IMPLEMENTATION OF THE ECRAD RADIATION MODULE USING PHYSICS INFORMED MACHINE LEARNING

Eduardo F. Miranda

Exame de Proposta de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Dr. Stephan Stephany e Dr. Roberto Pinto Souto aprovada em 27 de agosto de 2024.

URL of the original document:
<http://urlib.net/xx/yy>

INPE
São José dos Campos
2024

IMPLEMENTATION OF THE ECRAD RADIATION MODULE USING PHYSICS INFORMED MACHINE LEARNING

Eduardo F. Miranda

Exame de Proposta de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Dr. Stephan Stephany e Dr. Roberto Pinto Souto aprovada em 27 de agosto de 2024.

URL of the original document:
<http://urlib.net/xx/yy>

INPE
São José dos Campos
2024

Informar aqui sobre fontes financiadoras (a modificação desta linha deve ser feita no arquivo publicacao.tex).

ABSTRACT

This thesis demonstrates the use of a Machine Learning (ML) approach to optimize part of the ecRad radiation module employed in numerical weather and climate models, which is implemented using a numerical algorithm and is very computationally demanding. The ecRad radiation module is employed in an operational model of the European Centre for Medium-Range Weather Forecasts (ECMWF), but in this work, it will be developed and tested as a stand-alone implementation. The gas-optical scheme of the radiation module would be replaced by a ML-based implementation, following the current trend applied to model standard microphysics modules. In particular, it is intended to explore Physics-Informed Machine Learning (PIML) alternatives for the gas-optical scheme, starting with a former class of PIML methods, the Physics-Informed Neural Networks (PINNs). However, preliminary work has already been carried out for the gas-optical scheme using a ML approach implemented by a Deep Neural Network (DNN), reproducing the work of some authors. An incremental approach is then proposed for the thesis, starting with a non-PIML DNN, followed by a PINN implementation, and then exploring other PIML implementations. A previous of this thesis work applied a PINN approach to a simpler problem related to the one-dimensional Burgers' Equation, modeled by a partial differential equation. Test results and analyses are presented here for both previous works. An additional point is how to integrate the ML code, implemented in the Python environment and its associated libraries, with the ecRad radiation module written in Fortran 90.

Keywords: Physics-informed Machine Learning. Radiation Module. Deep Neural Network. Physics-informed Neural Network. High Performance Computing.

IMPLEMENTAÇÃO DO MÓDULO DE RADIAÇÃO ECRAD USANDO APRENDIZADO DE MÁQUINA INFORMADO PELA FÍSICA

RESUMO

Esta tese demonstra o uso de uma abordagem de aprendizado de máquina para otimizar parte do módulo de radiação ecRad empregado em modelos numéricos de previsão de tempo de clima, o qual emprega um algoritmo numérico que requer muito processamento. O módulo de radiação ecRad é usado num modelo operacional do European Centre for Medium-Range Weather Forecasts (ECMWF), mas neste trabalho será desenvolvido e testado como uma implementação independente. O esquema gás-ótico do módulo de radiação será substituído por uma implementação baseada em aprendizado de máquina, seguindo a tendência atual aplicada a módulos padrão da microfísica de modelos. Em particular, pretende-se explorar alternativas de Physics-Informed Machine Learning (PIML) para o esquema gás-ótico, começando com uma classe inicialmente proposta de métodos PIML, as Physics-Informed Neural Networks (PINNs). Entretanto, um estudo preliminar já foi desenvolvido para o esquema gás-ótico usando uma abordagem de aprendizado de máquina implementado por uma rede neural profunda (Deep Neural Network - DNN), reproduzindo o trabalho de alguns autores. Uma abordagem incremental é então proposta para a tese, começando com a DNN, fora do escopo de PIML, seguida de uma abordagem PINN e depois explorando outras classes de métodos PIML. Outro trabalho preliminar da tese aplicou uma abordagem PINN para um problema mais simples relacionado à equação unidimensional de Burgers, que é modelada por uma equação diferencial parcial. Resultados de testes e análises são apresentados aqui para ambos esses trabalhos preliminares. Um ponto adicional é como integrar o código de aprendizado de máquina, implementado no ambiente Python e suas bibliotecas associadas, com o módulo de radiação ecRad escrito em Fortran 90.

Keywords: Aprendizado de Máquina Informado pela Física. Módulo de Radiação. Rede Neural Profunda. Rede Neural Informada pela Física. Processamento de Alto Desempenho.

LIST OF FIGURES

Page

2.1 Schematic showing an example structure and loss function, for the PINN approach. The symbol θ represents the optimization algorithm. The ϵ represents the smallest threshold error in the optimization algorithm.
Source: Adapted from Meng et al. (2020). 14
2.2 Schematic illustrating the data flow in a structure of a conventional numeric gas optics scheme. "T" is temperature, "p" is pressure, "Gas" represents relative abundance, "Band" corresponds to the LW and SW radiation bands, "g-points" correspond to "k-terms" of the correlated k-distribution method, “Level” corresponds to atmospheric layers or vertical grid points within a column representing altitude or pressure, and "look-up tables" (LUT) correspond to the look-up table kernels of the RRTMGP model. The Figure 2.3 complements this Figure.
Source: Adapted from Ukkonen and Hogan (2023). 22
2.3 Simplified schematic, with only one hidden layer, illustrating the data flow in a gas optics DNN framework. "W" represents weight, and "b" represents bias. The other terms are described in the Figure 2.2.
Source: Adapted from Ukkonen and Hogan (2023). 23

3.1 PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 128x64. 30
3.2 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 128x64). 30
3.3 PINN predicted solution $u(t, x)$ for viscosity $0.1/\pi$ and problem size 128x64. 30
3.4 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 128x64). 30
3.5 PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 256x128. 31
3.6 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 256x128). 31
3.7 PINN predicted solution $u(t, x)$ for viscosity $0.1/\pi$ and problem size 256x128. 32

3.8 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 256x128). 32
3.9 PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 512x256. 33
3.10 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 512x256). 33
3.11 PINN predicted solution $u(t, x)$ for viscosity $0.1/\pi$ and problem size 512x256. 33
3.12 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 512x256). 33
3.13 PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 256x100 (black dots denote the 2,000 randomly assigned CPs). 35
3.14 Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 256x100). 35
3.15 Relative L2 error (%) in function of number of neurons and hidden layers. The simulation ran on the SDumont. 36
3.16 Processing times (seconds) in function of number of neurons and hidden layers. The simulation ran on the SDumont. 37
3.17 Relative L2 error (%) in function of number of neurons and dataset size. The number of hidden layers is set to 8. The simulation ran on the SDumont. 37
3.18 Processing times (seconds) in function of number of neurons and dataset size. The number of hidden layers is set to 8. The simulation ran on the SDumont. 39

LIST OF TABLES

Page

2.1 Network architectures used for different optical depths. Source: Adapted from Ukkonen et al. (2020). 21

3.1 Comparison of elapsed times (average of 3 runs) for the PINN and the 4 SINDy models for kinematic viscosity of the fluid of $0.01/\pi$, and for execution on the local machine (PC). 26

3.2 Comparison of elapsed times (average of 3 runs) for the PINN and the 4 SINDy models for kinematic viscosity of the fluid of $0.1/\pi$, and for execution on the local machine (PC). 27

3.3 Comparison of the results of the parameter discovery for the 1D Burgers' equation using PINN (in blue) and the 4 SINDy versions (kinematic viscosity of the fluid of $0.01/\pi$). 28

3.4 Comparison of the results of the parameter discovery for the 1D Burgers' equation using PINN (in blue) and the 4 SINDy versions (kinematic viscosity of the fluid of $0.1/\pi$). 28

3.5 Relative L2 errors and DNN training times for different number of neurons and hidden layers. On the color scale, the best values are highlighted in green. The simulation ran on the SDumont. 36

3.6 Relative L2 errors and DNN training times for different number of neurons and dataset size. The number of hidden layers is set to 8. On the color scale, the best values are highlighted in green. The simulation ran on the SDumont. 38

3.7 Prediction times for different number of neurons and hidden layers. On the color scale, the best values are highlighted in green. The simulation ran on the SDumont. 38

3.8 The performance analysis using the gprof tool of the offline ecRad example with the original numerical gas-optical scheme. The "routine" column is described in the Table 3.10. The remaining columns are the standard output of gprof. The total execution time is 1.55 s. Times are for one run. 40

3.9 The performance analysis using the gprof tool of the offline ecRad example with the DNN gas-optical scheme. The "routine" column is described in the Table 3.10. The remaining columns are the standard output of gprof. The total execution time is 1.50 s. Times are for one run. 40

3.10 Complement of the Tables 3.8 and 3.9 showing the names of the source files and the names of the routines (gprof standard). 41

CONTENTS

Page

1 INTRODUCTION 1
1.1 Physics-Informed Machine Learning (PIML) 1
1.2 Physics-Informed Neural Networks 2
1.3 ecRad Radiation Scheme 6
1.4 Objectives of the Thesis Research 7

2 METHODOLOGY 9
2.1 PINN Implementation for the 1D Burgers Equation 9
2.1.1 PINN Implementation 10
2.1.2 PINN Parameter Discovery 12
2.1.3 NM-based Parameter Discovery 15
2.2 DNN Preliminary Approach for ecRad 18
2.2.1 DNN-based Gas Optics Implementation 19
2.3 Computing Environment 23

3 RESULTS 25
3.1 PINN Implementation for the 1D Burgers Equation 25
3.1.1 Comparison of PINN and SINDy parameter discovery 25
3.1.1.1 Problem Size 128x64 29
3.1.1.2 Problem Size 256x128 31
3.1.1.3 Problem Size 512x256 32
3.1.2 Evaluation of PINN Hyperparameters and CP Size 34
3.1.2.1 Visual Assessment of PINN 34
3.1.2.2 Influence of the Layers and Neurons in the PINN 34
3.1.2.3 Influence of Neurons and the CP in the PINN 35
3.1.2.4 PINN Prediction Times 38
3.2 DNN-Based Gas-Optics Implementation 39

4 THESIS PROPOSAL 43
4.1 Work Plan 44
4.2 Schedule 45

REFERENCES 47

1 INTRODUCTION

This thesis aims to explore Physics-Informed Machine Learning (PIML) alternatives to replace part of the ecRad radiation numerical module, which is used by the global weather and climate model Integrated Forecasting System (IFS) of the European Centre for Medium-Range Weather Forecasts (ECMWF). The IFS model was developed and maintained jointly by the ECMWF, based in Reading, England, and Météo-France, based in Toulouse, France. PIML is described in section 1.1, while the initial class of Physics-Informed Neural Networks (PINN) methods is described in section 1.2. The ecRad radiation module appears in section 1.3. Former work in this thesis research included a PINN implementation for a test case, shown in this document. The objectives of this thesis are then shown in section 1.4.

1.1 Physics-Informed Machine Learning (PIML)

PIML is a set of methods and tools that systematically integrate Machine Learning (ML) algorithms with physical constraints and mathematical models developed in scientific and engineering domains. As opposed to purely data-driven methods, PIML models can be trained from additional information obtained by enforcing physical laws such as energy and mass conservation. More broadly, PIML models can include abstract properties and conditions such as stability, convexity, or invariance. The basic premise of PIML is that the integration of ML and physics can yield more effective, physically consistent, and data-efficient models. Recent advances in PIML for dynamical system modeling and control includes: (i) physics-informed learning for system identification; (ii) physics-informed learning for control; (iii) analysis and verification of PIML models; and (iv) physics-based digital twin, which consists of parameterized components whose physical properties can be altered to provide data from source systems that are identical to the target system (NGHIEM et al., 2023).

According to Karniadakis et al. (2021), despite great progress in simulating multiphysics problems using the numerical discretization of partial differential equations (PDEs), one still cannot seamlessly incorporate noisy data into existing algorithms, mesh generation remains complex, and high-dimensional problems governed by parameterized PDEs cannot be tackled. Moreover, solving inverse problems with hidden physics is often prohibitively expensive and requires different formulations and elaborate computer codes. In this context, ML emerges as a promising alternative, being a current research thread. However, training deep neural networks may require big data, which is not always available for scientific problems. Instead, such networks can be trained from additional information obtained by enforcing the physical laws, for example, at random points in the continuous space-time domain. Such physics-informed learning integrates noiseless or noisy data and mathematical models implemented by neural networks or other ML methods. Moreover, it may be possible to design specialized network architectures that automatically satisfy some of the physical invariants for better accuracy, faster training and improved generalization. Considering PIML approaches, some of the prevailing trends, capabilities and limitations in embedding physics into ML are discussed. Some PIML applications both for forward and inverse problems are presented, including discovering hidden physics and tackling high-dimensional problems.

PIML is a methodology that improves model accuracy and interpretability by combining physics concepts and ML techniques, enables more robust predictions while reducing the need for large amounts of training data, and has numerous potential applications in science and engineering, such as modeling physical systems, solving partial differential equations, and performing inverse analysis and optimization. Some approaches used in PIML can be classified as: physics-constrained ML, physics-guided ML, physics-encoded ML, data augmentation through physics principles, transfer learning from physics-based synthetic data to experimental data, and delta-learning physics correction to improve physics generalization and delta-learning unknown physics to represent unmodeled physical phenomena (TRONCI et al., 2024). In addition, there is also a need for new PIML standardized frameworks and patterns, as well as new PIML approaches, to enable scalable and robust implementations.

A first PIML approach was given by the Physics-Informed Neural Networks (PINN), which employ DNN for problems related to the solving of Partial Differential Equation (PDE), as described below.

1.2 Physics-Informed Neural Networks

Previous work related of this thesis research evaluated the parameter discovery of the data-driven one-dimensional Burgers' Equation (1D Burgers) using a PINN implementation.

Concerning PINNs, many simulations are mathematically modeled by PDEs, which have derivatives in space and time. Typically, the coefficients of these derivatives are unknowns that express physical properties of the problem being modeled by the PDE, which is usually solved by standard Numerical Models (NM) like the finite difference method. Recent work has proposed to solve a PDE, or discover the parameters of the PDE based on data, using Deep Neural Network (DNN). The universal approximation theorem states that a DNN can approximate any continuous function, as long as the network has a sufficient number of hidden layers and employs nonlinear activation functions (HORNIK et al., 1989). In any case, for parameter resolution or discovery, depending on the architecture employed, the DNN training input is provided by sample points, which compose a subset of the full set of known points of the function in the space and time domain. These sample points can be conveniently selected in order to increase its number of data related to initial (IC) and boundary conditions (BC). These sample points are then called Collocation Points (CP) (BASDEVANT et al., 1986), a name that came from standard NMs.

However, either for solving a PDE or for parameter discovery of the PDE, the training phase of DNN requires a high number of sample points in order to obtain accurate solutions. Sample points can be obtained either by observation or using synthetic data from a known model. PINNs, which are DNN embedding underlying physical equations as prior knowledge, were proposed in order to make feasible the use of a lower number of sample points in the training phase (CUOMO et al., 2022).

