Development and Assessment of Machine Learning Techniques for Non-Intrusive Probabilistic Surrogate Modeling of High-Fidelity Nuclear Reactor Simulations

by

Brandon LaFleur

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Nuclear Engineering and Radiological Sciences)
in the University of Michigan
2024

Brandon LaFleur

blafleur@umich.edu

ORCID ID: 0000-0002-6020-6401

DEDICATION

To my son, my wife, my parents, and my siblings

ACKNOWLEDGMENTS

This dissertation represents the culmination of years of dedicated work and collaboration. Many of the individuals who made this work possible and improved its quality are likely unaware of how much I have benefited from their expertise and wisdom, both technically and personally. Each of them played a pivotal role in my success during the process and will no doubt continue to play a role in my future endeavors.

I would like to express my appreciation to my Ph.D. advisor, Professor Annalisa Manera, and to my university committee members, Professor Karthik Duraisamy and Professor Brendan Kochunas. I value their feedback, criticisms, and dedication to my success. I particularly thank Professor Duraisamy, whose methods formed the underlying basis for the work herein. Their role in this process greatly shaped my final product. I also would like to thank Professor Kiedrowski, who was a great help in putting together the final product and who improved almost every aspect of my work. I owe a special debt of gratitude to my laboratory advisor, Dr. Brian Aviles, whose patience and wisdom kept me aligned throughout the entire process. He influenced my growth as a scholar, engineer, and person in more ways than he knows.

I would also like to acknowledge my employer, the Naval Nuclear Laboratory, for their financial support, which opened the initial door for me to pursue this personal goal. I also want to express a special thanks to my many talented coworkers at the Naval Nuclear Laboratory who unknowingly shaped my technical growth and curiosity. They had a permanent impact on me through discussions, teachings, and collaborations. Many of these people were instrumental in expanding my knowledge and fostering my passion for exploration.

Additionally, this research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517.

Finally, I will forever be grateful to my family members. My wife, best friend, and biggest advocate, Lakhena, was always in my corner offering unconditional support. My parents, Jamie and Doug, had complete faith in me, especially at those times when I did not have faith in myself. My siblings, Cameron, Chloe, Kara and Kanika, always cheered me on with their unwavering encouragement. Last but not least, I have deep appreciation for my friends, who were a constant source of joy and levity. All of these people provided me with camaraderie and kept me whole.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ACRONYMS

INL Idaho National Laboratory. 71, 79, 102, 112, 170

KL-divergence Kullback-Leibler divergence. 61, 62, 65, 66, 131

LSTM long short term memory. 10, 39, 96

MAE Mean Absolute Error. 53, 92, 120, 123, 126, 135

MAPE Mean Absolute Percentage Error. 53, 120, 123, 126, 128, 135, 136

MLP multi-layer perceptron. 9, 45, 75, 76, 80

MOC method of characteristics. 16

MOPED MOdel Priors with Empirical Bayes using DNN. 65, 66, 131

MPACT Michigan Parallel Characteristics Transport Code. 12, 14, 16, 20–25, 27, 29, 66, 68, 75, 78–80, 87, 95, 170

MSE Mean Squared Error. 37, 52, 53, 64, 85, 92, 95, 120, 121, 126

NIDS Non-Linear Independent Dual System. 12, 14, 31, 47–52, 66, 68, 73, 76, 87–89, 92, 93, 95, 97, 98, 100, 101, 104, 106, 107, 112, 117, 118, 122–127, 131–134, 137–139, 158–161, 163–167, 170, 172, 197

ORNL Oak Ridge National Laboratory. 16, 24

PCE polynomial chaos expansion. 10

PDE partial differential equation. 6

POD Proper Orthogonal Decomposition. 7–9, 11, 13, 30–35, 39, 43, 66, 158, 167

POD-NN Proper Orthogonal Decomposition neural network. 8, 35, 36, 38, 39, 164

QOI quantity of interest. 3, 7, 11, 33, 55, 56, 70, 167

RIA reactivity insertion accident. 12, 14, 25, 66, 68, 71, 72, 77, 78, 95, 97, 138, 158

ROM reduced-order model. 1–3, 5–15, 29–35, 38–40, 43, 47–49, 52–54, 56–58, 62, 66, 68, 69, 71, 73, 75, 77–80, 82–93, 95–102, 105, 107, 112–116, 118–128, 131–140, 157–168, 170–174, 196, 199

SDF signed distance function. 13, 48–50, 97, 105, 106, 114, 136–139, 158, 165, 166

SVD singular value decomposition. 9, 30, 32, 35, 39, 43

TCAE temporal autoencoder. 9, 45, 74, 76, 80, 81, 89

TML transient multilevel. 22, 23

UQ uncertainty quantification. 10, 11, 54–56, 59, 68, 70, 78, 82, 84–86, 89, 90, 92, 93, 97, 160

VERA Virtual Environment for Reactor Analysis. 27, 68, 78

VI variational inference. 12, 13, 56, 60–62, 64–66, 131, 132, 168

ABSTRACT

With the continuing advancement of computational resources, high-fidelity simulations of neutron transport play an increasingly important role in the design and analysis of nuclear reactor cores. Because of the inherent non-linear interdependency of the flux solution on the coolant properties, neutron transport solvers are often coupled to subchannel thermal-hydraulics or computational fluid dynamic solvers to capture the necessary physics.

The challenges present in numerical simulations required for nuclear reactor design, specifically the high computational cost and dimensionality encountered, are not unique to nuclear engineering. A full-order model is often expensive to evaluate in many engineering disciplines, particularly if the governing equations contain non-linear terms. The discretization of these partial differential equations leads to large systems of coupled equations.

This limitation has led to the development and deployment of reduced-order modeling techniques. For decades, reduced-order models (ROM) have experienced a wide variety of successes in many fields and have been demonstrated for a wide range of applications. These techniques are not typically applied in the nuclear engineering field, particularly in production environments. Importantly, they have not been applied to high-fidelity multiphysics simulations of nuclear reactors.

This work investigates the current state-of-the-art of ROMs and investigates their applicability to commonly encountered nuclear reactor design applications. Specifically, multi-stage convolutional neural network-based ROMs and the newly proposed Non-

Linear Independent Dual System (NIDS) algorithm. The following chapters contain a discussion of traditional intrusive projection-based ROMs and works its way to non-intrusive neural network-based ROM methods. This work includes discussions on the theory, merits, challenges, and limitations associated with various methodologies. Furthermore, the uncertainty associated with reducing high-dimensional multiphysics problems is quantified using probabilistic modeling techniques combined with neural network-based ROMs. Specifically, variational inference approaches were applied to the ROMs.

Using current state-of-the-art methods in non-intrusive ROMs, coupled with variational inference methods, ROMs are developed for two representative classes of nuclear engineering problems. The first application is a coupled MPACT/CTF model representing a single-assembly configuration experiencing a reactivity insertion accident via rod ejection. The state variables of interest are time-dependent relative pin powers. The second application is a 3D quarter-core MC21 depletion model. The state variables of interest are isotopic depletion trajectories. The performance of associated ROMs are assessed to evaluate the efficacy of using non-intrusive neural network-based ROMs in production design environments. In all contexts analyzed, NIDS methods are shown to outperform convolutional neural network-based algorithms for nuclear engineering applications and perform to a level acceptable in certain production design environments. Finally, a new Python package, Parody, is introduced to facilitate the assessment of ROMs and its potential use for further study of ROMs for nuclear applications is presented and discussed.

CHAPTER 1

Introduction

This chapter first motivates the interest in studying reduced-order modeling in the context of nuclear engineering. Then, it explains how similar challenges existing in the computational fluid dynamics (CFD) community have benefited from the development of capable reduced-order model (ROM) techniques. Next, this chapter will explore what work has been done in the radiation transport community using ROM techniques and how they differ from what has been accomplished in other communities. This includes a brief historical review of these ROM methods and how they have evolved over the last decade within and without the nuclear industry. Then, a discussion follows of the high level approaches that are explored in this work and what representative nuclear reactor analysis problems this work applies the ROM techniques to. Finally, the contributions this work represents along with the organization of the thesis will be summarized.

1.1 Motivation

1.1.1 Nuclear Reactor Design Challenges

Nuclear engineering, in the context of reactor design, deals with a wide range of physics processes and associated methodologies to model them. Primary physics of interest include neutron transport, reactor plant kinetics, material science, fluid dynamics, heat transfer, and structural analysis. A key trade-off present in nuclear engineering (as it is in most engineering disciplines) is developing physics solvers with the right balance of accuracy and computational cost.

Some of the more common types of analysis performed by a nuclear designer include optimization, uncertainty quantification, design space exploration, plant analysis, and operational and manufacturing support. The list below highlights some classes of problems nuclear designers often find themselves solving:

1. Finding optimal physical arrangements of structural material, reactor coolant, fuel, and neutron poisons to support operational goals by performing core design optimization.
2. Predicting plant dynamics during routine operation using plant analysis codes.
3. Quantifying the impact of deviations from assumed tolerances during the manufacturing of reactor components.
4. Supporting transport or long-term storage of potentially critical nuclear material by doing critical configuration optimizations.
5. Determining what level of fidelity is needed to make engineering decisions by performing physics down-selection analyses.
6. Analyzing spaces where radioactive material and human workers are present to support shielding applications.
7. Analyzing undesirable events to ensure the safety of the public and employees who interact with nuclear technology in the event of an accident.

Each of these analyses typically require many calculations of very similar physical systems under slightly perturbed initial conditions. These analyses are critical to evaluating the safety and performance of nuclear systems, as well as for designing new and more efficient reactors. By reducing the computational cost of simulations, ROMs can enable engineers to perform more detailed and thorough analyses of nuclear systems. Before exploring how to reduce their cost, we first discuss the variety of ways that neutron transport is modeled in practice.

In reactor design, neutron transport theory is the fundamental physics that describes the behavior of neutrons in a nuclear reactor. In Reference [1], Sanchez provides a discussion on the derivation of approaches used in general-purpose production calculations. Larson [2] offers a review of the advancements made in the previous 30 years for a variety of simulation techniques in the radiation transport community. Larsen et al. [2] extends the history that Lawrence [3] wrote, which provides a similar summary for common nodal methods and their histories from the perspective of the industry in 1985. Cho [4] presents a brief overview of the steps involved in typical modern reactor design and analysis, and provides derivations of commonly used multigroup transport equations and nodal methods.

In its full implementation, the neutron transport equation is 7 dimensional - 3 dimensions in space, 2 in direction, 1 in energy, and 1 in time. There is a large body of work devoted to discretizing and solving this equation along all dimensions. As a result, there are many methods available that describe the spatial and temporal evolution of neutrons within a reactor. These include point kinetics, diffusion theory, deterministic transport, and Monte Carlo methods. Each solver has its own strengths and weaknesses, which can impact both the accuracy and computational cost of the simulation. However, all are used in industry during some phase of the design and operational assessment phases. Which method that is deployed depends heavily on the problem at hand. For example, the point kinetics equations are relatively cheap computationally and are often used for transient analysis of reactor plants undergoing a plant evolution due to the fact that they are quick to solve and can be tuned to capture global power output as a function of time quite well. However they provide no information on the spatially dependent flux or power generation within a reactor. Therefore this approach, although used effectively throughout the industry for safety analysis, will not be the subject of the work herein that is focused on full-field ROM predictions of 3D quantity of interest (QOI)s.

Deterministic transport methods that solve the neutron transport equation directly, or the less accurate but cheaper diffusion equation, are used when static analyses are required that seek information on spatially dependent information within a reactor (such as power generation, fluence, decay heat, etc.) and can provide accurate solutions to the neutron transport equation. Diffusion theory assumes that the angular flux is well-approximated as a linear function of the neutron direction, which allows for a less expensive version of the transport equation to emerge at the cost of less accuracy. Monte Carlo methods represent the highest level of accuracy in practice, and simulate the underlying behavior of radiation which the transport equation describes the average behavior of. Monte Carlo methods require no discretization of the dimensions of the neutron transport equation, and can capture the full range of neutron behavior in a reactor based on the average behavior of simulated random walks. However, they are the most computationally intensive in practical environments and can be prohibitively expensive for many applications. Figure 1.1 shows a cartoon that depicts the relative cost and accuracy of the main classes of techniques used in industry to model reactor and/or plant behavior.

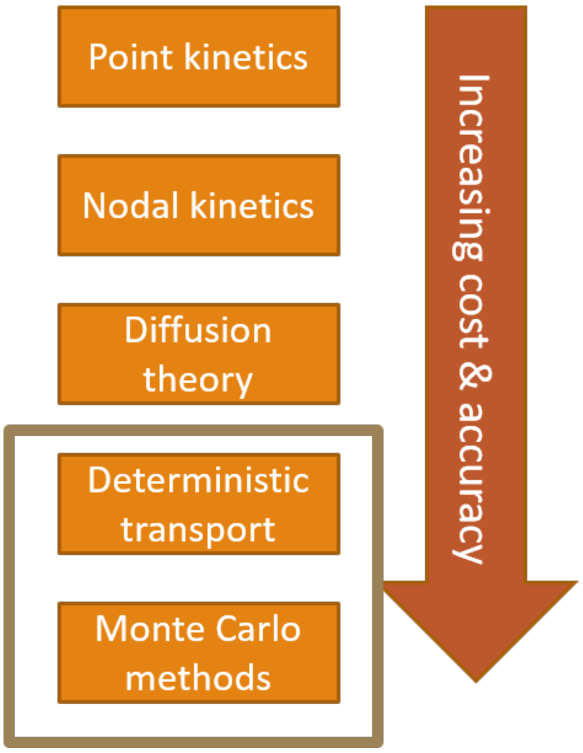


Figure 1.1: Classes of nuclear solvers and their relative cost/accuracy.

Neutron transport codes approximately solve the neutron transport equation. As mentioned above, this equation requires up to 7 dimensions to resolve (however for most applications involving full-field solutions, this includes just 6 dimensions - 2 in angle, 3 in space, and 1 in energy) and can result in huge systems of equations that take a significant amount of computational resources to produce converged solutions even when using state-of-the-art iterative solvers on state-of-the-art high performance computers. This work will not go into detail on methods of solving the neutron transport equation as there is a plethora of work over many decades devoted to this field. Section 2.1 provides more details on the methods employed in the full order model (FOM)s used in this work. For details on common approaches, the reader is pointed to other references. In Reference [4], Cho also provides an overview of important tasks and steps involved in nuclear design and analysis of reactor technology, as well as derivations of multigroup transport equations, multigroup diffusion equations, and nodal diffusion methods.

In addition to the challenges involved in capturing the behavior of neutron transport alone, modeling tools are often coupled to subchannel or CFD codes. These codes are used to simulate the flow of coolant and the heat transfer between coolant and the nuclear fuel. It is important to do so because the behavior of neutron transport in a reactor is highly dependent on the properties of the surrounding coolant for thermal reactors. These codes solve equations related to the conservation of mass, momentum, and energy. For more details on the nuclear and thermal hydraulics codes used in this work refer to Chapters 3 and 4.

