TNC optimisation approach.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	0.0049	2.7090	12.3896	0.0159	2.2301	0.0750
	Min.	5.34E–04	8.60E–02	1.89E–01	1.29E–03	2.43E–01	2.50E–02
	Avg.	0.0028	1.2111	2.9183	0.0059	1.0588	0.0484
Relative error (%)	Max.	0.1232	0.4097	0.0639	0.3009	0.2675	0.0679
	Min.	1.26E–02	1.08E–02	9.41E–04	2.83E–02	2.94E–02	2.17E–02
	Avg.	0.0580	0.1504	0.0147	0.1142	0.1313	0.0445

Table 12. Performance (absolute errors and relative error percentages) per parameter for the PINN model of the recuperated Brayton cycle that used only the Adam optimisation algorithm.

		$\dot{m}$ (kg/s)	h (kJ/kg)	p (kPa)
Absolute error	Max.	7.046	22.389	171.208
	Min.	0.420	0.417	19.630
	Avg.	3.162	6.379	85.745
Relative error (%)	Max.	1.127	3.128	0.803
	Min.	0.0813	0.0609	0.1062
	Avg.	0.5353	0.9266	0.4545

Table 13. Performance (absolute errors and relative error percentages) per parameter for the PINN model of the recuperated Brayton cycle that used the hybrid Adam-TNC optimisation approach.

		$\dot{m}$ (kg/s)	h (kJ/kg)	p (kPa)
Absolute error	Max.	6.753	22.133	145.228
	Min.	1.013	0.448	19.887
	Avg.	2.920	6.155	80.600
Relative error (%)	Max.	1.080	3.092	0.767
	Min.	0.171	0.065	0.108
	Avg.	0.499	0.893	0.429

The results presented in Table 10 and Table 11 illustrate that both PINN models for the heat exchanger network can produce results for the mass flow rates, pressures, and enthalpies that are within 0.16% of the benchmark solutions. A comparison of the results generated using only the Adam optimiser and the hybrid Adam-TNC optimisation approach shows no significant difference in the accuracy of the two optimisation approaches.

Similarly, the results shown in Table 12 and Table 13 illustrate that both PINN models for the recuperated Brayton cycle could make predictions that are in close agreement with the benchmark solutions. The two PINN models for the recuperated Brayton cycle produced average relative errors of less than 0.93% for the mass flow rates, pressures, and enthalpies. Once again, there is no significant difference in accuracy between using only the Adam optimiser and the hybrid Adam-TNC optimisation approaches.

These results illustrate that the PINN modelling methodology can successfully be applied to model thermofluid systems to generate accurate predictions. Interestingly, the PINN models of the recuperated Brayton cycle produced predictions with larger relative errors than the PINN models of the heat exchanger network. This can likely be attributed to the additional non-linearities present in the recuperated Brayton cycle as a result of the inclusion of the turbomachine performance characteristics. The additional non-linearities increase the complexity of the relationships being represented by the PINN models.

4.3.5 PINN performance: Optimiser selection

Figure 15 and Figure 16 provide training histories for the unsupervised training processes of the four different PINN models.

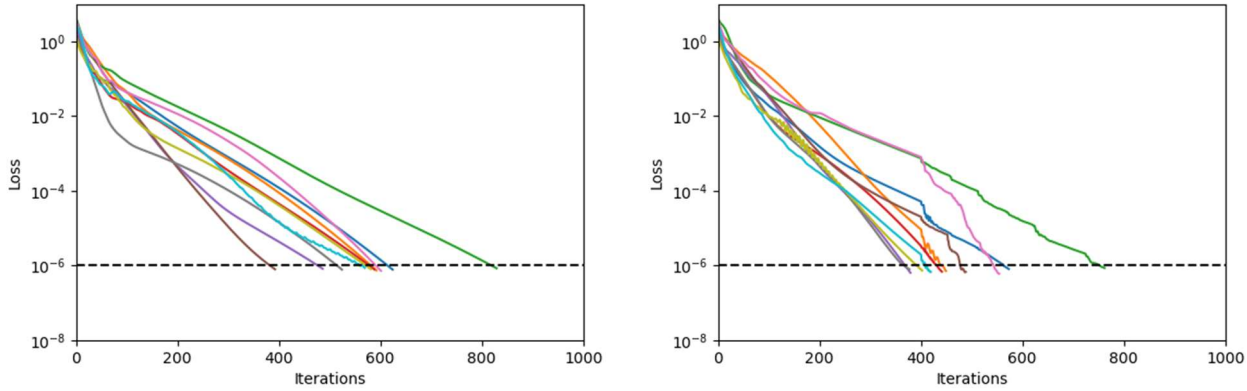


*Figure 15. Training histories for the unsupervised training of the PINN models for the heat exchanger network. **(a)** Training using only the Adam optimisation algorithm. **(b)** Training using the hybrid Adam-TNC approach.*

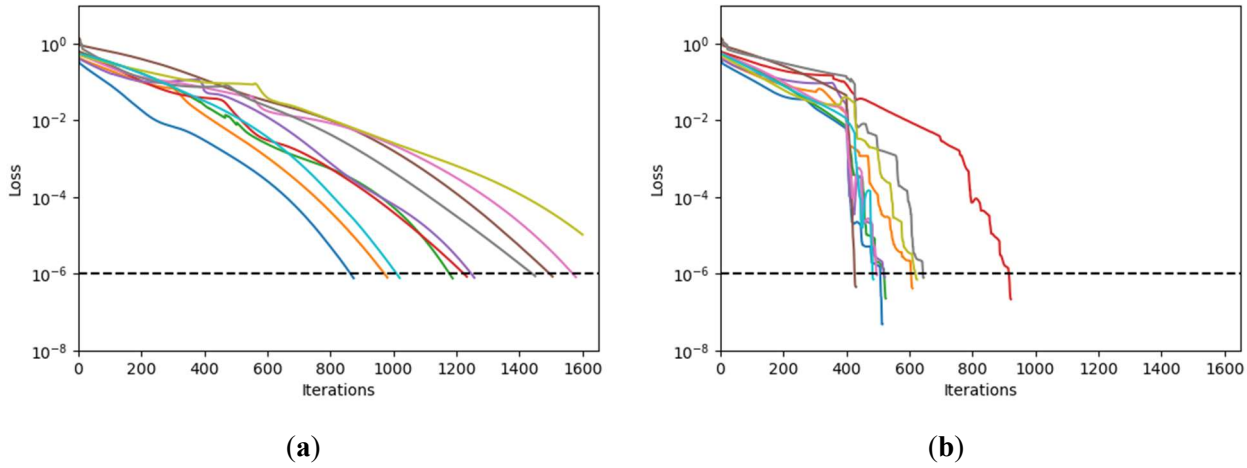


*Figure 16. Training histories for the unsupervised training of the PINN models for the recuperated Brayton cycle. **(a)** Training using only the Adam optimisation algorithm. **(b)** Training using the hybrid Adam-TNC approach.*

The training histories show that, on average, the PINN models that used the hybrid Adam-TNC optimisation approach converged to an acceptable total model loss in fewer iterations than those that used only the Adam optimiser. This is true for the heat exchanger network and the recuperated Brayton cycle. This finding is corroborated by the results shown in Table 14 which provides a summary of the number of training iterations each PINN model required to reach convergence. However, the effect of the hybrid Adam-TNC optimisation approach was significantly more prominent for the recuperated Brayton cycle. Using the second-order hybrid Adam-TNC optimisation approach reduced the average number of iterations by 95 in the case of the heat

exchanger network and by 690 in the case of the Brayton cycle. This demonstrates that the use of the hybrid Adam-TNC optimisation approach can offer a reduction in the computational expense of training PINNs.

Furthermore, the PINN model for the Brayton cycle that used the hybrid Adam-TNC optimisation approach reached the required tolerance for all samples within the maximum number of iterations, while the model that only used the Adam optimiser was unable to do so for one of the samples. Therefore, the hybrid Adam-TNC optimisation approach is the more desirable approach for training PINN models for integrated thermofluid systems.

Table 14. Number of iterations required to train the various PINN models.

Thermofluid network	Optimisation approach	Max. iterations	Min. iterations	Avg. iterations
Heat exchanger network	Adam only	829	392	579
	Adam and TNC	762	377	484
Recuperated Brayton cycle	Adam only	1600	874	1268
	Adam and TNC	922	431	578

4.3.6 PINN performance: Computational requirements

Once trained, the PINN models could make predictions for the two thermofluid systems in a fraction of the time taken by the conventional process models, as is shown in Table 15. On average, the PINN model of the heat exchanger network made predictions 120 times faster than the conventional process model for the same system. The PINN model of the Brayton cycle made predictions 88 times faster than the conventional process model for the same system.

Table 15. Time taken to generate solutions via the conventional process models and PINN models.

Thermofluid network	Model type	Max. time (s)	Min. time (s)	Avg. time (s)
Heat exchanger network	Conventional process model	0.5369	0.2890	0.3976
	Trained PINN	0.0040	0.0030	0.0033
Recuperated Brayton cycle	Conventional process model	0.5690	0.1396	0.2904
	Trained PINN	0.0092	0.0010	0.0033

Furthermore, the time taken for a trained PINN model to make a prediction is in the order of 1×10^{-3} seconds. This is a significant improvement over the conventional process models which required order 1×10^{-1} seconds to generate a solution. If these modelling approaches are to be applied to a more complex thermofluid network, the time taken for the conventional model to generate a solution will scale non-linearly and it would likely

require approximately 1×10^{1} seconds to generate a solution. By contrast, the inference speed of PINN models remains constant, regardless of the complexity of the network, provided that the GPU RAM is not exceeded [46]. This means that for a problem that requires 10 model calls it would take 1×10^{2} seconds to generate a solution using a conventional process model, but only 1×10^{-2} seconds using a PINN model.

While the presented results do not include the time taken to train the PINN models, the significant difference in inference speed suggests that the PINN modelling methodology will offer significant reductions in computational expense when the problem is scaled up to include large numbers of samples for surrogate modelling. This highlights the potential for PINN surrogate models as a valuable engineering tool in component and system design and optimisation, as well as in real-time simulation for anomaly detection, diagnosis, and forecasting.

An outline of the application of the PINN modelling methodology to a simplified industrial-scale biomass-fired boiler is provided in the following chapter. The PINN modelling methodology was applied to the biomass-fired boiler to investigate the performance of the PINN methodology for a case study that requires the consideration of complex component characteristic relationships.

Chapter 5. PINNs applied to a thermofluid network with complex component characteristics

Following the successful application of the PINN modelling methodology to two thermofluid systems with complex network structures but simple component characteristic calculations, the PINN methodology was applied to an additional case study which includes more complex component characteristic relationships. A simplified model of an industrial-scale biomass-fired boiler was selected for this purpose. The simplified boiler system was selected as it requires the consideration of complex component characteristic calculations due to the inclusion of radiative heat transfer. Furthermore, this system requires the consideration of fluid properties for gas mixtures. This chapter provides a description of the simplified boiler system, as well as an overview of the PINN model setup for this case study. It also presents the results generated with these PINN models and provides a discussion of the PINN model performance.

5.1 Case study description

A simplified industrial-scale sugarcane bagasse-fired boiler was selected as the final case study to be modelled using the PINN methodology. This system was analysed from the outlet of the combustion chamber and consists of a furnace as well as a single radiative and convective superheater heat exchanger. The two fluid flow streams in this system are the flue gas flow stream and the steam (or water) flow stream. The flue gas comprises of the products of the combustion process, which include a hot gas mixture and fly ash particles. Heat is transferred from the hot flue gas to the water flow stream. The layout of the simplified boiler system is shown in Figure 17.

This case study leveraged an existing boiler process model module developed by Rousseau [47] during the construction of both the conventional physics-based model and the PINN model. The existing boiler process model can be found in Appendix C (pg. 87). This model only considered the mass and energy balance equations. For this reason, the PINN model of the simplified boiler system predicts only the enthalpies at each node and the mass flow rates through each of the elements for both the flue gas and water flow streams.

The PINN models were trained for two operating conditions corresponding to different boiler loads. The boiler loads were specified in terms of its Maximum Continuous Rating (MCR), which refers to the maximum continuous steam output per unit time that a boiler can deliver at a specified temperature and pressure. For this study, the boiler was modelled at its maximum load case of 100% MCR and a low load case of 62% MCR. The input features for the PINN models of the simplified boiler system, and their values for the two load cases, are given in Table 16.

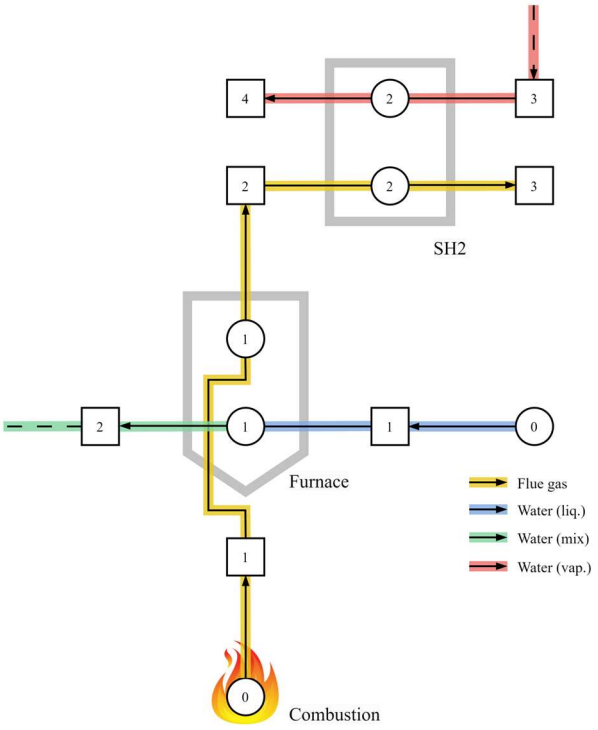


Figure 17. Thermofluid network model superimposed onto the process flow diagram for the simplified boiler system. Superheater - SH.

Table 16. Input features for the simplified boiler with their values for the two load cases.

Parameter	**Symbol**	**Unit**	**100% MCR**	**62% MCR**
Ambient air temperature	T_{amb}	K	298	305
Fuel mass flow rate on an ash-free basis	$\dot{m}_f$	kg/s	12.92	8.20
Adiabatic flame temperature	$T_{fg,aft}$	K	1628	1611
Flue gas mixture enthalpy at the adiabatic flame temperature	$h_{fg,aft}$	kJ/kg	2319	2309
Mass flow rate of the flue gas leaving the combustion zone	$\dot{m}_{fg,aft}$	kg/s	51.73	32.98
Fly ash mass flow rate	$\dot{m}_{fa}$	kg/s	0.55	0.35
Furnace pressure	$p_{fg,aft}$	kPa	101.325	101.325
Final steam mass flow rate	$\dot{m}_{ms}$	kg/s	28.8	18.8
Final steam pressure	p_{ms}	kPa	2773	3095
Desired final steam pressure	T_{ms}	K	675	675
Feedwater pressure	p_{fw}	kPa	3053	3053
Feedwater temperature	T_{fw}	K	493	481

In addition to the input features specified in Table 16, the PINN model requires the gas mixture compositions for the ambient air, combustion air, and flue gas for each load case. These gas mixture compositions were defined as fixed variables and were not passed into the PINN as input features. To obtain the composition of the combustion air, the composition of the flue gas, and the mass flow rate of the flue gas leaving the combustion chamber, a single pass of the combustion species mass balance calculation was performed for the simplified boiler system. This calculation ensures that the total mass of each element is conserved during the combustion reaction. The combustion species mass balance calculation requires a fuel composition to be specified, and the composition of the sugarcane bagasse fuel used for this case study is given in Table 17.

Table 17. Biomass fuel composition.

Constituent	**C**	**H**	**O**	**N**	**S**	**Ash**	**Moisture**
Units	wt %	wt %	wt %	wt %	wt %	wt %	wt %
Value	0.20787	0.02717	0.21069	0.00161	0.00020	0.04729	0.49904

A single pass of the adiabatic flame temperature calculation was performed to calculate the adiabatic flame temperature and the enthalpy of the flue gas mixture at the adiabatic flame temperature. The adiabatic flame temperature is the maximum theoretical temperature of the flue gas at the exit of the combustion zone. The combustion species mass balance calculation and the adiabatic flame temperature calculation were performed by implementing functions from the existing boiler process module developed by Rousseau [47]. The detailed procedure for both calculations is described by Rousseau et al. [48].

The generic forms of the mass and energy residual loss functions, given by Equations (17) and (18) (pg. 22) were applied to each component in the water flow path, and across the furnace in the flue gas flow path. However, the mass and energy balance equations for the flue gas stream across the superheater were adjusted slightly to account for the fly ash particles that are suspended within the flue gas mixture as well as the effect of air that may leak into the gas ducts from the surroundings, known as ingress air. Additionally, the energy balance equation was adjusted to account for the fact that no work is done by the fluid in the superheater. The term representing the rate of work done by the fluid, $\dot{W}$, was therefore set to zero. The resulting mass and energy balance equations for the flue gas stream in the superheater are given as follows:

$$\dot{m}_{fg,out} - \dot{m}_{fg,in} = \dot{m}_{ia,HX} \tag{33}$$

$$\dot{m}_{fg,out} h_{fg,out} - \dot{m}_{fg,in} h_{fg,in} = \dot{m}_{ia,HX} h_{ia} + \dot{Q}_{fg,HX} - 0 - \dot{m}_{fa} c_{p,ash} (T_{fg,out} - T_{fg,in}) \tag{34}$$

In Equations (33) and (34), $\dot{m}_{fg,in}$ and $\dot{m}_{fg,out}$ refer to the mass flow rates of the flue gas mixtures (excluding the fly ash) into and out of the heat exchanger, respectively, and $\dot{m}_{ia,HX}$ refers to a specified mass flow rate of ingress air into the heat exchanger. In Equation (34), $h_{fg,in}$ and $h_{fg,out}$ refer to the enthalpy of the flue gas

mixture at the inlet and outlet of the heat exchanger, respectively, h_{ia} refers to the enthalpy of the ingress air, $\dot{Q}_{fg,HX}$ refers to the rate of heat transfer to the flue gas mixture, $\dot{m}_{fa}$ refers to the specified fly ash mass flow rate, $c_{p,ash}$ is the specific heat capacity of the ash, and $T_{fg,in}$ and $T_{fg,out}$ refer to the temperature of the flue gas at the inlet and the outlet of the heat exchanger, respectively.