PINNs can be used for solving the direct problem, also called inference or solution, where the PDE and parameters are known, but not the result of the simulation. In the inverse problem, also called system identification or discovery, the CPs are used to discover the PDE parameters that best fit the exact dataset, thus finding the governing equations that rule the considered dynamic system modeled by the PDE. Dynamic systems are a diverse and well-studied class of mathematical objects used to model systems that change over time. Once identified, these equations can be utilized to predict future states, inform control inputs, or facilitate theoretical research using analytical approaches. Also considering the inverse problem, in the case of noisy datasets, PINNs can also perform data-driven parameter discovery, thus obtaining a more accurate model that allows to reproduce the set of sample points with less or no noise (ZHOU; XU, 2024).

PINNs are employed for unsupervised learning when trained only on physical equations, and for supervised learning when dealing with noisy data or solving an inverse problem, and generally the related physical equations are used in the loss function (CUOMO et al., 2022). Furthermore, PINN can be used in cases where the model (or the PDE that describes it) is known, to reduce the size of the dataset required to train the DNN, thereby increasing efficiency, or when there is noise in the sample, and we want the underlying physical law to help deal with it.

Models using PINN are effective and efficient for ill-posed and inverse problems, and when combined with domain decomposition, can scale to larger problems (SHARMA et al., 2023). Neural network-based regression algorithms can be implemented effectively and simply, even without the use of mesh discretization. Future PINN research topics include operator regression, the search for new variables, intrinsic representations, and equivariant neural network topologies with integrated physical restrictions.

Raissi et al. (2019) published an article on PINNs with 8956 citations (as of May 2024). The paper describes PINNs as DNNs trained to tackle supervised learning tasks when dealing with noisy data or inverse issues, and unsupervised learning when trained simply on physical equations, but conforming with physical rules, which are commonly described by nonlinear PDEs (CUOMO et al., 2022).

It also describes the use of DNNs to solve PDEs and obtain physics-informed surrogates of the physical model that are fully differentiable in all coordinates and free parameters. PINNs form a new class of data-efficient universal function approximators, which can be effectively trained using small datasets, and which may encode any underlying physical law.

DNN training data can be randomly sampled from observational data, or through simulations using synthetic data generated by a NM. Except for the latter, as long as a sufficient number of CP is available, a standard DNN can solve the PDE, but otherwise a PINN would be required. A PINN uses a specific loss function incorporating the related PDE and its parameters, in such a way that during the training phase using the set of CP, the applicable physical law is incorporated (CUOMO et al., 2022).

The most common PINN architectures are Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Newer architectures are Auto-Encoder (AE), Deep Belief Network (DBN), Generative Adversarial Network (GAN) and Bayesian Deep Learning (BDL) (VLADIMIROVA et al., 2018).

Cuomo et al. (2022) published an article with 737 citations (as of April 2024) that provides a complete analysis of PINNs, and discusses the benefits and drawbacks of various PINN variations, including variational hp-VPINN, conservative PINN (CPINN), and physically constrained DNNs (PCNN). Additionally, PINNs can function as Reinforcement Learning (RL) agents. According to the same study, most of the research focused on PINN customization through the use of different activation functions, DNN architectures, gradient optimization techniques, and loss functions. Many PINN applications have been proposed and, according to the case, a PINN implementation can be used with advantage as a substitute to a standard numerical method, as in the case of the Finite Element Method (FEM).

Considering that PINNs were recently proposed, there is still room for improvements and also for new theoretical considerations. Kim et al. (2021) proposed a general taxonomy of PINN conceptual levels, based on a literature review about dynamic systems: (i) what type of DNN is used, (ii) how physical knowledge is represented, and (iii) how physical information is integrated. The PINN is not the only DNN approach used to solve PDEs, although it seems to be the leading one in terms of number of articles. The PINN mainstream is still the PDE direct problem, but the number of works proposing PINNs to solve PDE inverse problems has been increasing. For instance, the use of CP is emphasized in some articles (MENG et al., 2020; YANG; PERDIKARIS, 2019; RAISSI et al., 2019). Other works present the Conservative PINN (CPINN) (JAGTAP et al., 2020), and the Physically Restricted Neural Networks (PCNNs) (ZHU et al., 2019; SUN et al., 2020; LIU; WANG, 2021).

PINNs may model the PDE with unknown initial and boundary conditions (called soft BC), as detailed in Raissi et al. (RAISSI et al., 2019), a work that proposed the name Physics-Informed Neural Network (PINN). There are also PCNNs, a class of *data-free* PINNs, which impose known initial and boundary conditions via a customized DNN architecture (hard BC) that also include the PDE in the loss function. Soft restrictions in DNNs are commonly stated as penalty terms in the loss function. These penalties urge the network to adhere to the intended behavior while allowing for some flexibility. Hard constraints, in turn, are often defined as specific conditions that network output parameters must meet, which are indirectly enforced using techniques such as the penalty approach and the improved Lagrangian method, which penalize violations of the constraints, effectively transforming hard constraints into soft ones during optimization. (LU et al., 2021; NANDWANI et al., 2019; YAO et al., 2023).

Recently, numerous frameworks have been presented, including the Deep Ritz Method (DRM) (E; YU, 2017), in which the loss function value is defined as the energy required to solve the problem. There are alternative implementations based on the Galerkin method, also known as the Petrov-Galerkin method, in which the loss function value is calculated by multiplying the residue by a test function. This approach results in the Deep Galerkin Method (DGM) when the residue is volumetric

(SIRIGNANO; SPILIOPOULOS, 2018). Given that a Galerkin technique is utilized with CP, it can be considered a version of PINN, namely a hp-VPINN (Kharazmi et al.) (KHARAZMI et al., 2021).

Challenges and future prospects for PINN include its use and implementation for a variety of application involving real-world equations. Applications include everything from convergence and stability to boundary condition management, DNN design, general PINN architecture design, and more. PINN have the potential to be a useful tool for solving high-dimensional PDEs that are important in physics, engineering, finance, and other areas. However, PINNs generally cannot accurately approximate PDE solutions in comparison to other specific NMs that are optimized for a single PDE (CUOMO et al., 2022). It is worth to note that some PINN implementations that appear in the literature lack reproducibility and documentation, restricting new potential users to employ them.

The extension of PINN applications for 2D or 3D problems poses additional challenges since training complexity grows, requiring more complex architectures and larger batch sizes. As a consequence, training may be constrained by GPU memory, and longer training times are required for convergence to the solution (NANDI et al., 2021). There is a trend to include PINN into standard libraries written in Fortran and C/C++, as well as integrating PINN solvers into older high performance computing (HPC) applications (MARKIDIS, 2021). For instance, PINN may be implemented on modern HPC clusters using the Horovod distributed training framework (SERGEEV; BALSO, 2018).

1.3 ecRad Radiation Scheme

This thesis proposed a PIML-base implementation of part of the ecRad radiation module, specifically the numerical gas-optics scheme, which is processing-demanding.

Atmospheric radiation is the most influential scheme in numerical climate and weather models, requiring the solving of multiple instances of the radiative transfer equation to simulate absorption and scattering processes of the incident solar radiation, and also of the back-scattered terrestrial radiation. Cloud coverage and surface optical properties are considered, as well as the multiple greenhouse gases. Standard implementations of the IFS model do not solve the radiation module for every timestep and grid point (in latitude and longitude). The ecRad radiation module was developed by ECMWF, being operational with the IFS since 2017 (HOGAN; BOZZO, 2018). It is extremely adjustable, with options for gas optics, cloud optics, aerosol optics, and radiative transfer solvers that represent cloud heterogeneity for different cases of cloud coverage.

Several authors proposed to replace the original numerical radiation module gas-optical scheme (RRTMGP) by DNN models in order to obtain beter computational performance (UKKONEN; HOGAN, 2023; VEERMAN et al., 2021; MEYER, 2022). The kernel of the RRTMGP scheme is written in Fortran 90, and performs a 3D linear interpolation of the optical depth of the atmosphere for the considered 2D grid point using a lookup table indexing temperatures, pressures and mixing fractions. Among other libraries, these DNN implementations of the gas-optical scheme as the TensorFlow library of the Python environment to perform training and validation of neural networks using a known dataset. The resulting models were then reproduced in Fortran 90 codes that embed the weights, biases, activation functions, etc. Thus, these codes can be coupled to the rest of the radiation module reproducing the operations that the DNN models would apply in the input data of the gas-optical scheme.

One of these works using DNN (CHEVALLIER et al., 2000) presented a speedup of 7 using a DNN over a standard NM to estimate parameters for the Longwave Radiative Transfer model used by the IFS model of the European ECMWF. This shows the potential of using DNNs to obtain the parametric representations for the modeling of various atmospheric processes. Krasnopolsky and Fox-Rabinovitz (2006) also cites speedups between 10 and 10^5 using a DNN over a standard NM in the parametrization of physical modules in oceanic and atmospheric models. Ukkonen and Hogan (2023) demonstrate speedups of about 3 when using DNN in the optical gas scheme, compared to the conventional RRTMGP scheme, promoting the acceleration of the overall ecRad module by a factor of 1.5.

1.4 Objectives of the Thesis Research

There is a trend in weather and climate numerical models to replace standard microphysics modules by AI-based modules that are faster or even more accurate, as in the case of this work for the radiation module. Numerical models simulate the dynamics and the microphysics of the atmosphere and are written in Fortran 77 or 90. The model dynamics simulates the atmosphere as a fluid subjected to governing equations of fluid dynamics and thermodynamics, which are discretized for a 3D grid, usually composed of a 2D latitude-longitude grid for several vertical levels of pressure. The model physics simulate different processes like turbulence or radiation, being approximated or parameterized independently for each 2D grid point, and thus called subgrid processes.

This thesis work aims to develop a PIML-based implementation to replace the gas-optical scheme of the ecRad radiation module. The current numerical ecRad module implements a processing-demanding numerical model and therefore cannot be executed on every time steps and every grid points of the IFS ECMWF model. There are some works of different authors ((CURCIC, 2019; KRASNOPOLSKY et al., 2008; KRASNOPOLSKY, 2013; UKKONEN et al., 2020; UKKONEN; HOGAN, 2023; VEERMAN et al., 2021)) showing the feasibility of DNN-based implementations to replace the original gas-optical scheme in order to optimize its computational performance.

This thesis proposes an incremental approach for the AI-based radiation module implementations, starting with a non-PIML Deep Neural Network (DNN), followed by a PINN implementation, and then other PIMLs implementations. The simple no-PIML DNN approach was already implemented for an example test case, as shown here, as a starting point. Further tests involving DNN-only implementations will come after, and PINN-based and other PIML-based are certainly foreseen.

The proposed PIML-based implementation of the ecRad radiation module can be partially employed in the microphysics of the MPAS (Model for Prediction Across Scales) atmospheric model that was chosen for MONAN (Model for Ocean-laNd-Atmosphere predictioN) currently being developed by CPTEC/INPE and other Brazilian institutions.

2 METHODOLOGY

This chapter presents in the next section the methodology employed in former work, related for the application of a PINN to the solving of the 1D Burgers Equation, including the associated problem of parameter discovery. The following section presents a teste case that uses a DNN implementation to replace the numerical implementation of the gas-optical scheme that is part of the ecRad radiation module of the IFS model used in the ECMWF.

2.1 PINN Implementation for the 1D Burgers Equation

In former work related to the application of a PINN related to the 1D Buergers equation, three problems were proposed: the use of PINN for the solution of the 1D Burgers' Equation, the discovery of PDE parameters using PINN, and the discovery of PDE parameters using NM.

The problems require data-driven parameter discovery for a given 1D Burgers' Equation, which estimates the velocity field u of a fluid along the dimension x and over time t (Equation 2.1). In this equation, two coefficients of the differential operators are defined as the parameters λ_1 and λ_2. The latter is the kinematic viscosity of the fluid (ν), and the velocity (u) subscripts denote partial differentiation in time and space, respectively, as u_t (du/dt), u_x (du/dx), and u_{xx} (d^2u/dx^2).

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0 \tag{2.1}$$

In the considered problem, the Burgers' Equation spatial and time domain, and the initial (IC) and boundary conditions (BC), are:

$$x \in [-1, 1],\ t \in [0, 1], \tag{domain}$$
$$u(0, x) = -\sin(\pi x), \tag{IC}$$
$$u(t, -1) = u(t, 1) = 0. \tag{BC}$$

Two sets of parameters were used, both using $\lambda_1 = 1$, but with $\lambda_2 = 0.01/\pi$ (expressing low viscosity) for the first, and $\lambda_2 = 0.1/\pi$ for the second (high or usual viscosity). It is intended to demonstrate that small viscosity values in the Burgers' equation can cause discontinuities, which are interpreted as shock waves, and which become more pronounced as the viscosity value decreases, thus making modeling using standard

NMs more difficult as it requires more precise spatial and temporal resolutions. In any case, some datasets were generated for the PINN and NM implementations with problem size (dimension x by time t) 128x64, 256x100, 256x128, and 512x256 (some were used in specific cases). The datasets were generated using the numerical GQM, which is an iterative numerical algorithm that approximates the definite integral of a function as a weighted sum of the functions values at specified points of the integration domain (BURKARDT, 2013). The method takes into account the domain, initial and boundary conditions shown above for generating the dataset. The dataset contains the spatial values x, the time values t, and the exact solution $u(t, x)$.

2.1.1 PINN Implementation

A part of the code implemented in this work was adapted from Raissi et al. (2019), as described ahead in this work, using Python 3.7 and the framework TensorFlow 1.15 for execution on GPUs. The PINN code is based on an MLP, with an input layer of 2 neurons, a number of hidden layers ranging from 1 to 8, each one with a number of neurons ranging from 10 to 30, and an output layer with a single neuron. The loss function is based on Mean Squared Error (MSE), and is detailed in subsection 2.1.2. Minimization of the loss function is performed by an optimization method, the generalized Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm, a quasi-Newton method. All hidden layers employ the hyperbolic tangent as the activation function.

Listing 2.1 - Snippet of Python code that implements $u(t, x)$, seen in the listing as `net_u(t, x)`.

```
def neural_net(X, weights, biases):
  num_layers = len(weights) + 1
  H = 2.0 * (X - lb) / (ub - lb) - 1.0
  for l in range(0, num_layers - 2):
    W = weights[l]
    b = biases[l]
    H = tf.tanh(tf.add(tf.matmul(H, W), b))
  W = weights[-1]
  b = biases[-1]
  Y = tf.add(tf.matmul(H, W), b)
  return Y

def net_u(t, x):
  u = neural_net(tf.concat([x, t], 1), weights, biases)
  return u
```

The implemented DNN MLP uses the L-BFGS optimizer provided by the SciPy[1] framework, which was configured to a maximum of 50,000 iterations (a value commonly used by SciPy for such optimizer) and also to stop iterations when the hardware floating point precision lower limit is reached, in this case IEEE-75432 single precision binary floating point. The batch size used is the same size as the CP set, which results in one iteration per training epoch. Some code snippets extracted from the implementation are shown in the Listings 2.1, 2.2, and 2.3.