In summary, designers often require high fidelity simulations of high dimensional systems, and often many independent realizations of those systems, in order to capture the requisite physics to design safe and operable reactors. This section motivated the high cost associated with simulating the requisite physics needed for nuclear reactor design and analysis. The next section will introduce the ROM concept and present it as a solution to the ever-increasing desire to simulate nuclear reactors in many configurations.

1.1.2 Reduced-Order Modeling in Nuclear Engineering

These challenges are not unique to the nuclear industry. Many engineering problems arise in the form of discretized partial differential equation (PDE)s that result in large systems of coupled equations. These systems can be thought of as a simple mapping of a set of inputs, $\boldsymbol{\mu}$, to a set of outputs, $\boldsymbol{y}$.

Numerical analysis methods for solving FOM systems of equations have been studied for decades and great strides have been made in their efficiency, sophistication, and robustness for a wide range of problems. However, two drivers can cause traditional solutions to these PDEs to become too computationally expensive. First, as models become more complex both in the physics captured and in the desired fidelity of results, the underlying systems of equations can become too large to solve on available computing resources in a reasonable amount of time. Second, if the application for the simulators requires a high number of evaluations, then the application of the model in a production environment can become too expensive, even if the evaluation of a single model is not.

For these applications where the FOM is prohibitively expensive to evaluate, a ROM can be a promising approach to reduce the computational cost of simulations in many fields, including nuclear reactor design and analysis. ROMs are simplified models that approximate the behavior of complex systems using a reduced number of degrees of freedom. These models are obtained by projecting the original high-dimensional problem onto a lower-dimensional subspace, which allows for faster and more efficient computations. The two circled methods in Fig. 1.1, "deterministic transport" and "Monte Carlo methods", represent the two most accurate but expensive methods used to solve the neutron transport equation. Replacing or augmenting these two classes of solvers will be the focus of the work herein. Chapter 2 explicitly describes the methods often employed to model neutron transport in reactor analysis. Later sections of this work will discuss various classes of ROMs that have different strengths and weaknesses depending on how they are deployed in the context of nuclear reactor design.

In general, ROMs use the same sets of inputs but require a lower-order model to predict the output quantities of interest. In the context of nuclear reactor modeling, the inputs, $\boldsymbol{\mu}$, are parameters such as material properties, geometries, cross sections, coolant temperatures and pressures, etc. The output quantities of interest, $\boldsymbol{y}$, could be the magnitude and locations of the peak power, departure from nucleate boiling, time to the peak power, distributions of the coolant temperature and density, isotopic concentrations, groupwise flux distributions, reactivity, etc.

For a ROM to be considered useful, $\hat{\boldsymbol{y}}$ should be as close to $\boldsymbol{y}$ as possible, while still keeping the ROM significantly less expensive to evaluate than the FOM. When these outputs are scalar quantities of interest, these ROMs are also often called surrogate models. This work deals only with full-field ROMs to predict one or more 3D QOI. In this work we explore current state-of-the-art methodologies for constructing ROMs and apply them to certain problems of interest in the nuclear engineering industry. Before discussing the detailed theory underpinning the ROM methods used in this work, the next section will provide a brief discussion of relevant research in the ROM field.

1.2 Relevant Research

1.2.1 General Reduced-Order Modeling Research

Projection-based ROMs use a reduced-order subspace to project the FOM equations onto a lower-order manifold, solve this new set of equations, and then project the low-order solution back into the original space. Proper Orthogonal Decomposition (POD) is a particularly well-researched method.

Benner [5] wrote a survey of state-of-the-art methods in projection based parametric model reduction and summarizes the important role these classes of problems play in engineering design. Rathinam [6] investigates basic properties of the POD method, including error analysis and the impact of perturbations to the training data. Willcox et al.

[7] and Carlberg [8] provide excellent summaries of the POD method in their background sections and are recommended by this author for a good first exposure to the primary ideas of projection based reduced-order modeling techniques. Parish et al. [9] wrote another work that has an excellent discussion of the primary ideas in projection based reduced-order modeling, while also introducing the Adjoint Petrov-Galerkin method as an improvement to existing approaches. Finally, Ghavamian [10] demonstrates the discrete empirical interpolation method (DEIM) approach to dealing with non-linearities in the FOM equations and applies this approach to two application problems.

These references explore the basics behind POD, and also contain descriptions of its variants that address issues commonly seen in projection-based ROMs. For example, using a linear projection-based method for non-linear operators often produces unstable results and can actually make the ROM more expensive than the FOM. Methods such as DEIM are used to make non-linear projection-based ROMs practical (see [5] and [10]). See Section 2.2.1 for a description of the POD method.

Within the last 10 years, neural network-based methods have been explored to generate ROMs and can provide a number of significant benefits over linear-based projection methods (such as POD, POD-DEIM, and other projection-based methods). For example, it can sometimes be difficult to implement intrusive projection-based ROMs because it requires access to the FOM operators and source code, which can be difficult for some commercial codes or legacy software that are currently deployed in production settings. Additionally, most projection-based ROMs employ a linear trial subspace, which can be limiting if the problem at hand cannot be easily reduced to such a subspace [11]. Therefore, in an effort to both simplify the application of employing ROMs and improve on the performance, non-linear trial subspaces have been explored, which can be both fully non-intrusive and do not suffer from the consequences of a linear trial subspace for a problem that it is ill-suited for. For more information on the application of neural networks for projection-based reduced-order modeling, see [12], [11], [13], or [14]. Section 2.2.3 provides a summary of the base Proper Orthogonal Decomposition neural network

(POD-NN) algorithm.

Other methods which do not utilize singular value decomposition (SVD) for computing the reduced subspace exist and are completely non-intrusive. These methods do not perform a projection analogous to the POD methods, but instead rely wholly on neural networks to perform the reduction in dimensionality. [15] demonstrated that a fully non-intrusive neural network-based architecture could be used for parameter mapping and time series prediction for some problems. This work mapped an input parameter, such as the Reynolds number, to the fully reconstructed time evolution of some dynamical system using a series of neural networks. As described in [15], the components of this neural network are as follows.

1. a convolutional autoencoder (CAE) performs spatial compression for each timestep and for each set of input parameters.

2. a temporal autoencoder (TCAE) performs temporal compression for each scalar input parameter. It used the compressed code from step (1) as its training input and output.

3. a multi-layer perceptron (MLP) performs the mapping of some set of inputs to the compressed code of the TCAE.

In this way, new parameter prediction can be performed using the MLP to reconstruct the latent code of the TCAE, which in turn reconstructs the latent code of the CAE, which can then be decoded into the full-order solution of the dynamical system using the CAE's decoder. The work in Chapter 3 fully describes this method and utilizes a modified version of this scheme on a coupled nuclear/thermal-hydraulics application problem.

These convolutional neural network (CNN) based ROMs have seen success in a diverse set of applications, particularly in the creation of flow fields for CFD applications. Bhatnagar [16] predicted velocity and pressure fields in unseen flow conditions and geometries given the shape of the object. Bhatnagar [16] used the flow solutions over airfoils with varying shapes as training data. Guo [17] used CNNs to predict non-uniform steady laminar flow in both 2D and 3D domains. Specifically, they used CNN based ROMs to predict flow over 2D primitive shapes (such as triangles, pentagons, and hexagons) and 2D car prototype shapes, as well as a 3D test involving randomly placed primitive objects in space. Time-varying applications make use of CNNs for spatial deconstruction and time series modeling methods, such as long short term memory (LSTM) layers, for stepping forward in the temporal dimension [18]. LSTMs are specialized types of recurrent units that are designed to capture and manage long-term dependencies in sequential data. However, these approaches can see limitations based on their large memory requirements.

Another class of algorithms centers around the notion that neural networks can be seen as universal approximators of continuous functions, and with enough neurons can approximate non-linear continuous operators [19], [20]. These algorithms are able to addresses some of the shortcomings present in CNN based ROMs ([20], [21], [22], [23], [24], [25]). Specifically, they are capable of handling FOMs that are defined on non-regular Cartesian grids while also requiring much less memory than CNN based ROMs.

1.2.2 Reduced-Order Modeling in Nuclear Engineering

In the context of nuclear engineering applications, much work has been done to reduce the burden of FOMs by employing surrogate models to predict scalars, such as global eigenvalues or global peaking parameters, and to optimize loading patterns. Prince and Regusa [26] has performed parametric uncertainty quantification (UQ) using proper generalized decomposition on a neutron diffusion problem, and Gilli [27] has demonstrated the use of non-intrusive polynomial chaos expansion (PCE) on criticality problems. Saleem [28] demonstrated a global parameter prediction framework using neural networks. Specifically, the framework predicts three quantities of interest: power peaking factors, fuel cycle length, and control rod bank level. Yamamoto [29] and Meneses et al. [30] performed core loading optimization using various machine learning methods. Ikonen and

Tulkki [31] performed analyses that demonstrated the importance of input interactions in the performance of nuclear fuel. Finally, Brown and Zhang [32] performed a basic Monte Carlo UQ analysis on a large-scale high-fidelity transient large-reactivity insertion analysis for a commercial core.

Recently, work has also been done to create intrusive ROMs specifically tailored to various forms and applications of the neutron transport equation. German [33] applied POD on multigroup diffusion eigenvalue problem benchmarks. German [34] also has implemented intrusive ROM methods to a 2D molten salt reactor benchmark problem. Tano [35] used a simple dense artificial neural network to speed up transport sweeps. Behne et al. [36] proposed multiresolution POD for neutron transport problems that have QOIs that span many orders of magnitude. Halvic et al. [37] used POD and Gaussian Processes (GP)s to model radiation transport through the atmosphere. And Anderson [38] introduced CrudNET, which was a CNN-based surrogate model used to predict crud deposition in a 2D lattice model.

These studies demonstrate the need for less expensive methods to tackle nuclear engineering problems that require high-fidelity results. They illustrate that there is a need for ROMs and surrogate models to fill the gap as the nuclear reactor design community continues to perform more and more analyses requiring quick turnaround models, such as UQ and input interaction analyses.

1.3 Thesis Contributions

Much work has been done in the ROM community to address the need to reduce the dimensionality of physics problems, but progress is just beginning in the transfer of these methods to the nuclear reactor design community. This area has a strong need for coupled thermal hydraulics/neutronics multiphysics simulations due to the tight interaction between thermal-hydraulic and nuclear physics, which poses an additional layer of complexity when converging solutions. This work assesses the efficacy of applying state-of-the-art ROM techniques to coupled thermal hydraulic/neutronic transport multiphysics problems and builds upon available previous work.

Specifically, to the author's knowledge, this work marks the first application of fully non-intrusive neural network-based ROMs whose uncertainty is quantified by variational inference (VI) techniques applied to representative nuclear design application problems. We apply the state-of-the-art Non-Linear Independent Dual System (NIDS) methodology, originally derived for application in the CFD community, to neutronics applications. Furthermore, we introduce uncertainty to the ROMs in the form of Bayesian neural networks (BNN)s via VI techniques. This represents the first application of VI techniques to non-intrusive neural-network based ROMs to estimate their uncertainty.

This work demonstrates these concepts by applying these methods to two nuclear applications of interest. The first is a single-assembly Michigan Parallel Characteristics Transport Code (MPACT) model undergoing a reactivity insertion accident (RIA). The inference time for neural networks is on the order of seconds to minutes, while the FOM runs take about an hour on 48 CPUs. These ideas are then scaled to a large 3D model in the second application to demonstrate their viability in a production environment. The second application is a depletion study using a 3D quarter-core MC21 model. The inference time for these neural networks are on the order of 5-10 minutes, while the run time for the associated FOMs are approximately 30-40 hours on 960 CPUs. Gains in calculation time vary significantly depending on the application, the desired accuracy, and the size of the dataset required to train the neural networks. However, this work saw wall clock speed ups of 60x to 240x for offline neural network inference.

The first application problem demonstrates the ability to properly capture the multi-physics present in an RIA that a coupled neutronics/CFD FOM can capture. The second application problem extends these ideas to a much larger full core model. Together, these problems constitute a contribution to the field of nuclear engineering in that the use of CNN and neural operator-based ROMs is new in the field of nuclear engineering. As mentioned previously, work in this field using ROMs has thus far been mostly focused on intrusive POD based methods for small benchmark problems. And, these methods are further restricted to linear dimensionality reduction methods. The methods explored in this work are neural network-based, employing nonlinear activation functions allowing the neural network to find nonlinear mappings between the full-order and reduced-order spaces. And, this work applies these methods to larger problems more representative of the size and scale encountered in production environments.

Furthermore, the methods of the CFD community from which these methods are derived are adjusted to suit the unique needs of nuclear engineering applications. Specifically, this is accomplished by leveraging the signed distance function (SDF) concept to encode geometric information about a reactor design. The SDF construct is traditionally used to define an object's boundary with a fluid in a CFD calculation. However, by changing its definition to instead encode the proximity to non-fuel elements in the spatial dimension, a nuclear reactor's core layout may be efficiently communicated to a neural network.

In addition, the application of BNNs will be explored. Specifically, VI will be used to quantify the epistemic uncertainty present in a few neural network-based ROMs. This method assumes a non-deterministic viewpoint of neural networks. Instead of a point estimate for the weights and biases of the networks, they are assumed to be random variables which can be sampled. In this way, they can provide estimates on the accuracy of the trained model.

Currently in the nuclear reactor design community little production level work is completed using multiphysics solvers without stacked uncertainties or within the best estimate plus uncertainty (BEPU) paradigm. Design spaces are explored, but often not to completeness (i.e., casualty optimization studies can rely heavily on engineering judgement and past assumptions). The type of work presented herein represents steps towards remedying these current deficiencies. If successful, this work could lead to more realistically deployable non-intrusive application opportunities to a larger pool of reactor simulation problems. It contributes to the growing body of work surrounding reduced-order modeling for reactor design calculations, specifically focused on the non-intrusive, and nonlinear methodologies areas of research. Finally, it does so by demonstrating these methods on problems of significant size to demonstrate the impressive scalability of the NIDS modeling architecture.

1.4 Thesis Organization

Chapter 2 details the background and primary neural network methodologies and architectures for the analyses in the thesis. Chapter 3 outlines and summarizes the results for the first application problem, a single assembly MPACT RIA analysis. Chapter 4 outlines and summarizes the results for the second application problem, a 3D quarter-core MC21 depletion analysis. Chapter 5 summarizes the work performed to support this work, discusses the various advantages and disadvantages of the methods used, and provides suggestions on where this research could reasonably be used in production nuclear engineering environments. Chapter 5 also contains suggestions for how the methods used herein could be further developed.

Additionally, for reproducibility and potential extensions of this work, the Appendix A provides details on the source code used and generated as part of creating this work. Appendix B discusses a Python tool, Parody, which was created to support the analyses. Parody provides an abstract wrapper around PyTorch to help perform ROM research faster and deploy various ROM methods to arbitrary FOM results.

CHAPTER 2

Background Theory and Algorithms

This chapter introduces and discusses the methods and theory underpinning the primary research of this work. First, Section 2.1 introduces the full order model (FOM)s used throughout this work. Then Section 2.2 discusses projection based reduced-order models and their evolution to non-intrusive neural network-based reduced-order modeling methods. These non-intrusive reduced-order models will form the basis for the remainder of this dissertation. See Chapters 3 and 4 for the application of these methods to the nuclear models of interest. Finally, Section 2.3 summarizes the methods available and chosen to transform the deterministic neural networks into their Bayesian versions, which are used for uncertainty estimation.