The mass and energy balance equations presented in Equations (33) and (34) were non-dimensionalised and rearranged to produce the following residual loss functions for analysing the flue gas flow stream across the superheater:

$$f_{mass,SH2} = \dot{m}^*_{fg,out} - \dot{m}^*_{fg,in} - \dot{m}^*_{ia,HX} \tag{35}$$

$$f_{energy,SH2} = \dot{m}^*_{fg,out} h^*_{fg,out} - \dot{m}^*_{fg,in} h^*_{fg,in} - \dot{m}^*_{ia,HX} h^*_{ia} - \frac{\dot{Q}_{fg,HX}}{\dot{m}_\infty h_\infty} + \frac{\dot{m}_{fa} c_{p,ash} (T_{fg,out} - T_{fg,in})}{\dot{m}_\infty h_\infty} \tag{36}$$

The required heat transfer component characteristics for the furnace and superheater were calculated using functions from the existing boiler process model developed by Rousseau [47]. The furnace was modelled using the projected method, also known as the Gurvich method. This semi-empirical method assumes that the heat transfer in the furnace is dominated by radiation and that the wall temperature of the surface surrounding the furnace is low enough (compared to the flue gas temperature) that any thermal radiation emissions from the wall surface can be neglected. The Gurvich method applies the following empirical relationship to describe the ratio of the flue gas temperature at the exit of the furnace $T_{fg,fet}$ to the adiabatic flame temperature $T_{fg,aft}$ [48]:

$$\frac{T_{fg,fet}}{T_{fg,aft}} = \left(M \left(\frac{\varepsilon_f}{Bo} \right)^{0.6} + 1 \right)^{-1} \tag{37}$$

where M is the flame centre modification factor, ε_f is the furnace effective emissivity, and Bo is the Boltzmann number. The Boltzmann number is defined as:

$$Bo = \frac{\varphi \dot{m}_f \overline{vc}}{\sigma \psi A_{rad} T^3_{fg,aft}} = \frac{T_g}{T_{fg,aft} - T_{fg,fet}} \tag{38}$$

In Equation (38), φ is the heat preservation coefficient, $\dot{m}_f$ is the mass flow rate of the fuel, $\overline{vc}$ is the mean overall heat capacity based on the fuel flow rate, σ represents the Stefan-Boltzmann constant, A_{rad} is the total surrounding projected area, and ψ an area-weighted furnace efficiency factor.

The mean overall heat capacity $\overline{vc}$ is calculated as follows:

$$\overline{vc} = \frac{\dot{m}_{fg,aft}(h_{fg,aft} - h_{fg,fet}) + \dot{m}_{fa} c_{p,ash}(T_{fg,aft} - T_{fg,fet})}{\dot{m}_f (T_{fg,aft} - T_{fg,fet})} \tag{39}$$

The flame centre modification factor M used in Equation (37) is derived from the following empirical relationship:

$$M = A - B(X_r + \Delta X) \tag{40}$$

where X_r is the normalised burner height, A and B are empirical constants that depend on the type of fuel being used, and ΔX is a correction factor for the actual flame position, which depends on the burner type and tilt.

Using the values for $T_{fg,aft}$ and $h_{fg,aft}$ obtained from the adiabatic flame temperature calculation, Equations (37) - (40) can be solved simultaneously to obtain the values required to calculate the total rate of heat transfer from the flue gas mixture $\dot{Q}_{fg,FUR}$ and the total rate of radiant heat projected from the gas volume to the surrounding surfaces $\dot{Q}_{rad}$.

The superheater was analysed using a radiative-convective heat exchanger model. The complexities introduced by the radiative-convective heat exchanger are illustrated in Figure 18, which provides a schematic of the relevant heat transfer phenomena and fluid flow paths for a generic radiative-convective heat exchanger model.

In Figure 18, $\dot{Q}_{conv}$ and $\dot{Q}_{gas}$ refer to the combination of radiative and convective heat transfer to the heat exchanger surfaces from the hot flue gas directly surrounding them. The radiative heat transfer being considered here is gas radiation from the fly ash particles suspended in the flue gas. $\dot{Q}_{wall}$ and $\dot{Q}_{roof}$ refer to the combination of radiative and convective heat transfer to the walls and roof surrounding the cavity in which the heat exchanger is located. In addition to the heat extracted from the flue gas, several radiative heat transfer mechanisms must be considered. $\dot{Q}_{direct}$ refers to the direct radiation from the flame in the furnace. As indicated by $\dot{Q}_{bypass}$, a portion of this direct radiation bypasses the current heat exchanger. $\dot{Q}_{back,in}$ refers to the direct radiation from the heat exchanger that is upstream in the flue gas flow path to the current heat exchanger, while $\dot{Q}_{back,out}$ refers to the direct radiation from the current heat exchanger to the heat exchanger that is downstream in the flue gas flow path.

Rousseau et al. [48] describe the detailed calculation procedures for both the projected method furnace model and the radiative-convective heat exchanger model.

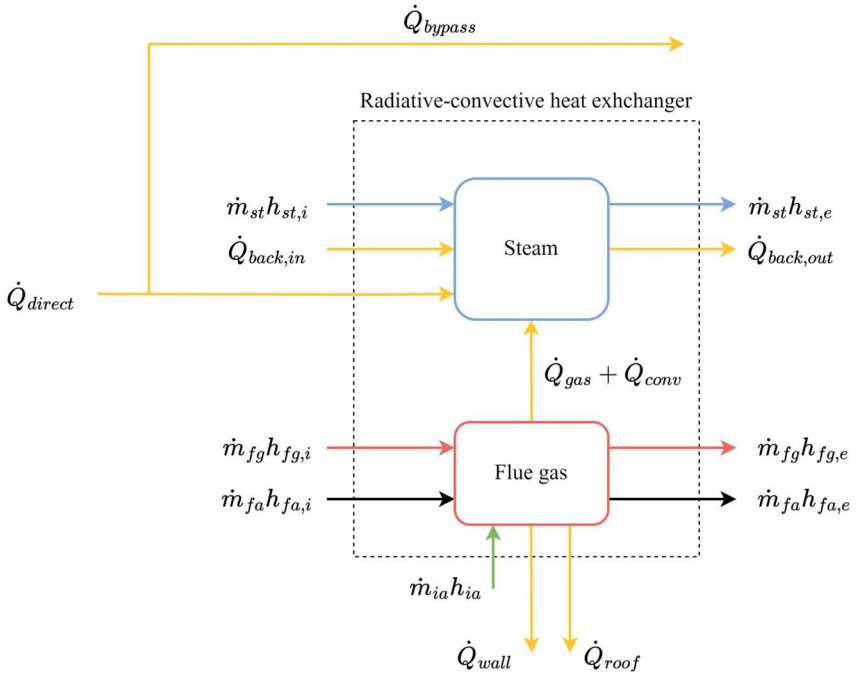


Figure 18. Heat transfer phenomena and fluid flow paths in a generic radiative-convective heat exchanger.

This case study requires the calculation of gas mixture fluid properties due to the flue gas, as well as for the ambient air and the combustion air. The bivariate spline approach for calculating fluid properties, outlined in Chapter 4.3.1 (pg. 33), was therefore extended to include gas mixtures, in addition to pure fluids. To achieve this, the reference properties to which bivariate spline functions were fitted were calculated using an existing gas mixture property module developed by Laubscher and Rousseau [49], instead of *CoolProp*. The gas mixture property module can be found in Appendix B (pg. 84).

Fluid property polynomials were generated for a specified gas mixture to evaluate the performance of this fluid property calculation approach for gas mixtures. The composition of the selected gas mixture corresponds to the composition of the flue gas for the 100% MCR sample and is given in Table 18.

Table 18. Specified gas mixture composition used to evaluate the fluid property polynomials for gas mixtures.

Constituent	**H2O**	**CO2**	**N2**	**SO2**	**O2**
Units	wt %	wt %	wt %	wt %	wt %
Value	0. 1996	0.1902	0.5751	0.0001	0.0349

The true fluid properties were calculated as functions of pressure and temperature, rather than pressure and enthalpy. The true fluid property relationships for the specified gas mixture are shown in Figure 19. The fluid properties for the gas mixture present highly linear relationships as the existing gas mixture property module

uses ideal gas relationships to calculate the gas mixture property values. The density relationship was used to evaluate the accuracy of the polynomial predictions, and the true density values, estimated density values, and the error between the two are shown in Figure 20. The estimated values match the true values well, particularly for higher temperatures. Therefore, the method of calculating fluid properties with interpolated bivariate spline polynomials was successfully extended to estimate the fluid property values for gas mixtures, in addition to pure fluids. This method was therefore implemented in the PINN model of the simplified boiler system.

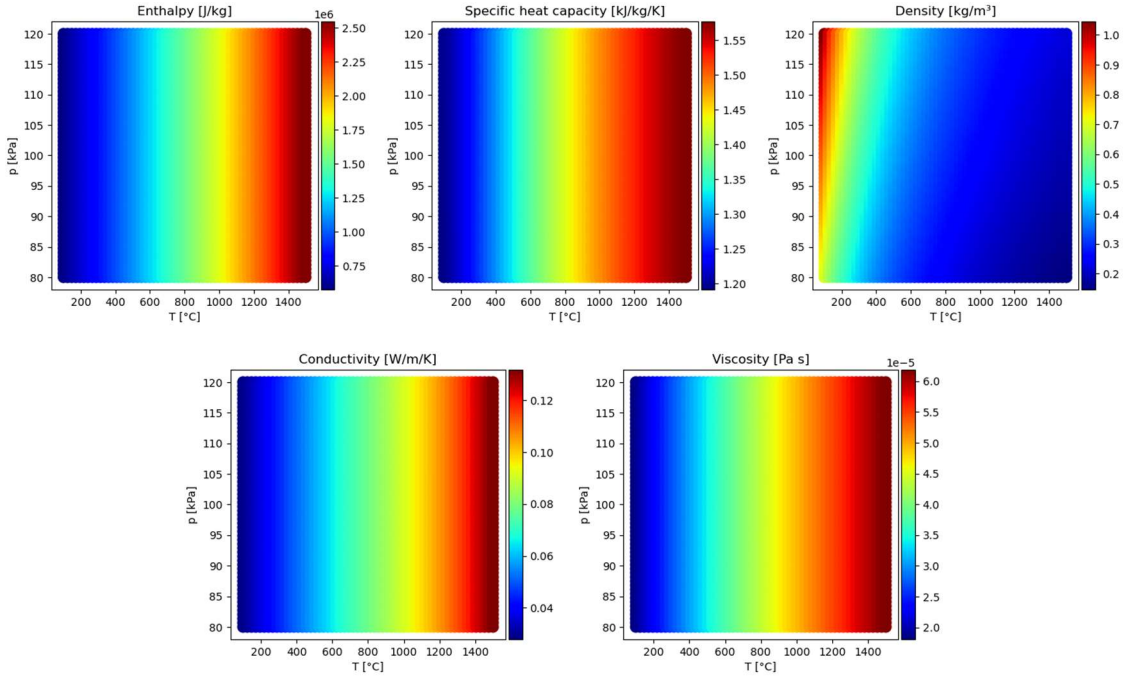


Figure 19. Fluid property relationships for a specified gas mixture as functions of pressure and temperature.

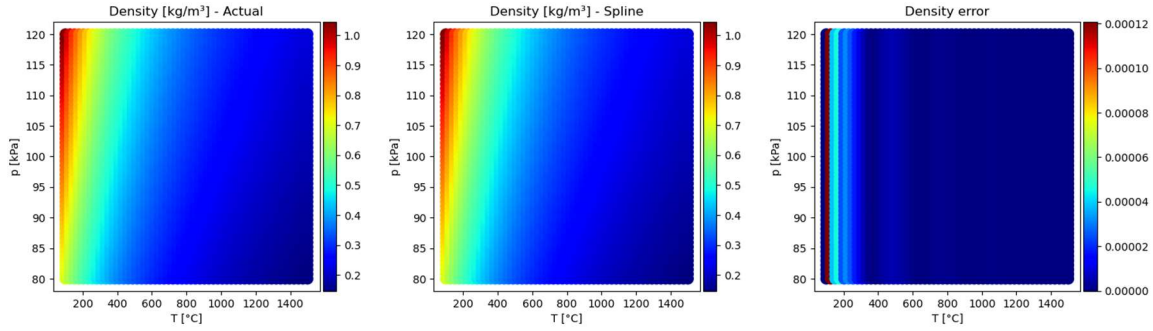


Figure 20. Comparison of density values for a specified gas mixture generated via an existing gas properties module and a bivariate spline.

A conventional, physics-based model of the simplified boiler system was developed in *Python 3.10.13* using the direct matrix inversion technique described in Chapter 3.1 (pg. 13). This model was used to generate the benchmark solution against which the PINN model predictions were compared. While this model used the

existing boiler process model functions developed by Rousseau [47] to calculate the heat transfer component characteristics, these functions were adjusted to use the interpolated gas mixture property polynomials.

The PINN model of the simplified boiler system was developed in *Python 3.10.13* using the *TensorFlow 2.9.3* framework.

5.2 Results and discussion

The results are presented and discussed in three sections, namely, (i) the results of the loss function validation procedure, (ii) the results of the hyperparameter search process, and (iii) the performance of the PINN model.

5.2.1 Loss function validation

To validate the PINN loss function for the simplified boiler system, the benchmark solutions for the 100% MCR and 62% MCR cases were passed through the loss function and the combined residuals and overall loss values were calculated. These values were averaged across the two samples and the results are given in Table 19. With an average overall loss value of only 2.4E–13, the loss function was shown to accurately represent the physical processes present in the simplified boiler system.

Table 19. Average combined residuals and overall PINN loss for the simplified boiler.

Residual	$J_{mass,st}$	$J_{energy,st}$	$J_{mass,fg}$	$J_{energy,fg}$	J_{loss}
Value	1.1E–16	1.9E–13	0.0E+00	4.8E–14	2.4E–13

5.2.2 Hyperparameter search results

The hyperparameter search process for the PINN model of the simplified boiler system was conducted using the data for the 100% MCR load case. Like with the two previous PINN models, the sample used for the hyperparameter search process was randomly selected. As with all other neural network models developed in this study, a coarse grid search was implemented to identify the best-performing network architecture for the PINN model of the simplified boiler system. For this process, the learning rate was fixed at 1E–04, and the model was trained for 400 iterations using only the Adam optimisation algorithm. The tanh activation function was used as an initial starting point for the PINN model. The choice of activation function was later refined. Network depths of 1, 2, and 3 hidden layers were tested and network widths of 8, 16, and 32 neurons were tested for each network depth. The results of the architecture hyperparameter search are shown in Table 20. The network architecture consisting of 3 hidden layers containing 32 neurons each produced the smallest total model loss and was therefore selected for the PINN model of the simplified boiler system.

Table 20. Total model losses for different PINN architectures for the simplified boiler system.

	Number of hidden layers		
Number of neurons per layer	**1**	**2**	**3**
8	5.1E–02	4.4E–02	4.1E–02
16	2.0E–04	2.9E–05	1.9E–05
32	5.2E–09	2.3E–12	8.1E–13

Once the PINN model architecture was selected, another hyperparameter search process was implemented to identify the best-performing activation function. The model was again trained using a learning rate of 1E–04; however, the model was only trained for 200 iterations during this process. The results of this hyperparameter tuning process are shown in Table 21. The exponential linear unit (ELU) activation function produced the smallest final model loss and was therefore selected for the PINN model of the simplified boiler system.

Table 21. Final model losses for the PINN model of the simplified boiler system using different activation functions.

Activation function	**Final model loss**
ReLU	1.3E–03
ELU	1.3E–09
Tanh	7.1E–08
Sigmoid	1.4E–02

5.2.3 PINN model performance

The PINN model for the simplified boiler system was trained using the hybrid Adam-TNC optimisation approach outlined in Chapter 4.2 (pg. 31). The model was first trained with the Adam optimiser for 80 iterations before the TNC optimiser was implemented to train the model to its loss tolerance of 1E–06.

The solutions generated by the trained PINN model of the simplified boiler system for both the 100% MCR and 62% MCR cases were compared to benchmark solutions that were generated using the conventional, physics-based thermofluid process model of the same system. The results of this comparison are presented in Table 22. These results show that the PINN model of the simplified boiler system can produce results for the mass flow rates, and enthalpies that are all within 0.095% of the benchmark solutions. This demonstrates that the PINN methodology can accurately capture complex and highly non-linear thermofluid component characteristic relationships, such as the relationships presented by radiative heat transfer. Therefore, the PINN methodology can be applied successfully as a numerical solver for the balance equations applied to integrated thermofluid systems which contain complex component characteristic relationships.

Table 22. Performance (absolute errors and relative error percentages) per parameter for the PINN model of the simplified boiler.

	Load case	$\dot{m}_{steam}$ (kg/s)	h_{steam} (kJ/kg)	$\dot{m}_{flue\ gas}$ (kg/s)	$h_{flue\ gas}$ (kJ/kg)
Absolute error	100% MCR	0.0022	0.2424	0.0009	0.3547
	62% MCR	0.0046	0.9540	0.0036	1.4414
Relative error (%)	100% MCR	0.0099	0.0084	0.0017	0.0215
	62% MCR	0.0298	0.0331	0.0109	0.0950

The training histories for the unsupervised training of the PINN models for both the 100% MCR and 62% MCR load cases are shown in Figure 21. The steep drop and rapid decrease of the model loss that occurs after 80 training iterations indicates the significant impact of the second-order TNC optimiser on increasing the computational speed of the training process. This aligns with the results presented in Chapter 4.3.5 (pg. 42), which highlight the reduction in computational expense offered by the hybrid Adam-RNC optimisation approach.

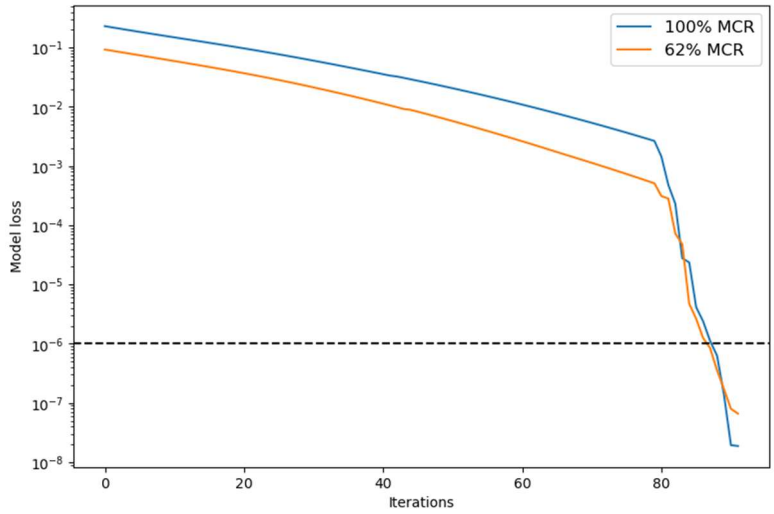


Figure 21. Training histories for the unsupervised training of the PINN model for the simplified boiler.

Once trained, the PINN model of the simplified boiler system could make predictions in a fraction of the time of the conventional process models, as is shown in Table 23. On average, the PINN model of the simplified boiler system made predictions 865 times faster than the conventional process model.

Table 23. Time taken to generate solutions via the conventional process model and PINN model of the simplified boiler system.