[1]https://docs.scipy.org/doc/scipy/reference/Optimize.minimize-bfgs.html

Listing 2.2 - Snippet of Python code that implements $f(t, x)$, seen in the listing as `net_f(x, t)`. The `net_u` function is shown in Listing 2.1. The `tf.gradients` is part of the TensorFlow framework and is used in the gradient-based training and optimization algorithm.

```
def net_f(x, t):
  lambda_1 = lambda_1
  lambda_2 = tf.exp(lambda_2)
  u = net_u(t, x)
  u_t = tf.gradients(u, t)[0]
  u_x = tf.gradients(u, x)[0]
  u_xx = tf.gradients(u_x, x)[0]
  f = u_t + lambda_1 * u * u_x - lambda_2 * u_xx
  return f
```

Listing 2.3 - Snippet of Python code that implements the loss function (`loss`). The `u` is the exact solution. The `net_u` function is shown in Listing 2.1. The `net_f` function is shown in Listing 2.2.

```
u_tf = u  # exact solution
u_pred = net_u(t, x)  # predicted solution using the DNN
f_pred = net_f(t, x)  # predicted solution using the PDE
loss = (tf.reduce_mean(tf.square(u_tf - u_pred)) +
             tf.reduce_mean(tf.square(f_pred)))
```

As already mentioned, in the proposed approach, in a first step, the PINN is trained to obtain the parameters of the differential operators, and then the resulting PINN model, which incorporates these parameters, is used to generate a 1D velocity field in the space domain and of time, which will be compared with the exact 1D field generated by GQM.

2.1.2 PINN Parameter Discovery

The PINN training dataset is provided by a set of CPs that were randomly selected from the exact dataset generated by the GQM, for the considered domain and time interval. For a given training iteration k, the value of the loss function to be minimized is given by the sum of two Mean Square Errors (MSE), MSE_u and MSE_f (Equation 2.3), which can be seen in Figure 2.1, and the implementation is related to the code snippet in the Listing 2.3.

The MSE_u represents how well the PINN output matches the exact data points of the solution for the set of CPs. It is calculated as the mean squared error of the difference between the exact solution and the PINN predicted output for each CP of the set. In the equation, N is the number of CPs, the term $[u(t_u^i, x_u^i)]$ refers to the exact solution for each CP, and the term $[u^i]$ refers to if the solution predicted by DNN, thus the difference $[u(t_u^i, x_u^i) - u^i]$ measures the prediction error. The MSE_u can be seen in the Figure 2.1 and in the Listing 2.1.

The MSE_f represents how well the automatically calculated derivatives match the actual PDE terms at the CPs within the domain. The mean squared error of the physics-based regularization term $([f(t_u^i, x_u^i)])$ enforces the compliance to the governing physical equations or constraints. It can be seen in the Figure 2.1 and in the code snippet shown in the Listing 2.2.

The MSE is used in the gradient descent optimization algorithm. The algorithm includes derivatives and automatic differentiation (BAYDIN et al., 2018) to calculate the gradient[2] of the error function with respect to the parameters of the PINN, to update the network weights and biases at each iteration. The gradient provides guidance to adjust the parameters in value and direction in order to minimize the error.

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx} \tag{2.2}$$

$$MSE^k = MSE_u^k + MSE_f^{k-1} \tag{2.3}$$

where

$$MSE_u^k = \frac{1}{N}\sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2$$

and

$$MSE_f^{k-1} = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2$$

According to the work of Raissi et al. (2019), the standard division of the CP dataset into training, validation and testing is not used in this specific PINN model. Instead, a random sample of CPs from the dataset is taken and used to train the PINN.

[2]https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/gradients

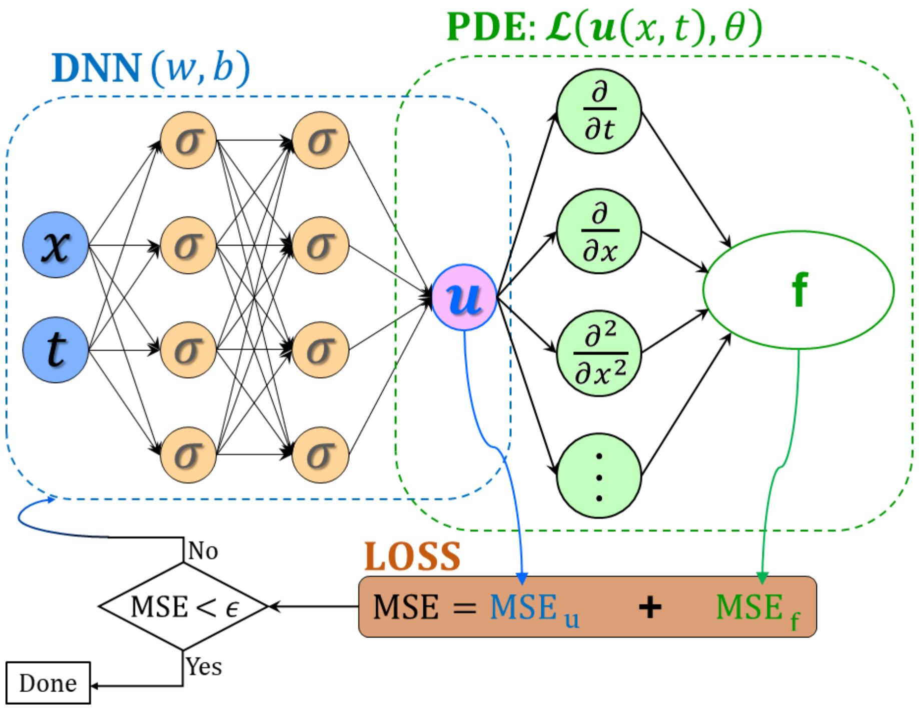
PDE: $\mathcal{L}(u(x,t),\theta)$
DNN (w,b)
x
t
σ
u
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
f
LOSS
MSE = MSE$_u$ + MSE$_f$
MSE < ϵ
No
Yes
Done

Figure 2.1 - Schematic showing an example structure and loss function, for the PINN approach. The symbol θ represents the optimization algorithm. The ϵ represents the smallest threshold error in the optimization algorithm.
Source: Adapted from Meng et al. (2020).

During training, a specific validation set is not used. After training, the network is used to generate two results: (*i*) the discovery of the PDE parameters, and (*ii*) a prediction of the PDE solution.

Once the parameters have been discovered, in a second step, the resulting PINN model is used to predict the velocity field for all points in the defined space and time domain. This predicted velocity field can then be compared with the exact velocity field (generated by GQM), using a relative error to evaluate the accuracy of the prediction, as suggested in Xu et al. (2022) (XU et al., 2022). Such a relative error is defined by Equation 2.4.

$$R_{L2} = \frac{\|\widehat{U} - U\|}{\|U\|} \tag{2.4}$$

The $\|U\|$ represents the Euclidean L2 norm, which is given by the Euclidean distance from the vector coordinate to the origin of the vector space, i.e. it is the square root of the sum of the squared components of the vector. In the one-dimensional case, the L2 norm is simply given by the value of the velocity. The $\|\widehat{U} - U\|$ represents the application of the L2 norm to the deviation of the estimated velocity field $\widehat{U}$, with respect to the exact velocity field U, considering all dataset points in the space and time domains.

The PINN hyperparameters, like the number of hidden layers $N_l(l = 1, 2, ...)$, and the number of neurons in each layer $N_{le}(e = 1, 2, ...)$, can also be optimized by numerical experimentation (trial and error). Optimization of N_l and N_{le}, and eventually of other hyperparameters not employed here, is still an unsolved problem, being made empirically (XU et al., 2022).

As any neural network, PINNs are subjected to overfitting or underfitting. Overfitting means that the DNN performs very well using the training set, but lacks performance using different, new input data of the test set, i.e. it does not generalize. On the other hand, underfitting means that the model performs poorly on both the training and test sets. Obviously, both overfitting and underfitting imply in poor performance of the neural network (KOEHRSEN, 2018).

2.1.3 NM-based Parameter Discovery

The inverse problem solved by the implemented PINN to find the coefficients of the differential operators is then compared to the chosen NM-based method, the Sparse Identification of Nonlinear Dynamical Systems (SINDy) method (BONINSEGNA et al., 2018). This is a method developed for solving the identification problem of dynamical systems, which are systems that change over time. SINDy uses sparse regression to create a linear combination of basis functions to captures the dynamic behavior of the considered physical system. It employs observational or synthetic data to obtain the governing equations that fit such data of the dynamical systems. The discovered equations can then be used to predict future states, inform control inputs, or enable theoretical research using analytical techniques (BRUNTON et al., 2016).

Assuming a physical system that evolves in time t and space x defined by a collection of measurements $x(t) \in \mathbb{R}$. Time evolution of $x(t)$ is modeled by SINDy using a nonlinear function $f(x(t))$ (Equation 2.5). The vector $x(t) = [x_1(t), x_2(t), \dots x_n(t)]^\top$ represents the state of the physical system at time t. The problem solved by SINDy is shown in Equation 2.6, where the function $f(x(t))$ is expressed as a matrix Θ of basis functions applied to the input data X, i.e. $\Theta(X)$, which is then multiplied by a matrix Ξ of coefficients that weighs these basis functions. Along the iterations, these coefficients are optimized until achieving convergence by means of an objective function (Equation 2.7) that assess the correctness of the solution given by $[\Theta(X)\,\Xi]$ at a given iteration. It is assumed that $f(x(t))$ is usually sparse in the space of a suitable set of basis functions, i.e. that much of the coefficients of the matrix Ξ are zero.

$$\frac{d}{dt}x(t) = f(x(t)) \tag{2.5}$$

$$\dot{X} \; = \Theta\,(X)\;\Xi \tag{2.6}$$

SINDy applies a sparsity-promoting regression (such as LASSO (TIBSHIRANI, 2011)) to a library of nonlinear candidate functions produced from the measurements, but restricting the number of basis functions to obtain a compact representation of the function and avoid overfitting. SINDy has been effectively used to solve a variety of problems, including linear and nonlinear oscillators, chaotic systems, and fluid dynamics (BRUNTON et al., 2016). SINDy is implemented by the sparse regression package PySINDy[3] which includes several implementations for the sparse identification method of nonlinear dynamical systems from various authors, being comprehensive literature reviews available at (de Silva et al., 2020) and (KAPTANOGLU et al., 2022). Part of the code implemented in this work was reused and adapted from PySINDy library examples. PySINDy runs on CPU.

This work employs four of these implementations, known as Sparse Regression Optimizers (SRO), which are part of the parameter discovery package and can be selected separately, producing a result for each choice: Sequentially Threshold Least Squares (STLSQ), Orthogonal Least Squares of Forward Regression (FROLS), Sparse Relaxed Regularized Regression (SR3), and Sparse Stepwise Regression (SSR). To keep things simple, we'll treat each of these as a separate NM.

A code snippet of this framework is shown in Listing 2.4, where `optimizer` is the chosen optimization method (exemplified in the code as STLSQ), `SINDy` defines the model, `fit` discovers the parameters, and `print` shows the result.

[3]http://pysindy.readthedocs.io

Listing 2.4 - Python code snippet that implements the NM for data-driven PDE parameter discovery. The STLSQ is one of the optimizers in the PySINDy framework.

```
optimizer = ps.STLSQ(threshold=2, alpha=1e-5, normalize_columns=True)
model = ps.SINDy(feature_library=pde_lib,
                 optimizer=optimizer,
                 feature_names=["u"])
model.fit(u, t=dt)
model.print()
```

The dataset used by SINDy for each of the 4 optimizers is the same used in the PINN method, but employing the full dataset instead of the set of CPs. Parameter discovery using PINN required training the network with the set of CPs, use of the Burgers' PDE in the loss function, etc. After training, the predicted velocity field in space and time is then compared to the exact field. In the case of SINDy, the complete dataset is used to obtain the parameters, with no further comparison (BRUNTON et al., 2016). However, parameters discovered by PINN and SINDy are compared in order to evaluate the accuracy of both approaches using the coefficients of the known PDE as reference. Although possible, some optimizations were not made to SINDy in order to obtain better accuracy, being the default settings of the framework used. These possible optimizations can be performed as future work.

Except in the case of SINDy with the SR3 optimizer, the other optimizers employ the objective function that must be minimized, which can be seen in the Equation 2.7.

$$\| \ y \ - \ X \ w \ \|^2 \ + \ \alpha \ \| \ w \ \|^2 \tag{2.7}$$

Above, w is the weight matrix containing the set of basis functions that maps the dataset contained in X to the solution y, i.e. w corresponds to the 2nd member ($[\Theta(X)\,\Xi]$) of Equation 2.6, and α is a regularization coefficient applied to w, helping prevent overfitting.

In order to minimize the objective function, the STLSQ (BRUNTON et al., 2016) algorithm uses ridge regression with sequential threshold, and iteratively runs the least squares algorithm, but masking elements below a specific threshold. The FROLS (BILLINGS, 2013) algorithm is a greedy algorithm that iteratively selects the most correlated function in the library. The SSR (BONINSEGNA et al., 2018) is also a greedy algorithm that iteratively eliminates the smallest coefficients. A greedy algorithm is any algorithm that uses the heuristic of choosing the locally optimal solution at every iteration (BLACK, 2005). Regardless of not assuredly finding a global optimal solution, greedy heuristics can produce locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time. Along the iterations, the SSR and FROLS algorithms respectively truncate (or add) one nonzero coefficient at each algorithm iteration.

The SR3 algorithm (CHAMPION et al., 2019) is a relaxed and sparse regularized regression with linear (dis)equality constraints that attempts to minimize the objective function (Equation 2.8).

$$\begin{aligned} &0.5\|y - Xw\|^2 + \lambda R(u) + (0.5/\nu)\|w - u\|^2 \\ &\text{subject to } Cw = d \end{aligned} \tag{2.8}$$

over u and w, where $R(u)$ is a regularization function, C is a constraint matrix, and d is a vector of values.

2.2 DNN Preliminary Approach for ecRad

The problem requires the emulation, using DNN, of the gas-optical algorithm solution of a modern radiation scheme (RTE+RRTMGP), for a wide range of atmospheric conditions, with errors in the heating rates and top-of-atmosphere radiative forcing, typically below $0.1 K day^{-1}$ and $0.5 W m^{-2}$ (UKKONEN et al., 2020). What is written below is an explanation more linked to the numerical model, but which serves as a basis for emulation.

Unlike many other phenomena specified in dynamical models, such as clouds and convection, atmospheric radiative transfer is a well-studied issue that can be correctly represented. Modern radiation codes typically use the correlated k approximation and the k-distribution approach to handle the complexity (GOODY et al., 1989). The highly variable absorption coefficient spectrum as a function of wavelength, k(λ), is rearranged by k and replaced with a monotonically increasing function $k(g)$, where $g(k)$ is the cumulative distribution function. This function is integrated using a restricted number of quadrature points, referred to as g points. Compared to the line-by-line method, this method significantly reduces the number of monochromatic calculations needed to retrieve fluxes in the SW and LW spectra.