2.1 full-order Models

This section will introduce the theory behind the FOM tools used in the rest of this work. Although the intent of the non-intrusive methodologies is that the FOM tools can be treated as black boxes, it is critical to understand the underlying nature of the equations being solved so it is known what types of future problems the reduced-order model (ROM) methodologies might be effectively applied to and which are still considered open areas of research.

First, Section 2.1.1 introduces the governing equations used to describe neutron transport. Section 2.1.2 discusses the methods used to solve the neutron transport equation in the Michigan Parallel Characteristics Transport Code (MPACT) tool, which is the solver used in the first application problem in Chapter 3. Section 2.1.3 will discuss the tool, COBRA-TF (CTF), which is coupled to MPACT to provide thermal feedback. This coupling is the source of the non-linear nature of the problem dynamics.

Section 2.1.4 will discuss the Monte Carlo tool, MC21, which is the solver used in the second application problem in Chapter 4. The source of the non-linearity in this application problem stems from how MC21 and the Bateman equations interact, which are the governing equations describing isotopic depletion. These equations are discussed in Section 2.1.5.

2.1.1 Neutron Transport

MPACT is a 3D reactor transport code developed by Oak Ridge National Laboratory (ORNL) and the University of Michigan. MPACT utilizes a 2D/1D method to solve the neutron transport equation ([39], [40]). In the 2D/1D method, 2D method of characteristics (MOC) is used in the radial planes to capture the heterogeneity of neutron flux in the radial direction. Each pin cell is explicitly modeled, and sub-pin detail can be captured [32]. In the axial direction, a low-order transport solution is obtained through a pin-cell homogenized basis. [41].

The following sections provide enough details to understand the essential approaches used in MPACT to describe neutron transport in a reactor. However, for more details on how MPACT solves the neutron transport equation please see [39], [42] and [43].

2.1.1.1 Linear Boltzmann Equation

The 3D steady-state Linear Boltzmann Transport Equation is the governing equation defining how neutrons and photons transport through matter for nuclear reactor and shielding analyses. It is 6 dimensional. There are 3 dimensions in space,

$$\mathbf{x} = (x, y, z) \quad ,$$

with x, y, and z, being the spatial dimensions. There are 2 dimensions in direction. The polar cosine, μ, and azimuthal angle ω are used to define the 3D unit vector $\mathbf{\Omega}$ defined as,

$$\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z) = (\sqrt{1-\mu^2}\cos\omega, \sqrt{1-\mu^2}\sin\omega, \mu) \quad .$$

Finally, there is one dimension in energy, denoted as E. With these dimensions, it is possible to define the population of neutrons at a specific point in space, traveling in a specific direction, with a specific energy. Before presenting the full equation, a few more quantities are defined to establish the proper context.

If the spatial variables x, y, and z are displaced by small amounts, dx, dy, and dz, the spatial vector will sweep out an incremental volume $dV = dxdydz$. Similarly, by sweeping the angular variables, μ and ω, on the unit sphere by incremental amounts, $d\mu$ and $d\omega$, the directional vector $\mathbf{\Omega}$ creates an incremental solid angle $d\Omega$.

Macroscopic cross sections, $\Sigma_r(\mathbf{x}, E)ds$, are defined as the probability that a neutron at point $\mathbf{x}$ with energy E traveling an incremental distance ds experiences a particular nuclear interaction r. The types of interactions that are used in neutron transport equations for reactor design are,

$\Sigma_t(\mathbf{x}, E)ds =$ the macroscopic cross section for any interaction with a nucleus,

$\Sigma_s(\mathbf{x}, E)ds =$ the macroscopic cross section for a scattering interaction,

$\Sigma_\gamma(\mathbf{x}, E)ds =$ the macroscopic cross section for a capture interaction,

$\Sigma_f(\mathbf{x}, E)ds =$ the macroscopic cross section for a fission interaction.

These cross sections satisfy,

$$\Sigma_t(\mathbf{x}, E) = \Sigma_s(\mathbf{x}, E) + \Sigma_\gamma(\mathbf{x}, E) + \Sigma_t(\mathbf{x}, E) \quad .$$

The macroscopic differential scattering cross section is defined as the incremental probability that a neutron at location $\mathbf{x}$, traveling in direction $\mathbf{\Omega}'$, with energy E, traveling an incremental distance ds, will scatter into $d\Omega$ about $\mathbf{\Omega}$ and dE about E,

$$\Sigma_s(\mathbf{x}, \mathbf{\Omega}' \cdot \mathbf{\Omega}, E' \to E) ds d\Omega dE \quad .$$

This expression satisfies,

$$\int_0^\infty \int_{4\pi} \Sigma_s(\mathbf{x}, \mathbf{\Omega}' \cdot \mathbf{\Omega}, E' \to E) ds d\Omega dE = \Sigma_s(\mathbf{x}, E') \quad .$$

The fission spectrum, $\chi(\mathbf{x}, E)dE$ is defined as the probability that a fission neutron, emitted at location $\mathbf{x}$, will have energy between E and $E + dE$.

To complete the set-up for the linear Boltzmann equation, we next consider all neutrons located in a volume dV, about point $\mathbf{x}$, traveling in a solid angle between $\mathbf{\Omega}$ and $\mathbf{\Omega} + d\Omega$, and having energy between E and $E + dE$. This defines the angular neutron density, $N(\mathbf{x}, \mathbf{\Omega}, E, t)$. Finally, the angular flux is defined as,

$$\psi(\mathbf{x}, \mathbf{\Omega}, E) = vN(\mathbf{x}, \mathbf{\Omega}, E),$$

where $v = \sqrt{2E/m}$ is the neutron speed, with m being the mass of a neutron. The ψ quantity is important because the rates at which neutrons interact with matter are directly proportional to it.

With these definitions, the linear Boltzmann equation for ψ is given as

$$\begin{aligned}\mathbf{\Omega}\cdot\nabla\psi(\mathbf{x},\mathbf{\Omega},E)+&\Sigma_t(\mathbf{x},E)\psi(\mathbf{x},\mathbf{\Omega},E) = \\ &\int_0^\infty\int_{4\pi}\Sigma_s(\mathbf{x},\mathbf{\Omega'}\cdot\mathbf{\Omega},E'\rightarrow E)\psi(\mathbf{x},\mathbf{\Omega'},E')d\mathbf{\Omega'}dE'+ \\ &\frac{\chi(\mathbf{x},E)}{4\pi k_{eff}}\int_0^\infty\int_{4\pi}v\Sigma_f(\mathbf{x},E')\psi(\mathbf{x},\mathbf{\Omega'},E')d\mathbf{\Omega'}dE', \\ &\text{with}\quad x\in V,\mathbf{\Omega}\in 4\pi, 0<E<\infty \quad ,\end{aligned} \tag{2.1}$$

with boundary conditions,

$$\psi(\mathbf{x},\mathbf{\Omega},E)=0,\mathbf{x}\in\partial V,\mathbf{\Omega}\cdot\mathbf{n}<0,0<E,\infty \quad . \tag{2.2}$$

Equations (2.1) and (2.2) define an eigenvalue problem for the eigenfunction ψ and the eigenvalue k_{eff}. To solve this set of equations, each of the six variables must be discretized.

2.1.1.2 Discretization of the Linear Boltzmann Equation

First, the discretization in energy is discussed. The multigroup approximation is typically used, which breaks up Eq. (2.1) into a series equations split up by energy group. To do so, each quantity in Eq. (2.1) is integrated over an energy range. For example, the angular flux for group g is defined as,

$$\psi_g(\mathbf{x},\mathbf{\Omega})=\int_{E_g}^{E_g-1}\psi(\mathbf{x},\mathbf{\Omega},E)dE \quad . \tag{2.3}$$

A challenge in this process is what to do with the cross section terms when performing the integration over the other terms in Eq. (2.1). For example, the total macroscopic term is often represented as

$$\Sigma_{t,g}(\mathbf{x})=\frac{\int_{E_g}^{E_g-1}\Sigma_t(\mathbf{x},\mathbf{\Omega})\psi(\mathbf{x},\mathbf{\Omega},E)dE}{\int_{E_g}^{E_g-1}\psi(\mathbf{x},\mathbf{\Omega},E)dE} \quad . \tag{2.4}$$

However, this presents a problem as the right hand side of Eq. (2.4) depends on the solution to the continuous energy angular neutron flux and is unknown. To make this a tractable problem, the typical procedure for this problem involves making a separability assumption and defining the neutron spectrum, $\Psi(\mathbf{x}, E)$:

$$\psi(\mathbf{x}, \mathbf{\Omega}, E) \approx \Psi(\mathbf{x}, E) f(\mathbf{x}, \mathbf{\Omega}) \quad .$$

With these assumptions and similar procedures followed for the other terms in Eq. (2.1), the multigroup transport equation becomes

$$\begin{aligned} \mathbf{\Omega} \cdot \nabla \psi_g(\mathbf{x}, \mathbf{\Omega}) + & \Sigma_{t,g}(\mathbf{x}) \psi_g(\mathbf{x}, \mathbf{\Omega}) = \\ & \sum_{g'-1}^{G} \int_{4\pi} \Sigma_{s,g' \to g}(\mathbf{x}, \mathbf{\Omega}' \cdot \mathbf{\Omega}) \psi_{g',n}(\mathbf{x}, \mathbf{\Omega}') d\mathbf{\Omega}' + \\ & \frac{\chi_g(\mathbf{x})}{4\pi k_{eff}} \sum_{g'-1}^{G} \int_{4\pi} v \Sigma_{f,g}(\mathbf{x}) \psi_{g',n}(\mathbf{x}, \mathbf{\Omega}') d\mathbf{\Omega}' \\ & \text{with} \quad x \in V, \mathbf{\Omega} \in 4\pi, 1 \leq g \leq G \quad . \end{aligned} \tag{2.5}$$

The key decision in this approximation is the choice of the neutron spectrum, $\Psi(\mathbf{x}, E)$, to perform the multigroup cross section calculations.

Next, discretization in the angular dimension is performed. The method used in MPACT, and many other tools, is the discrete ordinates approximation. Here, the approximation is to allow neutrons to travel only in discrete directions. Let N denote the number of discrete angles, $\mathbf{\Omega}_n$, in a quadrature set, with $1 \leq n \leq N$. Associated with each direction is a weight term, w_n, representing the fraction of area that direction accounts for on the unit sphere. With these variables, discrete directions can be represented as

$$\psi(\mathbf{\Omega}_n) = \psi_n, 1 \leq n \leq N \quad ,$$

and angular integrals of can be represented as discrete summations using the discrete direction along with its weighting function, such as,

$$\int_{4\pi} \psi(\mathbf{\Omega}) d\Omega \approx \sum_{n=1}^{N} \psi_n w_n \quad .$$

Putting the multigroup and discrete ordinates approximations together yields the multigroup discrete ordinates equation,

$$\begin{aligned}\mathbf{\Omega}_n \cdot \nabla\psi_{g,n}(\mathbf{x}) + &\Sigma_{t,g}(\mathbf{x})\psi_{g,n}(\mathbf{x}) = \\ &\sum_{g'-1}^{G}\sum_{n'=1}^{N} \Sigma_{s,g'\to g}(\mathbf{x}, \mathbf{\Omega_{n'}} \cdot \mathbf{\Omega_n})\psi_{g',n'}(\mathbf{x})w_{n'} + \\ &\frac{\chi_g(\mathbf{x})}{4\pi k_{eff}} \sum_{g'-1}^{G}\sum_{n'=1}^{N} v\Sigma_{f,g}(\mathbf{x})\psi_{g',n'}(\mathbf{x})w_{n'} \\ &\text{with} \quad x \in V, 1 \le n \le N, 1 \le g \le G \quad .\end{aligned} \tag{2.6}$$

The choice of energy discretization and quadrature sets is dependent on the problem at hand, as well as the expertise and experience of the method and design engineers available. This topic has a long research history and is outside the scope of this work. The multigroup discrete ordinates equations retain the basic structure of the original Boltzmann transport equation, with the main difference being that neutrons can exist in discrete energies and travel in discrete directions [42].

2.1.2 MPACT

There is a large body of work detailing methods for solving the neutron transport equation. See [2], [44], [1], [45], [46] for more context on challenges, solutions, and reviews in the nuclear community at various points in its history. This work will not go into detail on the many methods available to solve this equation, except to summarize the methods used in the FOM tool MPACT used for the Chapter 3 analysis. These equations and their complete derivations can be found in the MPACT theory manual in [42].

MPACT uses a 2D/1D method, which essentially approximates Eq. (2.1) with a

2D radial transport equation coupled to a 1D axial transport equation. The 2D radial approximation uses the multigroup discrete ordinate equations presented in Eq. (2.6), while the 1D axial transport equation is approximated with a 1D low-order P$_N$ equation.

The 2D problem formulation is derived from Eq. 2.1, and includes an axial leakage term, J_z, which represents the leakage from the 2D plane to the axial plane:

$$\begin{aligned}\mathbf{\Omega_x}\frac{\partial\psi}{\partial x}(\mathbf{x},\mathbf{\Omega}) + \mathbf{\Omega_y}\frac{\partial\psi}{\partial y}(\mathbf{x},\mathbf{\Omega}) + \Sigma_t\psi(\mathbf{x},\mathbf{\Omega}) &= \frac{\Sigma_s}{4\pi}\int_{4\pi}\psi(\mathbf{x},\mathbf{\Omega}')d\Omega' \\ &+ \frac{v\Sigma_f}{4\pi k_{eff}}\int_{4\pi}\psi(\mathbf{x},\mathbf{\Omega}')d\Omega' - \frac{1}{4\pi}\left[\frac{\partial J_z}{\partial z}(\mathbf{x})\right] \quad .\end{aligned} \tag{2.7}$$

While the 1D axial transport equation includes leakage terms J_x and J_y,

$$\begin{aligned}\mu\frac{\partial\psi}{\partial z}(\mathbf{x},\mathbf{\Omega}) + \Sigma_t\psi(\mathbf{x},\mathbf{\Omega}) &= \frac{\Sigma_s}{4\pi}\int_{4\pi}\psi(\mathbf{x},\mathbf{\Omega}')d\Omega' \\ &+ \frac{v\Sigma_f}{4\pi k_{eff}}\int_{4\pi}\psi(\mathbf{x},\mathbf{\Omega}')d\Omega' - \frac{1}{4\pi}\left[\frac{\partial J_x}{\partial x}(\mathbf{x}) + \frac{\partial J_y}{\partial y}(\mathbf{x})\right] \quad .\end{aligned} \tag{2.8}$$

The axial equation is further simplified by making the P$_1$ or P$_3$ approximations to reduce computational cost. The details of this approximation are outside the scope of this work but the reader is directed to the MPACT theory manual for a complete derivation [42]. The primary takeaway of the 2D/1D approximation is that the two problems are coupled together through the current leakage terms.