Model type	**Time for 100% MCR case (s)**	**Time for 62% MCR case (s)**	**Avg. time (s)**
Conventional process model	2.7452	2.6288	2.6870
Trained PINN	0.0030	0.0031	0.0031

As with the PINN models of the heat exchanger network and the Brayton cycle, the time taken for the trained PINN model of the simplified boiler to make a prediction is in the order of 1×10^{-3} seconds. This is significantly quicker than the conventional process model which requires order 1×10^{0} seconds to generate a solution. For a thermofluid network with significantly more complex component characteristic relationships, the inference time of the PINN modelling methodology remained approximately constant, while the calculation time of the conventional modelling approach increased non-linearly by a factor of 1×10^{1}. This further supports the suggestion that PINNs will offer significant reductions in computational expense when analysing more complex thermofluid systems, and when the analyses are scaled up to include large numbers of samples for surrogate modelling.

In the following chapter, the extension of the PINN modelling methodology to the development of surrogate models is investigated. The heat exchanger network was selected as the case study for this purpose.

Chapter 6. PINNs applied to surrogate modelling

Thus far, the PINN modelling methodology was only applied to generate numerical solvers that solved the mass, momentum, and energy balance equations for single samples individually. In this chapter, the extension of the PINN modelling methodology to the development of surrogate models is now investigated. The heat exchanger network was selected as the case study for this purpose. This chapter includes an outline of the development of a conventional MLP surrogate model, a discussion of the MLP model performance, and an overview of how the PINN model of the heat exchanger network described in Chapter 4 (pg. 26) was extended to consider multiple simulation cases simultaneously. Finally, an evaluation of the application of the PINN modelling methodology to develop surrogate models for integrated thermofluid systems is provided here.

6.1 Surrogate modelling basics

The application of the PINN modelling methodology to the development of numerical solvers for integrated thermofluid systems required that the PINN models be trained using single samples only. However, the application of the PINN modelling methodology to surrogate modelling requires that the PINN surrogate models be trained on multiple data samples simultaneously. This is because surrogate models mimic the input-output behaviour of complex systems by learning a set of non-linear transformations which approximately describe the underlying relationships, thus expanding the mapping range of the model to wider ranges of inputs.

The input features for the heat exchanger network, listed in Table 1 (pg. 27), were varied using the LHS method to generate the sets of unique samples required to train both the MLP and PINN surrogate models. The physically realistic ranges within which the input features for the heat exchanger network can be varied to simulate different operating conditions, shown in Table 1 (pg. 27), were divided into two subsets, namely, a training envelope and a testing envelope. The training envelope contained the lower 75% of the parameter values, while the testing envelope contained the remaining 25%. Datasets generated within the training envelope were used to train and validate the surrogate models. Subsets of the training datasets were reserved as hold-out test sets that were used to evaluate the interpolation performance of the trained surrogate models. The testing envelope, which contained the upper 25% of the parameter values, was used to generate datasets with samples that fall outside of the range of training data. These datasets were used to evaluate the extrapolation performance of the surrogate models. Once generated, the various sets of input parameters were passed through the conventional, physics-based model for the heat exchanger network to calculate the corresponding labelled target data for the MLP surrogate model, as well as the benchmark solutions used to evaluate the predictive performance of both the MLP and PINN surrogate models.

When developing a surrogate model, it is important to consider the sources of uncertainty that might affect the reliability of its predictions. Sources of uncertainty can be categorised into two groups, namely, aleatoric

uncertainty and epistemic uncertainty [50]. Aleatoric uncertainty arises from inherent randomness or variability in a system under study and cannot be reduced. Epistemic uncertainty arises from a lack of knowledge or understanding about a system under study, and it can be reduced by gathering more data, improving the model, or increasing the understanding of the system being modelled. Noise is a factor that can significantly affect the uncertainty surrounding a model's prediction. Therefore, the effect of noise must be carefully considered during a surrogate model's development.

Two types of noise are typically considered when developing a surrogate model using machine learning techniques, namely, stochastic and deterministic noise. Stochastic noise refers to any random noise that may be present in the data [51]. Stochastic noise is therefore a source of aleatoric uncertainty. Such noise could be caused by a variety of reasons such as hysteresis in the process being modelled, errors in the recorded measurements, random variations within the data sample, or even errors due to numerical approximations, to name a few. Deterministic noise, however, refers to any uncertainty in the model predictions due to overfitting during training [51]. Deterministic noise can occur when the model configuration is more complex than is necessary to fit the underlying relationship or when the training data doesn't represent the design space sufficiently. The effect of deterministic noise can be reduced by adjusting the model complexity or by using a dataset which more accurately represents the design space, thus improving the knowledge of the underlying system. Deterministic noise is therefore a source of epistemic uncertainty.

In the context of neural networks, both stochastic and deterministic noise cause a decrease in a surrogate model's performance when making predictions for unseen data [52]. However, for data-driven surrogate models, stochastic noise can act as a regulariser to prevent overfitting. This is because including small variations in the training data can be seen as a form of data augmentation [30]. Using a larger and more diverse training dataset can prevent a model from memorising specific details of the training data, and the model is thus encouraged to recognise general patterns or trends that are likely to be present in unseen data, thus improving its ability to generalise to new, unseen data.

In this study, the data used to train the MLP and PINN surrogate models were obtained via simulation using the conventional physics-based model of the heat exchanger network. The value of the convergence tolerance applied to the balance equations in the conventional, physics-based thermofluid model was selected to be 1E-12. This small value was selected to reduce the likelihood of numerical errors in the simulation introducing substantial noise into the data. The effect of stochastic noise was therefore considered negligible in this study. To prevent overfitting and thus reduce the effect of deterministic noise on the performance of the MLP surrogate model, cross-validation was implemented during the hyperparameter tuning process as described in Chapter 3.2 (pg. 15).

6.2 MLP surrogate model

The MLP surrogate model developed in this study used the MSE loss function described in Equation (13) (pg. 19). The best-performing network configuration was identified using a hyperparameter search process to make suitable selections for the network depth, network width, size of the training dataset, activation function, learning rate, number of epochs, and batch size. The final model was then trained, and its predictive performance was tested for both interpolation and extrapolation cases.

6.2.1 Hyperparameter search results

A coarse grid search was implemented to find the best-performing network architectures for the MLP surrogate model of the heat exchanger network. The architecture selection process was completed using a sufficiently large dataset containing 2000 unique samples, with a mini-batch size of 128. During this process, the MLP model was trained for 800 epochs with a learning rate of 1E–04. The ReLU activation function was used for the input and hidden layers. The results of the architecture selection process are shown in Table 24.

Table 24. Total validation losses for different MLP surrogate model architectures.

	Number of hidden layers		
Number of neurons per layer	**1**	**2**	**3**
8	6.2E–03	7.6E–03	7.8E–03
16	1.0E–03	6.5E–04	8.8E–04
32	1.2E–04	1.1E–04	1.1E–04
64	7.2E–05	9.7E–05	3.8E–05

As with the neural network models described in Chapter 4 (pg. 26) and Chapter 5 (pg. 45), increasing the number of neurons per hidden layer had a greater impact on reducing the model losses than increasing the number of hidden layers for the MLP surrogate model. The most complex network architecture, consisting of 3 hidden layers with 64 neurons each, was selected as this architecture produced the lowest total validation loss.

Once the network architecture had been selected, the effect of the size of the training dataset on the MLP surrogate model was investigated. During this process, differently sized datasets were used to train and test the MLP surrogate model. In this study, training datasets containing 5, 10, 200, 500, 1000, and 2000 unique samples were tested. Each dataset was divided into subsets of training data, validation data, and testing data, as described in Chapter 3.2 (pg. 15). The MLP surrogate model was trained for 800 epochs with a learning rate of 1E–04 for the differently sized datasets. Pure stochastic gradient descent was implemented (i.e., a batch size of 1) to mitigate the effect of batch size selection on the training process. As a result, the trainable network parameters were updated after each sample, regardless of the size of the training dataset. Once the training process for the MLP surrogate models was complete, the training dataset was used to calculate the in-sample

error. The unseen hold-out testing dataset was used to calculate the out-of-sample error. The in-sample and out-of-sample errors for the different datasets are shown in Figure 22, from which it can be seen that the difference between the in-sample and out-of-sample errors is constant for datasets containing 1000 samples or more. This indicates that the dataset used to train the MLP surrogate model must contain a minimum of 1000 number of samples to prevent overfitting.

A complete dataset containing 1430 unique samples was generated to train and validate the MLP surrogate model. This complete dataset was generated within the training envelope described in Chapter 6.1 (pg. 56). By including 1430 unique samples in the complete dataset, it was ensured that reserving 70% of the total number of samples for the training dataset resulted in the training dataset containing 1000 unique samples as required. The remaining samples were split between a validation dataset that contained 10% of the total number of samples and a hold-out test set that contained 20% of the total number of samples. The validation set was used during the remaining stages of the hyperparameter tuning process, while the hold-out test set was used to evaluate the interpolation performance of the trained MLP surrogate model.

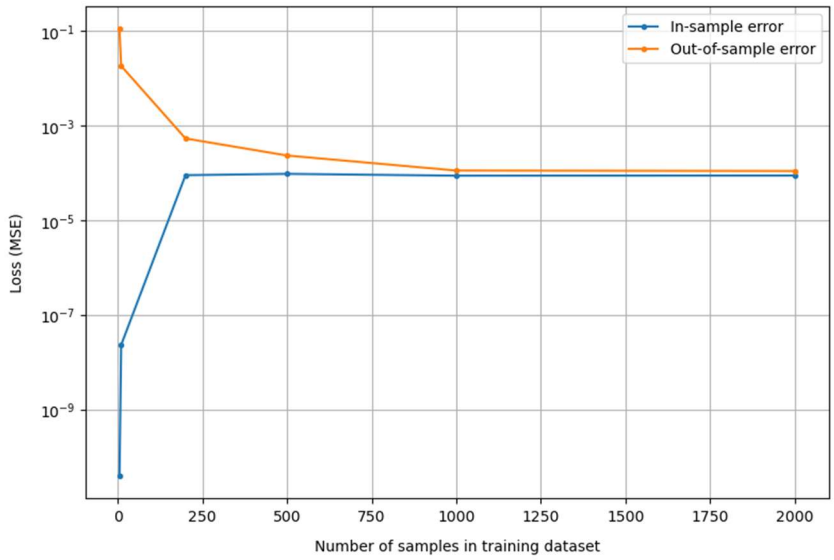


Figure 22. MLP surrogate model errors for differently sized training datasets.

Following the selection of the network architecture and the size of the training dataset, another hyperparameter search process was implemented to identify the best-performing activation function for the input layer and all hidden layers in the MLP surrogate model. For all activation functions shown in Table 25, the model was trained for 800 epochs with a learning rate of 1E–04. The final validation losses for each activation function were recorded and are shown in Table 25. The ReLU activation function produced the smallest final validation loss and was therefore selected for the MLP surrogate model of the heat exchanger network.

Table 25. MLP surrogate model validation errors for different activation functions.

Activation function	Final validation loss
ReLU	5.7E–05
ELU	8.0E–05
Tanh	1.4E–04
Sigmoid	9.5E–03
Swish	2.7E–04

Finally, the learning rate, number of epochs, and batch size were adjusted to find the optimal model configuration. The various combinations of hyperparameters that were tested are shown in Table 26.

Table 26. Various MLP surrogate model configurations with their respective hyperparameters.

Model	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**
Learning rate	1E–05	1E–05	1E–05	1E–04	1E–04	1E–04
Epochs	1500	1500	1500	1000	1000	1000
Batch size	32	64	128	32	64	128

The final training and validation losses for each of the six model configurations given in Table 26 are shown in Figure 23. From this figure, it can be seen that M4 produced the lowest training and validation errors. This configuration of hyperparameters was therefore selected for the MLP surrogate model. A summary of the final network architecture and hyperparameter configuration selected for the MLP surrogate model is given in Table 27. This model was then trained and tested to evaluate the performance of an MLP surrogate modelling approach for analysing integrated thermofluid systems.

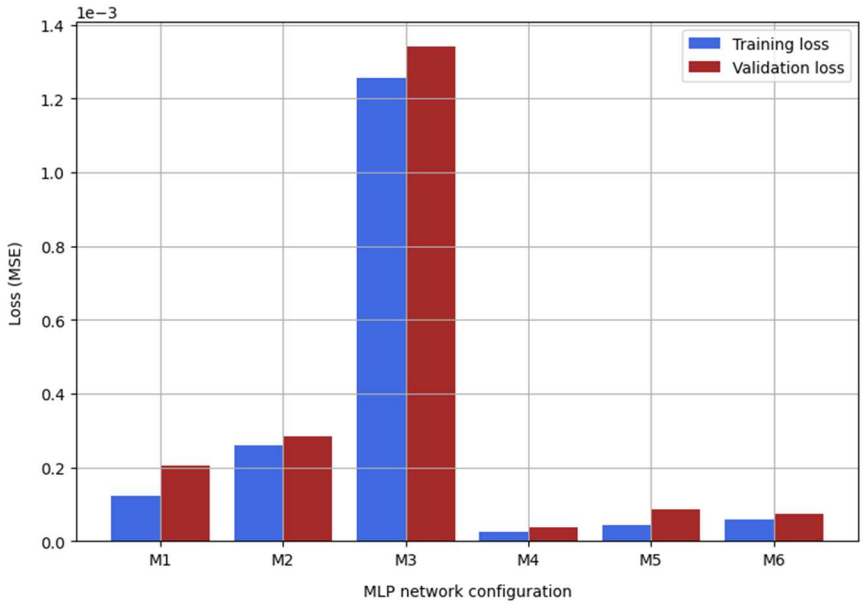


Figure 23. Validation and training losses for the six MLP surrogate model configurations.

Table 27. Final network architecture and hyperparameter configuration for the selected MLP surrogate model.

Hyperparameter	**M4**
Number of hidden layers	3
Number of neurons per hidden layer	64
Number of samples in the training dataset	1000
Activation function	ReLU
Learning rate	1E–04
Epochs	1000
Batch size	32

6.2.2 Model performance

The predictive performance of the trained MLP surrogate model was tested on samples within the training envelope (i.e., interpolation), and samples outside of the training envelope (i.e., extrapolation).

The ability of the trained MLP surrogate model to interpolate was tested by using the model to make predictions for an in-sample hold-out test set which had the same distribution as the training data and contained 286 samples (i.e., 20% of the training data). The extrapolation performance of the trained MLP surrogate model was tested using a dataset which also contained 286 samples. However, the distribution of these samples did not match the distribution of the training data. Instead, these samples were generated for the testing envelope which contained the upper 25% of the physically realistic range of input features, as described in Chapter 6.1 (pg. 56).

The predictions made by the MLP surrogate model were compared to benchmark solutions generated via the conventional, physics-based process model to assess their accuracy. The results of this comparison for the interpolation test set and the extrapolation test set are shown in Table 28 and Table 29, respectively. These tables present the average errors and relative error percentages for each set of conserved parameters.

Table 28. Interpolation performance (absolute errors and relative error percentages) per parameter for the MLP surrogate model of the heat exchanger network.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	0.1405	14.9138	107.9222	0.0492	8.7000	1.4139
	Min.	0.0001	0.2804	2.0986	0.0018	0.2398	0.0034
	Avg.	0.0038	1.6338	8.7927	0.0094	1.2170	0.2724
Relative error (%)	Max.	4.0356	2.2112	0.6719	1.2277	1.3053	1.3364
	Min.	0.0030	0.0365	0.0124	0.0387	0.0312	0.0034
	Avg.	0.0329	0.2174	0.0235	0.2052	0.1622	0.2686

The results given in Table 28 show that for samples that fall within the range of the training data, the largest average error between the MLP surrogate model predictions and the benchmark solutions was only 0.27%.

This demonstrates that the MLP surrogate model can interpolate for in-sample data and can be used to make predictions for the mass flow rates, pressures, and enthalpies with a high level of accuracy.

The accuracy of the MLP surrogate model for interpolation predictions is corroborated by Figure 24 which shows the target values, obtained from the benchmark solutions, plotted against the predictions made by the MLP surrogate model for the interpolation test set. In this figure, a single set of mass flow rate, enthalpy, and pressure values were considered for both the sCO2 and gas flow streams. The diagonal lines indicate how well the predictions matched the benchmark solutions. For the selected parameters, the majority of the predictions fall on the diagonal line, thereby indicating that they closely matched the benchmark solutions. There were some instances where there was a slight deviation of the predictions from the diagonal line, thus indicating a decrease in the performance of the MLP surrogate model for these samples. Notably, these samples occurred primarily near the upper and lower limits of the parameter values. This is expected, as less training data would have been generated for these regions, compared to the regions which correspond to the middle of value ranges, due to the normal distribution of training data. As a result, the model would have had the least amount of information about the behaviour of the heat exchanger network in these outer regions.

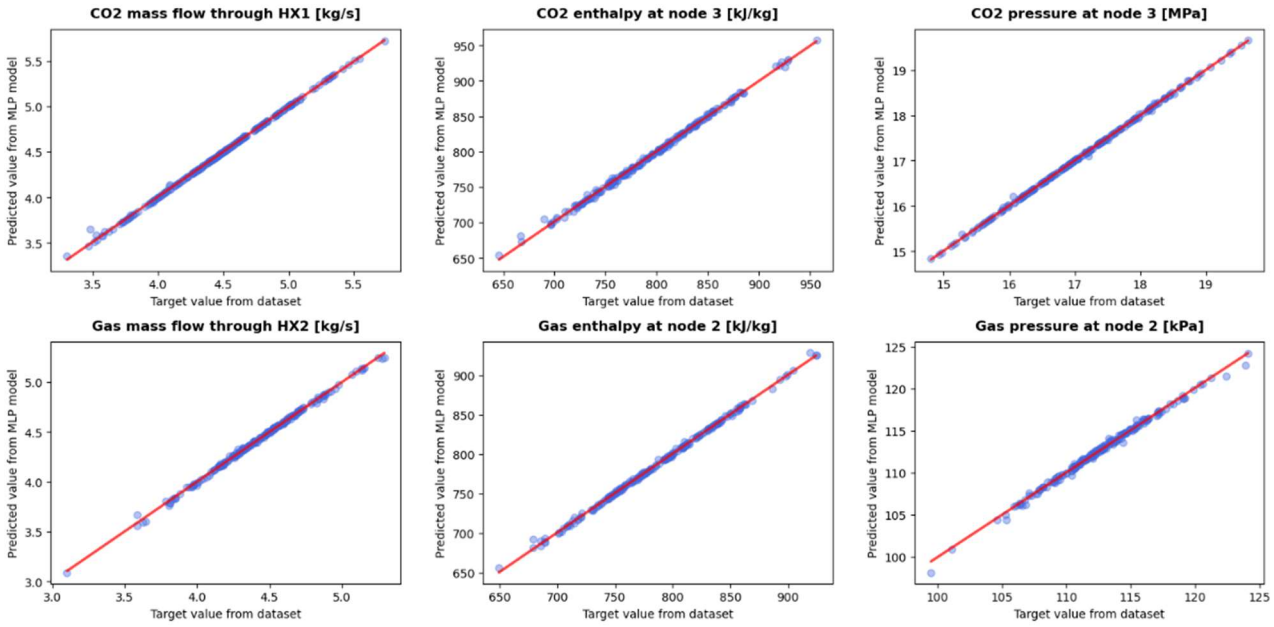


Figure 24. Target values from the training dataset plotted against predicted values obtained from the MLP surrogate model for the interpolation test set.