K-distributions can be applied to an inhomogeneous medium, such as the atmosphere, providing that the mapping of wavelengths to g-space is completely correlated for neighboring atmospheric layers, an approximation that allows fluxes and heating rates to be estimated with less than 1% uncertainty (FU; LIOU, 1992). Despite the effectiveness of the correlated k-distribution (CKD) method, radiation calculations in climate models are expensive and performed on a coarser horizontal and temporal grid than others. The ECMWF high-resolution forecast model calls the radiation scheme every 1 hour on a grid with 10.24 times fewer columns than the rest of the model (HOGAN; BOZZO, 2018).

Because radiation calculations in climate models are expensive, they appear to be a viable area of study for optimizing efficiency without losing accuracy, and one possible approach is the use of PIML. An interesting aspect about PIML that can be explored in research is the increase in performance of algorithms through the use of single-precision floating-point numbers and the GPU, which is a known feature in algorithms that use DNN. The preliminary work shows a DNN implementation replacing an existing numerical implementation of the gas-optical scheme (from RRTMGP) of the eCrad model, where performance gains were obtained in preliminary tests, paving the way for future work and a proposal for investigation and implementation of PIML.

2.2.1 DNN-based Gas Optics Implementation

As previously mentioned, the thesis proposes a PIML-based implementation to replace the numerical radiation module ecRad employed in the IFS model of the ECMWF. However, as a preliminary work, a DNN-based model of the RRTMGP gas-optical scheme was implemented for a test case involving a reduced database. The RRTMGP numerical code is part of ecRad module. The accuracy and performance of the DNN implementation were analyzed, taking as reference the corresponding numerical implementation RRTMGP.

The DNN implementation of the gas-optical scheme of the radiation module shown here, the original numerical code (RRTMGP) was replaced by a DNN model. The kernel of the original RRTMGP scheme was written in Fortran 90, and performs a 3D linear interpolation of the optical depth of the atmosphere for the considered 2D grid point using a lookup table indexing temperatures, pressures and mixing fractions. RRTMGP stand for Rapid Radiative Transfer Model for General circulation model (GCM) applications – Parallel (PINCUS et al., 2019). It is based on RRTM-G, the Rapid Radiative Transfer Model for GCM-Parallel applications (HOGAN et al., 2017). This is a free and open source package that predicts the optical properties of the gaseous atmosphere, includes the solver of the Radiative Transfer Equation called Radiative Transfer for Energetics (RTE), and is used in (numerical) climate and weather models (PINCUS et al., 2019). This scheme, like others, includes SW (shortwave) solar radiation and LW (longwave) thermal radiation solving. It employs a statistical k-distribution model based on advanced spectroscopy and numerous g-points, with 256 within 16 LW bands and 224 within 14 SW bands. The RRTMGP code implemented in this work was adapted from Ukkonen and Hogan (2023).

The DNN implementation of the gas-optical scheme was developed in a framework composed of the Python 3.12 and the TensorFlow 2.16 library in order to perform training and validation of a MLP neural network using a known dataset. The resulting trained model is saved to disk to later be used in the DNN implementation in Fortran 90 inserted in the radiation module. This Fortran 90 code can be run on CPU or GPUs using cuBLAS and OpenACC, while TensorFlow uses GPU. In the ecRad radiation module, the DNN code is based on Neural Fortran, being called RRTMGP-NN Curcic (2019).

The standalone ("offline") version of ecRad used in this work is modular, runs on a Unix-like platform, operates independently, and allows investigations and research related to cloud radiative effects without being tied to the operational forecasting model. It has four components: Gas Optics, Aerosol Optics, Cloud Optics, and Solver. ecRad offers several solvers, including McICA (Monte-Carlo Independent Column Approximation), Tripleclouds, and SPARTACUS (UKKONEN; HOGAN, 2023). There is a test version for practical test cases that may run on a standard laptop with GPU and minimum RAM, which was employed in this work and had its performance evaluation analyzed with the Gprof tool.

The input data to the DNN are the temperature, pressure and concentrations of each gas represented in the RRTMGP and, excluding oxygen and nitrogen which are assumed to be constant, this results in 18 LW and 7 SW inputs. In order to predict absorption and Rayleigh optical depths in SW and Planck and absorption optical depths in LW, different models are trained. These are multi-output DNNs that predict all g-points simultaneously, with output sizes of 256 LW or 224 SW. In the main code, written in Fortran, the originally numerical gas-optical implementation is replaced by the DNN implementation, also written in Fortran, which uses trained models saved on disk. The training of the DNN network to obtain the prediction model is implemented in Python using the TensorFlow library, and the final trained model is saved to disk for later use by the Fortran implementation of the main code. The MLP architectures (layers and neurons) that were used, are described in Table 2.1. In the implementation of the DNN network used in this work, a manual adjustment of the architecture was made, however it would be interesting to use an automated method in future work. One point that is particularly interesting for the study is that identifying an effective loss function is a significant task, since decreasing the instantaneous imprecision of a specific variable does not guarantee numerical stability over long time intervals, realistic variability, or conservation of energy, moisture, and momentum.

Predicted	Input	2 Layers	Output
LW absorption	18	58-58	256
LW emission	18	16-16	256
SW absorption	7	48-48	224
SW scattering	7	16-16	224

Table 2.1 - Network architectures used for different optical depths.
Source: Adapted from Ukkonen et al. (2020).

The underlying data flow in a framework of a conventional numerical gas-optical scheme (e.g., RRTMGP) is shown in Figure 2.2, and the gas-optical scheme it emulates is shown in Figure 2.3. In the numerical model, the code calculates the absorption by 1 to 2 major gas species by 3D interpolation in temperature, pressure, and the relative abundance (η) (PINCUS et al., 2019), and contributions from less important gases are computed separately using 2D interpolation, with a given band being written multiple times. In the case of DNN gas optics, all spectral points are predicted simultaneously from an input matrix that includes all gases.

For a given band, vertical level, and column, the RRTMGP calls interpolation kernels several times to compute gas-specific contributions to optical properties, accounting for temperature (T) and pressure (p) dependencies, as well as overlap of major gases in the band. These kernels loop over the g points in a single band, of which there are only 1-12 in the reduced k distributions.

The DNN improves performance by (i) predicting a vector containing all g points from a vector containing T, p, and the mixing ratios of all gases (the DNN implicitly treats gas interactions) and (ii) batching the computations for multiple atmospheric levels and columns by expressing the core DNN computations as matrix-matrix multiplications (between weights W and input matrices) that are delegated to a BLAS library.

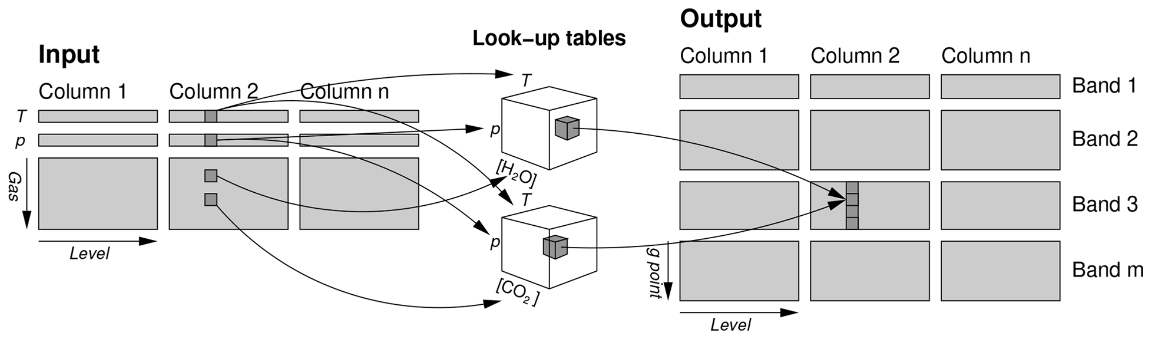

Figure 2.2 - Schematic illustrating the data flow in a structure of a conventional numeric gas optics scheme. "T" is temperature, "p" is pressure, "Gas" represents relative abundance, "Band" corresponds to the LW and SW radiation bands, "g-points" correspond to "k-terms" of the correlated k-distribution method, “Level” corresponds to atmospheric layers or vertical grid points within a column representing altitude or pressure, and "look-up tables" (LUT) correspond to the look-up table kernels of the RRTMGP model. The Figure 2.3 complements this Figure.
Source: Adapted from Ukkonen and Hogan (2023).

In the chosen DNN loss function (Equation 2.9), α is a coefficient representing a trade-off between heating rate and radiative forcing errors (e.g. 0.6 for LW and 0.2 for SW), y is the target vector, and $\hat{y}$ is the DNN output vector. It minimizes the error in the difference in y associated with different perturbation experiments as well as the mean squared error of y (scaled DNN outputs) and indirectly measures radiative forcing errors.

$$loss = \alpha \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + (1-\alpha) \sum_{\substack{i=1 \\ i \text{ odd}}}^{N} ((y_{i+1} - y_i) - (\hat{y}_{i+1} - \hat{y}_i))^2 \tag{2.9}$$

The training dataset used is derived from a variety of sources, including existing atmospheric profiles, future climate experiments, and reanalyses, which were synthetically supplemented by varying greenhouse gas concentrations, either manually or using hypercube sampling, as well as data provided by the Radiative Forcing Model Intercomparison Project (PINCUS et al., 2016), which includes 100 profiles and 18 perturbations. The profiles were designed to assess global mean clear-sky errors in instantaneous radiative forcing and should be well suited as an out-of-sample test for our purposes, and in addition, a different dataset based on the CAMS reanalysis (INNESS et al., 2019) is also used that uses the same approach as the IFS and the Correlated k-distribution Model Intercomparison Project (CKDMIP) (HOGAN; MATRICARDI, 2020), where only nine gases are considered, but the

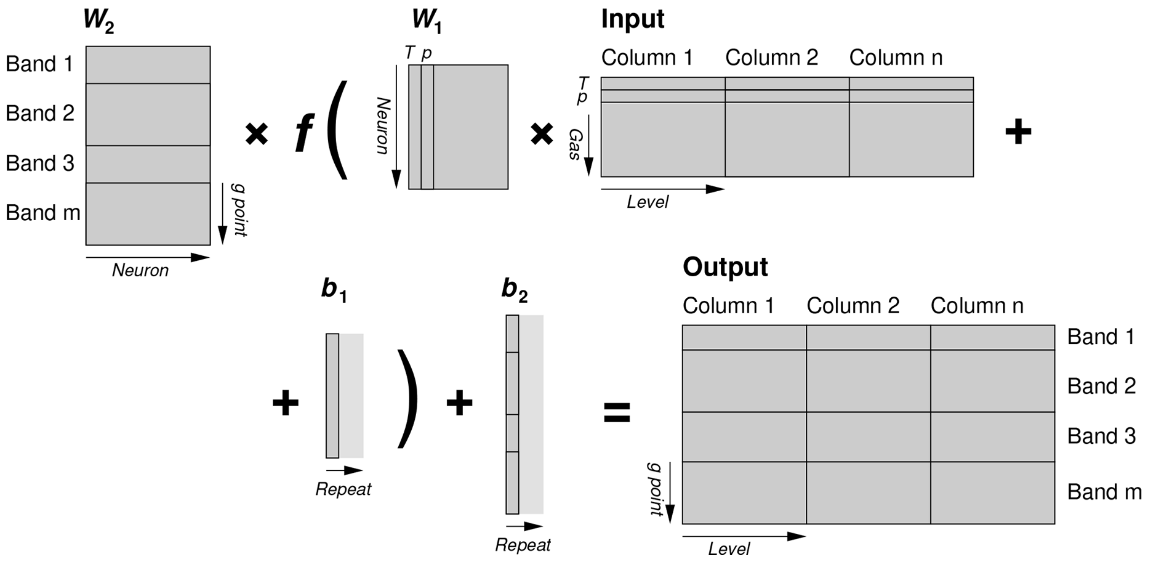

Figure 2.3 - Simplified schematic, with only one hidden layer, illustrating the data flow in a gas optics DNN framework. "W" represents weight, and "b" represents bias. The other terms are described in the Figure 2.2.
Source: Adapted from Ukkonen and Hogan (2023).

radiative forcing.

For performance analysis, Gprof was used, which is a performance analysis tool for Unix-based programs, developed as an expanded version of the previous "prof" tool, using a combination of instrumentation and sampling, and which can collect and print a limited number of call graphs. Two analyses were performed, one of the original numerical implementation of ecRad, and another of the version that uses the DNN implementation. The goal was to identify, in a general way, without going into too much detail, the main costly routines.

2.3 Computing Environment

The DNN, PINN and the four PySINDy codes[4] were executed in three different computing environments. It is important to note that the PINN model runs on GPU, DNN runs on CPU/GPU, and PySINDy runs on CPU. The first environment (used in 1D Burgers') is a PC with a 6-core Intel i7-9750H CPU with 6 cores, 8 GB of main memory, and an NVIDIA GTX 1050 GPU (768 CUDA cores and 3 GB of memory). The second (used in DNN RRTMGP) is a PC with 2-core Intel I7-7500U CPU with 2 cores, 16 GB of main memory, and an NVIDIA Geforce 940MX GPU (384 CUDA cores and 4 GB of memory). The third environment (used in DNN RRTMGP) is

[4]The codes are available at http://github.com/efurlanm/pd1b24

the Santos Dumont (SDumont) supercomputer of the National Scientific Computing Laboratory (LNCC), more specifically a single Bull Sequana X1120 processing node with two 24-core Intel Xeon Gold 6252 Skylake 2.1 GHz processors (total of 48 cores), 384 GB of main memory and four Nvidia Volta V100 GPUs, although only one GPU was employed. Both computing environments included the Python[5] 3.7 interpreter and the TensorFlow[6] v1.15 machine learning platform.

[5]http://www.python.org
[6]http://www.tensorflow.org

3 RESULTS

Results are presented in this chapter for the previous work, application of a PINN implementation for the 1D Burgers equation, and the current preliminary work, the DNN-based gas optics implementation.

3.1 PINN Implementation for the 1D Burgers Equation

The results for the 1D Burger's equation are divided into two parts of experiments: (i) the first compares the results of PINN with the 4 SINDy versions; (ii) the second evaluates the PINN implementation for various hyperparameters and CP sizes.

The first part uses datasets related to different problem sizes (x space versus t time discretization): 128x64, 256x128, and 512x256, while the second part uses only a 256x100 problem size. The GQM is used to generate the dataset corresponding to the exact solution (reference), which are then employed by the SINDy versions, and also to extract the set of CPs used by the PINN implementation. The MLP architecture used for the PINN implementation in the first part has one input layer, 3 hidden layers with 20 neurons each, and one output layer. The PINN undergoes training and PDE parameters are estimated, solving the inverse problem, and the resulting model estimates the solution (space versus time field). The processing-time data obtained in the all experiments is given by the average of 3 executions.