To allow for transient calculations, as performed in Chapter 3, the transient multilevel (TML) method is used. The primary idea of the TML method is to capture the flux change in space, energy, and angle in a time domain consistent with its physical variation during a transient [42]. Prior to describing the general scheme used by MPACT, two new concepts must be introduced: the coarse mesh finite difference (CMFD) method and the exact point kinetics equation (EPKE).

CMFD is a method used to speed convergence when solving the neutron transport equation. It was originally developed to accelerate the convergence of nodal diffusion problems in reactor analysis. In short, it is derived from a multigroup diffusion equation variant of the neutron transport equation. Its basic formulations consist of using the diffusion approximation and solving the resultant equations on a coarse mesh and mapping this information to and from the mesh used for the fine mesh transport solution. See the MPACT theory manual for more details [42].

The EPKE is derived from the integration of Eq. (2.1) over angle, space, and energy and is presented below:

$$\frac{dp(t)}{dt} = \frac{\rho(t) - \beta^{eff}(t)}{\Lambda(t)} p(t) + \frac{1}{\Lambda(0)} \sum_{\tau} \lambda_\tau(t) C_\tau(t) \quad , \tag{2.9a}$$

$$\frac{dC_\tau(t)}{dt} = \frac{\Lambda(0)}{\Lambda(t)} \beta_\tau^{eff}(t) p(t) - \lambda_\tau^{PK}(t) C_\tau(t) \quad , \quad \tau = 1, 2, ..., 6 \quad , \tag{2.9b}$$

where $p(t)$ represents the core wise amplitude, $C_\tau(t)$ represents the adjoint weighted precursor number density for delayed neutron group τ, β_τ^{eff} represents the adjoint flux weighted delayed neutron fraction, Λ is the neutron generation time, and λ_τ^{PK} are the delayed neutron constants.

The EPKE is used throughout the nuclear reactor design community for transient analysis of plant operations and accident analysis. It is useful as it can be quickly solved compared to higher fidelity approaches and provides accurate predictions for global changes in power. In MPACT, it is useful specifically for this reason. The TML method uses these three levels of fidelity together (transport methods, CMFD methods and the EPKE) to provide accurate and relatively cheap 3D transient solutions for reactor analysis.

The 3D transport solution (the 2D/1D method) uses coarse timesteps to capture the variation in sub-pin flux behavior, a 3D CMFD transient solver uses an intermediately sized timestep to capture the pin-wise scalar flux distribution changes, and the EPKE uses the smallest timestep in order to capture the time variation of the flux magnitude, which is driven by the prompt neutron generation time [42]. Figure 2.1 shows a cartoon of how the three solution methods compare to each other in timestep sizes.

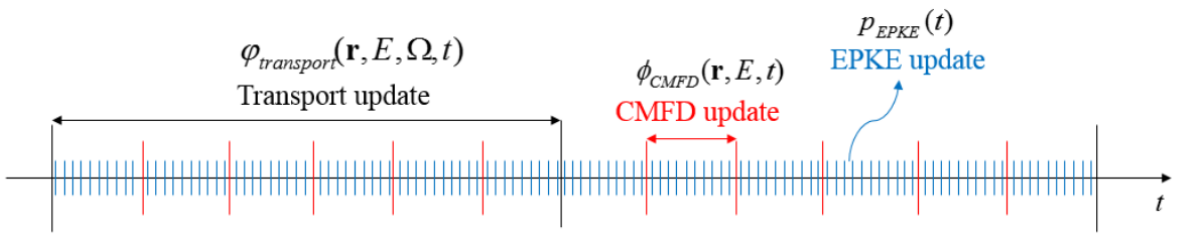

$\varphi_{transport}(\mathbf{r}, E, \Omega, t)$
Transport update
$\phi_{CMFD}(\mathbf{r}, E, t)$
CMFD update
$p_{EPKE}(t)$
EPKE update
t

Figure 2.1: Illustration of TML scheme [42].

To couple these three techniques together, an assumption of separability is made about the shape and amplitudes at each stage. For the transport solution coupled to the CMFD solution, the shape within a coarse mesh is assumed to be accurate from the transport solution while the amplitude function is corrected using the CMFD solution. Likewise, for the CMFD coupling to the EPKE the shape within the CMFD solution is assumed to be accurate while the global amplitude function is updated using the EPKE.

Figure 2.2 shows the entire coupled scheme, with more details about intermediate steps available in the MPACT theory manual [42].

Notice that Fig. 2.2 includes a "TH solve" in the transport step loop. During this solve step, a thermal solution for the coolant in the model is found using CTF and the cross section data is updated. The next section introduces the CTF tool's governing equations.

2.1.3 Cobra-CTF

COBRA-TF, also referred to as CTF, is a thermal-hydraulic simulation code designed for light water reactor vessel and core analysis. It was developed by the Reactor Dynamics and Fuel Modeling Group at North Carolina State University (established at Penn State) and also sees contributions from ORNL ([47]). CTF is a thermal hydraulic simulation code designed for the analysis of light-water reactors. It uses a two-fluid, three-field modeling approach to model the independent behavior of liquids, droplets, and vapor. CTF is able to model both the flow within each channel and the cross flow between

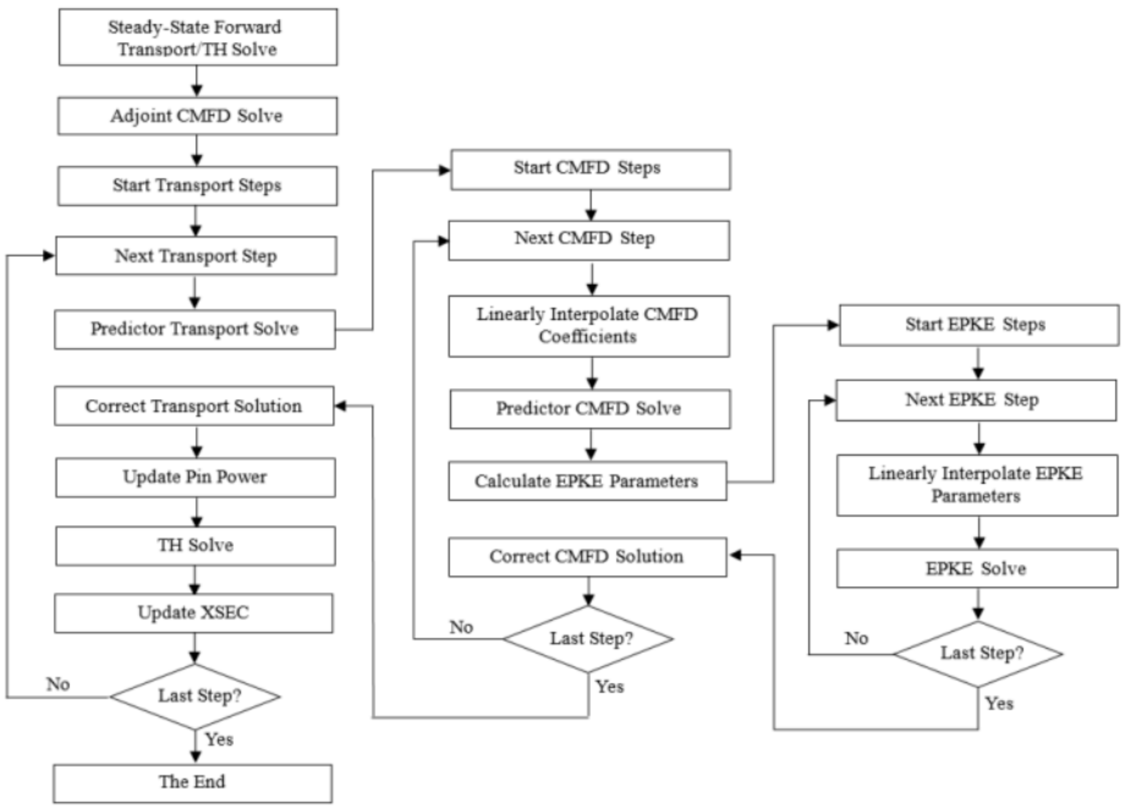


Figure 2.2: Flow chart for the transient multilevel method.

channels. Users and developers of VERA-CS have used it in a number of application problems (see [39], [32], and [43]).

The axial mesh for CTF is consistent with the axial mesh of MPACT, and the MPACT power is summed over each pin and passed to CTF. In this way, the geometries are consistent, and energy is conserved between the codes. MPACT receives average temperatures from CTF for each radial, axial, and azimuthal volume of each pin and cladding region, together with the average coolant density for each coolant segment. The energy deposition and cross-section information is then updated during each iteration between the codes until they converge on a solution. See [39] for more details on how MPACT and CTF are coupled.

This section summarizes the general conservation equations that CTF solves, as presented in [48] (i.e., the CTF theory manual). These equations are important in the context of this work because it is the coupling of the neutron transport and these subsequent equations which provide the non-linearity which makes a reactivity insertion accident (RIA) difficult to model. However, this section will not go into the various closure models, assumptions, and mesh discretization strategies used by CTF as they are outside the scope of this work. See [48] for a complete derivation of, and implementation details for, the solver methods used within CTF.

CTF conserves mass, momentum, and energy for three phases of flow (liquid film, liquid droplets, and vapor). Each phase is modeled with its own set of conservation equations. The general mass conservation equation is given by

$$\frac{\partial}{\partial t}(\alpha_k \rho_k) + \nabla \cdot (\alpha_k \rho_k \vec{V}_k) = L_k + M_e^T \quad . \tag{2.10}$$

Here, the k subscript refers to the type of flow (liquid film, droplets, or vapor). α refers to the specific volume, and ρ the density. The first term on the left hand side is the change of mass with time, and the second term is the advection of the field mass into or out of the volume with $\vec{V}_k$ being the field velocity. L_k represents mass transfer into or out of the phase k. M_e^T represents the mass transfer due to turbulent mixing and void drift. See [48] for the models used and available for each phase for L_k and M_e^T.

The general momentum conservation equation is given by,

$$\begin{aligned} \frac{\partial}{\partial t}(\alpha_k \rho_k \vec{V}_k) + \frac{\partial}{\partial x}(\alpha_k \rho_k u_k \vec{V}_k) + \frac{\partial}{\partial y}(\alpha_k \rho_k v_k \vec{V}_k) + \frac{\partial}{\partial z}(\alpha_k \rho_k w_k \vec{V}_k) \\ = \alpha_k \rho_k \vec{g} - \alpha_k \nabla P + \nabla \cdot \left[\alpha_k (\tau_k^{ij} + T_k^{ij})\right] + \vec{M}_k^L + \vec{M}_k^d + \vec{M}_k^T \quad . \end{aligned} \tag{2.11}$$

Here, the left-hand side describes the change in volume momentum with time and the advection of momentum. u_k, v_k, and w_k refer to the velocities in x, y, and z directions respectively for phase k. The right-hand side has a term for gravitational force $\vec{g}$, pressure force P, viscous τ, and turbulent shear stress T, momentum source/sink due to phase change $\vec{M}_k^L$, interfacial drag forces $\vec{M}_k^d$, and momentum transfer due to turbulent mixing $\vec{M}_k^T$.

The general energy conservation equation is given by,

$$\frac{\partial}{\partial t}(\alpha_k \rho_k h_k) + \nabla \cdot (\alpha_k \rho_k h_k \vec{V}_k) = -\nabla \cdot \left[\alpha_k(\vec{Q}_k + \vec{q}_k^T)\right] + \Gamma_k h_k^i + q_{wk}''' + \alpha_k \frac{\partial P}{\partial t} \quad . \qquad (2.12)$$

The left hand side terms are change of phase energy with respect to time and advection of phase energy into or out of the volume, with h representing enthalpy. The right-hand side represents the conduction $\vec{Q}_k$, turbulent heat flux $\vec{q}_k^T$, energy transfer due to phase change τ_k, volumetric wall heat transfer q_{wk}''', and the pressure work term.

Among others, CTF makes the following simplifying assumptions for the energy conservation equation. There is no volumetric heat generation within the fluid. There is no radiative heat transfer, pressure is uniform throughout the phases, and conduction in the fluids is zero. See [48] for a complete description of other assumptions and simplifications made for various models of all phenomena captured by these three conservation equations.

2.1.4 MC21

MC21 is a continuous energy Monte Carlo radiation transport code used to compute steady-state reaction rate distributions in three-dimensional models ([49]) and was developed by the Naval Nuclear Laboratory. It was specifically optimized to support large-scale reactor physics simulations, but the code can be used for any neutron or photon transport problem. It has been used to evaluate Virtual Environment for Reactor Analysis (VERA) core physics benchmark problem number 6 and 7 ([50] and [51] respectively) and has shown to produce state-of-the-art results when compared to other multiphysics solvers such as the VERA tool used in Chapter 3 (specifically MPACT and CTF - [39]).

For the Chapter 4 analysis, MC21 is used along with an internal depletion solver to model the change in isotopic concentrations as a function of time. The equations describing how depletion of nuclear reactor isotopes evolves with time are presented in the next section.

2.1.5 Bateman Equations

Nuclei production and the variation in their inventory during and after reactor operation are governed by the Bateman Equations [52] (originally described by Bateman [53]). The depletion analysis application problem using MC21 (see Chapter 4) relies on inline buildup equations which utilize these equations to compute isotopic abundances after each timestep. In general, the concentration for any individual isotope in a reactor is influenced by three phenomena:

- the generation by fission production,
- the generation from one or more parent nuclei by radioactive decay or neutron capture, or
- the disappearance by radioactive decay or neutron capture.

The equation used by reactor engineers to capture these phenomena for isotopes in a reactor can be generalized as,

$$\frac{dN_i}{dt} = \phi \sum_{hn} \sigma_f^{hn} y_{hn}^i N_{hn} + \sum_{j,k} (\alpha_i^j \lambda_{Pj} N_{Pj} + \phi \beta_i^k \sigma_c^{Pk} N_{Pk}) - (\sigma_a^i \phi + \lambda_i) N_i \quad . \tag{2.13}$$

The left hand side represents the change in the number density N of the isotope i. The first term on the right-hand side of Eq. (2.13) represents generation from heavy nuclei by fission, where ϕ is the neutron flux, the sum is over all heavy nuclei hn capable of undergoing fission, σ_f^{hn} is the microscopic fission cross section of each heavy nucleus, y_{hn}^i is the fission yield of heavy nuclei hn for fission product i, and N_{hn} is the concentration of heavy nuclei. The second term on the right-hand side of Eq. (2.13) is the generation of one or more parent nuclei by radioactive decay or neutron capture, where α_i^j and β_i^k are the branching ratios of the parents P_j and P_k to the fission product i. λ_{Pj} is the radioactive decay constant of parent P_j, N_{Pj} and N_{Pk} are the concentrations of parents

j and k, and σ_c^{Pk} is the microscopic capture cross section of parent Pk. The third term on the right hand side of Eq. (2.13) is the disappearance by radioactive decay or neutron capture, where σ_a^i is the microscopic capture cross section of fission product i, and λ_i is the radioactive decay constant of the fission product.