The results given in Table 29 show that for samples that fall outside of the range of the training data, the largest average error between the MLP surrogate model predictions and the benchmark solutions was 1.4%. This is approximately 5 times larger than the largest average error for the interpolation test cases. While these results indicate that the MLP surrogate model can still make relatively accurate predictions for cases that fall outside of the range of training data, the predictive performance of the model is reduced for extrapolation applications in comparison to interpolation applications.

Table 29. Extrapolation performance (absolute errors and relative error percentages) per parameter for the MLP surrogate model of the heat exchanger network.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	0.0812	11.1825	90.9284	0.0466	11.5041	3.6411
	Min.	0.0000	1.4090	0.3189	0.0092	0.4659	0.6722
	Avg.	0.0028	3.6069	6.1112	0.0225	4.4294	1.9641
Relative error (%)	Max.	0.9090	1.1310	0.3566	0.6429	1.2045	2.5858
	Min.	0.0004	0.1413	0.0012	0.1329	0.0481	0.4790
	Avg.	0.0947	0.3654	0.0520	0.3164	0.4622	1.3994

The results presented in Table 29 are corroborated by Figure 25 which shows the target values, obtained from the benchmark solutions, plotted against the predictions made by the MLP surrogate model for the extrapolation test set. In comparison to the plots shown in Figure 24, the predictions are significantly more scattered and deviate substantially from the diagonal line, indicating a greater discrepancy between the predictions and the target values.

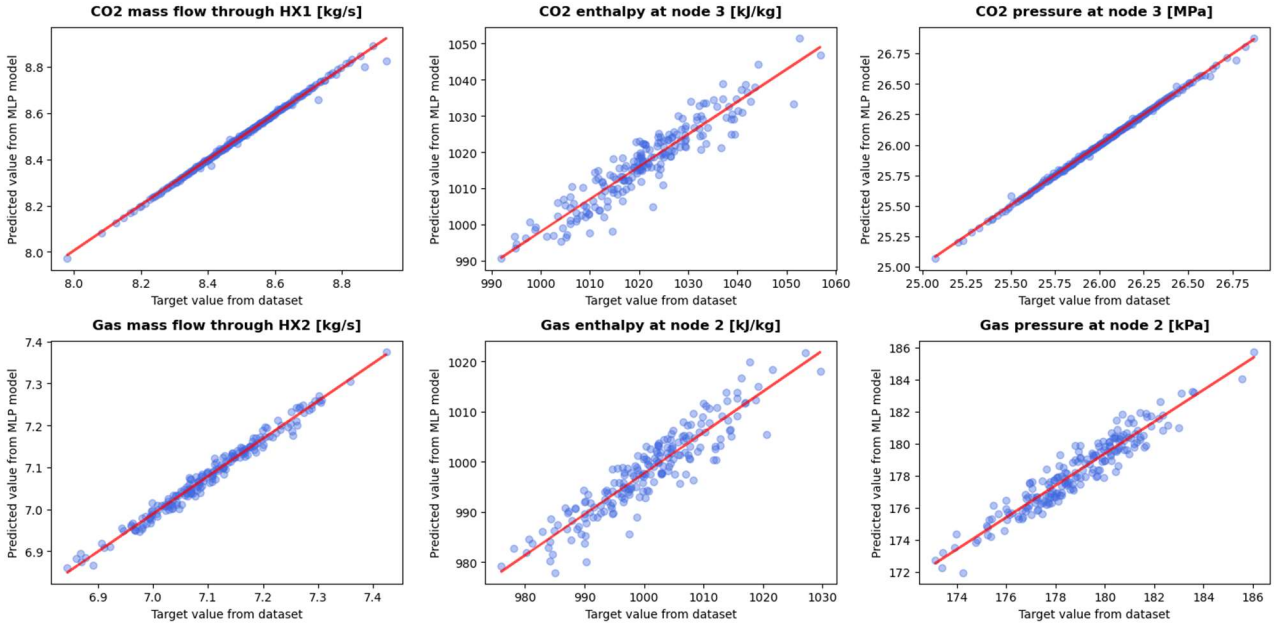


Figure 25. Target values from the training dataset plotted against predicted values obtained from the MLP surrogate model for the extrapolation test set.

Generating the trained MLP surrogate model involved creating the training dataset containing 1000 unique samples, as well as the training process itself. The time taken for each of these tasks is given in Table 30. The total time required to generate the full MLP surrogate model was 931.8 seconds, which equates to approximately 15.5 minutes.

Table 30. Time taken to generate the MLP surrogate model.

Time for data generation (s)	Model training time (s)	Total time to generate model (s)
550.1	381.7	931.8

The prediction times of the trained MLP surrogate model for both the interpolation and extrapolation test cases are given in Table 31. This table shows that the time taken for the trained MLP surrogate model to generate predictions is in the order of 1×10^{-3} seconds. This is quicker than the 1×10^{-1} seconds required to generate a solution with the conventional model of the same system, shown in Table 15 (pg. 43). This highlights the substantial computational advantage offered by ANNs.

Table 31. Time taken to generate predictions with the trained MLP surrogate model.

		Interpolation test cases	Extrapolation test cases
Time taken to generate prediction (s)	Max.	0.0048	0.0044
	Min.	0.0019	0.0019
	Avg.	0.0023	0.0021

6.3 PINN surrogate model

To generate the PINN surrogate model for the heat exchanger network, the PINN model for the same thermofluid system, described in Chapter 4 (pg. 26), was extended from considering single samples only to considering multiple simulation cases simultaneously. This required an adjustment of the *TensorFlow* tensor dimensions and several adjustments to various neural network hyperparameters. The final PINN surrogate model was then trained, and its predictive performance was tested for both interpolation and extrapolation cases.

6.3.1 Model development

In this study, the PINN surrogate model used the same network architecture as the MLP surrogate model to ensure that any performance differences between the two models could not be attributed to a difference in the complexity of the network architectures. The PINN surrogate model therefore contained 3 hidden layers with 64 neurons each.

The dimensions of the vectors used within the PINN loss function calculation were adjusted for the PINN surrogate model to ensure that the *TensorFlow* operations being implemented were executed across all samples as desired. The vectors within the PINN loss function were defined as column vectors with the dimensions given as $N\times1$, where N represents the number of unique samples being passed through the PINN at once. Additionally, the initial pre-training step was adjusted such that the model was trained to satisfy the boundary

conditions for all samples, rather than for just a single sample. This adjustment further reduced the size of the solution space within which the model was searching for the optimal network parameters.

During the initial training and testing of the PINN surrogate model, it became clear that reaching the desired loss tolerance of 1E–06 would be difficult using a large dataset of unique samples. The number of samples in the training dataset was therefore decreased to a smaller value, and datasets of 2, 5, and 10 samples were used to train the model. Full batch learning was implemented for these datasets, instead of pure stochastic learning. In this process, the entire dataset is passed through the neural network at once to calculate a single loss value. Full batch learning was implemented to improve the computational efficiency of the optimisation process. Furthermore, full batch learning was also used to avoid the PINN surrogate model becoming stuck at local minima during training [53].

Initially, the PINN surrogate model used the tanh activation function for the input and hidden layers, as this was the best-performing activation function for the PINN model developed for the analysis of single samples for the same thermofluid system (see Chapter 4.3.3 [pg. 38]). However, when using this activation for the PINN surrogate model, it was found that the model loss would stagnate during training. The model loss would stop decreasing once it had reached values in the order of 1E–03. The values at which the loss stagnated were significantly greater than the desired loss tolerance of 1E–06, indicating that the PINN model was not performing as desired. The problem of the loss stagnation disappeared when the ReLU activation function was used instead. The stagnation of the model loss can therefore be attributed to the saturation of the tanh activation function. Activation function saturation can occur for bounded activation functions such as tanh, which has a range of $y \in (-1,1)$. Because the tanh function has two asymptotic regions approaching -1 and 1, large changes in the incoming signal into the activation function do not result in large variations in the output values generated by applying the activation function. As a result, there were no substantial differences between the output signals for different samples, and the neural network could not distinguish between the different samples being passed through the neural network simultaneously. The phenomenon of activation function saturation is illustrated in Figure 26.

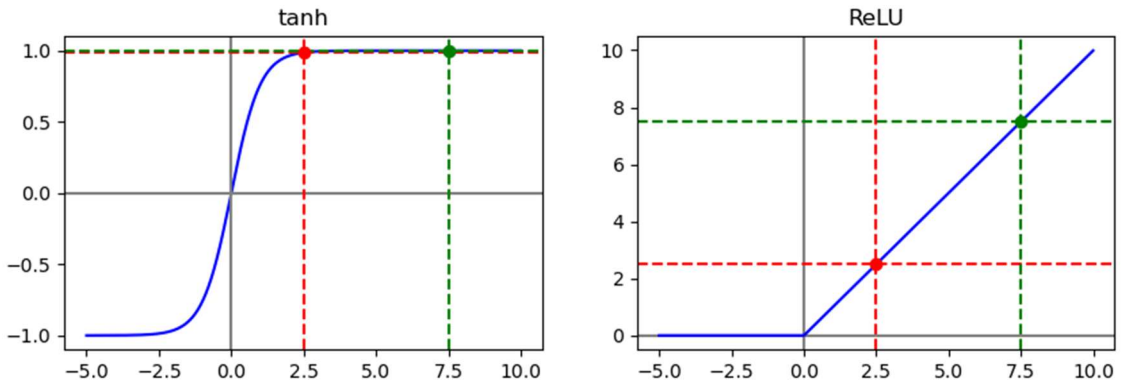


Figure 26. Visualisation of a saturating and non-saturating activation function.

Figure 26 also illustrates that saturation issues do not affect the unbounded ReLU function, as there are no limits imposed on the output values given positive input signals. Therefore, a non-saturating activation should be used for the PINN surrogate model of the heat exchanger network. The ReLU activation function was thus used for the PINN surrogate model. Using this activation function allowed the model loss to be trained to values below the stagnation point of 1E–03.

Despite the implementation of the non-saturating ReLU activation function, the model was still unable to reach the desired loss tolerance of 1E–06. The issue of the vanishing gradient problem was then investigated, as it is a phenomenon which commonly occurs when training neural networks. The vanishing gradient problem occurs when the gradients used to update the neural networks tend to zero, and "vanish" when backpropagated from the output layers to the preceding layers of the network. First, the non-saturating Leaky ReLU activation function was implemented as an alternative to the ReLU activation function. The Leaky ReLU function is generally preferred for cases where vanishing gradients might pose a problem, as it has a small slope for negative function inputs, rather than a value of zero. As a result, the Leaky ReLU activation function enables the calculation of gradient values for the region where the function inputs are negative. In this study, the Leaky ReLU function was implemented with a default slope of 0.3 for the negative input region. The use of the Leaky ReLU activation function resulted in a significant improvement in the success of the training process, and the model was trained to smaller loss values. In addition to the choice of activation function, the vanishing gradient problem can also be compounded by the complexity of the neural network. The more complex the network architecture, the more pronounced the effect of the vanishing gradient problem. The network architecture was therefore simplified; however, this had no noticeable effect on the loss values. The original architecture of 3 hidden layers with 64 neurons each was thus still used.

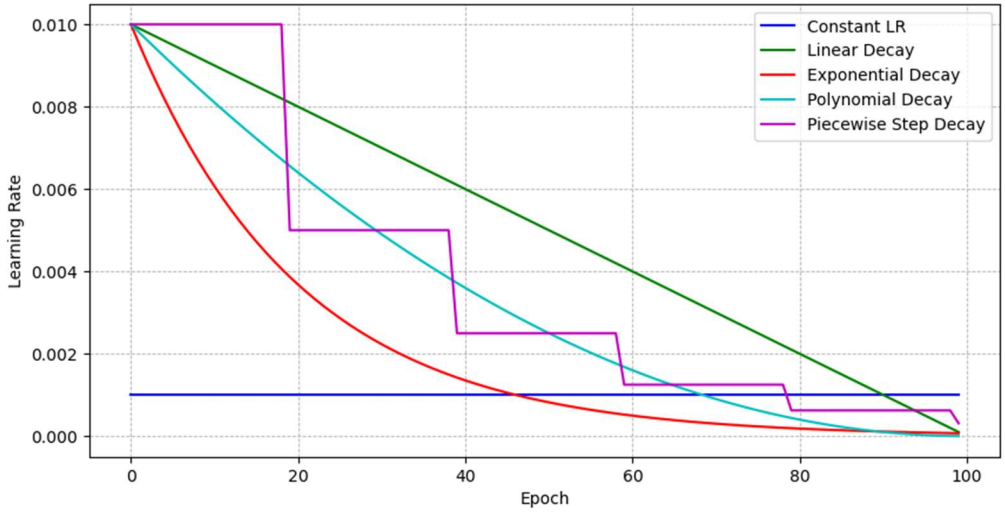


Figure 27. Various learning schedulers that can be applied to train neural networks.

Finally, a learning rate scheduling approach was tested to adjust the neural network learning rate as the training progressed. Learning rate scheduling approaches often lower the learning rate during training, allowing larger

updates to the network parameters during the early stages of training and refining the magnitude of the updates as the network parameters approach their optimal values. This can be helpful as it not only prevents the network from diverging, which can occur if the learning rate is too large, but also reduces the time taken to train the neural network, by allowing larger learning rates initially. A variety of learning rate schedules can be used, as shown in Figure 27.

In this study, a piecewise step decay was implemented, as this learning rate scheduling approach allows for the greatest control over the learning rate throughout the training process. The piecewise step decay learning rate schedule used to train the PINN surrogate model of the heat exchanger network is given in Table 32. Rather than using the given learning rate scheduling function in *TensorFlow* to adjust the learning rate within a single training loop, the training loop was restarted for each new set of learning rate and epoch parameters. A manual learning rate scheduling approach was implemented as it was found to accelerate the training procedure, in comparison to the built-in *TensorFlow* function. With the Leaky ReLU activation function and the piecewise step decay learning rate scheduling, the PINN surrogate model could only be trained to the desired loss tolerance of 1E–06 for datasets that contained two to five samples.

Table 32. Learning rate schedule for training the PINN surrogate model of the heat exchanger network.

Learning rate	Number of epochs
1E–03	1000
1E–04	800
5E–05	800
1E–05	800

6.3.2 Model performance

The final PINN surrogate model for the heat exchanger network was trained using a dataset containing five different samples. As shown in Table 33, the time required to generate the PINN surrogate model was 477.1 seconds, which equates to approximately eight minutes. This table also emphasises that no training data was generated to train the PINN surrogate model.

Table 33. Time taken to generate the PINN surrogate model.

Time for data generation (s)	Model training time (s)	Total time to generate model (s)
0	477.1	477.1

The solutions generated by the trained PINN surrogate model for the five training samples were then compared to the benchmark solutions generated via the conventional, physics-based model of the same system. Table 34 provides the absolute and relative errors of the solutions generated by the PINN surrogate model for the five training samples.

Table 34. Accuracy (absolute errors and relative error percentages) per parameter of the solutions to the training samples for the PINN surrogate model of the heat exchanger network.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	0.0057	1.5224	4.8370	0.0041	1.5991	0.0429
	Min.	0.0007	0.1581	0.2021	0.0006	0.3634	0.0105
	Avg.	0.0029	1.1184	2.4925	0.0023	0.9747	0.0249
Relative error (%)	Max.	0.1131	0.1998	0.0298	0.0880	0.2140	0.0427
	Min.	0.0177	0.0174	0.0012	0.0118	0.0411	0.0100
	Avg.	0.0625	0.1493	0.0155	0.0519	0.1287	0.0247

These results show that the PINN surrogate model of the heat exchanger network generated accurate solutions for the network mass flow rates, enthalpies, and pressures. The largest average error between the PINN surrogate model solutions and the benchmark solutions was only 0.15% for the five training samples. The performance of the PINN surrogate model is similar to the PINN model of the same system that considered single samples only (see Chapter 4.3.4 [pg. 40]). These results demonstrate that the PINN network parameters were successfully trained to generate accurate solutions that satisfy the mass, momentum, and energy balance equations for the heat exchanger network by considering five different samples simultaneously. Furthermore, despite using a limited number of samples for training, the PINN surrogate model was successfully trained using only the governing physics equations.

The predictive performance of the trained PINN surrogate model was then tested on five unseen samples that fell inside of the training envelope (i.e., interpolation), as well as five samples that fell outside of the training envelope (i.e., extrapolation). The accuracy of the predictions made by the PINN surrogate model was assessed by comparing the predictions against the benchmark solutions generated via the conventional, physics-based process model of the same system. The results of this comparison for the interpolation test set and the extrapolation test set are shown in Table 35 and Table 36, respectively.

Table 35. Interpolation performance (absolute errors and relative error percentages) per parameter for the PINN surrogate model of the heat exchanger network.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	0.7417	35.7362	548.7739	0.4447	39.1086	12.2660
	Min.	0.0164	15.7526	112.6328	0.0563	15.1173	2.8926
	Avg.	0.2948	24.7605	373.7970	0.2024	25.4417	6.0425
Relative error (%)	Max.	18.9105	5.1535	3.3669	11.2943	5.5911	12.8703
	Min.	0.3760	2.0116	0.7159	1.2075	1.9318	2.8395
	Avg.	6.7756	3.3446	2.2156	4.6031	3.4501	6.0507

Table 36. Extrapolation performance (absolute errors and relative error percentages) per parameter for the PINN surrogate model of the heat exchanger network.

		$\dot{m}_{CO2}$ (kg/s)	h_{CO2} (kJ/kg)	p_{CO2} (kPa)	$\dot{m}_{air}$ (kg/s)	h_{air} (kJ/kg)	p_{air} (kPa)
Absolute error	Max.	1.8446	119.4814	2454.2133	0.5207	74.3279	59.5787
	Min.	1.4301	99.9344	1476.6159	0.3440	52.3304	52.0603
	Avg.	1.6788	108.3803	2106.4433	0.4575	65.4193	55.7352
Relative error (%)	Max.	20.8615	12.1927	9.3750	7.1513	7.7943	41.8831
	Min.	17.3054	9.9892	5.7901	4.9569	5.4099	37.6574
	Avg.	19.6635	11.0260	8.0819	6.4250	6.8409	39.6370

In comparison to the accuracy of the solutions for the training samples, there is a significant reduction in the accuracy of the PINN surrogate model predictions for unseen samples, with the largest average being 6.8% and 39.6% for interpolation and extrapolation, respectively. The reduced performance on unseen data, in comparison to the samples used for training, indicates that the dataset containing five samples does not adequately represent the design space. Therefore, the amount of information that the PINN model is able to obtain from the training data is limited. As a result, the PINN surrogate model is likely underfitting the solutions space. Furthermore, because the PINN was trained on a limited number of samples, its ability to extrapolate could not be evaluated sufficiently. The usefulness of the PINN surrogate model for making predictions for unseen samples is thus limited based on the presented implementation and mentioned adjustments.