3.1.1 Comparison of PINN and SINDy parameter discovery

Tables 3.1 and 3.2 show the average of elapsed times of 3 runs on the local machine for the PINN (using GPU) and the 4 SINDy implementations (only CPU), for the 3 problem sizes and for the two cases of viscosity. In the case of PINN, training time to estimate the parameters and prediction time of the resulting 1D field using the trained model are shown. Processing times of SINDy versions are consistently lower than those of the PINN model, even considering that the latter executes using GPU. Therefore, future studies must consider the tradeoff between accuracy and processing times when comparing PINNs and numerical methods like SINDy.

Tables 3.3 and 3.4 show the results obtained for the PDE parameter discovery, comparing the PINN and the 4 SINDy implementations (STLSQ, FROLS, SR3 and SSR, discussed in subsection 2.1.3), and also considering 2 different viscosity values in the 1D Burgers' equation, $0.01/\pi$ and $0.1/\pi$. As already commented, 3 sizes of problems are considered: 128x64, 256x128 and 512x256.

Model	Elapsed [s]
128x64	
PINN Train	47.567
PINN Predict	0.371
STLSQ	0.031
FROLS	0.140
SR3	0.054
SSR	0.068
256x128	
PINN Train	53.033
PINN Predict	0.732
STLSQ	0.049
FROLS	0.071
SR3	0.098
SSR	0.086
512x256	
PINN Train	52.633
PINN Predict	3.067
STLSQ	0.105
FROLS	0.625
SR3	0.118
SSR	0.181

Table 3.1 - Comparison of elapsed times (average of 3 runs) for the PINN and the 4 SINDy models for kinematic viscosity of the fluid of $0.01/\pi$, and for execution on the local machine (PC).

Subscripts denote partial differentiation in time and space, e.g. u_x denotes du/dx, u_xx denotes d^2u/dx^2, and so on. -At the top of each table is shown the *Exact PDE* 1D Burgers' equation (e.g., $u_t + 1.0uu_x - 0.003183u_{xx} = 0$) used to generate the datasets.

In the case of PINN, training is done using 2,000 CPs randomly obtained from the dataset, regardless of the size of the problem. In the case of SINDy implementations, the complete dataset is used in the model to obtain the parameters. In 3.3 the viscosity value $(0.01/\pi)$ is lower than in the second $(0.1/\pi)$ in order to check if SINDy accuracy is compromised for small viscosity values, as confirmed by the results. PINN accuracy was good, even using the set of CPs extracted from the smaller dataset, and such accuracy improves with the problem size. 3.3 is for experiments with the higher viscosity value, showing that the accuracy of the SINDy implementations improved in comparison to the results of the preceding table, but are still lower than those obtained by PINN.

The next subsections present the solution field $u(t, x)$ generated by the PINN implementation for the 3 problem sizes, and for the 2 viscosity values. For each case,

Model	Elapsed [s]
128x64	
PINN Train	22.633
PINN Predict	0.361
STLSQ	0.013
FROLS	0.060
SR3	0.015
SSR	0.043
256x128	
PINN Train	36.100
PINN Predict	0.995
STLSQ	0.033
FROLS	0.074
SR3	0.015
SSR	0.060
512x256	
PINN Train	23.133
PINN Predict	3.020
STLSQ	0.086
FROLS	0.588
SR3	0.095
SSR	0.262

Table 3.2 - Comparison of elapsed times (average of 3 runs) for the PINN and the 4 SINDy models for kinematic viscosity of the fluid of $0.1/\pi$, and for execution on the local machine (PC).

there are figures showing the predicted solution $u(t, x)$ with marks representing the CPs, and a particular snapshot at time $t = 0.5$ comparing the PINN prediction and the exact solution. These figures allow a visual assessment of the accuracy of the solutions.

Correct PDE: 0.003183 u_xx - 1.0 uu_x   (viscosity = 0.01/pi)

Model	Discovered equation and parameters
	128x64 problem size
PINN	0.0033735 u_xx - 0.99912 uu_x
STLSQ	0.06420 u + 0.00505 u_xx - 1.06304 uu_x + + 0.00469 uuu_xx - 0.00001 uu_xxx
FROLS	-0.418 u
SR3	0.064 u + 0.005 u_xx - 1.063 uu_x + 0.005 uuu_xx
SSR	0.064 u + 0.005 u_xx - 1.063 uu_x + 0.005 uuu_xx
	256x128 problem size
PINN	0.0031779 u_xx - 0.99942 uu_x
STLSQ	0.00395 u_xx - 1.00869 uu_x + 0.00126 uuu_xx
FROLS	0.015 u + 0.004 u_xx - 1.003 uu_x
SR3	0.004 u_xx - 1.009 uu_x + 0.001 uuu_xx
SSR	0.011 u + 0.004 u_xx - 1.011 uu_x + 0.001 uuu_xx
	512x256 problem size
PINN	0.0031403 u_xx - 0.99850 uu_x
STLSQ	0.00339 u_xx - 1.00534 uu_x + 0.00041 uuu_xx
FROLS	0.006 u + 0.004 u_xx - 1.006 uu_x
SR3	0.003 u_xx - 1.005 uu_x
SSR	0.006 u + 0.003 u_xx - 1.007 uu_x

Table 3.3 - Comparison of the results of the parameter discovery for the 1D Burgers' equation using PINN (in blue) and the 4 SINDy versions (kinematic viscosity of the fluid of $0.01/\pi$).

Correct PDE: 0.03183 u_xx - 1.0 uu_x   (viscosity = 0.1/pi)

Model	Discovered equation and parameters
	128x64 problem size
PINN	0.0318582 u_xx - 0.99928 uu_x
STLSQ	0.03222 u_xx - 1.00012 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.003 u + 0.032 u_xx - 1.002 uu_x
	256x128 problem size
PINN	0.0318372 u_xx - 0.99924 uu_x
STLSQ	0.03193 u_xx - 1.00002 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.001 u + 0.032 u_xx - 1.000 uu_x
	512x256 problem size
PINN	0.0318292 u_xx - 0.99936 uu_x
STLSQ	0.03186 u_xx - 1.00001 uu_x
FROLS	0.032 u_xx - 1.000 uu_x
SR3	0.032 u_xx - 1.000 uu_x
SSR	0.032 u_xx - 1.000 uu_x

Table 3.4 - Comparison of the results of the parameter discovery for the 1D Burgers' equation using PINN (in blue) and the 4 SINDy versions (kinematic viscosity of the fluid of $0.1/\pi$).

3.1.1.1 Problem Size 128x64

Figures 3.1, 3.2, 3.3, and 3.4 show, for the problem size 128x64, the solutions for the viscosities $0.01/\pi$ and $0.1/\pi$. Figures 3.1 and 3.3 show the predicted spatio-temporal solution $u(t, x)$, and the CPs used to training are represented as dark marks on the graph. Since the dataset is relatively small and the number of CP is 2,000, the small marks are aligned and may not appear random at first. At the top and bottom of the figure, ($x = 1.0$ and $x = -1.0$), are the CPs related to the boundary conditions (BC) of the PDE, on the left side ($t = 0, 0$) are the CPs relative to the initial conditions (IC), and in the center of the figure ($x = 0.0$) the complex non-linear behavior (sometimes called formation of a shock wave) of the Burgers' equation with lower viscosity is represented. The remaining randomly selected CPs appear on the graph. Although it was not done this way, it is possible to select more points in the BC and IC regions to reinforce network training, if necessary.

Figures 3.2 and 3.4 show a snapshot in time comparing the superimposed solution for the PINN method and the exact numerical solution. It is possible to see that even using a few CPs, the PINN was able to accurately capture the complex non-linear behavior of the Burgers' equation, which presents the formation of an accentuated internal layer around t = 0.4, which is generally difficult to solve with traditional NM and requires a laborious spatio-temporal discretization of the equation. In this scenario, we can directly address the nonlinear problem without needing to commit to any prior assumptions, linearization, or local time step.

Due to the number of CPs being constant even when the size of the dataset varies, great variation in the discovery of parameters in the PINN model is not expected. In the case of the NMs seen in Tables 3.3 and 3.4, the variation is expected because the NM uses the complete dataset (does not use CP). In addition, there is also variation related to each NM algorithm. For the Table 3.3 with lower viscosity, it is possible to observe that for 128x64 (total of 8,192 data points) the accuracy of the NM was not good, unlike the PINN model which obtained good accuracy even with a smaller amount (2000 CPs) of training data. As for Table 3.4 with higher viscosity, it is possible to observe that for 128x64 the NM showed better accuracy, especially FROLS and SR3, approaching the PINN model.

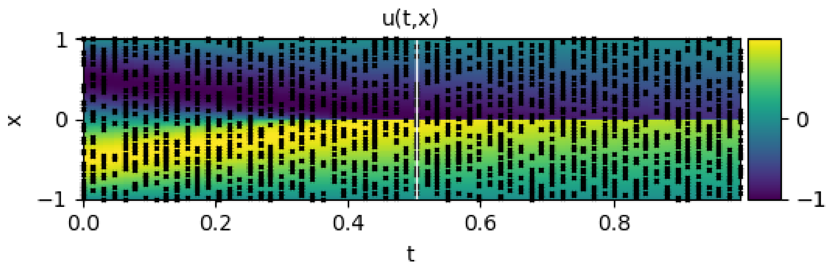

Figure 3.1 - PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 128x64.

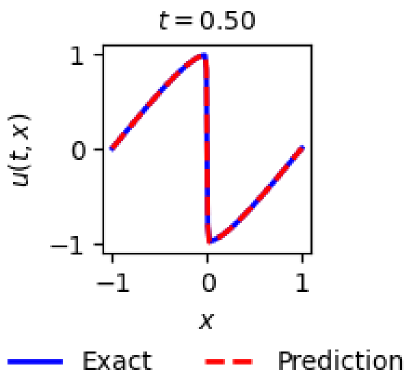

Figure 3.2 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 128x64).

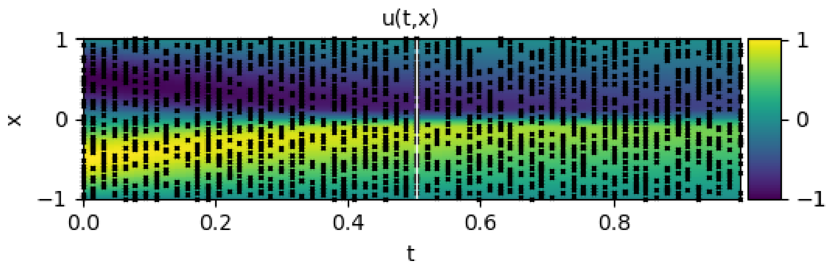

Figure 3.3 - PINN predicted solution $u(t, x)$ for viscosity $0.1/\pi$ and problem size 128x64.

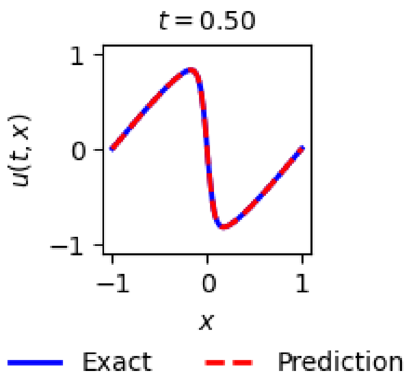

Figure 3.4 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 128x64).

3.1.1.2 Problem Size 256x128

The Figures 3.5, 3.6, 3.7 and 3.8 show, for the problem size 256x128, the solutions for the viscosities $0.01/\pi$ and $0.1/\pi$. Much of what was presented in the previous section also applies to this section. In Figures 3.5 and 3.7 one can initially observe a better distribution of CPs in a more random way, due to the fact that the set of data to be larger and the size of the CP set remains at 2,000 CPs, allowing the CP to be distributed a little better in the graph. Figures 3.6 and 3.8 show the same behavior as the previous subsection, even because the amount of CP is the same. For the Table 3.3 with lower viscosity, and 256x128 problem size, the NMs were not accurate. For low viscosity, in most cases, regardless of the dataset, the PINN model showed better accuracy. For the Table 3.4 with higher viscosity, and 256x64 problem size, most of the NMs behaved well, differently from what occurs in the case with lower viscosity.

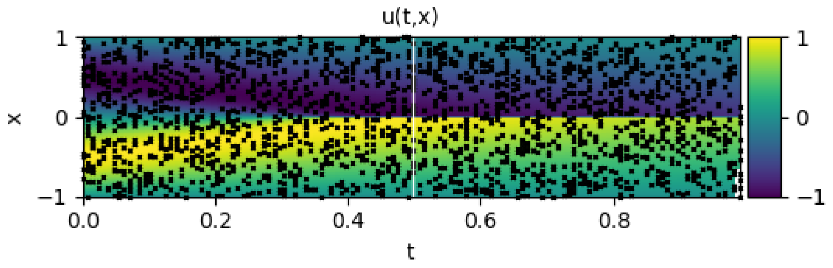

Figure 3.5 - PINN predicted solution $u(t,x)$ for viscosity $0.01/\pi$ and problem size 256x128.

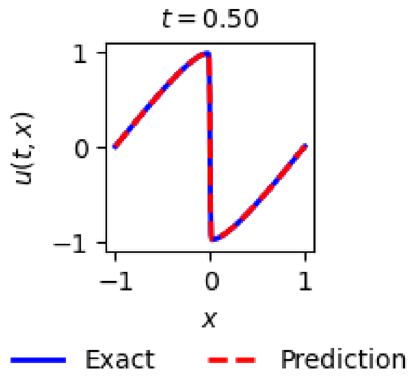

Figure 3.6 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 256x128).

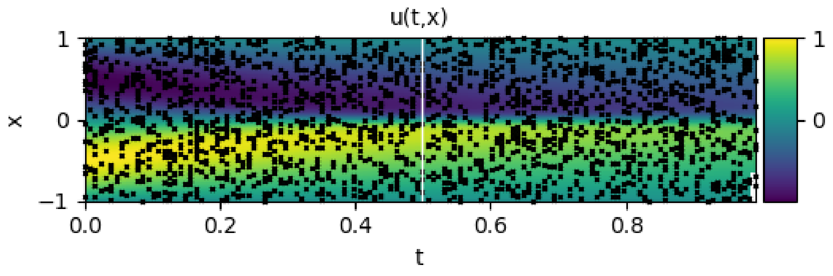

Figure 3.7 - PINN predicted solution $u(t, x)$ for viscosity $0.1/\pi$ and problem size 256x128.

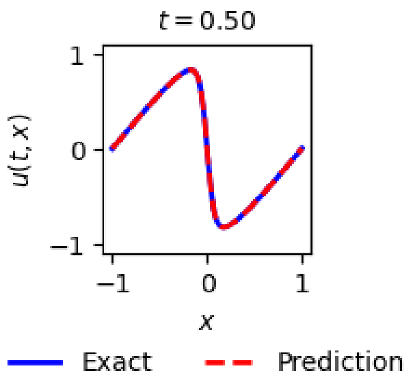

Figure 3.8 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 256x128).