Capturing behavior for each isotope of interest in a reactor creates a system of coupled differential equations that can be solved for any isotope at any time, given the flux and current isotopic concentrations of all other isotopes. In a nuclear reactor the flux, ϕ, depends on the power distribution, which depends on the thermal feedback within the core as well as any way in which a reactivity control mechanism (such as soluble boron or control rod mechanisms) reacts to the reactivity of the core. All of these physics and control phenomena (neutronics, isotopic concentration decay/production rates, thermal distributions, and reactivity control feedback) are tightly coupled to each other. In a FOM, the way this is typically solved is with an iterative coupling of physics codes. It is this system that the ROMs in Chapter 4 is the focus of.

During a depletion step, the Bateman equations are discretized and solved over some period of time, defining a timestep. In each timestep, the flux, ϕ in Eq. (2.13) is assumed to be constant with time. The most accurate method for incorporating time dependence of flux into a depletion calculation would be to perform transport calculations at many intermediate steps and use the resulting flux values during the depletion calculation. This would result in very small timesteps to resolve the dependence of flux on composition because of the non-linear interdependence of flux and number densities of isotopes. To reduce the cost and allow for larger timesteps, in this work the so-called "predictor-corrector" technique is used in MC21.

As outlined in [42] for MPACT (but implemented in the same manner as in MC21), the predictor-corrector method works by computing a predicted nuclide concentration for a given time step and then a corrected nuclide concentration. First, a typical depletion calculation is performed to compute the particle number densities at the end of the timestep, using the flux, ϕ_1, and cross section values from the beginning of the timestep to do so. Call the resulting number densities N_1. Next, a new flux solution, ϕ_2 and cross sections are computed using the end of time step number densities, N_1. Then, a corrector step performs a depletion from the beginning of timestep number densities using ϕ_2 to compute new end of timestep number densities, N_2. The final number densities are then the average of the predicted, N_1, and corrected, N_2, concentrations.

The Bateman equations and a FOM solver (in the case of this work, MC21), work together using this predictor-corrector procedure to capture the isotopic trajectories during a depletion analysis. The term "power history" refers to the primary driver of this trajectory and is defined by a reactor power executed over a specific length of time. These tools, MC21 and the Bateman Equation solvers, are the focus of Chapter 4.

2.2 Reduced-Order Models

Reduced-order modeling is the transformation of high-dimensional models into meaningful representations of reduced dimensionality [54]. Well-known and researched linear ROM techniques can be classified into Krylov subspace methods, balanced truncation, and Proper Orthogonal Decomposition (POD). In reference [55], Bai summarizes the Lanczos process based Krylov subspace technique for reduced-order modeling and provides a review of other Krylov subspace techniques. In reference [56], Gugercin and Antoulas provide a survey of model reduction methods using balanced truncation in 2007, with [57] providing a good tutorial for those new to both reduced-order modeling and balanced truncation in particular. POD will be discussed in Section 2.2.1. Other methods include reduced basis methods, dynamic mode decomposition and Koopman operator methods, Gaussian processes, or polynomial chaos expansion. Each method is potentially useful in certain contexts; however, this paper will focus largely on projection-based ROMs.

In reference [58], Antoulas discusses the two basic families of singular value decomposition (SVD) and Krylov based reduced-order models, weighing the strengths and weaknesses of each and how they may be combined to realize the benefits of both. In reference Lu et al. [59] published another survey paper which highlights common methods for model reduction, focusing on the POD method. This author recommends this resource for those interested in a thorough introduction to the POD method contrasted against other common methods. In reference, Chen et al. [60] proposed a continuous reduced-order modeling approach, as opposed to other methods which reduce dimensionality of the discretized space.

Each of these references observes the model reduction task in a slightly different light but with significant overlap. Taken together, they help the reader understand the intuition and algorithmic steps needed to perform effective model order reduction. The next section provides a basic discussion of the POD method and will walk through how it relates to the growth of the field into the neural network-based non-intrusive reduced-order modeling methods. Finally, it introduces all the methods used in the application sections of this dissertation (i.e., the convolutional neural network (CNN) and Non-Linear Independent Dual System (NIDS) based ROMs.

2.2.1 Projection-Based Reduced-Order Models

Projection-based ROM methods are a common way to avoid the cost of expensive FOMs. One of the most well-researched areas is POD derived methods, which have been applied in a wide variety of applications. Alsayyari [61] used a POD approach to build a surrogate for a fast spectrum molten salt reactor. Ghavamian et al. [10] applied the discrete empirical interpolation method (DEIM) method to two solid mechanics problems. Rahman [62] created a surrogate model for a 2D flow field of ocean currents. This application showed good performance for a problem with dynamics evolving over a large range of temporal and spatial scales. Although these methods are quite common, they have not been widely applied in the nuclear engineering field outside of research contexts, despite their potential computational cost savings.

Only in the last few years have intrusive POD based ROMs been seriously explored for radiation transport applications. German [33] applies POD on multi-group diffusion eigenvalue problem benchmarks and shows great agreement. In another paper, German [34] also implemented an intrusive ROM method applied to a 2D molten salt reactor problem under a handful of scenarios to evaluate its effectiveness. Tano and Regusa [35] used a simple dense artificial neural network to speed up transport sweeps by replacing the Gaussian elimination process used in typical neutron transport methods with a neural network. Later, Behne and Regusa[36] introduced multi-resolution POD ROMs applied to neutron transport problems with large streaming paths that lead to large changes in order of magnitude of quantities of interest. Halvic et al. [37] created a ROM to model radiation transport through the atmosphere using SVD as a reduced subspace calculator and Gaussian processes as the coefficient dynamics regressor.

Notice that these previous works either used POD derived methods as their basis for subspace selection ([34], [34], [63], [37]) or sought to improve convergence speeds of currently used numerical methods algorithms ([63]). The former is valuable work for when the problem dynamics can be accurately captured with a linear subspace. There are a huge class of problems in nuclear engineering that are static, linear problems well suited for this type of analysis; however, when the problem express strong nonlinear behavior these approaches are expected to struggle. In reference, Behne [63] also showed great improvements for convergence performance, but represents an intrusive approach to deploying neural networks for reactor analysis. This would be an excellent approach when seeking faster convergence for FOMs; however, if the goal is to treat the FOM as a black box to generate a flexible ROM for nonlinear problems, other approaches are required.

The following section will start with a description of the POD algorithm as a basis for the introduction of neural network-based non-intrusive ROMs. These non-intrusive neural network-based ROMs are then applied to nuclear applications to assess their applicability within the nuclear community. These methods, and the research herein, fill a gap in the class of projects represented in the previous paragraph. The methods herein represent non-intrusive neural network-based ROM algorithms capable of capturing nonlinear dynamics in 3 dimensions for an arbitrary number of quantity of interest (QOI)s.

2.2.2 Proper Orthogonal Decomposition

Projection-based methods consist of an offline phase and an online phase. During the offline phase, a series of FOM results are calculated for multiple input parameter instances with the aim of exploring the design space of interest. This collection of FOM results, often called a snapshot matrix, is used to construct a lower-dimensional subspace of the problem space. During the online phase, the chosen ROM algorithm seeks to construct this space using the snapshot matrix. The new system, in the lower order subspace, can then be used to realize low-dimensional solutions to new input parameter combinations in the design space. The ROM algorithm is then used to project this low-order solution from the lower-dimensional space back into the full-order space, thus producing a full-order result for downstream analysis.

This section explores basic non-intrusive ROM methods, built up from the well-studied ROM method, POD. These non-intrusive methods all rely on neural networks to do the mapping from the full-order space to the low order space and back again. The next sections build from the POD method to various non-intrusive neural network-based architectures, such as CNN and neural operator-inspired architectures. These algorithms are then implemented for nuclear applications of interest (see Chapters 3 and 4).

POD is a widely studied method in the ROM community. It is often ineffective at handling non-linear operators and requires adjustments to make it effective for most real-world applications. However, its basic formulation provides a useful context for the CNN and hypernetwork-based ROMs explored later in this work.

Let a generic FOM be represented as

$$\frac{d\mathbf{u}(\mathbf{x},\mathbf{t})}{dt} = L\mathbf{u}(\mathbf{x},\mathbf{t}) + \mathbf{N}(\mathbf{u}(\mathbf{x},\mathbf{t}),\mu) \quad , \tag{2.14}$$

where

$\mathbf{u}$ = state variable of interest

t = time - let the problem be discretized into n_t timesteps

x = spatial location - let the problem be discretized into n_x spatial locations

μ = input parameters which define a full-order model execution

L = linear operator

N = non-linear operator.

The POD algorithm first decomposes $\mathbf{u}(\mathbf{t})$ into a linear combination of a collection of spatially dependent functions, $\phi(\mathbf{x})$ (sometimes called modes), and time-dependent basis coefficients $\mathbf{a}(\mathbf{t})$ such that

$$\mathbf{u}(\mathbf{x},\mathbf{t}) \approx \sum_{\mathbf{k=1}}^{\mathbf{r}} \mathbf{a_k}(\mathbf{t})\phi_{\mathbf{k}}(\mathbf{x}) \quad , \tag{2.15}$$

where

k = the k^{th} mode

r = number of modes included in the ROM formulation.

Plugging Eq. (2.15) into Eq. (2.14) allows us to rewrite the FOM as

$$\frac{d\mathbf{a}(\mathbf{t})}{dt} \approx \mathbf{\Phi_r^T L \Phi_r} + \mathbf{\Phi_r^T N}(\mathbf{\Phi_r a}(\mathbf{t}),\mu) \quad , \tag{2.16}$$

where

$\mathbf{a}$ = $[a_1, a_2, ..., a_r] \in \mathbb{R}^{n_r \times n_t}$, basis coefficients

$\mathbf{\Phi}$ = $[\phi_1, \phi_2, ..., \phi_r] \in \mathbb{R}^{n_x \times n_r}$, spatial modes.

The ROM is considered useful if $r \ll n_x$ and the ROM still retains acceptable accuracy for the application in question. Put another way, aim for the dimensionality of the ROM to be much less than that of the original FOM problem with minimal degradation of accuracy when compared to the FOM. To determine the spatial modes Φ_r, the eigenmodes are extracted via an SVD on the snapshot matrix. The snapshot matrix is constructed so that it includes the state variables of the FOM for many parametrically unique runs, with the goal of adequately exploring the design space during the offline phase. After creating the snapshot matrix and performing the SVD decomposition, the resulting left-singular matrix is truncated to include only the most important modes. A nice property of the POD algorithm is that when adding more modes the error, quantified via the Frobenius norm, never increases.

The terms in the linear portion of Eq. (2.16) ($\mathbf{\Phi_r^T L \Phi_r}$) can be computed in an offline phase, whereas the terms in the nonlinear portion of Eq. (2.16) ($\Phi_r^T N(\mathbf{\Phi_r a(t)}, \mu)$) must occur in an online phase. Therefore, using this formulation of POD has the shortcoming that the presence of a non-linear operator can cause the ROM to be more expensive than the FOM. Methods such as the discrete empirical interpolation method ([10]) (and others; see [7] for a review of projection-based methods and various solutions to this non-linearity problem) exist to mitigate this shortcoming. However, for the purpose of this discussion, the simple POD framework is adequate.

2.2.3 Neural Network-Based Projection Methods

One disadvantage of POD with Galerkin projection as presented here is that it is considered an intrusive method. In this context, intrusive refers to the act of recasting the governing equations into the lower-dimensional space. It often requires access to the underlying code of the FOM. This can make the deployment of these ROM solutions difficult in production environments where legacy software is used. Proper Orthogonal Decomposition neural network (POD-NN) is one way to bypass these constraints ([14], [12]). POD-NN algorithms retain the essential components of POD, with the exception that they use neural networks to capture the dynamics of the basis coefficients in time instead of the lower-dimensional representation of the FOM (that is, Eq. (2.16)).

Using other constructs in neural networking, such as CNN based neural networks or operator based neural networks, can see further improvements over the POD-NN method. This section will discuss each of these methods in turn; however, Before discussing these methods in detail the next section will discuss the basic mechanics of neural networks in general.

2.2.3.1 Neural Networks

A neural network is a type of machine learning model. It contains a single or multiple layers of connected nodes, called neurons, which are organized into layers. Each neuron takes an input, linearly transforms it, and applies a non-linear transformation to produce an output. It then feeds this output into the input of the next layer. In supervised learning, the model learns to adjust the weights and biases of the linear transformation to minimize the difference between the predicted and known outputs during the training process. The operations performed for a single-layered neural network are

$$z_i = \sum_{j=1}^{m_i} w_{ij} x_j + b_i \quad , \tag{2.17}$$

where:

z_i	$=$ Output of a neuron in the i-th hidden layer
m_i	$=$ Number of neurons in the previous layer
w_{ij}	$=$ The weight connecting neuron j in the previous layer to neuron i in the current layer
b_i	$=$ The bias term of neuron i in the current layer.

After the output z_i is computed, an activation function, $f(\cdot)$ is performed on the output to introduce a non-linearity into the neural network model. The activation function is defined as

$$a_i = f(z_i) \quad ,$$

where

$a_i =$ Output of a neuron in the i-th hidden layer after the activation function is applied.

Typical activation functions include the rectified linear unit (often referred to as "ReLU"),

$$f(z_i) = \max(0, z_i) \quad ,$$

the sigmoid function,

$$f(z_i) = \frac{1}{1 - e^{-z_i}} \quad ,$$

and tanh function,

$$f(z_i) = \tanh(z_i) \quad .$$

These operations (the linear and non-linear transformations to the inputs) can be chained together to create neural networks with an arbitrary number of layers. The final layer of the neural network is called the output layer, denoted as $\hat{y} = [a_1, ..., a_n]$. For binary classification, this usually produces just one output. For multiclass classification or regression, this can include as many outputs as is required for the application of interest.

The task of computing the specific values used for the weights and biases (w_{ij} and b_i) is performed using the backpropagation algorithm, which is a supervised learning algorithm that updates the weights of neurons in a neural network so that the network can learn to approximate the output of a given input ([64], [65]). First, a loss is computed which represents the difference between the known output and the output produced by the neural network. The goal of the backpropagation algorithm is to minimize this loss value. For regression problems, this is often a distance metric such as Mean Squared Error (MSE),

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad ,$$

where

$\mathcal{L}$ = The loss of the neural network.

n = Number of output values.

Using algorithms such as gradient descent or its variants, such as adaptive moment estimation (ADAM) ([66]), the weights and biases are updated using the gradients of the loss function with respect to the parameters:

$$w_{ij} \leftarrow w_{ij} - \alpha\frac{\partial \mathcal{L}}{\partial w_{ij}} \tag{2.18a}$$

$$b_i \leftarrow b_i - \alpha\frac{\partial \mathcal{L}}{\partial b_i} \tag{2.18b}$$

where

α = The learning rate, which controls the size of the weight update.

Loss gradients ($\frac{\partial \mathcal{L}}{\partial w_{ij}}$, $\frac{\partial \mathcal{L}}{\partial b_i}$) are calculated layer by layer starting with the output layer. For example, first the derivative of the loss function with respect to the final activation function is computed. Then, these gradients are propagated backward through the network. For each neuron, the gradient with respect to its output using the chain rule is computed. This gradient is then used to update the gradients of neurons in the previous layer. This process is carried out sequentially all the way backwards through the neural network until the contribution of each weight and bias to the overall loss is known. Once these values are known, Eqs (2.18a) and (2.18b) are executed for all weights and biases.