Despite all the adjustments to the PINN surrogate model, it could not be trained to the desired loss tolerance of 1E–06 for datasets which contained more than five samples. Instead, datasets containing a larger number of samples could only be trained to final losses of approximately 1E–05, which did not yield suitable model performance and accuracy. This problem could not be resolved within the scope of the present study and should be investigated further.

A potential reason for this phenomenon is that the solution space of network parameters is not robust enough to find a unique solution that satisfies multiple samples due to the substantial number of degrees of freedom in the mass, energy, and momentum balance equations. A possible solution to the encountered phenomenon involves including additional information about the underlying physics in the training process of the surrogate model by using a limited amount of labelled training data to train the model in addition to the governing equations. This would introduce an additional component into the overall loss calculation, as the difference between the predictions and the labelled target data would be used to evaluate the quality of the predictions, in addition to the set of governing equations.

The idea of combining a limited amount of labelled training data to train the model with the governing equations to train the PINN model is illustrated in Figure 28, which shows the proposed adjustments to the structure of the PINN. Because the governing equations would be used in conjunction with the labelled training

data, less training data could likely be used to train the adjusted PINN model in comparison to a conventional MLP surrogate model. The adjusted PINN model would therefore still address the limitations related to data acquisition that are often encountered with conventional ANN surrogate models.

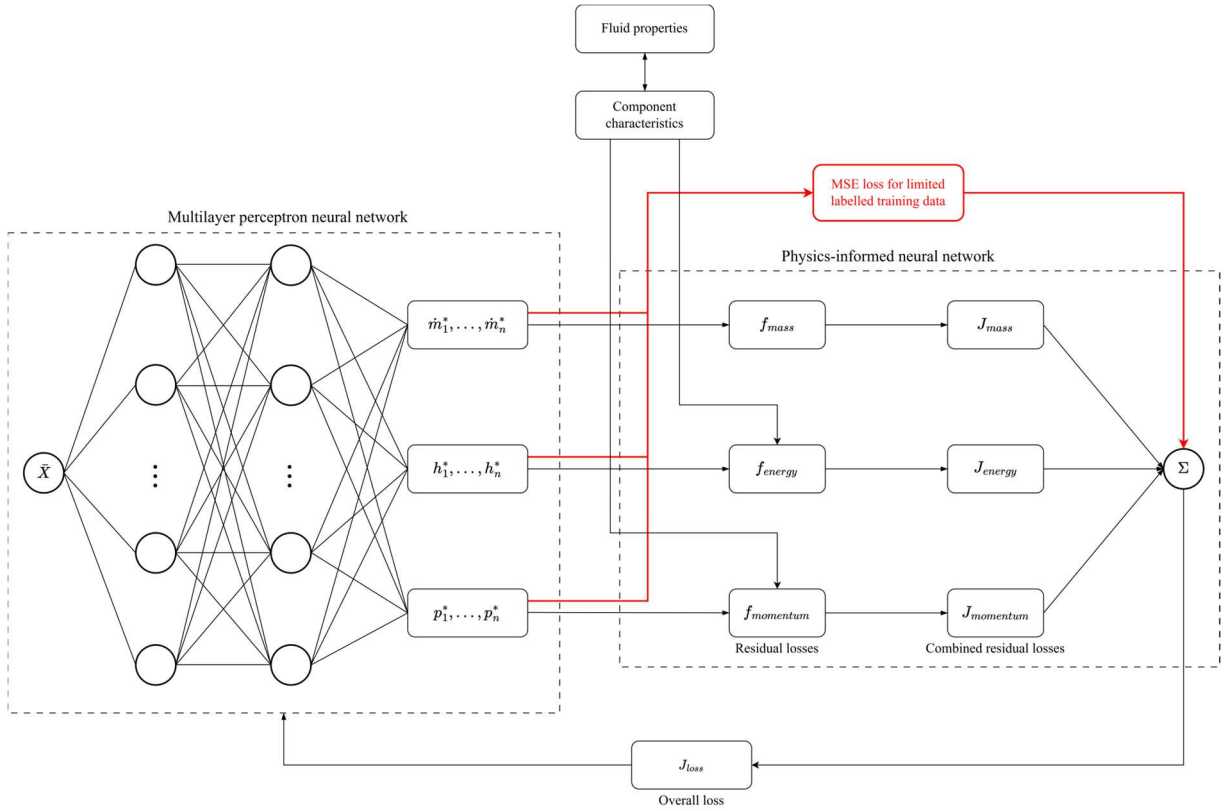


Figure 28. Proposed structure for an adjusted PINN that uses some labelled data in addition to the governing physics equations during training.

Chapter 7. Conclusions and recommendations

In this chapter, a summary of the key findings and conclusions of this study is presented. A brief discussion of recommendations for future work that builds on this research is also included here.

7.1 Conclusions

In this study, the PINN modelling methodology was applied to train ANNs to analyse three different integrated thermofluid systems, namely: (i) a heat exchanger network with two different fluid streams; (ii) a recuperated closed Brayton cycle containing various turbomachines and heat exchangers, and (iii) a simplified boiler consisting of a furnace and radiative-convective superheater. The three case studies were selected as they required the consideration of important phenomena typically encountered in integrated thermofluid systems modelling including (i) interaction between different fluid streams via heat transfer; (ii) components in both series and parallel configurations; (iii) turbomachines with associated non-linear performance characteristics; (iv) complex component characteristics, such as radiative heat transfer; (v) fluid property calculations for pure fluids in the subcritical, two-phase, and supercritical regions; and (vi) fluid property calculations for gas mixtures. The PINN models developed for these three case studies were employed as numerical solvers and thus analysed single samples only.

The PINN loss functions that represent the mass, energy, and momentum balance equations must be written in non-dimensional form to ensure that the various components of the loss function are of similar orders of magnitude. Applying this formulation of the equations ensured that all components of the loss function were considered equally during the PINN training process. It was also shown that to ensure consistent performance and accurate predictions from the PINN models, the trainable network parameters should first be trained using a supervised pre-training step before the PINN loss function, consisting of the governing physics equations, is implemented. During the supervised pre-training step, the desired output values at all the nodes and elements throughout the thermofluid system were set equal to prescribed boundary values obtained from the input vector. The PINNs were then trained to predict these values. Furthermore, to implement the PINN modelling methodology successfully, the *CoolProp* fluid property library could not be used to perform the fluid property calculations. Instead, sets of interpolated polynomial functions were developed for each case study and used to estimate the required fluid properties. In comparison to the *CoolProp* fluid property library, using these polynomial functions offered a significant increase in the computational speed of the fluid property calculations, enabled vectorised calculations, and automatically clipped the fluid property values to fall within a realistic range.

This study also demonstrated that the use of a hybrid first- and second-order optimisation approach that combined the Adam and TNC optimisation algorithms reduced the number of training iterations by as much as 690 in comparison to using the purely first-order Adam optimiser. This therefore showed that the use of the

hybrid Adam-TNC optimisation approach provided a significant computational advantage over the pure Adam optimisation approach and accelerated the training process of the PINN models.

The solutions generated via the trained PINN models were compared to benchmark solutions generated via conventional, physics-based thermofluid process models. The largest average relative errors between the solutions generated with the PINN models and the benchmark solutions were 0.16%, 0.93%, and 0.095% for the heat exchanger network, Brayton cycle, and simplified boiler, respectively. Such small errors indicate that the solutions generated via the PINN models matched the benchmark solutions very well, thus highlighting the accuracy of the PINN models. Overall, it was shown that the PINN modelling methodology can be applied successfully to analyse integrated thermofluid systems by solving the non-dimensionalised forms of the mass, energy, and momentum balance equations in the neural network loss function.

Furthermore, it was shown that the time taken for the trained PINN models of all three thermofluid systems to generate solutions is in the order of 1×10^{-3} seconds. This is a significant improvement over the conventional process models which required 1×10^{-1} seconds to generate a solution in the case of the heat exchanger network and Brayton cycle and 1×10^{0} seconds in the case of the more complex boiler system. This demonstrates that the inference speed of the PINNs remains constant, regardless of the complexity of the thermofluid system. In contrast, the time taken to generate a solution for a more complex system with the conventional modelling approach scales non-linearly. The significant difference in inference speed, particularly for the more complex thermofluid system, suggests that the PINN modelling methodology can offer significant reductions in computational expense when the problem is scaled up to include large numbers of samples for surrogate modelling.

The possibility of using the PINN modelling methodology for surrogate modelling applications was investigated using the heat exchanger network as the selected case study. A conventional MLP surrogate model of the same system was also developed to investigate the limitations of conventional ANNs for analysing thermofluid systems. It was shown that the MLP surrogate model required a training dataset containing 1000 samples. The time taken to generate these training samples via the conventional physics-based process model accounts for approximately 60% of the total time taken to develop the entire MLP surrogate model. This demonstrates the significant, and often prohibitive, computational expense of obtaining enough data to train MLP surrogate models for integrated thermofluid systems. For samples that fall within the range of the training data, the largest average error between the MLP surrogate model predictions and the benchmark solutions was only 0.27%. This demonstrates that the MLP surrogate model is capable of interpolation and can be used to make predictions with a high level of accuracy. However, it was shown that the predictive performance of the MLP surrogate model declined when making predictions for samples that fall outside of the range of training data.

The PINN surrogate model of the heat exchanger network was developed by extending the previous PINN model of this system to consider multiple samples simultaneously. This involved the following changes: (i) converting all vectors used within the PINN loss function to column vectors, (ii) implementing a non-saturating activation function to ensure that the output signals for various samples differed enough for the model to distinguish between them, and (iii) using a piecewise constant decay learning rate scheduling approach to refine the magnitude of the learning rate as training progressed. Despite these adjustments, the PINN surrogate model could only be trained successfully using a dataset containing a maximum of five samples.

The solutions generated by the trained PINN surrogate model for the five training samples were then compared to the benchmark solutions generated via the conventional, physics-based model of the same system. The largest average error between the PINN surrogate model solutions and the benchmark solutions was only 0.15% for the five training samples. This demonstrates that the PINN surrogate model was successfully trained using only the governing physics equations. However, there was a significant reduction in the accuracy of the PINN surrogate model predictions for unseen samples, with the largest average being 6.8% and 39.6% for interpolation and extrapolation, respectively. The reduced performance on unseen data, in comparison to the samples used for training, indicates that the dataset containing five samples does not adequately represent the design space and the amount of information that the PINN model is able to obtain from the training data is therefore limited. Furthermore, because the PINN was trained on a limited number of samples, its ability to extrapolate could not be evaluated sufficiently. The usefulness of the PINN surrogate model for making predictions for unseen samples is limited based on the presented implementation and mentioned adjustments. However, this limitation could not be resolved within the scope of the present study and should be investigated further.

Despite its current limitations, the significant increase in computational speed offered by the PINN modelling methodology when applied to the analysis of integrated thermofluid systems suggests that this is a promising modelling technique that should be explored further. The PINN models developed in this study therefore lay the foundation for applying this modelling technique to more complex thermofluid systems, as well as to problems that require real-time predictions (such as anomaly detection, diagnosis, and forecasting) or for multi-query analysis problems (such as those presented by system and component design and optimisation).

7.2 Recommendations

The current study was limited to the analysis of steady-state integrated thermofluid systems. However, thermofluid systems are also operated under transient conditions, particularly during startup and shutdown procedures. The analysis of transient systems is even more computationally expensive than that of steady-state systems. Due to the computational advantage offered by the PINN modelling methodology in comparison to

conventional methods, it would be valuable to investigate whether this modelling methodology could be used to analyse the transient operation of integrated thermofluid systems as well.

Additionally, the scope of the current study was limited to fully connected neural networks only. Future work could therefore investigate the effect of different network structures, such as graph convolutional neural networks and convolutional structures, on the training and performance of the PINN models for integrated thermofluid systems.

Results of the current study showed the significant increase in inference speed that can be achieved with PINN models. To capitalise on this increase in speed, it would be worthwhile to address the limitations of the PINN surrogate model in future work. This could include investigating the effect of combining the MLP and PINN training strategies, as outlined in Chapter 6.3.2 (pg. 67), to allow the PINN surrogate model to be trained using an increased number of samples, thus improving the model's ability to generalise to unseen samples that are both within and beyond the envelope of training data.

List of References

[1] R. Laubscher and P. Rousseau, "Application of mixed-variable physics-informed neural networks to solve normalised momentum and energy transport equations for 2D internal convective flow," *arXiv,* 2021.

[2] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems," *arXiv,* 2022.

[3] 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.

[4] J. Ortega, S. Khivsara, J. Christian, C. Ho, J. Yellowhair, and P. Dutta, "Coupled modeling of a directly heated tubular solar receiver for supercritical carbon dioxide Brayton cycle: Optical and thermal-fluid evaluation," *Applied Thermal Engineering,* vol. 109, pp. 970-978, 2016.

[5] Ansys. "Ansys fluent fluid simulation software." https://www.ansys.com/products/fluids/ansys -fluent (accessed 28 June, 2022).

[6] X. Wei, V. Manovic, and D. P. Hanak, "Techno-economic assessment of coal- or biomass-fired oxy-combustion power plants with supercritical carbon dioxide cycle," *Energy Conversion and Management,* vol. 221, 2020.

[7] AspenTech. "Aspen Plus®." https://www.aspentech.com/en/products/engineering/aspen-plus (accessed 26 June, 2022).

[8] M. Rauch, A. Galović, and Z. Virag, "Optimization of combined Brayton-Rankine cycle with respect to the total thermal efficiency," *Transactions of Famena,* vol. 40, pp. 1-10, 2016.

[9] MathsWorks. "MatLab " https://www.mathworks.com/products/matlab.html (accessed 28 June, 2022).

[10] B. P. Baillie, H. Ravichandar, I. Salehi, A. P. Dani, and G. M. Bollas, "Approaches for creation and evaluation of computationally efficient thermofluid system models," *IFAC-PapersOnLine,* vol. 51, no. 18, pp. 868-873, 2018.

[11] P. Ding, X.-H. Wu, Y.-L. He, and W.-Q. Tao, "A fast and efficient method for predicting fluid flow and heat transfer problems," *Journal of Heat Transfer,* vol. 130, 2008.

[12] M. Hosoz and H. M. Ertunc, "Artificial neural network analysis of an automobile air conditioning system," *Energy Conversion and Management,* vol. 47, no. 11-12, pp. 1574-1587, 2006.

[13] R. A. Haffejee and R. Laubscher, "Application of machine learning to develop a real-time air-cooled condenser monitoring platform using thermofluid simulation data," *Energy and AI,* vol. 3, 2021.

[14] M. Fast and T. Palmé, "Application of artificial neural networks to the condition monitoring and diagnosis of a combined heat and power plant," *Energy,* vol. 35, no. 2, pp. 1114-1120, 2010.

[15] G. Waxenegger-Wilfing, K. Dresia, J. C. Deeken, and M. Oschwald, "Heat transfer prediction for methane in regenerative cooling channels with neural networks," *Journal of Thermophysics and Heat Transfer,* vol. 34, no. 2, pp. 347-357, 2020.

[16] Y. Shi, W. Zhong, X. Chen, A. B. Yu, and J. Li, "Combustion optimization of ultra supercritical boiler based on artificial intelligence," *Energy,* vol. 170, pp. 804-817, 2019.

[17] S. Singh and H. Abbassi, "1D/3D transient HVAC thermal modeling of an off-highway machinery cabin using CFD-ANN hybrid method," *Applied Thermal Engineering,* vol. 135, pp. 406-417, 2018.

[18] A. Pacheco-Vega, M. Sen, K. T. Yang, and R. L. McClain, "Neural network analysis of fin-tube refrigerating heat exchanger with limited experimental data," *International Journal of Heat and Mass Transfer,* vol. 44, pp. 763-770, 2001.

[19] 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, 2020.

[20] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *Journal of Computational Physics,* vol. 394, pp. 56-81, 2019.

[21] S. Cai, Z. Mao, ZhichengWang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: A review," *Acta Mechanica Sinica,* vol. 37, pp. 1727-1738, 2021.

[22] E. Ang and B. F. Ng, "Physics-Informed Neural Networks for flow around airfoil," in AIAA SciTech Forum, San Diego, California, United States of America, 2022.

[23] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Computer Methods in Applied Mechanics and Engineering,* vol. 360, 2020.

[24] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *Journal of Computational Physics,* vol. 404, 2020.

[25] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," *arXiv,* 2020.

[26] R. Laubscher, "Simulation of multi-species flow and heat transfer using physics-informed neural networks," *Physics of Fluids,* vol. 33, no. 8, 2021.

[27] S. Markidis, "The Old and the New: Can physics-informed deep-learning replace traditional linear solvers?," *Front Big Data,* vol. 4, 2021.

[28] Y. A. Çengel and M. A. Boles, Thermodynamics An Engineering Approach, 7th ed. Boston, USA: McGraw-Hill, 2010.

[29] CoolProp. "Welcome to CoolProp." http://www.coolprop.org/# (accessed 16 August, 2022).

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. Cambridge, USA: MIT Press, 2016.

[31] A. Géron, N. Tache, Ed. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2 ed. Sebastopol: O'Reilly Media, 2019.

[32] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[33] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural computation,* vol. 8, no. 7, pp. 1341-1390, 1996.

[34] M. N. Thombrea, H. A. Preisiga, and M. B. Addisa, "Developing Surrogate Models via Computer Based Experiments," *Computer Aided Chemical Engineering,* vol. 37, pp. 641-646.

[35] M. D. Mckay, R. J. Beckman, and W. J. Conover, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics,* vol. 42, no. 1, pp. 55-61, 1979.

[36] A. Lee. "pyDOE: The experimental design package for python." pyDOEHosted.com. https://pythonhosted.org/pyDOE/ (accessed. 08 August 2022).

[37] SciPy v1.12.0 Manual. "Statistical functions (scipy.stats)." Docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/stats.html (accessed. 20 November 2023).

[38] Y. Ahn *et al.*, "Review of supercritical CO2 power cycle technology and current status of research and development," *Nuclear Engineering and Technology,* vol. 47, no. 6, pp. 647-661, 2015.

[39] P. Rousseau, (2021) sCO2_SimpleRecuperatedBrayton.ipynb (Revision 1) [Python notebook source code]. See Appendix A.

[40] Y. A. Çengel and A. Ghajar, Heat and Mass Transfer: Fundamentals and Applications, 5th ed. New York, USA: McGraw-Hill Education, 2015.