3.1.1.3 Problem Size 512x256

The Figures 3.9, 3.10, 3.11 and 3.12 show, for the problem size 512x256, the same behavior as the previous subsection. For Tables 3.3 and 3.4, 512x256 problem size, the NMs showed similar behavior to the previous section, with the SSR method standing out.

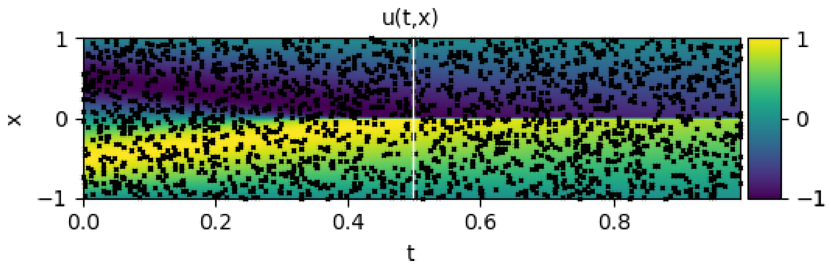

Figure 3.9 - PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 512x256.

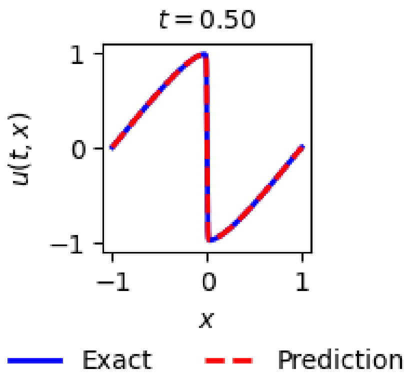

Figure 3.10 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 512x256).

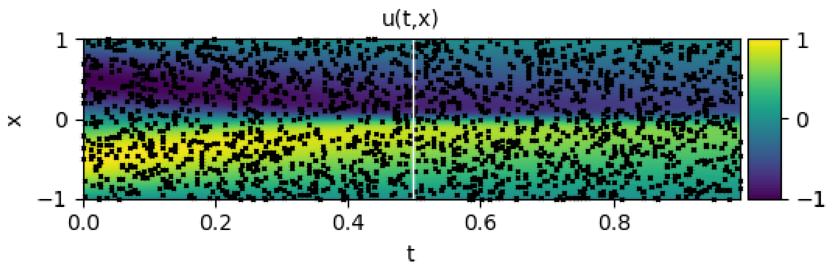

Figure 3.11 - PINN predicted solution $u(t, x)$ for viscosity $0.1/\pi$ and problem size 512x256.

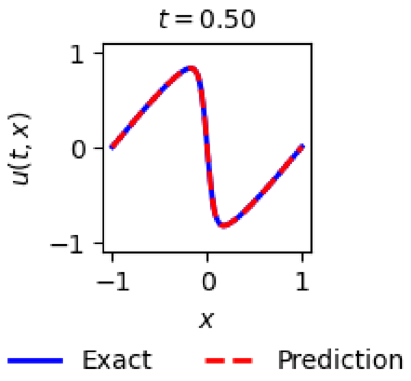

Figure 3.12 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.1/\pi$ and problem size of 512x256).

3.1.2 Evaluation of PINN Hyperparameters and CP Size

PINN processing time and accuracy is evaluated using the model generated with different sets of hyperparameters and CP set sizes in experiments executed on SDumont. Resulting predicted 1D fields are presented in figures, which are similar to those of the previous section. Following, PINN accuracy and processing times are evaluated as a function of the number of neurons per hidden layer, the number of hidden layers, and also the size of the CP set.

3.1.2.1 Visual Assessment of PINN

In this section, the network architecture is the same employed for the preceding comparisons between PINN and SINDy implementations (single input and output layers and 4 hidden layers with 20 neurons each). A visual assessment of predictive accuracy for problem size 256x100 and fluid viscosity of 0.01 is shown in Figure 3.13, with time t on the horizontal axis and spatial coordinate x on the vertical axis. The color scale refers to the velocity $u(t, x)$. The black dots on the graph represent the 2,000 CPs randomly generated from the dataset and used for training. The Figure 3.14 shows a specific snapshot at $t = 0.5$, where it is possible to observe the overlapping solutions for PINN and GQM (exact). For this specific result, the equation obtained by PINN is $u_t + 0.99967uu_x - 0.0030988u_{xx} = 0$, while the exact PDE is $u_t + uu_x - 0, 0031831u_{xx} = 0$. Thus, the PINN could identify the underlying PDE with good accuracy.

3.1.2.2 Influence of the Layers and Neurons in the PINN

For the results presented below, the hyperparameters N_l (number of hidden layers) and N_{le} (number of neurons per hidden layer) were varied, as well as the number of CPs for training. The Table 3.5 shows the relative L2 errors and training times of the PINN, for different hyperparameters used: 10, 15, 20, 25, and 30 neurons per hidden layer, and 1, 2, 4, 6, and 8 hidden layers. The number of CPs was set at 2,000. All values shown here are the average of 3 runs. In this table it is possible to observe that there is a tendency for the best values to be concentrated in the center, probably because there is a problem of underfitting or overfitting in the values at the edges of the table. For future work, it would be interesting to better evaluate this behavior. One of the highlights is that the smallest error is obtained with 6 hidden layers, not 8. In this specific case, increasing the number of layers not only does not increase accuracy, it also worsens performance.

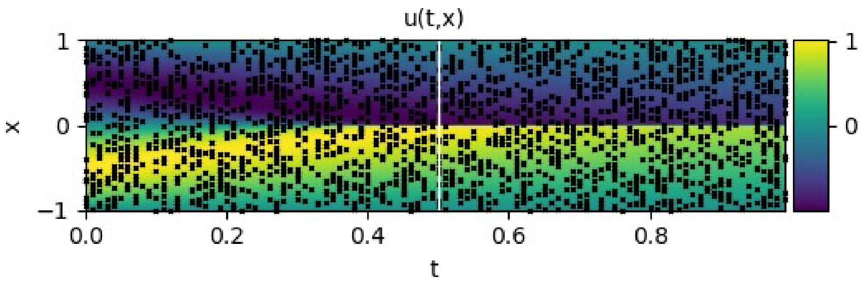

Figure 3.13 - PINN predicted solution $u(t, x)$ for viscosity $0.01/\pi$ and problem size 256x100 (black dots denote the 2,000 randomly assigned CPs).

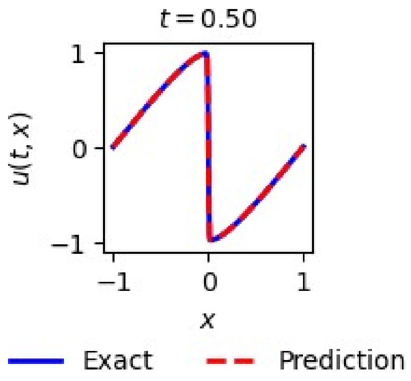

Figure 3.14 - Comparison of the solutions obtained by PINN (in red) and the exact numerical solution (in blue) for the $t = 0.5$ snapshot (viscosity of $0.01/\pi$ and problem size of 256x100).

The Figure 3.15 shows that the error for 1 hidden layer is high compared to the other number of layers. For 2 layers, there is a significant improvement in accuracy. For 4, 6, and 8 the gain in precision is not that great, but the curves are similar and are in the region of greater accuracy, showing that they would be the best choices.

The Figure 3.16 shows, for 4 hidden layers, a tendency to describe a curve that resembles a parabolic, with a minimum processing time of 20 neurons per hidden layer. This is probably due to the problem of underfitting and overfitting occurring at the beginning and at the end of the curve. Once again, for future work, it would be interesting to better evaluate this behavior.

3.1.2.3 Influence of Neurons and the CP in the PINN

The Table 3.6 shows the relative L2 errors and training times of the PINN, for different hyperparameters and number of CP: 10, 15, 20, 25, and 30 neurons per hidden layer, and 400, 800, 1200, 1600, and 2000 CP. The number of layers was set at 8. All values shown here are the average of 3 runs. In this table, as in the previous one, it is possible to observe that there is a tendency for the best values

Hidden layers	Number of neurons per hidden layer				
	10	15	20	25	30
	Relative L2 Error (%)				
1	18.54	17.77	17.80	17.53	17.47
2	3.70	2.52	1.52	1.77	1.59
4	0.30	0.41	0.44	0.39	0.16
6	0.22	0.19	0.10	0.18	0.17
8	0.26	0.13	0.19	0.16	0.23
	Training - processing time (seconds)				
1	4.2	5.4	5.0	21.7	9.4
2	35.9	51.8	39.3	55.5	70.7
4	51.2	43.1	33.4	40.9	47.4
6	59.7	40.3	42.5	35.2	38.6
8	58.7	60.0	58.6	54.4	84.5

Table 3.5 - Relative L2 errors and DNN training times for different number of neurons and hidden layers. On the color scale, the best values are highlighted in green. The simulation ran on the SDumont.

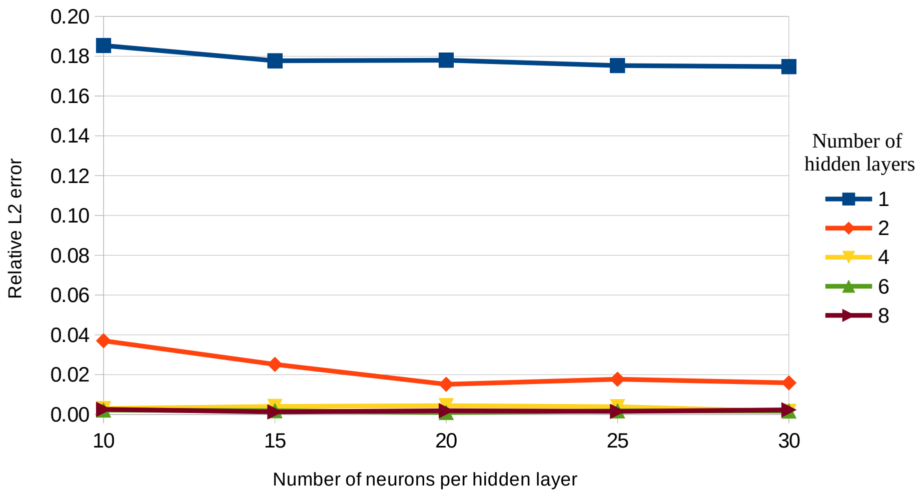

Figure 3.15 - Relative L2 error (%) in function of number of neurons and hidden layers. The simulation ran on the SDumont.

to be concentrated in the center, probably because the problem of underfitting or overfitting is occurring in the values at the edges of the table. Once again, for future work, it would be interesting to better evaluate this behavior. One of the highlights is that considering the smallest error and the shortest processing time, the best dataset size is 1600, and the best number of neurons per hidden layer is 20.

The Figure 3.18 shows for most curves a tendency to describe a curve that resembles

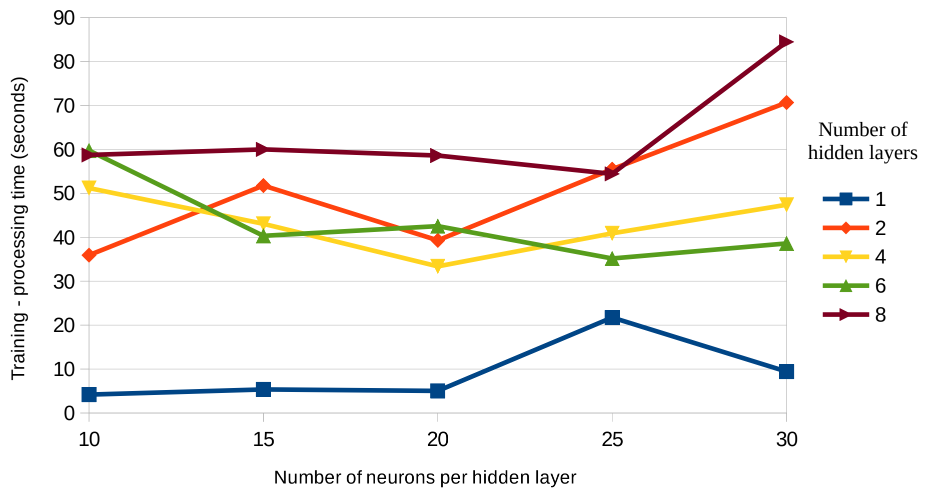

Figure 3.16 - Processing times (seconds) in function of number of neurons and hidden layers. The simulation ran on the SDumont.

a parabolic, probably due to the problem of underfitting and overfitting occurring at the beginning and end of the curve. Once again, for future work, it would be interesting to better evaluate this behavior. The shortest processing time occurs for 15 neurons per hidden layer, and 1600 CPs.

The Figure 3.17 shows that the error for 400 CPs is high compared to the others. 800 CPs presents a significant improvement in accuracy, and the other curves are relatively close, not presenting such a relative large accuracy gain.

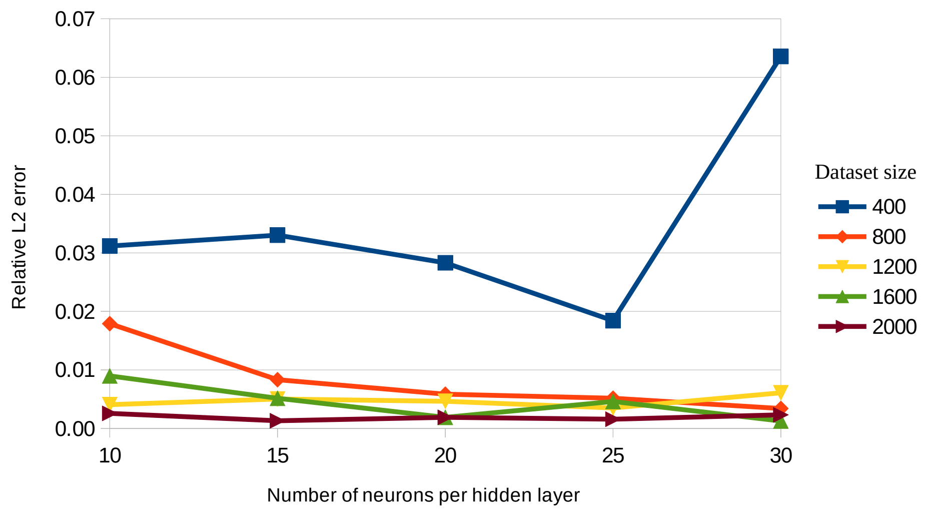

Figure 3.17 - Relative L2 error (%) in function of number of neurons and dataset size. The number of hidden layers is set to 8. The simulation ran on the SDumont.