2.2.3.2 Proper Orthogonal Decomposition and Neural Networks

Returning to the context of ROMs, POD-NN uses the idea of neural networks to find a mapping of the basis coefficients, $\mathbf{a}(\mathbf{t})$, at some time t to the next time $t+\Delta t$. The modes,

$\phi(\mathbf{x})$, remain the same and are still derived from the SVD of the snapshot matrix. Figure 2.3 (from [12]) shows how a densely connected network could move the basis coefficients forward in time. In this example, the problem is characterized by a Reynolds number that is used as one of the inputs to the model, but this could be generalized to any number of input parameters for some application of interest. Note that any time series model could be used in place of the dense network, such as a long short term memory (LSTM) layer or a non-neural network-based time-stepping method. The POD-NN method is non-intrusive and can be treated as a black box. All that is needed to train the algorithm is the snapshot matrix. No projections of the governing equations are performed and, therefore, no access to underlying code or matrices is required.

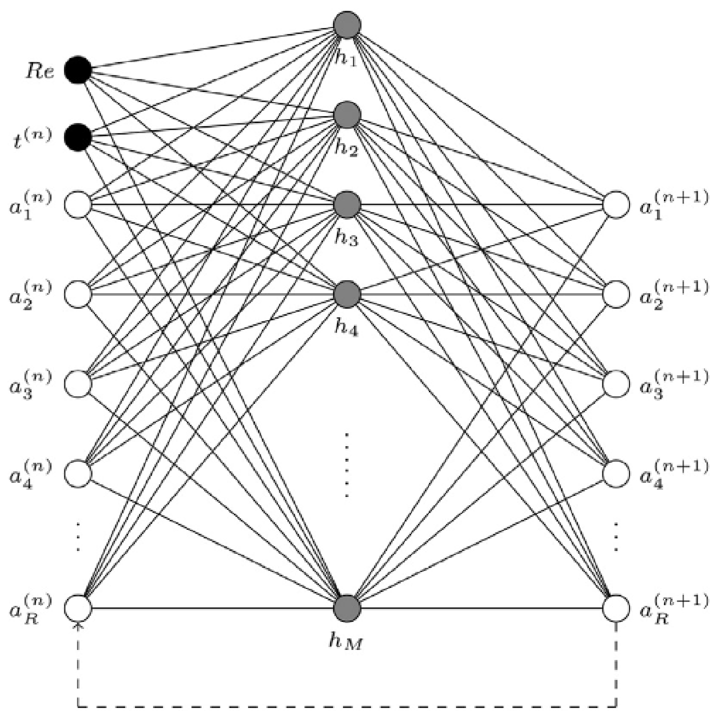


Figure 2.3: A simple POD-NN non-intrusive model order reduction technique ([12]).

A disadvantage of ROM methods that originate from the principles of POD is that spatial modes are computed using a linear decomposition method such as SVD and, therefore, the modes are restricted to a linear subspace. Another class of ROM methods utilizes CNN based autoencoders to perform dimensionality reduction and projection, which ignore the constraints of a linear subspace. Before discussing how the CNN autoencoders are structured, the autoencoder and CNN constructs are briefly described.

2.2.3.3 Autoencoders and Convolutional Neural Networks

An autoencoder is a type of neural network that is used for unsupervised learning ([67]). They consist of two neural networks trained simultaneously, an encoder and a decoder. The encoder compresses the input data into a lower-dimensional representation. This can be represented as $f(x) = h$ where f is the encoder model, x is the model input, and h is the lower-dimensional representation, often referred to as the "code" of the autoencoder. The decoder operates on this code to reconstruct the original input. This can be represented as $g(h) = \hat{x}$, where g is the decoder, and $\hat{x}$ is the reconstructed input. Taken together, traversing an autoencoder is represented as $\hat{x} = g(f(x))$. It is called an unsupervised learning method because all that is needed to train the networks are the original input data without labels or associated output data.

The objective of the autoencoder is to minimize the difference between the input data and its reconstruction, effectively learning to capture the most important features of the data in the encoding. The encoder and decoder from the equations above are composed of neural networks as described in Eq. (2.17), and follow the same procedures for training and optimization using the backpropagation algorithm. Autoencoders are widely used for tasks such as data compression, feature extraction, and anomaly detection.

Their value comes from their ability to find informative representations of data in a lower-dimensional space which can be used for classification, or in the case of ROMs, time stepping of a physics problem. An autoencoder, composed only of densely connected layers, quickly becomes too expensive and large to be used for common image classification or physics modeling problems, so the autoencoder construct is often combined with CNNs.

A CNN is a type of neural network that retains the spatial relationship between a structured input format. CNNs started to become ubiquitous with the creation of AlexNet [68], but were introduced much earlier ([69]) and have been deployed in a significant number of image recognition applications. These networks received continuous improvement over the last 15 years as the technology became better understood and computational power became more available. Some important papers in this history are referenced below. [70] investigated these methods to perform classification for categories with variations in their pose, lighting, and orientation. [71] explored the large hyperparemeter space which describes these architectures and quantified their impact on image recognition. [72] studied CNNs and their ability to learn to recognize features in large images. [73] and [74] investigated the effects of increasing the depth of their neural networks for the ImageNet image classification competition and found improved performance over other entries.

They are commonly used in image processing tasks because the spatial relationship between certain features of an image is often useful for the classification of images. Due to the way they sweep a set of trained filters with variable stride length, coupled with pooling techniques, they have an inherent ability to automatically select the most important features within images at various levels of spatial scale to best solve the application for which it is being trained.

There are three primary operations performed in a convolutional neural network. The following equations assume a 2D grid of information (i.e., an image) but these relationships can be adjusted for 1D or 3D applications by adjusting the dimensions of the kernel and convolving the kernel over three dimensions instead of two.

The first, the convolution, maps a kernel K with dimensions $k_h \times k_w$ onto the input 2D plane I. In this context, a kernel refers to a matrix (being 1, 2, or 3 dimensional) that is convolved over the input information (i.e., a 2D image). This involves element wise multiplication of the kernel with a region of the input, to produce an output feature map. In a CNN, the values of the kernel are the parameters that are optimized, and are adjusted during training to create the optimal kernels for the current application.

A convolutional operation is defined as,

$$O(x,y,c_{out}) = \sum_{i=-\frac{k_h}{2}}^{\frac{k_h}{2}} \sum_{j=-\frac{k_w}{2}}^{\frac{k_w}{2}} \sum_{c_{in}=0}^{C_{in}-1} I(x+i, y+j, c_{in}) \cdot K(i+\frac{k_h}{2}, j+\frac{k_w}{2}, c_{in}, c_{out}) \quad ,$$

where

O = is the output of the convolutional layer

x, y = are the x and y locations in the grid space

c_{out} = output channel dimension

c_{in} = input channel dimension.

After the convolution operation, as is typical with neural networks, a non-linear activation function is applied element-wise to the output features. A common activation function is the ReLU (defined in (2.2.3.1)), which is denoted as A_{ReLU} in the following equations. Finally, the third primary operation seen in typical CNNs is the pooling operation. Pooling is often used to reduce dimensions of the features during a forward pass. A max pooling operation takes the maximum value of the neural network layer, after convolution with the kernel and activation, and is performed for each output channel. For example, a max pooling with a window size of 2 (here window refers to the scale over which the maximum operation is performed), would look like,

$$\begin{aligned} O(x,y,c_{out}) = \max(&A_{ReLU}(2x, 2y, c_{out}), \\ &A_{ReLU}(2x+1, 2y, c_{out}), \\ &A_{ReLU}(2x, 2y+1, c_{out}), \\ &A_{ReLU}(2x+1, 2y+1, c_{out})) \quad . \end{aligned}$$

This example performs pooling with a window size of 2; however, kernels can be any size according to the application at hand (i.e., the kernel size and shape is a hyperparameter that is optimized). Another common pooling operation is the average pooling operation, where the same task is performed by averaging neighboring locations in the output feature instead of reducing the result to the maximum value over some region. Additional hyperparameters include step size of the convolutional operation and padding rules for when a kernel overlaps with the edge of the input feature.

CNN extension to the applied physics domain is natural in that solutions to physics problems often have a physical structure that contains useful information about the underlying dynamics. For example, in a nuclear reactor design, the spatial arrangement of fuel and poison pins is extremely important to predicting the shape of the neutron flux. Or, the proximity of some fuel pin to a radial water reflector will influence the neutron spectrum in nearby spatial regions.

CNN based autoencoders methods thus establish mappings between reduced-dimensional subspaces and input parameter spaces in a fully non-intrusive manner. CNN based autoencoder ROMs represent an improvement on POD inspired methodologies, as they are more suited to retain the spatial relationships present in physics problems and are non-linear in nature (due to the non-linear activation functions inherent within a neural network). This allows them to learn non-linear mappings between higher and lower dimensional spaces, unlike SVD, which is restricted to a linear mapping. CNN autoencoder based ROMs have been used in many applications with good success ([62], [11], [16], [18]). Two CNN autoencoder-based ROM architectures are used in this work and are described below.

2.2.3.4 Multi-Stage Convolutional Neural Network

Now that the basic mechanics of neural networks, autoencoders, and CNNs have been described, the next section will discuss how to put these pieces together to produce fully non-intrusive neural network-based ROMs.

The following multistage convolutional neural network was introduced in [15]. It assumes that the time-varying spatially dependent solution is on a uniform Cartesian grid and is evenly spaced in time. The state variable of interest is indicated by $q(i; \mu)$, where i is the time step, and $\mu \in \mathbb{R}^{n_\mu}$ is the set of n_μ input parameters. The entire time series sequence is defined by a set of input parameters μ and is denoted by $Q(\mu) = [q(1;\mu), ..., q(n_t;\mu)] \in \mathbb{R}^{nxn_t}$, where n_t is the total number of time steps and n is the number of spatial locations.

To map a set of input parameters μ to a sequence of output parameters Q, three layers of neural networks are constructed and trained independently. The first is an autoencoder that compresses individual snapshots into lower-dimensional representations. The second compresses a series of these compressed snapshots (i.e., the code of the first layer) into a temporally compressed representation. The third and final layer maps μ to the code of the second layer. Figure 2.4 (from [15]) shows a schematic of the interaction between the three layers. Figure 2.4 shows both the flow of data to train each individual network (shown by the black line) and how the three networks are connected during the prediction of new parameters (shown by the red line).

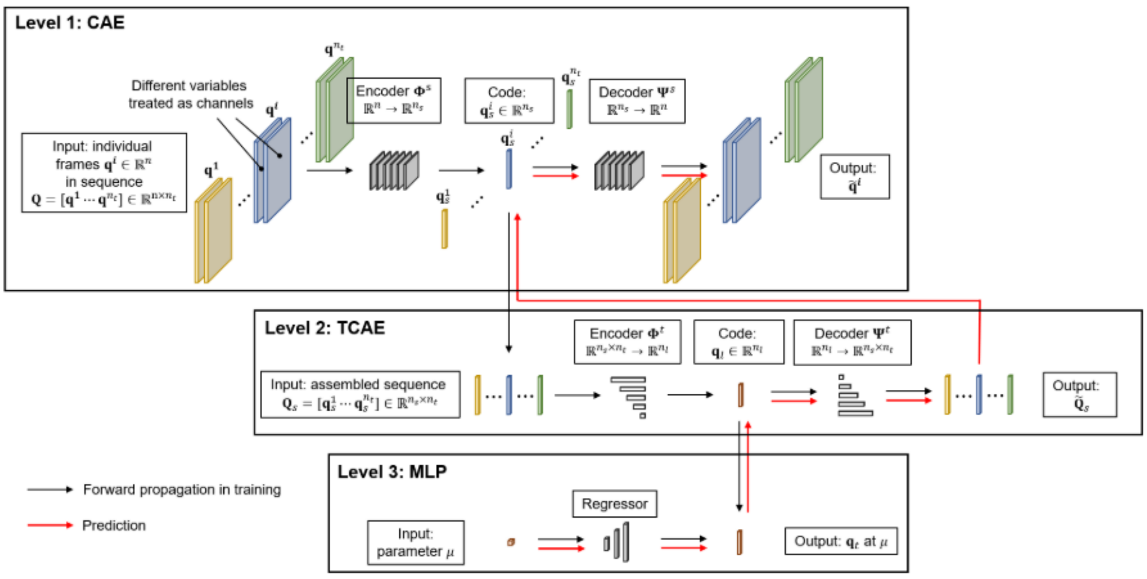

Level 1: CAE
Different variables treated as channels
Input: individual frames $\mathbf{q}^i \in \mathbb{R}^n$ in sequence $\mathbf{Q} = [\mathbf{q}^1 \cdots \mathbf{q}^{n_t}] \in \mathbb{R}^{n \times n_t}$
$\mathbf{q}^1$
$\mathbf{q}^i$
$\mathbf{q}^{n_t}$
Encoder $\mathbf{\Phi}^s$ $\mathbb{R}^n \to \mathbb{R}^{n_s}$
Code: $\mathbf{q}_s^i \in \mathbb{R}^{n_s}$
$\mathbf{q}_s^1$
$\mathbf{q}_s^i$
$\mathbf{q}_s^{n_t}$
Decoder $\mathbf{\Psi}^s$ $\mathbb{R}^{n_s} \to \mathbb{R}^n$
Output: $\tilde{\mathbf{q}}^i$
Level 2: TCAE
Input: assembled sequence $\mathbf{Q}_s = [\mathbf{q}_s^1 \cdots \mathbf{q}_s^{n_t}] \in \mathbb{R}^{n_s \times n_t}$
Encoder $\mathbf{\Phi}^t$ $\mathbb{R}^{n_s \times n_t} \to \mathbb{R}^{n_t}$
Code: $\mathbf{q}_t \in \mathbb{R}^{n_t}$
Decoder $\mathbf{\Psi}^t$ $\mathbb{R}^{n_t} \to \mathbb{R}^{n_s \times n_t}$
Output: $\tilde{\mathbf{Q}}_s$
Level 3: MLP
Input: parameter μ
Regressor
Output: $\mathbf{q}_t$ at μ
Forward propagation in training
Prediction

Figure 2.4: Generalized neural network architecture. Image from [15].

The convolutional autoencoder compresses each individual full-order state variable of interest distribution at some time step i into a lower-dimensional representation. This process trains both the encoder and the decoder. Let us denote the convolutional autoencoder (CAE) encoder by $\Phi_s(q(i;\mu))$ that maps $q \to q_s$, compressing the problem into n_s

dimensional space. Call the decoder of this level $\Psi_s(q_s(i;\mu))$ that maps $q_s \to \hat{q}$. The goal is to find Φ_s and Ψ_s, which produce $\hat{q} \approx q$. There are many hyperparameters to define at this stage, including the number of convolutional filters, the learning rate, the number of hidden layers, stride lengths, and up-sampling strategies during the decoding phase.