[41] SciPy v1.12.0 Manual. "scipy.optimize.fsolve." Docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html (accessed 20 November, 2023).

[42] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes, "A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics," *Computer Methods in Applied Mechanics and Engineering,* vol. 379, 2021.

[43] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[44] CoolProp. "Tabular Interpolation." http://www.coolprop.org/coolprop/Tabular.html (accessed 08 December, 2023).

[45] SciPy v1.12.0 Manual. "scipy.interpolate.RectBivariateSpline." Docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RectBivariateSpline.html (accessed 08 December, 2023).

[46] R. Laubscher, "Utilization of artificial neural networks to resolve chemical kinetics in turbulent fine structures of an advanced CFD combustion model," Doctor of Philosophy PhD, Faculty of Engineering, Stellenbosch University, Stellenbosch, 2017.

[47] P. Rousseau, (2021) BoilerProcessModels.py. (Revision 1) [Python notebook source code]. See Appendix C.

[48] P. Rousseau, R. Laubscher, and B. T. Rawlins, "Heat Transfer Analysis Using Thermofluid Network Models for Industrial Biomass and Utility Scale Coal-Fired Boilers," *Energies,* vol. 16, no. 4, 2023.

[49] R. Laubscher and P. Rousseau, (2021) *GasMixtureProperties.py*. (Revision 1) [Python notebook source code]. See Appendix B.

[50] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods," Machine Learning, vol. 110, no. 3, pp. 457-506, 2021.

[51] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*. AMLBook New York, 2012.

[52] X. Zhu and X. Wu, "Class Noise vs. Attribute Noise: A Quantitative Study," Artificial Intelligence Review, vol. 22, no. 3, pp. 177-210, 2004.

[53] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. The MIT Press, 2019.

Appendix A. Python code for the Brayton cycle model

The *Python* source code listing for the existing recuperated Brayton cycle model that was developed by Rousseau [39] is provided here.

```
# Preliminaries
import numpy as np
from scipy.optimize import fsolve
from CoolProp.CoolProp import PropsSI
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from CyclePlots import CyclePlots
kel = 273.15


# Momentum balance component characteristics
def momentum_components(m_dot, h, p, fluid, a_C, a_T, K_RXHP, K_H, K_RXLP, K_PC):

    # Preliminaries
    T = np.zeros(7)
    T = PropsSI('T','P',p,'H',h,fluid)

    # Compressor characteristics
    T_0i_C = T[1] # K
    p_0i_C = p[1] # kPa
    CM_C = (m_dot*np.sqrt(T_0i_C))/p_0i_C
    PR_C = a_C[0]+a_C[1]*CM_C+a_C[2]*np.power(CM_C,2)
    dp_C = p[1]*(PR_C-1)

    # Recuperator HP characteristics
    rho_RXHP = PropsSI('D','P',0.5*(p[2]+p[3]),'H',0.5*(h[2]+h[3]),fluid)
    dp_RXHP = K_RXHP*abs(m_dot)*m_dot/rho_RXHP

    # Heater characteristics
    rho_H = PropsSI('D','P',0.5*(p[3]+p[4]),'H',0.5*(h[3]+h[4]),fluid)
    dp_H = K_H*abs(m_dot)*m_dot/rho_H

    # Turbine characteristics
    T_0i_T = T[4] # K
    p_0i_T = p[4] # kPa
    CM_T = (m_dot*np.sqrt(T_0i_T))/p_0i_T
    PR_T = a_T[0]+a_T[1]*CM_T+a_T[2]*np.power(CM_T,2)
    dp_T = p[4]*(1/PR_T-1)

    # Recuperator LP characteristics
    rho_RXLP = PropsSI('D','P',0.5*(p[5]+p[6]),'H',0.5*(h[5]+h[6]),fluid)
    dp_RXLP = K_RXLP*abs(m_dot)*m_dot/rho_RXLP


    # Precooler characteristics
    rho_PC = PropsSI('D','P',0.5*(p[6]+p[1]),'H',0.5*(h[6]+h[1]),fluid)
    dp_PC = K_PC*abs(m_dot)*m_dot/rho_PC

    return dp_C, dp_RXHP, dp_H, dp_T, dp_RXLP, dp_PC
```

```
# Energy balance component characteristics
def energy_components(m_dot, h, p, fluid, b_C, b_T, eps_H, eps_PC, eps_RX, T_H, T_L):

    # Preliminaries
    T = np.zeros(7)
    T = PropsSI('T','P',p,'H',h,fluid)

    # Compressor characteristics
    T_0i_C = T[1] # K
    p_0i_C = p[1] # kPa
    CM_C = (m_dot*np.sqrt(T_0i_C))/p_0i_C
    eta_C = b_C[0]+b_C[1]*CM_C+b_C[2]*np.power(CM_C,2)
    s_1 = PropsSI('S','P',p[1],'H',h[1],fluid)
    h_2s = PropsSI('H','P',p[2],'S',s_1,fluid)
    w_C = (1/eta_C)*(h[1]-h_2s)

    # Recuperator HP characteristics
    cp_RXHP = PropsSI('C','P',0.5*(p[2]+p[3]),'H',0.5*(h[2]+h[3]),fluid)
    cp_RXLP = PropsSI('C','P',0.5*(p[5]+p[6]),'H',0.5*(h[5]+h[6]),fluid)
    cp_RXmin = min(cp_RXHP,cp_RXLP)
    q_RXHP = eps_RX*cp_RXmin*(T[5]-T[2])

    # Heater characteristics
    cp_H = PropsSI('C','P',0.5*(p[3]+p[4]),'H',0.5*(h[3]+h[4]),fluid)
    q_H = eps_H*cp_H*(T_H-T[3])

    # Turbine characteristics
    T_0i_T = T[4] # K
    p_0i_T = p[4] # kPa
    CM_T = (m_dot*np.sqrt(T_0i_T))/p_0i_T
    eta_T = b_T[0]+b_T[1]*CM_T+b_T[2]*np.power(CM_T,2)
    s_4 = PropsSI('S','P',p[4],'H',h[4],fluid)
    h_5s = PropsSI('H','P',p[5],'S',s_4,fluid)
    w_T = eta_T*(h[4]-h_5s)

    # Recuperator LP characteristics
    q_RXLP = eps_RX*cp_RXmin*(T[2]-T[5])

    # Precooler characteristics
    cp_PC = PropsSI('C','P',0.5*(p[6]+p[1]),'H',0.5*(h[6]+h[1]),fluid)
    q_PC = eps_PC*cp_PC*(T_L-T[6])

    return q_PC, w_C, q_RXHP, q_H, w_T, q_RXLP


# Momentum balance equations
def momentum_balance(p_x, h, fluid, a_C, a_T, K_RXHP, K_H, K_RXLP, K_PC, p_L):

    # Boilerplating
    p = p_x[:7]
    m_dot = p_x[7]

    # Preliminaries
    f = np.zeros(8)


    # Component characteristics
    dp_C, dp_RXHP, dp_H, dp_T, dp_RXLP, dp_PC = momentum_components(m_dot, h,  p, \
        fluid, a_C, a_T, K_RXHP, K_H, K_RXLP, K_PC)
```

```
    # Momentum balance residuals
    f[0] = ((p[0])-(p[6]))/p[0]
    f[1] = ((p[1])-(p_L))/p[1]
    f[2] = ((p[2])-(p[1]+dp_C))/p[2]
    f[3] = ((p[3])-(p[2]-dp_RXHP))/p[3]
    f[4] = ((p[4])-(p[3]-dp_H))/p[4]
    f[5] = ((p[5])-(p[4]+dp_T))/p[5]
    f[6] = ((p[6])-(p[5]-dp_RXLP))/p[6]
    f[7] = ((p[1])-(p[6]-dp_PC))/p[1]

    return f

# Energy balance equations
def energy_balance(h, p, m_dot, fluid, b_C, b_T, eps_H, eps_PC, eps_RX, T_H, T_L):

    # Preliminaries
    f = np.zeros(7)

    # Component characteristics
    q_PC, w_C, q_RXHP, q_H, w_T, q_RXLP = energy_components(m_dot, h, p, fluid, \                b_C, b_T, eps_
H, eps_PC, eps_RX, T_H, T_L)

    # Energy balance residuals
    f[0] = ((h[0])-(h[6]))/h[0]
    f[1] = ((h[1])-(h[6]+q_PC))/h[1]
    f[2] = ((h[2])-(h[1]-w_C))/h[2]
    f[3] = ((h[3])-(h[2]+q_RXHP))/h[3]
    f[4] = ((h[4])-(h[3]+q_H))/h[4]
    f[5] = ((h[5])-(h[4]-w_T))/h[5]
    f[6] = ((h[6])-(h[5]+q_RXLP))/h[6]

    return f


def solve_brayton_cycle(m_dot, p, h, p_L, T_H, T_L, fluid, a_C, b_C, a_T, b_T, \
            K_RXHP=0.5, K_H=0.5, K_RXLP=0.5, K_PC=0.5, \
            eta_G=0.9, eta_M=0.99, eps_H=0.85, eps_PC=0.85, eps_RX=0.95):

    # Initialize
    T = np.zeros(7)
    s = np.zeros(7)
    rho = np.zeros(7)
    cp = np.zeros(7)

    # Boilerplating
    p_x = np.append(p,m_dot)

    # Iterative solution
    it = 0
    err = 1e10


    while err > 1e-6 and it < 20:
        it = it+1
        # Save old value
        m_dot_old = m_dot
```

```
        # Solve momentum balance equations
        p_x = fsolve(momentum_balance, p_x, args=(h, fluid, a_C, a_T, K_RXHP,  K_H, \
                K_RXLP, K_PC, p_L))


        # Boilerplating
        p = p_x[:7]
        m_dot = p_x[7]

        # Solve energy balance equations
        h = fsolve(energy_balance, h, args=(p, m_dot, fluid, b_C, b_T, eps_H, \
            eps_PC, eps_RX, T_H, T_L))

        # Calculate err
        err = abs(m_dot-m_dot_old)/m_dot
        print('it = {:2}, err = {:.1e}'.format(it, err))

    # Post-processing
    T = PropsSI('T','P',p,'H',h,fluid)
    s = PropsSI('S','P',p,'H',h,fluid)
    rho = PropsSI('D','P',p,'H',h,fluid)
    cp = PropsSI('C','P',p,'H',h,fluid)
    dp_C, dp_RXHP, dp_H, dp_T, dp_RXLP, dp_PC = momentum_components(m_dot, h,  p, \                    fluid,
a_C, a_T, K_RXHP, K_H, K_RXLP, K_PC)
    q_PC, w_C, q_RXHP, q_H, w_T, q_RXLP = energy_components(m_dot, h, p, fluid, \
                    b_C, b_T, eps_H, eps_PC, eps_RX, T_H, T_L)

    PR_C = p[2]/p[1]
    PR_T = p[4]/p[5]
    Q_dot_PC = m_dot*q_PC
    W_dot_C = m_dot*w_C
    Q_dot_RXHP = m_dot*q_RXHP
    Q_dot_H = m_dot*q_H
    W_dot_T = m_dot*w_T
    Q_dot_RXLP = m_dot*q_RXLP
    w_G = eta_M*eta_G*(w_T+w_C)
    W_dot_G = m_dot*w_G
    W_dot_L = (W_dot_C+W_dot_T)-(W_dot_G)
    eta_th = (W_dot_C+W_dot_T)/Q_dot_H

    # Energy balance
    f_mom = momentum_balance(p_x, h, fluid, a_C, a_T, K_RXHP, K_H, K_RXLP,K_PC, p_L)
    f_ene = energy_balance(h, p, m_dot, fluid, b_C, b_T, eps_H, eps_PC, eps_RX, \
            T_H, T_L)


    Q_dot_net = Q_dot_PC+Q_dot_RXHP+Q_dot_H+Q_dot_RXLP
    W_dot_net = W_dot_C+W_dot_T
    dE_net = Q_dot_net-W_dot_net
    dp_M_net = dp_C+dp_T
    dp_L_net = dp_RXHP+dp_H+dp_RXLP+dp_PC
    dp_net = dp_M_net-dp_L_net


    return m_dot, p, h, T, s, cp, rho, Q_dot_PC, W_dot_C, Q_dot_RXHP, Q_dot_H, \                W_dot_T, Q_dot_R
XLP, W_dot_G, W_dot_L, eta_th


# Component characteristics
fluid = 'CO2'
a_C = np.array([3.947422e+00, 5.373162e+02, -8.776627e+05])
```

```
b_C = np.array([-1.942890e-16, 1.453743e+03, -5.936429e+05])
a_T = np.array([-8.437242e-23, 3.313531e-09, 9.730980e+06])
b_T = np.array([1.054712e-15, 3.199740e+03, -2.752242e+06])
K = 35      # Nom = 35, Min = 0, Max = 2250
K_RXHP = K
K_H = K
K_RXLP = K
K_PC = K
eta_G = 0.9
eta_M = 0.99
eps_H = 0.85    # Nom = 0.85, Min = 0.35, Max = 1.0
eps_PC = 0.85   # Nom = 0.85, Min = 0.4, Max = 1.0
eps_RX = 0.95   # Nom = 0.95, Min = 0, Max = 1.0

# Boundary values
T_L = 35+kel    # Nom = 35, Min = 15, Max = 220
T_H = 580+kel   # Nom = 580, Min = 330, Max = 1500
p_L = 7600e3    # Nom = 7600e3, Min = 3600e3, Max = 9950e3

# Guess values
m_dot = 350
p_H = 15000e3
p = np.array([p_L, p_L, p_H, p_H, p_H, p_L, p_L])
h = np.zeros(7)+PropsSI('H','P',p[2],'T',100+kel, fluid)

# Solve cycle
m_dot, p, h, T, s, cp, rho, Q_dot_PC, W_dot_C, Q_dot_RXHP, Q_dot_H, W_dot_T, \              Q_dot_RXLP,
W_dot_G, W_dot_L, eta_th = solve_brayton_cycle(m_dot, p, h, \
                p_L, T_H, T_L, fluid, a_C, b_C, a_T, b_T, K_RXHP, K_H, K_RXLP, K_PC, \
                   eta_G, eta_M, eps_H, eps_PC, eps_RX)

# Post-processing
dp_C, dp_RXHP, dp_H, dp_T, dp_RXLP, dp_PC = momentum_components(m_dot, h, p, \
         fluid, a_C, a_T, K_RXHP, K_H, K_RXLP, K_PC)
q_PC, w_C, q_RXHP, q_H, w_T, q_RXLP = energy_components(m_dot, h, p, fluid, \

         b_C, b_T, eps_H, eps_PC, eps_RX, T_H, T_L)
```

Appendix B. Python code for gas mixture properties

The *Python* source code listing for the existing gas mixture property calculation module that was developed by Laubscher and Rousseau [49] is provided here.

```
import CoolProp.CoolProp as CP
import numpy as np

#constants
R_u = 8.31446261815324

#enthalpy of formations
h_f0_CH4 = -74850
h_f0_O2 = 0
h_f0_N2 = 0
h_f0_CO2 = -393520
h_f0_H2O = -285830
h_f0_SO2 = -296810
h_f0_O = 240190
h_f0_H = 218000
h_f0_N = 472650

def molecular_weights(gasmixture={}):
    M = {}
    for key in gasmixture:
        M[key] = CP.PropsSI('M', key)
    return M

def mean_molecular_weight(gasmixture={}):
    MW = 0
    for key in gasmixture:
        MW += (gasmixture[key]/CP.PropsSI('M',key))
    return 1/MW

def gas_constant(gasmixture={}):
    return R_u/mean_molecular_weight(gasmixture)

def density(gasmixture={}, T=298.15, p=101325):
    return p/((R_u/mean_molecular_weight(gasmixture)) * T)

def mole_fractions(gasmixture={}):
    mole_fraction_dict = {}
    for key in gasmixture:
        mole_fraction_dict[key] = gasmixture[key] * \
            mean_molecular_weight(gasmixture)/CP.PropsSI('M',key)
    return mole_fraction_dict

def molar_concentrations(gasmixture={}, rho = 1):
    moles_frac = mole_fractions(gasmixture)
    molar_cons = {}
    for key in moles_frac:
        molar_cons[key] = moles_frac[key] * rho / CP.PropsSI('M',key)
    return molar_cons
```

```
def enthalpy(gasmixture={}, T=298.15, p=101325):
    molefrac = mole_fractions(gasmixture)
    enthalpy = 0
    for key in gasmixture:
        p_partial = molefrac[key]*p
        enthalpy += gasmixture[key]*(CP.PropsSI('H', 'T', T, 'P', p_partial, key) - \
                            CP.PropsSI('H', 'T', 298.15, 'P', 101325, key))
    return enthalpy

def specific_heat_pressure(gasmixture={}, T=298.155, p=101325):
    molefrac = mole_fractions(gasmixture)
    cp = 0
    for key in gasmixture:
        p_partial = molefrac[key]*p
        cp += gasmixture[key] * CP.PropsSI('CPMASS', 'T', T, 'P', p_partial, key)
    return cp

def specific_heat_volume(gasmixture={}, T=298.15, p=101325):
    molefrac = mole_fractions(gasmixture)
    cv = 0
    for key in gasmixture:
        p_partial = molefrac[key]*p
        cv += gasmixture[key] * CP.PropsSI('CVMASS', 'T', T, 'P', p_partial, key)
    return cv

def WilkeMethod(OutputLabel, gasmixture={}, T=298.15, p=101325):
    M_gas = molecular_weights(gasmixture)
    X_gas = mole_fractions(gasmixture)
    n = len(gasmixture)
    M = np.zeros(n+1)
    X = np.zeros(n+1)
    var = np.zeros(n+1)
    phi = np.zeros([n,n+1])

    # Allocate pure gas molar mass, volume fraction and variable values to arrays
    i = 0
    for key in gasmixture:
        i = i+1
        M[i] = M_gas[key]
        X[i] = X_gas[key]

        # Viscosity and conductivity not available for SO2 in CP. Use RL curve fits                from Refrop
        if ((key == 'SO2') and (OutputLabel == 'V')):
            var[i] = -1.35715e-6+4.89079e-8*T-5.88555e-12*T**2
        elif ((key == 'SO2') and (OutputLabel == 'L')):
            var[i] = -3.37104e-3+4.38025e-5*T+3.12276e-9*T**2
        else:
            var[i] = CP.PropsSI(OutputLabel, 'T', T, 'P', X_gas[key]*p, key)

    # Calculate phi matrix
    for i in range(1,n):
        for j in range(1,n):
            phi[i][j] = ((1+((var[i]/var[j])**0.5)*((M[j]/M[i])**0.25))**2) * \
              1/((4/(2**0.5))*((1+M[i]/M[j])**0.5))


    # Calculate mixture variable value
    var_mix = 0
    for i in range(1,n):
        sum = 0
```

```
        for j in range(1,n):
            sum = sum+X[j]*phi[i][j]
        var_mix = var_mix+(X[i]*var[i])/sum
    return var_mix

def viscosity(gasmixture={}, T=298.15, p=101325):
    viscosity = WilkeMethod('V', gasmixture, T, p)
    return viscosity

def conductivity(gasmixture={}, T=298.15, p=101325):
    conductivity = WilkeMethod('L', gasmixture, T, p)
    return conductivity
```