Dataset size	Number of neurons per hidden layer				
	10	15	20	25	30
	Relative L2 Error (%)				
400	3.12	3.30	2.83	1.84	6.36
800	1.79	0.83	0.59	0.52	0.34
1200	0.41	0.50	0.46	0.35	0.61
1600	0.90	0.51	0.19	0.46	0.13
2000	0.26	0.13	0.19	0.16	0.23
	Training - processing time (seconds)				
400	57.9	82.3	83.3	59.8	58.6
800	79.7	53.5	63.2	45.0	63.0
1200	63.7	52.2	43.8	42.1	56.8
1600	59.9	27.5	45.3	46.5	56.4
2000	58.7	60.0	58.6	54.4	84.5

Table 3.6 - Relative L2 errors and DNN training times for different number of neurons and dataset size. The number of hidden layers is set to 8. On the color scale, the best values are highlighted in green. The simulation ran on the SDumont.

3.1.2.4 PINN Prediction Times

Table 3.7 shows the prediction times of the trained PINN model in function of different number of hidden layers (1, 4, 8) and different numbers of neurons per hidden layer (10, 20, 30). As expected, such times are very close, differently from the corresponding training times, which are higher and differ too much. For instance, in the case of 4 hidden layers and 20 neurons per layer, training time was 33.4 s, while prediction time was only 0.724 s, or 2.2%. This is a common issue with neural networks, showing the convenience of avoiding re-training the network whenever possible.

Number of hidden layers	Number of neurons per hidden layer		
	10	20	30
1	0.647	0.636	0.675
4	0.704	0.724	0.705
8	1.092	0.867	0.789

Table 3.7 - Prediction times for different number of neurons and hidden layers. On the color scale, the best values are highlighted in green. The simulation ran on the SDumont.

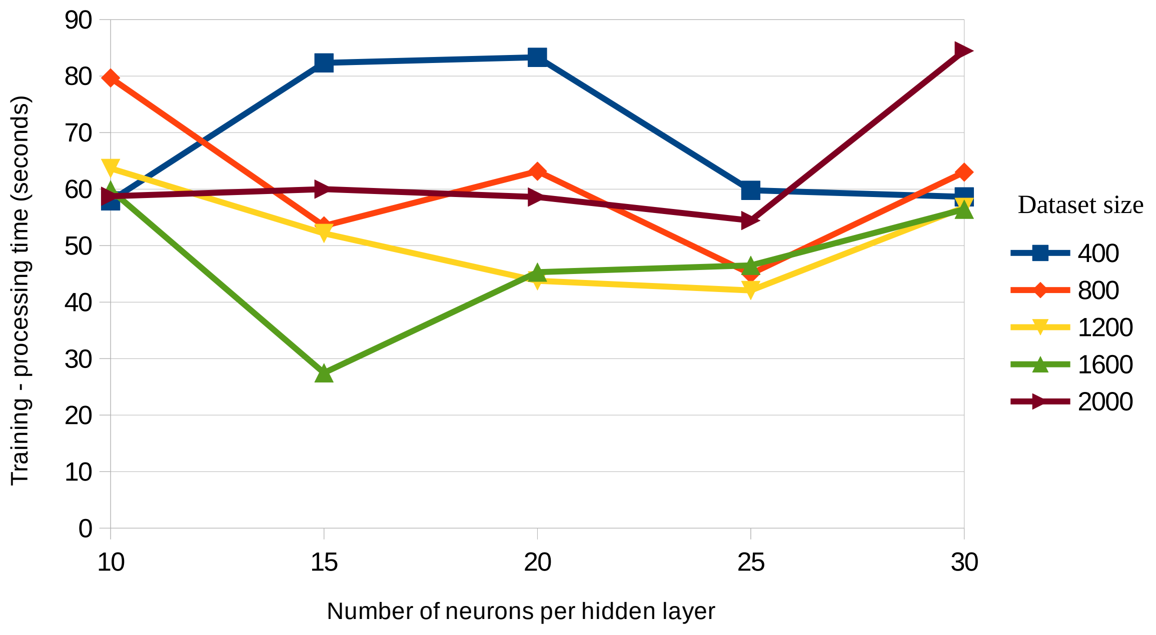

Figure 3.18 - Processing times (seconds) in function of number of neurons and dataset size. The number of hidden layers is set to 8. The simulation ran on the SDumont.

3.2 DNN-Based Gas-Optics Implementation

The gas-optical system of the radiation module was implemented using DNN, and the results were compared to those of the implementation using the traditional numerical model. TensorFlow was used to train the DNN, using a dataset derived from various sources. Gprof was used to evaluate the performance of the ecRad versions.

The result of eCrad running offline with example data is shown in Tables 3.8 and 3.9. The first table shows the performance analysis of the offline ecRad example with the original numerical scheme, using the gprof tool. The second table shows the gas-optical DNN version analysis. Times are for one run. The total execution time for the numerical version was 1.55 s, and the time for the DNN version was 1.50 s, showing a performance gain.

The "radiation_ifs_rrtm" is the interface to IFS implementation of RRTM-G and "gas_optics" is the gas absorption model implementation. "radiation_tripleclouds" is the implementation of the "Tripleclouds" SW and LW solver. "radiation_aerosol_-optics" is the implementation of the aerosol optical properties of the RRTM model. "radiation_two_fluxes" is the implementation of the two-flux approximation that simplifies radiative transfer calculations by considering only two primary radiation fluxes, upward and downward irradiance. Comparing the numerical implementation

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	routine
17.42	0.27	0.27	12	22.50	34.06	CloudsSW
12.90	0.47	0.20	12	16.67	49.17	GasOptics
12.90	0.67	0.20	12	16.67	30.10	CloudsLW
9.68	0.82	0.15	11	13.64	13.64	Aerosol
7.74	0.94	0.12	4817	0.02	0.02	TransSW

Table 3.8 - The performance analysis using the gprof tool of the offline ecRad example with the original numerical gas-optical scheme. The "routine" column is described in the Table 3.10. The remaining columns are the standard output of gprof. The total execution time is 1.55 s. Times are for one run.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	routine
16.67	0.25	0.25	12	20.83	50.83	GasOptics
16.67	0.50	0.25	12	20.83	40.00	CloudsSW
14.00	0.71	0.21	832	0.25	0.25	TransSW
9.33	0.85	0.14	12	11.67	11.67	Aerosol
6.00	0.94	0.09	12	7.50	20.00	CloudsLW

Table 3.9 - The performance analysis using the gprof tool of the offline ecRad example with the DNN gas-optical scheme. The "routine" column is described in the Table 3.10. The remaining columns are the standard output of gprof. The total execution time is 1.50 s. Times are for one run.

with that using DNN, a point that stands out is the performance increase of the "CloudsLW" LW (thermal-infrared) radiation solver Tripleclouds, which focuses on shallow cumulus clouds and aims to accurately represent the horizontal heterogeneity of subgrid clouds.

In terms of accuracy, the work of Ukkonen and Hogan (2023) shows in the results that the DNN-based model is safe and suitable for use in operational climate and meteorological models. Independent validation of DNN gas optics models was performed using data and methods from CKDMIP (HOGAN; MATRICARDI, 2020), using a distinct dataset that was not used in training. The DNN has close accuracy to the scaled-down RRTMGP numerical method used for training, notably in terms of warming rates, and that in both cases it is possible to observe improved bias and RMSE in upward fluxes when employing the DNN. The results for the three CKDMIP concentration scenarios (glacial maximum, pre-industrial and future) are

routine	name
CloudsSW	___radiation_tripleclouds_sw_MOD_solver_tripleclouds_sw
GasOptics	___radiation_ifs_rrtm_MOD_gas_optics
CloudsLW	___radiation_tripleclouds_lw_MOD_solver_tripleclouds_lw
Aerosol	___radiation_aerosol_optics_MOD_add_aerosol_optics
TransSW	___radiation_two_stream_MOD_calc_ref_trans_sw

Table 3.10 - Complement of the Tables 3.8 and 3.9 showing the names of the source files and the names of the routines (gprof standard).

comparable, with the DNN gas optics producing higher upward fluxes and similar warming rates. These preliminary results show a promising path for PIML research.

4 THESIS PROPOSAL

Physics-Informed Machine Learning (PIML) is a new paradigm and hot topic of research that combines data-driven and physics-based strategies, which started with Physics-Informed Neural Networks (PINNs), and is currently being expanded by further approaches.

This thesis aims to replace the ecRad radiation module of the Integrated Forecasting System (IFS), the operational weather and climate model of the European Centre for Medium-Range Weather Forecasts (ECMWF). Despite its influence for the atmospheric simulation, the current numerical ecRad module is very processing-demanding. Therefore, it is not usually executed for every timestep and grid point (in latitude and longitude) of the IFS model. Following the current trend in meteorological centers, it is expected to replace numerical modules that are part of the microphysics of weather and climate forecast models by AI-based implementations such as PIML-based.

In a first non-PIML attempt of the proposed thesis research, a Deep Neural Network (DNN) implementation of the gas-optical scheme of the radiation module was shown here. The original numerical code (RRTMGP) was replaced by a DNN model. The kernel of the RRTMGP scheme was written in Fortran 90, and performs a 3D linear interpolation of the optical depth of the atmosphere for the considered 2D grid point using a lookup table indexing temperatures, pressures and mixing fractions. The DNN implementation of the gas-optical scheme employed the TensorFlow library in the Python environment to perform training and validation of a neural network using a known dataset. The resulting trained model is written to disk to be used later by the Fortran 90 DNN implementation embedded in the radiation module. Performance and accuracy results were discussed, using the ecRad modules results as reference.

As already mentioned, this thesis proposes an incremental approach for the AI-based radiation module implementations, starting with DNN, followed by PINN, and then other PIMLs. The simple no-PIML DNN approach was already implemented for an example test case, as shown here, as a starting point. Further tests involving DNN-only implementations will come after, and PINN-based and other PIML-based are certainly foreseen. As previoulsy mentioned, the resulting future PIML-based implementation of the ecRad radiation module can be partially employed in the microphysics of the MPAS (Model for Prediction Across Scales) atmospheric model that was chosen for MONAN (Model for Ocean-laNd-Atmosphere predictioN) currently being developed by CPTEC/INPE and other Brazilian institutions.

The following workplan and schedule describe the steps/tasks envisaged in this thesis work,

4.1 Work Plan

Thesis title: Implementation of the ecRad Radiation Module Using Physics Informed Machine Learning

Workplan tasks:

- **Bibliographic research**: review of the literature related to PIML.
- **PIML radiation module**: implementation of a preliminary PINN version of the radiation module, including test of different problem instances, and comparison with the numerical model for numerical differences and computing performance.
- **Optimization of the preliminary PINN-based radiation module**: improvement of the accuracy of the PINN model assuming the numerical radiation module as reference, and using optimization of the neural network architecture and/or of the hyperparameters.
- **Other PIML approaches**: implementation of other PIML approaches (non PINN-based models) and comparison with the optimized PINN model.
- **Further case studies**: use of the PIML implementations (including PINN-based models) in other case studies, with comparisons with the corresponding numerical implementations.
- **Article submission**: submission of articles for conferences and indexed journals.
- **Thesis writing**: writing of the doctoral thesis and presentation slides.

4.2 Schedule

Tasks	2024	2025			2026	
	9-12	1-4	5-8	9-12	1-4	5-8
Bibliographic research	▨	▨	▨	▨	▨	▨
PIML radiation module	▨	▨	▨			
Module optimization			▨	▨		
Other PIML approaches				▨	▨	
Further case studies				▨	▨	▨
Article submission		▨	▨	▨	▨	▨
Thesis writing	▨	▨	▨	▨	▨	▨

Acknowledgment

Authors thank LNCC (National Laboratory for Scientific Computing) for grant 205341 AMPEMI (call 2020-I), which allows access to the Santos Dumont supercomputer (node of the SINAPAD, the Brazilian HPC system). This study was financed in part by the Coordination for the Improvement of Higher Education Personnel (CAPES), Brazil, finance Code 001, and also by the CNPq Project 446053/2023-6. The authors also thank the Brazilian Ministry of Science, Technology and Innovation, and the Brazilian Space Agency.

REFERENCES

BASDEVANT, C.; DEVILLE, M.; HALDENWANG, P.; LACROIX, J.; OUAZZANI, J.; PEYRET, R.; ORLANDI, P.; PATERA, A. Spectral and finite difference solutions of Burgers equation. **Computers & Fluids**, v. 14, p. 23–41, dec. 1986. Available from: http://www.researchgate.net/publication/222935980_Spectral_and_finite_difference_solutions_of_Burgers_equation. 3

BAYDIN, A. G.; PEARLMUTTER, B. A.; RADUL, A. A.; SISKIND, J. M. Automatic differentiation in machine learning: A survey. **Journal of machine learning research**, v. 18, n. 153, p. 1–43, 2018. Available from: https://www.jmlr.org/papers/v18/17-468.html. 13

BILLINGS, S. Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains. **Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains**, 2013. ISSN 9781119943594. 17

BLACK, P. E. **Greedy Algorithm**. 2005. Available from: https://xlinux.nist.gov/dads//HTML/greedyalgo.html. 18

BONINSEGNA, L.; NÜSKE, F.; CLEMENTI, C. Sparse learning of stochastic dynamical equations. **The Journal of Chemical Physics**, v. 148, n. 24, p. 241723, 2018. ISSN 0021-9606. Available from: https://doi.org/10.1063/1.5018409. 15, 17

BRUNTON, S. L.; PROCTOR, J. L.; KUTZ, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. **Proceedings of the National Academy of Sciences**, Proceedings of the National Academy of Sciences, v. 113, n. 15, p. 3932–3937, 2016. Available from: https://www.pnas.org/doi/10.1073/pnas.1517384113. 15, 16, 17

BURKARDT, J. **Investigating Uncertain Parameters in the Burgers Equation**. Mathematics Department, Ajou University, Suwon, Korea: [s.n.], 2013. Available from: https://people.sc.fsu.edu/~jburkardt/presentations/burgers_2013_ajou.pdf. 10

CHAMPION, K.; ZHENG, P.; ARAVKIN, A. Y.; BRUNTON, S. L.; KUTZ, J. N. **A Unified Sparse Optimization Framework to Learn Parsimonious Physics-Informed Models from Data**. 2019. Available from: https://arxiv.org/abs/1906.10612v2. 18

CHEVALLIER, F.; MORCRETTE, J.; CHéRUY, F.; SCOTT, N. A. Use of a neural-network-based long-wave radiative-transfer scheme in the ECMWF atmospheric model. **Quart J Royal Meteoro Soc**, v. 126, n. 563, p. 761–776, jan. 2000. ISSN 0035-9009, 1477-870X. {06}. Available from: https://rmets.onlinelibrary.wiley.com/doi/10.1002/qj.49712656318. 7

CUOMO, S.; COLA, V. S. D.; GIAMPAOLO, F.; ROZZA, G.; RAISSI, M.; PICCIALLI, F. Scientific Machine Learning Through Physics–Informed Neural

Networks: Where we are and What's Next. **J Sci Comput**, v. 92, n. 3, p. 88, sep. 2022. ISSN 0885-7474, 1573-7691. Available from: https://link.springer.com/10.1007/s10915-022-01939-z. 3, 4, 6