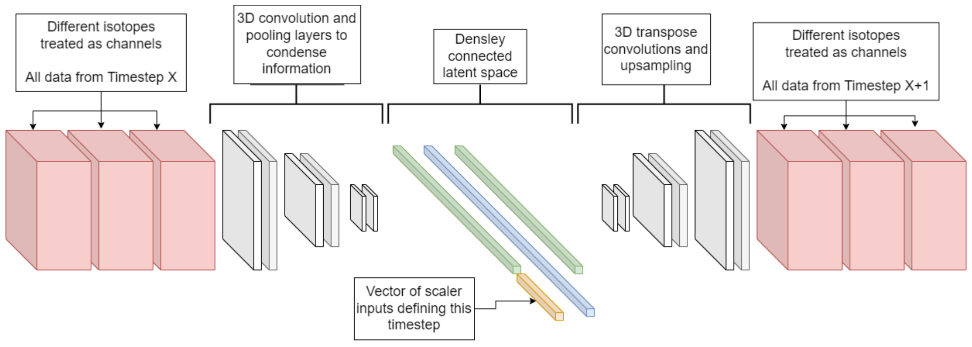
The temporal convolutional autoencoder compresses a series of codes from the first stage $Q_s(\mu) = [q_s(1;\mu), ..., q_s(n_t;\mu)]$ into a lower-dimensional representation. This process trains both the encoder and the decoder. Let us denote the temporal autoencoder (TCAE) encoder by $\Phi_l(Q_s(\mu))$ which maps $Q_s \to q_l$, and the decoder by $\Psi_l(q_l)$ that maps $q_l \to \hat{Q}_s$. An important feature of this convolutional network is that it uses causal convolutions in its convolutional layers. Causal convolutions ensure that the neural network does not violate the ordering in which the underlying data are modeled. In other words, $q_s(i;\mu)$ should not depend on any time steps following i, but should depend only on the proceeding time steps.

The final stage is a multi-layer perceptron (MLP), which maps μ onto $\hat{q}_l$. In the prediction of new parameters, after each stage is independently trained and fine-tuned, a new μ will be input into the MLP stage to produce a set of $\hat{Q}_s$. This $\hat{Q}_s$ will then be input into the decoder of TCAE (Ψ_l) to produce a set of $\hat{q}_s$. This $\hat{q}_s$ is then input into the decoder of the CAE (Ψ_s) to produce a set of $\hat{q}$.

2.2.3.5 CNN Latent Space Stepper

The CNN latent space stepper is a unique method introduced in this work to bypass one constraint of the previous method. One significant difference between the method proposed here and the multistage CNN network from section 2.2.3.4, is that the latter uses a second autoencoder to compress the spatial distribution latent space in the temporal dimension. However, for problems that attempt to predict a time-varying quantity where the drivers of the underlying physics can change at each timestep, this is not sufficient. The method proposed here shows that these CNN based surrogate methods can be extended to the useful application of predicting phenomena where the inputs of the underlying physics may change with each time step. Other methods can do this, but this method is proposed and used here because of its simplicity and effectiveness in fitting the nuclear application of interest.

The framework of the CNN latent space stepper is comprised of an encoding neural network, a concatenation of the latent space with scalar input data (in Chapter 4 application, inputs are power history and timestep length), followed by a decoding neural network that predicts the next time step's state variable of interest. Figure 2.5 shows a cartoon representation of this architecture.



Different isotopes treated as channels
All data from Timestep X
3D convolution and pooling layers to condense information
Densley connected latent space
3D transpose convolutions and upsampling
Different isotopes treated as channels
All data from Timestep X+1
Vector of scaler inputs defining this timestep

Figure 2.5: Illustration of the latent space stepping neural network architecture.

More formally, the architecture can be described as follows. A convolutional encoder compresses each FOM result in some time step i into a lower-dimensional representation. In Figure 2.5 this is represented as the pink boxes. The channels of the input layer of this encoder are all the state variables of interest. For example, they could be isotopic concentrations, power level, or coolant temperature. Denote the encoder by $\Omega(T(i))$, where i is the timestep, and T is the collection of state variable distributions. Ω maps $T(i)$ onto $T_s(i)$, a latent space representation of the state variable of interest. In Figure 2.5 this is represented by the first green line. In this case $T \in \mathbb{R}^{n \times n_t}$, where n is the spatial dimensionality and n_t is the number of state variables in the problem.

Once in the flattened latent space, T_s, these data are concatenated with a vector representation of the input parameter scalars, μ, which describe the timestep i. In Figure 2.5 this is represented by the yellow line. The first demonstration of this method will include only a varying depletion power fraction as part of μ. In this case, μ is a singlevalued scalar representing the depletion power fraction.

After this concatenation, a decoder is used to map this latent space representation of the current timestep's $T(i)$ and μ onto the next timestep's isotopic distribution, $T(i+1)$. In Figure 2.5 this is represented by the second green line and the subsequent decoder. Call the decoder $\Theta(T_s(i))$. In a strict sense, it is not an autoencoder as the inputs and outputs to the network are not identical; however, it mimics the intent of an autoencoder in that it reduces the input to a lower dimensional latent space. In this application, the latent space is then used to step the transient forward in time using that timestep's unique scalar inputs. Θ maps $T_s(i)$ to $\hat{T}(i+1)$.

After the network is trained, predictions would be performed by first taking the current timestep, compressing it with the encoder, attaching the timestep-specific scalar input data onto the latent space, and finally decoding it to the next timestep distribution. The process in full is expressed as

$$T(i+1) = \Theta(\Omega(T(i)), \mu) \quad . \tag{2.19}$$

2.2.4 Non-linear Independent Dual System

One disadvantage of the CNN based methods is that they require spatial discretization to be on a regular Cartesian grid and to be constant across all FOM realizations and new predictions for the ROM. This is a significant problem in many engineering applications where the geometry changes from input to input, such as core optimization, or when the FOM results are composed of measurement data where the sensor locations change between different training cases. Another primary disadvantage is their computational memory requirements. CNN based ROMs do not scale well to large physics problems, as their layers become too unwieldy to contain in memory. This pitfall is demonstrated later in this work for the 3D MC21 application problem in Section 4.2.1. NIDS is a recently proposed method that builds on the ideas of neural operator methods and uses the idea of implicit neural representations to define and capture the geometric constraints of a problem [75]. NIDS is one approach that can avoid the two primary issues with CNN based ROMs.

Neural operator methods take advantage of the notion that neural networks can be seen as universal approximators of continuous functions and, with enough neurons, can approximate non-linear continuous operators ([19], [20]). These algorithms can address some of the shortcomings present in CNN based ROMs ([20], [21], [22], [23], [24], [25]). Specifically, they are able to handle FOMs that are defined on non-regular Cartesian grids and require much less memory than CNN based ROMs. Methods such as these, as well as other physics-informed neural network frameworks ([76], [77], [78]) attempt to solve an underlying set of physics equations (or at least obtain information from the underlying governing physics) or obtain relationships on a continuous geometric basis which allows them to be unconstrained by the Cartesian grid constraints of CNN based methods.

Implicit neural representations can also be used to describe the geometric properties of a FOM application. The signed distance function (SDF) is an example of an implicit representation construct that can be used to represent the surface of a shape by a continuous field where the magnitude of the SDF at any point represents the distance from the surface of some object, and the sign indicates whether the region is inside or outside of the shape. In this way, the SDF field can be used to encode the shape of objects in some problem space. This method has been used for various aerodynamic applications in which the shape of the object of interest is varied during training and prediction. Work such as [18], [16], [79], and [80] represent a few applications in which the SDF was used to capture geometric shapes that had impacts on the state variables of interest.

As described in [16], the SDF can be simply represented by Eq. (2.20):

$$SDF(x) = \begin{cases} d(x, \partial\Omega) & x \notin \Omega \\ 0 & x \in \partial\Omega \\ -d(x, \partial\Omega) & x \in \Omega \end{cases} \tag{2.20}$$

Where:

Ω	= The object boundary
x	= Geometric location
$d(x, \partial\Omega)$	= The shortest geometric distance between x and the surface of the object

Including a SDF in a computational fluid dynamics (CFD) application is valuable because it allows for the encoding of the shape of some object within a flow field. In nuclear engineering applications, the SDF can be modified to represent the minimum geometric distance in the x-y plane from a poison or reflector. In this way, each fuel pin will know how far away the strong absorbing or scattering materials are. The inclusion of a SDF of this nature proved important in determining the correct depletion trajectory in the 3D MC21 application. This application includes a complex checkerboard arrangement of poison pins intermixed with fuel pins. Although not pursued in this work, future work could include using this type of SDF to encode fuel and poison arrangements and allow for quick design space exploration of fuel and poison loading arrangements.

NIDS seeks to create a ROM using this idea of implicit neural representation to allow discretization-independent learning and prediction for unseen or variable geometric fields [25]. NIDS also leverages the idea of a hypernetwork from the neural operator work ([20], [22], [23]). A hypernetwork is a neural network that uses a smaller network to generate the weights and biases of a larger network layer by layer [81]. NIDS leverages the hyperparameter idea by breaking up the inputs to an engineering problem into two pieces, each with its own neural network.

The first network operates on global parameters and is used to predict the weights and biases of a linear output layer, while the final hidden state is the output of the spatial network (which takes geometric data such as x, y, z locations, and SDF information) [25]. Figure 2.6 shows a representation of this idea. Notice that the spatial network outputs the required shape as the final hidden layer values, and the parameter network output is reshaped to form the weight and bias matrices that are combined with the hidden layer values to produce a final output.

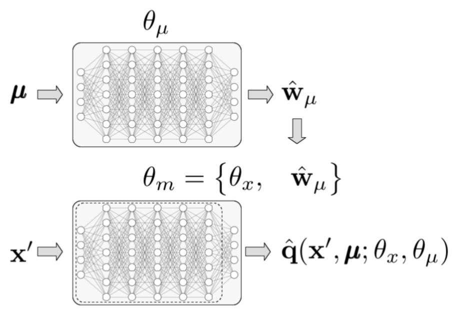

θ_μ
$\boldsymbol{\mu}$
$\hat{\mathbf{w}}_\mu$
$\theta_m = \{\theta_x, \quad \hat{\mathbf{w}}_\mu\}$
$\mathbf{x}'$
$\hat{\mathbf{q}}(\mathbf{x}', \boldsymbol{\mu}; \theta_x, \theta_\mu)$

Figure 2.6: Schematic diagram of a NIDS ROM ([25]).

NIDS, as defined in [25], is summarized below. The spatial network is defined as

$$N_x(x) \triangleq h_x \tag{2.21}$$

Where:

$N_x(x)$	= Spatial network
x	= Spatial information
h_x	= Hidden vector for the final output layer

The parameter network is defined as

$$N_\mu(\mu) \triangleq w_\mu = W_\mu, b_\mu \tag{2.22}$$

Where:

$N_x(x)$	= Parameter network
μ	= Parameter information
w_μ	= Output of parameter network
W_μ	= Reshaped output to be of shape $n_q \times n_h$

The resulting vectors from these two networks (h_x, W_μ, and b_μ) are combined to produce the final predictions as follows:

$$\hat{q}(x,\mu) \triangleq W_\mu h_x + b_\mu \tag{2.23}$$

The spatial network takes pointwise spatial location information, x, as input and outputs a hidden vector h_x. The parameter network takes in parametric information about the FOM, μ, and outputs a vector, $w_\mu \in \mathbb{R}^{(n_q \times n_h)+n_q}$, which is reshaped into weight, W_μ, and bias, b_μ, matrices. To summarize, the parameter matrix produces weight and bias values while the spatial network produces hidden vector values. These two outputs are linearly combined to produce final predictions. In other words, the two networks combine to produce the final output layer of the neural network. In this context the name, Nonlinear Dual Independent System, is made apparent. There are two systems which are independently created (the parameter and spatial networks) in a non-linear fashion (due to the non-linear activation functions in the neural networks).

Formulated in this way, NIDS is suitable for the prediction of new steady-state parameters. However, with a simple change, it can be formulated to perform transient analysis. Both application problems in this work can be interpreted as transient analyses, so this adaptation was necessary to be able to apply this algorithm to nuclear application problems. The first application problem (see Chapter 3) is a transient application, since it analyses the time-dependent dynamics of a rod ejection event.

The second application problem (see Chapter 4) is computed with steady-state solvers to resolve the neutron flux for each depletion timestep. However, it is more precisely interpreted as a time-dependent problem, as it is intended to track the isotopic depletion as a function of varying power history with time. It is important to realize that the isotopic depletion takes place on a timescale that is orders of magnitude slower than that of neutron transport. This allows for decoupling the depletion and neutronics solves.

To use NIDS in a transient context, instead of predicting the state variable of interest given some spatial and parametric input, the *change* in value is predicted. For example, for the Chapter 4 MC21 depletion calculation, instead of predicting the isotopic concentration at some location, x, given some parametric inputs, the NIDS model predicts the *change* in isotopic concentration. For this application problem, the intent is to be able to predict how isotopic concentrations change given an arbitrary power history (different power levels at each timestep). In this way, the NIDS model can step through time and predict how isotopic concentrations change given arbitrary power histories.

The requirements for this transient adaptation are simple. The NIDS algorithm and the model architecture do not need to be changed. Only the preparation of data needs to be changed. Appendix B describes the Python tool Parody that was created to support this work. Parody supports transient and steady-state NIDS applications for 1, 2, and 3 spatial dimensions. See Appendix B for details on how the Parody infrastructure was designed to quickly accommodate the prepossessing required to support transients NIDS.

2.2.5 Metrics

At the global level, the agreement between the relative power distribution of a ROM and its FOM can be defined in a few ways. This work will use some or all of these metrics during comparisons between FOM and ROMs for subsequent chapters.

The first and most common metric seen for regression problems such as this is the MSE. The MSE is also known as an L2 loss and is calculated by computing the square of the difference between the predicted and actual value and averaging that difference,

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(p_i - \hat{p}_i)^2 \quad , \tag{2.24}$$

where:

MSE	= mean squared error between the ROM and the FOM
y_i	= quantity of interest at spatial-temporal location i produced by the FOM
$\hat{y}_i$	= quantity of interest at spatial-temporal location i produced by the ROM.

This is the loss term that is used for all deterministic neural network training in this work and is implied when discussing metrics such as "validation loss" of a neural network. Here and for all metrics defined below, the value of n is the sum total of all spatial locations in all time steps in a single, or set of, FOM realizations.

A related metric is the Mean Absolute Error (MAE). The MAE is also known as the L1 loss. It is related to the MSE except that it does not square the difference but takes the absolute value of the difference in its summation. The MAE is defined as

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \quad . \tag{2.25}$$

The RMSE is the square root of the MSE and is often used to assess neural network training when observing the validation dataset performance,

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \quad . \tag{2.26}$$

Finally, Eq. (2.27) shows the Mean Absolute Percentage Error (MAPE), which normalizes MAE by the maximum value in the collection. This last metric highlights relative error performance, which will be useful in Chapter 4 because average isotopic concentrations can range from 10^{-2} to 10^{-15} $\frac{\#}{\text{barn-cm}}$. The MAPE provides a way to capture how well individual isotopes are predicted compared to each other,

$$\text{MAPE}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n} \frac{|y_i - \hat{y}_i|}{max(|y_i|)} \quad . \tag{2.27}$$

These metrics can also be filtered for some power level threshold. This technique is used during ROM assessment in Chapter 3. For example, $\hat{y}_i$ and y_i can be filtered to include the top $x\%$ power generation regions (for example, the top 10% power generation regions), as this is often of interest for a nuclear designer. This is valuable because a designer may not always be interested in how well the ROM performs for the regions that do not produce the majority of the power. These regions are typically not limiting from a performance perspective. These metrics will be used for individual realizations as well as averaged over a collection of FOM and ROM realizations. When each metric is used in the remainder of this work, its context and use case will be discussed appropriately.