Appendix C. Python code for the boiler process model

Excerpts of the *Python* source code for the existing boiler process model module developed by Rousseau [47] are provided here. These excerpts were implemented in both the conventional physics-based thermofluid process model and the PINN model of the simplified boiler system developed in this study.

```
from matplotlib.pyplot import hot
import numpy as np
from numpy.core.arrayprint import repr_format
import scipy as sp
import math
import CoolProp.CoolProp as cp
import GasMixtureProperties as gmp
from scipy.optimize import fsolve
kel = 273.15

def barometric_pressure(H):
    return 101325*((288-0.0065*H)/288)**5.256

def air_mass_fraction(w):
    # Preliminaries
    M = {'H2O':cp.PropsSI('M','H2O'), 'N2':cp.PropsSI('M','N2'), \
        'O2':cp.PropsSI('M','O2')}    #[kg/mol]
    keylist = []
    for i,k in enumerate(M):
        keylist.append(k)
    M_air = 0.2101*M['O2']+(1-0.2101)*M['N2']    #[kg/mol]

    Y_N2 = (1-0.2101)*M['N2']/M_air
    Y_O2 = 0.2101*M['O2']/M_air
    Y_ha = {}
    Y_ha['O2'] = Y_O2/(Y_N2+Y_O2+w)
    Y_ha['N2'] = Y_N2/(Y_N2+Y_O2+w)
    Y_ha['H2O'] = w/(1+w)

    return Y_ha

def fuel_mass_fraction(Y_DAF, Y_H2O=0.0, Y_ASH=0.0, Y_UC=0.0):
    f = 1-Y_H2O-Y_ASH
    Y = {}
    for key in Y_DAF:
        Y[key] = f*Y_DAF[key]
    Y['C'] = Y['C']-Y_UC
    Y['H2O'] = Y_H2O
    Y['ASH'] = Y_ASH
    Y['UC'] = Y_UC
    return Y

def combustion_smb(Y_f, alpha=1.0, w_air=0.0):
    # Preliminaries
    M = {'C':12.011e-3,'H':1.00795e-3,'O':15.9995e-3,'N':14.0065e-3,'S':32.065e-3, \
        'H2O':cp.PropsSI('M','H2O'),'CO2':cp.PropsSI('M','CO2'), \
        'N2':cp.PropsSI('M','N2'),'SO2':cp.PropsSI('M','SO2'), \
        'O2':cp.PropsSI('M','O2')}    #[kg/mol]
    keylist = []
```

```
for i,k in enumerate(M):
    keylist.append(k)
M_air = 0.2101*M['O2']+(1-0.2101)*M['N2']    #[kg/mol]
# Check fuel mixture mass fractions
Y_f_tot = 0.0
for key in Y_f:
    Y_f_tot += Y_f[key]
if (abs(Y_f_tot-1) > 1e-3):
    message = 'Sum of fuel mixture mass fractions = ' + str(Y_f_tot)
    raise Exception(message)

# Check excess air ratio
if (alpha < 1.0):
    message = 'Excess air ratio must be greater than one'
    raise Exception(message)

# Molar amounts of reactants in fuel [mol/kg_f]
N_f = {}
N_f_tot = 0.0
for i in range(0,5):
    N_f[keylist[i]] = Y_f[keylist[i]]/M[keylist[i]]
    N_f_tot += N_f[keylist[i]]
N_f['H2O'] = Y_f['H2O']/M['H2O']
N_f['UC'] = Y_f['UC']/M['C']

# Molar amounts in combustion air [mol/kg_f]
N_ca = {}
N_st_O2 = N_f['C']+0.25*N_f['H']-0.5*N_f['O']+N_f['S']
N_st_N2 = 3.7619*N_st_O2
N_ca['O2'] = alpha*N_st_O2
N_ca['N2'] = 3.7619*alpha*N_st_O2
N_ca['H2O'] = 4.7619*alpha*w_air*(M_air/M['H2O'])*N_st_O2
N_ca_tot = N_ca['O2']+N_ca['N2']+N_ca['H2O']

# Molar amounts in flue gas [mol/kg_f]
N_fg = {}
N_fg['H2O'] = N_f['H2O']+0.5*N_f['H']+4.7619*alpha*w_air*(M_air/M['H2O'])*N_st_O2
N_fg['CO2'] = N_f['C']
N_fg['N2'] = 0.5*N_f['N']+3.7619*alpha*N_st_O2
N_fg['SO2'] = N_f['S']
N_fg['O2'] = (alpha-1)*N_st_O2
N_fg_tot = N_fg['H2O']+N_fg['CO2']+N_fg['N2']+N_fg['SO2']+N_fg['O2']

# Mol fractions in combustion air [mol/mol_fg]
X_ca = {}
X_ca_tot = 0.0
X_ca['O2'] = N_ca['O2']/N_ca_tot
X_ca['N2'] = N_ca['N2']/N_ca_tot
X_ca['H2O'] = N_ca['H2O']/N_ca_tot
X_ca_tot = X_ca['O2']+X_ca['N2']+X_ca['H2O']
if (abs(X_ca_tot-1) > 1e-3):
    message = 'Sum of combustion air mol fractions = ' + str(X_ca_tot)
    raise Exception(message)

# Molecular weight of combustion air [kg/mol]
M_ca = 0.0
M_ca = X_ca['O2']*M['O2']+X_ca['N2']*M['N2']+X_ca['H2O']*M['H2O']
```

```
    # Mol fractions in flue gas [mol/mol_fg]
    X_fg = {}
    X_fg_tot = 0.0
    for i in range(5,10):
        X_fg[keylist[i]] = N_fg[keylist[i]]/N_fg_tot
        X_fg_tot += X_fg[keylist[i]]
    if (abs(X_fg_tot-1) > 1e-3):
        message = 'Sum of combustion flue gas mol fractions = ' + str(X_fg_tot)
        raise Exception(message)

    # Molecular weight of flue gas [kg/mol]
    M_fg = 0.0
    for i in range(5,10):
        M_fg += X_fg[keylist[i]]*M[keylist[i]]

    # Mass fractions in combustion air [kg/kg_ca]
    Y_ca = {}
    Y_ca_tot = 0.0
    Y_ca['O2'] = X_ca['O2']*M['O2']/M_ca
    Y_ca['N2'] = X_ca['N2']*M['N2']/M_ca
    Y_ca['H2O'] = X_ca['H2O']*M['H2O']/M_ca
    Y_ca_tot = Y_ca['O2']+Y_ca['N2']+Y_ca['H2O']
    if (abs(Y_ca_tot-1) > 1e-3):
        message = 'Sum of combustion air mass fractions = ' + str(Y_ca_tot)
        raise Exception(message)

    # Mass fractions in flue gas [kg/kg_fg]
    Y_fg = {}
    Y_fg_tot = 0.0
    for i in range(5,10):
        Y_fg[keylist[i]] = X_fg[keylist[i]]*M[keylist[i]]/M_fg
        Y_fg_tot += Y_fg[keylist[i]]
    if (abs(Y_fg_tot-1) > 1e-3):
        message = 'Sum of flue gas mass fractions = ' + str(Y_fg_tot)
        raise Exception(message)

    # Mass flow ratios [kg/kg_f]
    TAR = N_st_O2*M['O2']+N_st_N2*M['N2']
    DAR = N_ca['O2']*M['O2']+N_ca['N2']*M['N2']
    HAR = N_ca['O2']*M['O2']+N_ca['N2']*M['N2']+N_ca['H2O']*M['H2O']
    FGR = N_fg['H2O']*M['H2O']+N_fg['CO2']*M['CO2']+N_fg['N2']*M['N2']+ \
          N_fg['SO2']*M['SO2']+N_fg['O2']*M['O2']

    return Y_ca, Y_fg, X_ca, X_fg, TAR, DAR, HAR, FGR, N_f, N_ca, N_fg

def HHV_solid_fuel(N_f, N_fg):
    ''' Calculate the theoretical Higher Heating Value of any solid fuel '''
    HHV = N_f['H']*gmp.h_f0_H+N_f['O']*gmp.h_f0_O+N_f['N']*gmp.h_f0_N \
          -0.5*N_f['H']*gmp.h_f0_H2O-(N_fg['CO2']+N_f['UC'])*gmp.h_f0_CO2- \
          N_fg['SO2']*gmp.h_f0_SO2
    return HHV
def HHV_fuel(N_f, M_fuel, h_f0_fuel, N_fg):
    ''' Calculate the Higher Heating Value of any non-solid fuel '''
    N_f_fuel = 1.0/M_fuel
    HHV = N_f_fuel*h_f0_fuel-0.5*N_f['H']*gmp.h_f0_H2O-N_fg['CO2']*gmp.h_f0_CO2
    return HHV

HHV_C = 32.763e6 #[J/kg]
```

```
def aft(m_dot_f, m_dot_pa, m_dot_sa, m_dot_da, m_dot_fg, m_dot_fa, m_dot_ba, HHV, \
        Y_f, Y_ca, Y_fg, p, T_f, T_pa, T_sa, T_da, T_ba, cp_f=710, cp_ash=710):
    # HHV of carbon
    HHV_C = 32.763e6 #[J/kg]

    # Enthalpies for sensible energy flows
    h_f = cp_f*(T_f-(25+kel))
    h_pa = gmp.enthalpy(Y_ca, T=T_pa, p=p)
    h_sa = gmp.enthalpy(Y_ca, T=T_sa, p=p)
    h_da = gmp.enthalpy(Y_ca, T=T_da, p=p)
    h_ba = cp_ash*(T_ba-(25+kel))

    def energy_balance(T_aft):

        # Enthalpies at T_aft
        h_fg_aft = gmp.enthalpy(Y_fg, T=T_aft[0], p=p)
        h_fa_aft = cp_ash*(T_aft[0]-(25+kel))

        # Energy balance
        LHS = m_dot_fg*h_fg_aft+m_dot_fa*h_fa_aft+m_dot_ba*h_ba
        RHS = m_dot_f*(h_f+HHV-Y_f['UC']*HHV_C)+m_dot_pa*h_pa+m_dot_sa*h_sa+ \
              m_dot_da*h_da

        return [LHS-RHS]

    # Solve energy balance
    T_aft = fsolve(energy_balance,np.array([1500+kel]))[0]

    # Post processing
    h_fg_aft = gmp.enthalpy(Y_fg, T=T_aft, p=p)
    h_fa_aft = cp_ash*(T_aft-(25+kel))

    return T_aft, h_fg_aft

def fg_mass_balance(m_dot_fgi, m_dot_ia, Y_fgi, Y_ia):
    # Overall mass balance
    m_dot_fge = m_dot_fgi+m_dot_ia

    # Species mass balance
    Y_fge = {}
    Y_fge['H2O'] = (m_dot_fgi*Y_fgi['H2O']+m_dot_ia*Y_ia['H2O'])/m_dot_fge
    Y_fge['CO2'] = (m_dot_fgi*Y_fgi['CO2'])/m_dot_fge
    Y_fge['N2'] = (m_dot_fgi*Y_fgi['N2']+m_dot_ia*Y_ia['N2'])/m_dot_fge
    Y_fge['SO2'] = (m_dot_fgi*Y_fgi['SO2'])/m_dot_fge
    Y_fge['O2'] = (m_dot_fgi*Y_fgi['O2']+m_dot_ia*Y_ia['O2'])/m_dot_fge

    # Check mass balance
    Y_fge_tot = 0.0
    for key in Y_fge:
        Y_fge_tot += Y_fge[key]
    if (abs(Y_fge_tot-1) > 1e-3):
        message = 'Sum of flue gas outlet mass fractions = ' + str(Y_fge_tot)
        raise Exception(message)

    return m_dot_fge, Y_fge
```

```
# Constants for furnace analysis
wall_fouling_factor = {'gas_oil':0.65, "heavy_oil":0.55, "coal":0.4, \
    "grate_fuels":0.6, "other_fuels":0.2, "open_pitch":0.2, "refractory":0.1}
fly_ash_diameter = {'tubular_ball_mill':13e-6, 'hammer_mill':16e-6, \
    'grate_firing':20e-6}   #[m]
coke_constant_1 = {'anthracite':1.0, 'other_coals':0.5}
coke_constant_2 = {'suspension_firing':0.1, 'grate_firing':0.03}
rho_ash = 2200  # [kg/m^3]
cp_ash = 710    #[J/kgK]
sigma_sb = 5.6704e-8    #[W/m2·K^4]
A_prime = {'gas_fuel_oil':0.54,'high_reactive_coal':0.59,'low_reactive_coal':0.56, \
        'grate_stoker':0.59}
B_prime = {'gas_fuel_oil':0.20,'high_reactive_coal':0.50,'low_reactive_coal':0.50, \
        'grate_stoker':0.50}

def angular_coefficient_tubewalls(b, d):
    return 1.15484*pow(b/d,-0.850475)

def non_uniformity_factor(fuel, h_hf):
    # Check correct fuel type
    if (fuel != 'lignite') and (fuel != 'coal') and (fuel != 'oilgas'):
        message = fuel + ' is not a valid fuel in non_uniformity_factor function'
        raise Exception(message)

    c_1 = {'lignite':1.4728, 'coal':1.9745, 'oilgas':1.5214}
    c_2 = {'lignite':1.3324, 'coal':1.8080, 'oilgas':2.9035}
    lam = {'lignite':1.7131, 'coal':0.9821, 'oilgas':1.5750}
    mu = {'lignite':0.2275, 'coal':0.2275, 'oilgas':0.2675}
    sig = {'lignite':0.1907, 'coal':0.2371, 'oilgas':0.1968}

    return c_1[fuel]*(lam[fuel]/2)*np.exp((lam[fuel]/2)*(2*pow(mu[fuel],c_2[fuel])+ \
        lam[fuel]*pow(sig[fuel],2)-2*h_hf))\*sp.special.erfc((pow(mu[fuel], \
        c_2[fuel])+lam[fuel]*pow(sig[fuel],2)-h_hf)/(1.4142*sig[fuel]))

def reradiation_coefficient(fuel, T_fe):
    ''' Reradiation coefficient for platen (for non-platen: beta = 1.0) '''
    # Check correct fuel type
    if (fuel != 'solid_fuel') and (fuel != 'heavy_oil'):
        message = fuel + ' is not a valid fuel in reradiation_coefficient function'
        raise Exception(message)

    if fuel == 'solid_fuel':
        if T_fe < 1060+kel:
            beta = 1.0
        else:
            beta = -2.18793e-6*pow(T_fe-kel,2)+4.36338e-3*(T_fe-kel)-1.16738
    elif fuel == 'heavy_oil':
        if T_fe < 950+kel:
            beta = 0.9
        else:
            beta = -1.43393e-6*pow(T_fe-kel,2)+2.53702e-3*(T_fe-kel)-0.217727

    return beta

def gurvich_gas_temperature(T_aft, T_fe, M_flame):
    T_g = T_aft*pow((pow(M_flame*T_fe/T_aft,5/3))/(pow(1-T_fe/T_aft,2/3)),0.25)

    return T_g
```

```
def highash_fluegas_emissivity(m_dot_fg, m_dot_fa, Y_fg, T, p_fur, S_fur, d_fa, \
      x1_coke, x2_coke):
    # Coefficients for wsggm of gas emissivity
    b_0 = np.array([0.0, 0.130, 0.595, 0.275])
    b_1 = np.array([0.0, 0.265, -0.150, -0.115])   # [1/1000K]
    k_gi = np.array([0.0, 0.0, 0.824, 25.907])  # [1/(m*bar)]

    # Flue gas mol fractions
    X_fg = gmp.mole_fractions(Y_fg)
    A_p = (3/2)/(rho_ash*d_fa)  #[m^2/kg_p]
    L_p = m_dot_fa/(m_dot_fg/gmp.density(Y_fg, T=T, p=p_fur))   #[kg_p/m^3]

    # Gas emissiviy
    X_fg_H2O = X_fg['H2O']
    X_fg_CO2 = X_fg['CO2']
    epsilon_gi = np.zeros(4)
    epsilon_gi = (b_0+b_1*T/1000)*(1-np.exp(-k_gi*((X_fg_H2O+X_fg_CO2)*(p_fur/1e5))
                *S_fur))  # p_fur needed in [bar]
    epsilon_g = np.sum(epsilon_gi)
    K_g = -np.log(1-epsilon_g)/S_fur
    Q_abs = (0.275*(1e6*d_fa)**0.298)-(0.305)
    Q_bsc = ((6.2188e-3)-(1.0492e-2*(1e6*d_fa))+(7.287e-3*(1e6*d_fa)**2)-(2.1925e-5*\
     (1e6*d_fa)**3))/((1.851e-1)-(2.0405e-3*(1e6*d_fa)**2)+(6.254e-4*(1e6*d_fa)**3))
    gamma = (1+2*Q_bsc/Q_abs)**0.5
    beta = (gamma-1)/(gamma+1)
    k_coke = 10.2*x1_coke*x2_coke   # [1/(m*MPa)]
    K_coke = k_coke*(p_fur/1e6) #[1/m]
    phi_gp =(K_g+K_coke+Q_abs*A_p*L_p)*S_fur*gamma
    epsilon_gp = (1-beta)*((1-np.exp(-phi_gp))/(1+beta*np.exp(-phi_gp)))

    return epsilon_gp

def conv_fluegas_emissivity(m_dot_fg, m_dot_fa, Y_fg, T, p_fur, S_fur, d_fa, \
      x1_coke, x2_coke):
    # Flue gas mol fractions
    X_fg = gmp.mole_fractions(Y_fg)

    # Gas emissiviy
    X_fg_H2O = X_fg['H2O']
    X_fg_tri = X_fg['CO2']+X_fg['H2O']+X_fg['SO2']
    k_g = ((7.8+16*X_fg_H2O)/(3.16*pow(X_fg_tri*(p_fur/1e6)*S_fur,0.5))-1)* \
        (1-0.37*T/1000)
    R_fg = gmp.gas_constant(Y_fg)
    rho_fg_n = 101325/(R_fg*(0.0+kel))
    k_fa = (48350*rho_fg_n)/pow((T**2)*((d_fa/1e-6)**2),1/3)
    mu_fa = m_dot_fa/(m_dot_fg+m_dot_fa)
    k_coke = 10.2   #[1/(m*MPa)]
    k_fur = k_g*X_fg_tri+k_fa*mu_fa+k_coke*x1_coke*x2_coke
    epsilon_gp = 1-np.exp(-k_fur*(p_fur/1e6)*S_fur)

    return epsilon_gp

def fet_projected(m_dot_f, m_dot_fg, m_dot_fa, Y_fg, f_radloss, T_aft, p_fur, \
    A_fur, Vol_fur, xi_fur, x_fur, d_fa, x1_coke, x2_coke, M_flame, high_ash=1):
    # Initialize local variables that are set in the sub-function so that they will
      be available locally

    epsilon_fur = epsilon_g = h_fg_aft = h_fg_fe = h_fa_aft = h_fa_fet = Q_dot_fg \