CURCIC, M. A parallel Fortran framework for neural networks and deep learning. **SIGPLAN Fortran Forum**, v. 38, n. 1, p. 4–21, mar. 2019. ISSN 1061-7264, 1931-1311. Available from: https://dl.acm.org/doi/10.1145/3323057.3323059. 8, 20

de Silva, B. M.; CHAMPION, K.; QUADE, M.; LOISEAU, J.-C.; KUTZ, J. N.; BRUNTON, S. L. **PySINDy: A Python Package for the Sparse Identification of Nonlinear Dynamics from Data**. arXiv, 2020. Available from: http://arxiv.org/abs/2004.08424. 16

E, W.; YU, B. **The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems**. arXiv, sep. 2017. ArXiv:1710.00211 [cs, stat]. Available from: http://arxiv.org/abs/1710.00211. 5

FU, Q.; LIOU, KN. On the correlated k-distribution method for radiative transfer in nonhomogeneous atmospheres. **Journal of Atmospheric Sciences**, v. 49, n. 22, p. 2139–2156, 1992. 19

GOODY, R.; WEST, R.; CHEN, L.; CRISP, D. The correlated-k method for radiation calculations in nonhomogeneous atmospheres. **Journal of Quantitative Spectroscopy and Radiative Transfer**, v. 42, n. 6, p. 539–550, dec. 1989. ISSN 0022-4073. Available from: https://www.sciencedirect.com/science/article/pii/0022407389900447. 18

HOGAN, R.; AHLGRIMM, M.; BELJAARS, A.; Paul Berrisford. Text, **Radiation in Numerical Weather Prediction**. 2017. Available from: https://www.ecmwf.int/en/elibrary/80347-radiation-numerical-weather-prediction. 19

HOGAN, R. J.; BOZZO, A. A Flexible and Efficient Radiation Scheme for the ECMWF Model. **J. Adv. Model. Earth Syst.**, v. 10, n. 8, p. 1990–2008, aug. 2018. ISSN 19422466. Available from: http://doi.wiley.com/10.1029/2018MS001364. 6, 19

HOGAN, R. J.; MATRICARDI, M. Evaluating and improving the treatment of gases in radiation schemes: The Correlated K-Distribution Model Intercomparison Project (CKDMIP). **Geosci. Model Dev.**, v. 13, n. 12, p. 6501–6521, dec. 2020. ISSN 1991-9603. Available from: https://gmd.copernicus.org/articles/13/6501/2020/. 22, 40

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. Available from: https://www.sciencedirect.com/science/article/pii/0893608089900208. 3

INNESS, A.; ADES, M.; Agustí-Panareda, A.; BARRÉ, J.; BENEDICTOW, A.; BLECHSCHMIDT, A.-M.; DOMINGUEZ, J. J.; ENGELEN, R.; ESKES, H.; FLEMMING, J.; HUIJNEN, V.; JONES, L.; KIPLING, Z.; MASSART, S.;

PARRINGTON, M.; PEUCH, V.-H.; RAZINGER, M.; REMY, S.; SCHULZ, M.; SUTTIE, M. The CAMS reanalysis of atmospheric composition. **Atmos. Chem. Phys.**, v. 19, n. 6, p. 3515–3556, mar. 2019. ISSN 1680-7324. Available from: https://acp.copernicus.org/articles/19/3515/2019/. 22

JAGTAP, A.; KHARAZMI, E.; KARNIADAKIS, G. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. **Computer Methods in Applied Mechanics and Engineering**, v. 365, p. 113028, 2020. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0045782520302127. 5

KAPTANOGLU, A. A.; de Silva, B. M.; FASEL, U.; KAHEMAN, K.; GOLDSCHMIDT, A. J.; CALLAHAM, J. L.; DELAHUNT, C. B.; NICOLAOU, Z. G.; CHAMPION, K.; LOISEAU, J.-C.; KUTZ, J. N.; BRUNTON, S. L. PySINDy: A comprehensive Python package for robust sparse system identification. **Journal of Open Source Software**, v. 7, n. 69, p. 3994, 2022. ISSN 2475-9066. Available from: http://arxiv.org/abs/2111.08481. 16

KARNIADAKIS, G. E.; KEVREKIDIS, I. G.; LU, L.; PERDIKARIS, P.; WANG, S.; YANG, L. Physics-informed Machine Learning. **Nat Rev Phys**, v. 3, n. 6, p. 422–440, may 2021. ISSN 2522-5820. Available from: https://www.nature.com/articles/s42254-021-00314-5. 1

KHARAZMI, E.; ZHANG, Z.; KARNIADAKIS, G. E. hp-VPINNs: Variational Physics-Informed Neural Networks With Domain Decomposition. **Computer Methods in Applied Mechanics and Engineering**, v. 374, p. 113547, feb. 2021. ISSN 00457825. ArXiv:2003.05385 [cs, math]. Available from: http://arxiv.org/abs/2003.05385. 6

KIM, S.; KIM, I.; LEE, J.; LEE, S. Knowledge Integration into deep learning in dynamical systems: an overview and taxonomy. **Journal of Mechanical Science and Technology**, v. 35, 2021. Available from: https://link.springer.com/article/10.1007/s12206-021-0342-5. 5

KOEHRSEN, W. Overfitting vs. underfitting: A complete example. **Towards Data Science**, v. 405, 2018. Available from: http://www.pstu.ac.bd/files/materials/1566949131.pdf. 15

KRASNOPOLSKY, V. M. **The Application of Neural Networks in the Earth System Sciences: Neural Networks Emulations for Complex Multidimensional Mappings**. Dordrecht: Springer Netherlands, 2013. (Atmospheric and Oceanographic Sciences Library, v. 46). ISBN 978-94-007-6072-1 978-94-007-6073-8. Available from: https://link.springer.com/10.1007/978-94-007-6073-8. 8

KRASNOPOLSKY, V. M.; FOX-RABINOVITZ, M. S. A new synergetic paradigm in environmental numerical modeling: Hybrid models combining deterministic and machine learning components. **Ecological Modelling**, v. 191, n. 1, p. 5–18, jan. 2006. ISSN 0304-3800. {04}. Available from: https://www.sciencedirect.com/science/article/pii/S0304380005003455. 7

KRASNOPOLSKY, V. M.; Fox-Rabinovitz, M. S.; BELOCHITSKI, A. A. Decadal Climate Simulations Using Accurate and Fast Neural Network Emulation of Full, Longwave and Shortwave, Radiation. **Monthly Weather Review**, v. 136, n. 10, p. 3683–3695, oct. 2008. ISSN 1520-0493, 0027-0644. Available from: https://journals.ametsoc.org/view/journals/mwre/136/10/2008mwr2385.1.xml. 8

LIU, D.; WANG, Y. A Dual-Dimer Method for Training Physics-Constrained Neural Networks with Minimax Architecture. **Neural Networks**, v. 136, p. 112–125, apr. 2021. ISSN 08936080. ArXiv:2005.00615 [cs, stat]. Available from: http://arxiv.org/abs/2005.00615. 5

LU, L.; PESTOURIE, R.; YAO, W.; WANG, Z.; VERDUGO, F.; JOHNSON, S. G. **Physics-Informed Neural Networks with Hard Constraints for Inverse Design**. arXiv, feb. 2021. Available from: http://arxiv.org/abs/2102.04626. 5

MARKIDIS, S. The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers? **Frontiers in Big Data**, Frontiers, v. 4, 2021. ISSN 2624-909X. Available from: https://www.frontiersin.org/articles/10.3389/fdata.2021.669097. 6

MENG, X.; LI, Z.; ZHANG, D.; KARNIADAKIS, G. E. PPINN: Parareal Physics-Informed Neural Network for time-dependent PDEs. **Computer Methods in Applied Mechanics and Engineering**, v. 370, p. 113250, 2020. ISSN 00457825. ArXiv:1909.10145 [physics, stat]. Available from: http://arxiv.org/abs/1909.10145. vii, 5, 14

MEYER, D. Machine Learning Emulators for Numerical Weather Prediction—Applications to Parametrization Schemes. 2022. 7

NANDI, T.; HENNIGH, O.; NABIAN, M.; LIU, Y.; WOO, M.; JORDAN, T.; SHAHNAM, M.; SYAMLAL, M.; GUENTHER, C.; VANESSENDELFT, D. **Progress Towards Solving High Reynolds Number Reacting Flows in SimNet**. [S.l.], 2021. Available from: https://www.osti.gov/servlets/purl/1846970. 6

NANDWANI, Y.; PATHAK, A.; SINGLA, P. A Primal-Dual Formulation for Deep Learning with Constraints. 2019. 5

NGHIEM, T. X.; DRGOÑA, J.; JONES, C.; NAGY, Z.; SCHWAN, R.; DEY, B.; CHAKRABARTY, A.; CAIRANO, S. D.; PAULSON, J. A.; CARRON, A.; ZEILINGER, M. N.; CORTEZ, W. S.; VRABIE, D. L. **Physics-Informed Machine Learning for Modeling and Control of Dynamical Systems**. arXiv, jun. 2023. Available from: http://arxiv.org/abs/2306.13867. 1

PINCUS, R.; FORSTER, P. M.; STEVENS, B. The Radiative Forcing Model Intercomparison Project (RFMIP): Experimental protocol for CMIP6. **Geosci. Model Dev.**, v. 9, n. 9, p. 3447–3460, sep. 2016. ISSN 1991-9603. Available from: https://gmd.copernicus.org/articles/9/3447/2016/. 22

PINCUS, R.; MLAWER, E. J.; DELAMERE, J. S. Balancing Accuracy, Efficiency, and Flexibility in Radiation Calculations for Dynamical Models. **Journal of Advances in Modeling Earth Systems**, v. 11, n. 10, p. 3074–3089, 2019. ISSN 1942-2466. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1029/2019MS001621. 19, 20, 21

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. **Journal of Computational physics**, Elsevier, v. 378, p. 686–707, 2019. Available from: https://www.sciencedirect.com/science/article/pii/S0021999118307125. 4, 5, 10, 13

SERGEEV, A.; BALSO, M. D. **Horovod: Fast and Easy Distributed Deep Learning in TensorFlow**. arXiv, 2018. Available from: http://arxiv.org/abs/1802.05799. 6

SHARMA, P.; EVANS, L.; TINDALL, M.; NITHIARASU, P. Stiff-PDEs and Physics-Informed Neural Networks. **Arch Computat Methods Eng**, v. 30, n. 5, p. 2929–2958, jun. 2023. ISSN 1886-1784. Available from: https://doi.org/10.1007/s11831-023-09890-4. 4

SIRIGNANO, J.; SPILIOPOULOS, K. DGM: A deep learning algorithm for solving partial differential equations. **Journal of Computational Physics**, v. 375, p. 1339–1364, dec. 2018. ISSN 00219991. Available from: http://arxiv.org/abs/1708.07469. 6

SUN, L.; GAO, H.; PAN, S.; WANG, J.-X. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. **Computer Methods in Applied Mechanics and Engineering**, v. 361, p. 112732, apr. 2020. ISSN 00457825. Available from: https://linkinghub.elsevier.com/retrieve/pii/S004578251930622X. 5

TIBSHIRANI, R. Regression Shrinkage and Selection via The Lasso: A Retrospective. **Journal of the Royal Statistical Society Series B: Statistical Methodology**, v. 73, n. 3, p. 273–282, 2011. ISSN 1369-7412. Available from: https://doi.org/10.1111/j.1467-9868.2011.00771.x. 16

TRONCI, E. M.; DOWNEY, A. R.; MEHRJOO, A.; CHOWDHURY, P.; COBLE, D. Physics informed machine learning part I: Different strategies to incorporate physics into engineering problems. In: **Conference Proceedings of the Society for Experimental Mechanics Series**. [S.l.]: Springer Nature Switzerland, 2024. 2

UKKONEN, P.; HOGAN, R. J. Implementation of a machine-learned gas optics parameterization in the ECMWF Integrated Forecasting System: RRTMGP-NN 2.0. **Geoscientific Model Development**, v. 16, n. 11, p. 3241–3261, jun. 2023. ISSN 1991-959X. Available from: https://gmd.copernicus.org/articles/16/3241/2023/. vii, 7, 8, 20, 22, 23, 40

UKKONEN, P.; PINCUS, R.; HOGAN, R. J.; NIELSEN, K. P.; KAAS, E. Accelerating Radiation Computations for Dynamical Models With Targeted Machine Learning and Code Optimization. **Journal of Advances in Modeling Earth Systems**, v. 12, n. 12, p. e2020MS002226, 2020. ISSN 1942-2466. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1029/2020MS002226. ix, 8, 18, 21

VEERMAN, M. A.; PINCUS, R.; STOFFER, R.; van Leeuwen, C. M.; PODAREANU, D.; van Heerwaarden, C. C. Predicting atmospheric optical properties for radiative transfer computations using neural networks. **Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences**, Royal Society, v. 379, n. 2194, p. 20200095, feb. 2021. Available from: https://royalsocietypublishing.org/doi/10.1098/rsta.2020.0095. 7, 8

VLADIMIROVA, M.; ARBEL, J.; MESEJO, P. Bayesian neural networks become heavier-tailed with depth. In: **NeurIPS 2018 - Thirty-second Conference on Neural Information Processing Systems**. Montréal, Canada: [s.n.], 2018. p. 1–7. Available from: https://hal.science/hal-01950658. 4

XU, S.; SUN, Z.; HUANG, R.; DILONG, G.; YANG, G.; JU, S. A practical approach to flow field reconstruction with sparse or incomplete data through physics informed neural network. **Acta Mechanica Sinica**, v. 39, nov. 2022. Available from: https://link.springer.com/article/10.1007/s10409-022-22302-x. 14, 15

YANG, Y.; PERDIKARIS, P. Adversarial Uncertainty Quantification in Physics-Informed Neural Networks. **Journal of Computational Physics**, v. 394, p. 136–152, 2019. ISSN 00219991. ArXiv:1811.04026 [physics, stat]. Available from: http://arxiv.org/abs/1811.04026. 5

YAO, Y.; ZHONG, X.; ZHENG, Y.; WANG, Z. A Physics-Incorporated Deep Learning Framework for Parameterization of Atmospheric Radiative Transfer. **Journal of Advances in Modeling Earth Systems**, v. 15, n. 5, p. e2022MS003445, 2023. ISSN 1942-2466. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1029/2022MS003445. 5

ZHOU, W.; XU, Y. F. **Data-Guided Physics-Informed Neural Networks for Solving Inverse Problems in Partial Differential Equations**. arXiv, jul. 2024. Available from: http://arxiv.org/abs/2407.10836. 3

ZHU, Y.; ZABARAS, N.; KOUTSOURELAKIS, P.-S.; PERDIKARIS, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. **Journal of Computational Physics**, v. 394, p. 56–81, oct. 2019. ISSN 0021-9991. Available from: https://www.sciencedirect.com/science/article/pii/S0021999119303559. 5