     = Q_dot_rf = Q_dot_ww = Q_dot_fe = Q_dot_loss_fur = vc = phi_fur = Bo = 0.0
```

```
    # Total furnace radiative projected area
A_rad_fur = 0.0
for key in x_fur:
    A_rad_fur += x_fur[key]*A_fur[key]

# Furnace efficiency factor - psi
psi_fur = {}
psi_eff_fur = 0.0
for key in x_fur:
    psi_fur[key] = x_fur[key]*xi_fur[key]
    psi_eff_fur += (psi_fur[key]*A_fur[key])/A_rad_fur

# Furnace mean beam length
A_tot_fur = 0.0
for key in A_fur:
    A_tot_fur += A_fur[key]
S_fur = 3.6*Vol_fur/A_tot_fur

# Guess values
T_fe = 0.7*T_aft

# Boilerplating
x = np.array([T_fe])

def fet_gurvich(x):

    # Set variables as non-local so that they will be available in outer function
    nonlocal epsilon_fur, epsilon_g, h_fg_aft, h_fg_fe, h_fa_aft, h_fa_fet, \
        Q_dot_fg, Q_dot_rf, Q_dot_ww, Q_dot_fe, Q_dot_loss_fur, vc, phi_fur, Bo

    # Boilerplating
    T_fe = x[0]

    # Preliminaries
    f = np.zeros(1)

    # Furnace gas emissivity
    if high_ash == 1:
        epsilon_g = highash_fluegas_emissivity(m_dot_fg, m_dot_fa, Y_fg, T_fe, \
                p_fur, S_fur, d_fa, x1_coke, x2_coke)
    else:
        epsilon_g = conv_fluegas_emissivity(m_dot_fg, m_dot_fa, Y_fg, T_fe, \
                p_fur, S_fur, d_fa, x1_coke, x2_coke)
    epsilon_fur = 1/(1+psi_eff_fur*(1/epsilon_g-1))

    # Boltzmann number
    h_fg_aft = gmp.enthalpy(Y_fg, T=T_aft, p=p_fur)
    h_fg_fe = gmp.enthalpy(Y_fg, T=T_fe, p=p_fur)
    h_fa_aft = cp_ash*(T_aft-(25+kel))
    h_fa_fet = cp_ash*(T_fe-(25+kel))
    vc = (m_dot_fg*(h_fg_aft-h_fg_fe)+m_dot_fa*(h_fa_aft-h_fa_fet))/(m_dot_f* \
        (T_aft-T_fe))
    phi_fur = 1-f_radloss
    Bo = (phi_fur*m_dot_f*vc)/(psi_eff_fur*A_rad_fur*sigma_sb*np.power(T_aft,3))

    f[0] = T_fe-T_aft/(M_flame*(np.power(epsilon_fur/Bo,0.6))+1)

    return f
```

```
    # Solve for T_fe
    x = fsolve(fet_gurvich, x)

    # Boilerplating
    T_fe = x[0]

    if high_ash == 1:
        epsilon_g = highash_fluegas_emissivity(m_dot_fg, m_dot_fa, Y_fg, T_fe, \
            p_fur, S_fur, d_fa, x1_coke, x2_coke)
    else:
        epsilon_g = conv_fluegas_emissivity(m_dot_fg, m_dot_fa, Y_fg, T_fe, p_fur, \
            S_fur, d_fa, x1_coke, x2_coke)

    # Average gas temperature
    T_g = gurvich_gas_temperature(T_aft, T_fe, M_flame)
    T_g = np.power((phi_fur*m_dot_f*vc*(T_aft-T_fe))/(psi_eff_fur*A_rad_fur* \
        sigma_sb*epsilon_fur),0.25)

    # Furnace heat transfer
    Q_dot_fg = m_dot_fg*(h_fg_aft-h_fg_fe)+m_dot_fa*(h_fa_aft-h_fa_fet)
    Q_dot_fur = phi_fur*Q_dot_fg
    Q_dot_loss_fur = (1-phi_fur)*Q_dot_fg
    q_fur = Q_dot_fur/A_rad_fur     # Based on total radiation area

    # Heat transfer through furnace exit
    eta_fe = non_uniformity_factor('coal', 1.0)     # Positioned at furnace outlet
    beta_fe = reradiation_coefficient('solid_fuel', T_fe)  # Taken at T_fe
    Q_dot_fe = beta_fe*eta_fe*(psi_fur['furnaceexit']*A_fur['furnaceexit'])/ \
            (psi_eff_fur*A_rad_fur)*Q_dot_fur

    # Heat transfer to refractory
    eta_rf = non_uniformity_factor('coal', 0.0) # Assumed near bottom of furnace
    Q_dot_rf = eta_rf*(psi_fur['refractory']*A_fur['refractory'])/ \
            (psi_eff_fur*A_rad_fur)*Q_dot_fur

    # Heat transfer to tubewalls
    Q_dot_ww = Q_dot_fur-(Q_dot_fe+Q_dot_rf)

    return T_fe, h_fg_fe, T_g, Q_dot_fg, Q_dot_fur, Q_dot_ww, Q_dot_rf, Q_dot_fe, \
        Q_dot_loss_fur, q_fur, A_rad_fur, A_tot_fur

def bypass_direct_radiation(N_l, S_l, S_t, epsilon_g, beta, Q_dot_ri):
    b_l = (N_l-1)*S_l
    phi = pow((b_l/S_t)**2+1,0.5)-b_l/S_t
    Q_dot_bp = (Q_dot_ri*(1-epsilon_g)*phi)/beta

    return Q_dot_bp

def back_direct_radiation(W_e, H_e, epsilon_g, T_fg):
    A_e_fg = W_e*H_e
    Q_dot_back = 0.5*epsilon_g*sigma_sb*A_e_fg*pow(T_fg,4)

    return Q_dot_back

def Gnielinski_external_tube_bank(d_OD, S_t, S_l, N_l, A_open, inline, theta, \
        m_dot, rho, cp, mu, k, T_g, T_w):
    a_Gni = S_t/d_OD
    b_Gni = S_l/d_OD
```

```
    if b_Gni < 1:
        psi = 1-math.pi/(4*a_Gni*b_Gni)
    else:
        psi = 1-math.pi/(4*a_Gni)
    L_Gni = math.pi/2*d_OD
    v = m_dot/(rho*A_open)
    v_psi = v/psi
    Re = ((rho*v*L_Gni)/(psi*mu))*math.sin(theta*(2*math.pi/360))
    Pr = (cp*mu)/k
    Nu_l_lam = 0.664*pow(Re,0.5)*pow(Pr,1/3)
    Nu_l_turb = (0.037*pow(Re,0.8)*Pr)/(1+2.443*pow(Re,-0.1)*(pow(Pr,2/3)-1))
    Nu_l0 = 0.3+pow(pow(Nu_l_lam,2)+pow(Nu_l_turb,2),0.5)
    if inline == 1.0:
        f_A = 1+(0.7*(b_Gni/a_Gni-0.3))/(pow(psi,1.5)*pow((b_Gni/a_Gni+0.7),2))
    else:
        f_A = 1+2/(3*b_Gni)
    Nu_0bundle = ((1+(N_l-1))/N_l)*f_A*Nu_l0
    Nu = pow(T_g/T_w,0.12)*Nu_0bundle
    htc = Nu*(k/L_Gni)

    return v, v_psi, Re, Pr, Nu, htc

def Gnielinski_internal_tube(d_ID, roughness, A_ff, m_dot, rho, cp, mu, k):
    v = m_dot/(rho*A_ff)
    Re = (rho*v*d_ID)/mu
    Pr = (cp*mu)/k
    f = pow(0.79*math.log(Re-1.64),-2)
    Nu = ((f/8)*(Re-1000)*Pr)/(1+12.7*pow(f/8,0.5)*(pow(Pr,2/3)-1))
    htc = Nu*(k/d_ID)

    return v, Re, Pr, Nu, htc, f

def effective_radiation_htc(epsilon_w, epsilon_g, T_g, T_w):
    htc_rad = (1+epsilon_w)/2*epsilon_g*sigma_sb*((pow(T_g,4)-pow(T_w,4))/(T_g-T_w))

    return htc_rad

def radiative_convective_hx(m_dot_fgi, Y_fgi, m_dot_fa, p_fgi, p_fge, T_fgi, T_fge, \
      m_dot_ia, Y_ia, T_ia, Q_dot_ri, m_dot_st, p_sti, p_ste, h_sti, h_ste, W_i, \
      H_i, W_e, H_e, L_duct, R_dep_o, R_dep_i, epsilon_w, k_tube, theta, N_t, N_l, \
      S_t, S_l, pp, L_avg, d_OD, t_wall, roughness, eta_fin_o, AfoverA_o, \
      eta_fin_i, AfoverA_i, inline, platen, psi, beta, A_wall, T_wall, A_roof, \
      T_roof, d_fa, x1_coke, x2_coke, high_ash=1):

    # Initialize local variables that are set in the sub-function so that they will
      be available locally
    T_fo = p_fg = T_fg = rho_fg = mu_fg = cp_fg = k_fg = p_st = T_st = rho_st = \
      mu_st = cp_st = k_st = S_mbl = epsilon_g = Q_dot_bp = Q_dot_abs = Q_dot_back \
      = W_ave = H_ave = A_open = v_fg = v_psi_fg = Re_fg = Pr_fg = Nu_fg = \
      htc_conv_fg = htc_rad_fg = htc_fg = A_ho = eta_o = UA_fg = NTU_fg = \
      epsilon_hxfg = d_ID = A_ff = v_st = Re_st = Pr_st = Nu_st = htc_st = f_st \
      = A_hi = eta_i = R_fi = R_fo = R_w = UA_st = NTU_st = epsilon_hxst = Q_dot_ex \
      = Q_dot_re = Q_dot_wall = Q_dot_roof = Q_dot_fg = Q_dot_st = dp_fg = dp_st = \
      T_wo = T_wi = T_fi = 0.0

    # Overall mass balance (Ingress air assumed to be at inlet)
    m_dot_fge, Y_fge = fg_mass_balance(m_dot_fgi, m_dot_ia, Y_fgi, Y_ia)
```

```
    # Check mass balance
    Y_fge_tot = 0.0
    for key in Y_fge:
        Y_fge_tot += Y_fge[key]
    if abs(Y_fge_tot-1) > 1e-3:
        message ='Sum of flue gas outlet mass fractions in heat exchanger model = '\
                + str(Y_fge_tot)
        raise Exception(message)

    # Input enhalpies
    h_fgi = gmp.enthalpy(Y_fgi,T=T_fgi,p=p_fgi)
    h_fai = cp_ash*(T_fgi-(25+kel))
    h_ia = gmp.enthalpy(Y_ia, T=T_ia, p=p_fgi)
    T_sti = cp.PropsSI('T','H',h_sti,'P',p_sti,'Water')

    # Output enhalpies
    h_fge = gmp.enthalpy(Y_fge,T=T_fge,p=p_fge)
    h_fae = cp_ash*(T_fge-(25+kel))
    T_ste = cp.PropsSI('T','H',h_ste,'P',p_ste,'Water')

    # Flue gas side average properties
    p_fg = 0.5*(p_fgi+p_fge)
    T_fg = 0.5*(T_fgi+T_fge)
    rho_fg = gmp.density(Y_fge, T_fg, p=p_fg)
    mu_fg = gmp.viscosity(Y_fge, T_fg, p=p_fg)
    cp_fg = gmp.specific_heat_pressure(Y_fge, T_fg, p=p_fg)
    k_fg = gmp.conductivity(Y_fge, T_fg, p=p_fg)

    # Steam side average properties
    p_st = 0.5*(p_sti+p_ste)
    h_st = 0.5*(h_sti+h_ste)
    T_st = cp.PropsSI('T','H',h_st,'P',p_st,'Water')
    rho_st = cp.PropsSI('D','H',h_st,'P',p_st,'Water')
    mu_st = cp.PropsSI('V','H',h_st,'P',p_st,'Water')
    q = cp.PropsSI('Q','H',h_st,'P',p_st,'Water')
    if q > 0.0 and q < 1.0:
        cp_st = 1e10
    else:
        cp_st = cp.PropsSI('CPMASS','H',h_st,'P',p_st,'Water')
    k_st = cp.PropsSI('L','H',h_st,'P',p_st,'Water')
    if cp_st < 0:
        print(h_st, p_st, cp_st, k_st, mu_st)

    # Flue gas side geometry
    W_ave = 0.5*(W_i+W_e)
    H_ave = 0.5*(H_i+H_e)
    A_open = W_ave*H_ave
    if platen == 1.0:
        A_ho = 2*H_ave*d_OD*N_l*N_t     # NB changed to H_ave from L_avg
    else:
        A_ho = math.pi*d_OD*(L_avg*N_l*N_t)

    # Steam side geometry
    d_ID = d_OD-2*t_wall
    A_ff = N_t*pp*math.pi/4*d_ID**2    # Free flow area of all the tubes
    A_hi = math.pi*d_ID*(L_avg*N_l*N_t)

    # Outlet initial values
    T_fo = 0.5*(T_fg+T_sti)
```

```
    # Boilerplating
    x = np.array([T_fo])

    def balance_equations(x):
        ''' Internal function to solve balance equations together with component
        characteristics '''

        # Set variables as non-local so that they will be available in outer function
        nonlocal T_fge, h_fae, T_ste, p_fg, T_fg, rho_fg, mu_fg, cp_fg, k_fg, \
          p_st, T_st, rho_st, mu_st, cp_st, k_st, S_mbl, epsilon_g, Q_dot_bp, \
           Q_dot_abs, Q_dot_back, W_ave, H_ave, A_open, v_fg, v_psi_fg, Re_fg, \
            Pr_fg, Nu_fg, htc_conv_fg, htc_rad_fg, htc_fg, A_ho, eta_o, UA_fg, \
             NTU_fg, epsilon_hxfg, d_ID, A_ff, v_st, Re_st, Pr_st, Nu_st, htc_st, \
              f_st, A_hi, eta_i, R_fi, R_fo, R_w, UA_st, NTU_st, epsilon_hxst, \
               Q_dot_ex, Q_dot_re, Q_dot_wall, Q_dot_roof, Q_dot_fg, Q_dot_st, \
                dp_fg, dp_st, T_wo, T_wi, T_fi

        # Preliminaries
        f = np.zeros(1)

        # Boilerplating
        T_fo = x[0]

        # Effective gas emissivity
        S_mbl = 0.9*d_OD*((4*S_t*S_l)/(math.pi*d_OD**2)-1)
        if high_ash == 1:
            epsilon_g = highash_fluegas_emissivity(m_dot_fge, m_dot_fa, Y_fge, \
                T_fg, p_fg, S_mbl, d_fa, x1_coke, x2_coke)
        else:
            epsilon_g = conv_fluegas_emissivity(m_dot_fge, m_dot_fa, Y_fge, T_fg, \
                p_fg, S_mbl, d_fa, x1_coke, x2_coke)

        # Bypass direct radiation heat transfer
        Q_dot_bp = bypass_direct_radiation(N_l, S_l, S_t, epsilon_g, beta, Q_dot_ri)

        # Absorbed direct radiation heat ransfer
        Q_dot_abs = Q_dot_ri-Q_dot_bp

        # Back direct radiation from gas volume to next surface
        Q_dot_back = back_direct_radiation(W_e, H_e, epsilon_g, T_fg)

        # Gnielinski correlation for external tube bank htc
        v_fg, v_psi_fg, Re_fg, Pr_fg, Nu_fg, htc_conv_fg = \
         Gnielinski_external_tube_bank(d_OD, S_t, S_l, N_l, A_open, inline, theta,\
            m_dot_fge, rho_fg, cp_fg, mu_fg, k_fg, T_fg, T_fo)

        # Effective external radiation htc
        htc_rad_fg = effective_radiation_htc(epsilon_w, epsilon_g, T_fg, T_fo)

        # Total external htc and effectiveness
        if platen == 1.0:
            htc_fg = (math.pi*d_OD)/(2*S_l)*htc_conv_fg+htc_rad_fg
        else:
            htc_fg = htc_conv_fg+htc_rad_fg
        eta_o = 1-AfoverA_o*(1-eta_fin_o)
        UA_fg = psi*eta_o*htc_fg*A_ho
        NTU_fg = UA_fg/(m_dot_fge*cp_fg)
        epsilon_hxfg = 1-math.exp(-NTU_fg)
```

```
    # Gnielinski correlation for internal tube flow htc
    v_st, Re_st, Pr_st, Nu_st, htc_st, f_st = \
        Gnielinski_internal_tube(d_ID, roughness, A_ff, m_dot_st, rho_st, \
          cp_st, mu_st, k_st)

    # Internal htc and effectiveness
    eta_i = 1-AfoverA_i*(1-eta_fin_i)
    R_fi = R_dep_i
    R_fo = R_dep_o
    R_w = (math.log(d_OD/d_ID))/(2*math.pi*k_tube*(L_avg*N_l*N_t))
    UA_st = psi/(1/(eta_i*htc_st*A_hi)+R_fi/(eta_i*A_hi)+R_w+R_fo/(eta_o*A_ho))
    NTU_st = UA_st/(m_dot_st*cp_st)
    if NTU_st < 0:
        print(UA_st, m_dot_st, cp_st)
    epsilon_hxst = 1-math.exp(-NTU_st)

    # Heat transfer rates
    Q_dot_ex = epsilon_hxfg*m_dot_fge*cp_fg*(T_fgi-T_fo)
    Q_dot_re = Q_dot_bp+Q_dot_back
    Q_dot_wall = htc_fg*A_wall*(T_fg-T_wall)
    Q_dot_roof = htc_fg*A_roof*(T_fg-T_roof)
    Q_dot_fg = -(Q_dot_ex+Q_dot_back+Q_dot_wall+Q_dot_roof)
    Q_dot_st = epsilon_hxst*m_dot_st*cp_st*(T_fo-T_sti)

    # Pressure drops
    dp_fg = 0.0     # !!!
    dp_st = 0.0     # !!!

    # Detail wall temperatures
    T_wo = T_fo-R_fo/(eta_o*A_ho)*Q_dot_st
    T_wi = T_wo-R_w*Q_dot_st
    T_fi = T_wi-R_fi/(eta_i*A_hi)*Q_dot_st

    # Energy and momentum balance equations
    f[0] = (Q_dot_st)-(Q_dot_ex+Q_dot_abs)

    return f

# Solve energy and momentum balance equations with hx component characteristics
x = fsolve(balance_equations, x)

# Boilerplating
T_fo = x[0]

return Q_dot_fg, Q_dot_re, Q_dot_wall, Q_dot_roof, Q_dot_st, dp_fg, dp_st, \
       T_fo, T_wo, T_wi, T_fi
```