2.3 Uncertainty Quantification

In this work, neural networks are being trained to make predictions of new parameters for nuclear simulations. Before these methods can be applied in practice, there must be confidence in their prediction accuracy. There are many methods available to quantify model uncertainty in neural network predictions ([82]) including MC dropout, Markov chain Monte Carlo, Bayes by Backprop, Laplacian approximations, ensemble techniques, and others. Section 2.3.1 offers a brief summary of each method, and covers Bayes by Backprop in more detail, which is the method used in this work to quantify the epistemic uncertainty of neural network predictions.

Separate from model uncertainty are forward uncertainty quantification (UQ) and input-sensitivity studies that are often performed in the engineering community. In this work, representative forward UQ and input sensitivity studies are performed to assess the applicability of the ROM algorithms to the nuclear reactor design and analysis community (see Chapters 3 and 4). UQ and sensitivity analysis refer to two similar but different activities in engineering and design. UQ is concerned with understanding the uncertainty of the model output due to uncertain inputs. Input-sensitivity studies are concerned with assessing the impact on the output by individual inputs.

One tool for setting performance limits in the context of UQ analyses is Wilks' formula. In the nuclear power industry, Wilks' formula for setting tolerance limits has gained popularity because the U.S. Nuclear Regulatory Commission allows licensing decisions based on best estimate plus uncertainty analyses. In these contexts, where numerous uncertain inputs are sampled and their impact on the output QOI is observed, nuclear reactor designers require a way to quantify the limiting behavior. Because the probability distribution of the output QOI is unknown, non-parametric methods are required to determine confidence bounds on a sample from the unknown distribution. Here, Wilks' formula can be used to determine the sample size required to estimate statistical parameters within a certain degree of confidence. It fits well into the regulatory framework because it incorporates both a tolerance and a confidence in its prediction [83].

The term "rank" is used to describe which order estimate from Wilks' formula is desired. For example, an order estimate of an upper tolerance limit uses the maximum of the sample, the second order estimate uses the second largest, etc. [83]. When the rank is 1, the Wilks' computation is given by the relationship,

$$\alpha > P^N \quad ,$$

where

$100(1-\alpha)$	= confidence desired (typically 95%)
$100P$	= tolerance limit
N	= number of samples.

This concept is best illustrated with an example. Typically, a rank 1, 95/95 ($\alpha = 5$, and $P = 95$) value is used. For a maximum response with a 95% confidence level and 95% probability, the rank 1 evaluation of Wilks' formula requires that 59 samples are collected.

In other words, this means that if 59 calculations are performed, the highest result is within the upper 5% range of the true output distribution with at least a 95% confidence level. The required samples are 93 and 124 for second- and third-order evaluations of Wilks' formula, respectively for 95/95 estimates. [84] has demonstrated that although the Wilks' formula provides a good estimate of the 95/95 limit, a Monte Carlo-based UQ procedure is preferable in some nuclear engineering contexts. This work will reference

Wilks' formula results (using 59 samples), but will also perform Monte Carlo-based forward UQ due to the ease of generating large numbers of FOM results for the application problems in Chapters 3 and 4.

Separate, but related to, UQ analyses are input sensitivity studies, which are also performed for some application problems in this work. Input sensitivity studies are interested in understanding the impact of a single input value on output QOIs. These studies hold all input values constant, while varying one single value across its allowable range. Input sensitivities are of interest to this work because they allow us to understand whether the ROM is capable of separating contributions from individual input parameters. More details are given in Sections 3.4.3 and 3.5.3, where input sensitivity analyses are performed for all input variables of interest for the application problem.

2.3.1 Bayesian Neural Network Theory

In addition to exploring neural network ROMs, this work also explores combining the reduced-order modeling algorithms above with variational inference (VI). This technique is one method that can be used to assign uncertainty to deep neural networks in a Bayesian context. In the context of deep learning, specifically this is used to quantify the uncertainty of the model due to the uncertainty of the weights and biases. This section provides a brief summary of Bayesian interpretations for a neural network-based ROM before describing the specific algorithms used.

The field of UQ identifies two primary categories of uncertainty that can emerge in physics models: aleatoric and epistemic uncertainty. Aleatoric uncertainty encompasses the uncertainty associated with the intrinsic randomness of a system that remains irreducible even with perfect knowledge. For instance, the interaction of a neutron with a Uranium 235 atom contains aleatoric uncertainty, as the occurrence of an interaction stems from a stochastic process. Due to the probabilistic nature of a system, the aleatoric uncertainty cannot be diminished, although it can be quantified in some cases [85]. Epistemic uncertainty is associated with limited knowledge or information about a system. In theory, epistemic uncertainty can be mitigated by acquiring additional data. This type of uncertainty arises from an incomplete understanding of the true underlying data-generating process and can potentially be diminished through increased data collection or measurements. Epistemic uncertainty is frequently related to the selection of model input, the constraints of the chosen model, or an insufficiency of measurement data.

In summary, epistemic uncertainty refers to the reducible part and aleatoric refers to the irreducible part of the uncertainty associated with creating and using a physics model. [85] provides an introduction to this topic in the context of machine learning. In the work herein, epistemic uncertainty is estimated for a subset of applications and ROMs presented throughout. Specifically, epistemic uncertainty is estimated by taking on a Bayesian interpretation of neural networks. Recall that a neural network is a series of linear transformations coupled with non-linear transformations. For ease of reading, Eq. (2.17) is repeated here in compact form,

$$\mathbf{y} = \sigma(\mathbf{W_1}\mathbf{x} + \mathbf{b})\mathbf{W_2} \quad , \tag{2.28}$$

where W_1 represents the weight matrix, b represents the biases, and σ represents the nonlinear activation function. The primary idea behind Bayesian interpretations of neural networks is that instead of computing a point estimate of the weights and biases of the model through stochastic gradient descent, we interpret the parameters as probabilistic variables. These probabilistic variables have an associated probability distribution defined by its own parameters. For example, an individual weight in a model could be interpreted as a Gaussian distribution with a certain mean and standard deviation. The goal of any deep learning Bayesian algorithm is thus to estimate the probability distribution parameters for each model weight and bias value. When it comes time to perform model inference, these distributions are then sampled many times and an estimate of the distribution of expected outputs can be made.

What follows in this section is a discussion that recasts the ROMs into a Bayesian framework. To begin, the aim of a ROM model is to create a function f, with parameters ω (the weights, $\mathbf{W_1}$ and $\mathbf{W_1}$, and biases, b), which can produce an output, y_i, from a set of inputs x_i,

$$y_i = f^{\omega}(x_i) \quad .$$

Call the training dataset $D = \{X, Y\} = \{(x_i, y_i)\}_{i=1}^{N}$, for N training data samples, to denote the set of known inputs and outputs, X and Y. A Bayesian interpretation defines the likelihood of the model, $p(y|x, \omega)$. In other words, this is the probability of producing the outputs y given a set of inputs x and model parameters ω. Using the likelihood of the model, for a given test sample, x^*, a test output, y^*, can be estimated by

$$p(y^*|x^*, D) = \int p(y^*|x^*, \omega)p(\omega|D)d\omega \quad , \tag{2.29}$$

which is the probability of producing the output y^* given a set of inputs x^* for a model trained with training data D. In words, Eq. (2.29) is equivalent to averaging the predictions from an ensemble of networks (a collection of model outputs sampled from $p(y^*|x^*, \omega)$) weighted by the probability that the true model is characterized by parameters ω, $p(\omega|D)$. The primary challenge associated with this formulation is the process of calculating the posterior, $p(\omega|D)$. The posterior distribution of the model parameters can be rewritten using Bayes theorem (note that D is expanded into its constituent elements for clarity),

$$p(\omega|X, Y) = \frac{p(Y|X, \omega)p(\omega)}{p(Y|X)} \quad .$$

Computing this in practice is difficult and typically intractable in the context of neural networks because neural networks often have millions of parameters. There are many approaches to handling and modeling uncertainty for machine learning in this context and, in particular, deep learning. [82] provides a comprehensive review of UQ of epistemic uncertainty under these contexts. Some of the key approaches to estimating the uncertainty of Bayesian neural networks (BNN) are highlighted here.

2.3.1.1 Bayesian Neural Network Approaches

Markov Chain Monte Carlo is one way to compute the posterior. It approximates a probability distribution taking an initial sample of the posterior, and then applying a stochastic transition repeatedly to map out the true posterior. However, it is slow and computationally expensive compared to other methods [86]. Chandra [87] provides a Python based summary and demonstration of Bayesian neural networks using Markov Chain Monte Carlo approaches. Despite their high costs, there has been recent attempts to exploit the symmetry of neural networks to define more efficient sampling methods for faster convergence [88].

To combat the high cost of Markov Chain Monte Carlo, MC dropout was introduced as an effective technique that has been widely used to help solve over-fitting problems in neural networks and compute prediction uncertainty [89]. During training, dropout randomly removes some neurons within the neural network, which can be viewed as a way to reduce co-tuning of a neural network [90] (i.e., it forces the model to learn multiple representations/relationships between inputs and outputs instead of “memorizing” or overfitting to the training data). During inference, neurons are “dropped out” with the same probability as during training. In this way, multiple predictions for the same input behave as an ensemble of models and can be used to estimate the epistemic uncertainty of the model.

Model ensembles are another method that can estimate the uncertainty of a model. In essence, these methods seek to combine multiple independent models trained on a given task in various ways. Deep ensembles [91] train multiple models of the same architecture on the same data using different random seeds to start the parameter training. At inference time each model makes a prediction and the results are statistically combined.

Batch ensembles [92] are similar to deep ensembles with the exception that they are trained using different batches of the training data. They are more attractive than deep ensembles because the training time for each ensemble model will be reduced because of the reduced datasets used for each model.

Finally, probabilistic ensembles [93] train a single probabilistic model. At inference time, outputs from the model are sampled multiple times and the results are statistically combined. These are more attractive because they have a more solid theoretical interpretation in a Bayesian framework, despite being typically more complicated to implement than other approaches. On a model by model basis they are also more expensive to train because they will have more parameters than their deterministic counterparts. This approach will be used during calculations of the VI BNNs used in this work.

Laplacian approximations [94] build Gaussian distributions around the true posterior using Taylor expansions about the maximum a posteriori estimate. Finally, Deep Gaussian processes are another method to acquire model uncertainty. See [95] for details on the Gaussian process modeling method, and [96] for their extension into deep Gaussian processes. These represent a hierarchy of Gaussian processes. As [96] describes, there are three types of connected nodes in a deep Gaussian model. The input node, hidden node (which can be abstracted to be multiple nodes if desired) and the output node. Each node becomes an independent Gaussian process. The attraction to this type of model is that it has been shown to approximate a true posterior of a probability distribution. However, this work employed VI to estimate epistemic model uncertainty. The next section will discuss why it was chosen and how it can be integrated into neural networks models.

2.3.1.2 Variational Inference

VI is another method to estimate the posterior. It is used here because its implementation lends itself to large neural networks easily (i.e, it is scalable to situations with millions of parameters), the algorithm requires small changes to the underlying source code, it is able to incorporate any potential posterior distribution regardless of its complexity, and its training time is small compared to other methods (in particular Markov chain Monte Carlo, which can be intractable for large neural networks). Compared to ensemble methods, VI also has a more firm mathematical foundation describing its interpretations. Ensemble methods have a more empirical interpretation. For these reasons VI was chosen for this work.

In VI, because the posterior is intractable, it is replaced with a distribution of known form (that is, a variational distribution), $q(w|\theta)$, characterized by θ. In practice, this is usually a Gaussian distribution characterized by a mean, μ, and standard deviation, ρ. In VI based models, computing the posterior is considered an optimization problem that can take advantage of the stochastic gradient descent that is performed by training standard neural network models to train against a loss function that characterizes the difference between the true posterior and the variational posterior of known form.

The Kullback-Leibler divergence (KL-divergence) is used [97] to quantify the similarity between a known and unknown probability distribution. The KL-divergence, sometimes referred to as relative entropy, is a mathematical metric that quantifies the difference between two probability distributions. It is a common metric used to describe the similarity of the variational distribution and the posterior of interest for VI problems. Intuitively, the KL-divergence can be understood as a quantity that describes how likely the observations in one distribution would be produced in another. See [98] for a discussion of various interpretations of VI and KL-divergence in different probabilistic contexts. In VI, the goal is to minimize the KL-divergence between the variational and the true posterior probability distributions. In general, KL-divergence between two probability distributions ($q(x)$ and $p(x)$) is defined as

$$KL(q(x)||p(x)) = \int q(x) \log \frac{q(x)}{p(x)} dx \quad .$$

In the context of estimating the posterior distribution of neural network parameters, this becomes

$$KL(q(\omega|\theta)||p(\omega|D)) = \int q(\omega|\theta) \log \frac{q(\omega|\theta)}{p(\omega|D)} d\omega \quad . \tag{2.30}$$

The following steps will demonstrate how to convert this into actionable expressions that can be used within the PyTorch neural network framework used for other ROMs in this work. A key result of this process is that a practitioner can optimize the weights of a neural network using a loss term that does not include the true posterior distribution $p(\omega|D)$. To begin with, the following equations expand and rewrite the KL-divergence used for VI,

$$\begin{aligned}
KL(q(\omega|\theta)||p(\omega|D)) &= \int q(\omega|\theta) \log \frac{q(\omega|\theta)}{p(\omega|D)} d\omega \\
&= \int q(\omega|\theta) \log \frac{q(\omega|\theta)}{p(D|\omega)p(\omega)} p(D) d\omega \\
&= \int q(\omega|\theta) [\log q(\omega|\theta) - \log p(D|\omega) - \log p(\omega) + \log p(D)] d\omega \\
&= KL(q(\omega|\theta)||p(\omega)) + \int q(\omega|\theta) \left[\log p(D|\omega) + \log p(D)\right] d\omega \; . \qquad (2.31)
\end{aligned}$$

Note that $\log p(D)$ does not depend on the model parameters. By taking advantage of the fact that the integral over a probability distribution is 1, i.e., $\log p(D) \int q(\omega|\theta) d\omega = \log p(D)$, Eq. (2.31) can be rewritten as

$$KL(q(\omega|\theta)||p(\omega|D)) = KL(q(\omega|\theta)||p(\omega)) + \mathbb{E}_{q(\omega|\theta)}[\log p(D|\omega)] + \log p(D) \quad . \tag{2.32}$$

The two terms on the left of the right-hand side of Eq. (2.32) are known as the "Variational free energy", $\mathcal{F}$, or the negative of the evidence lower bound (ELBO). Variational free energy is

$$\mathcal{F}(D, \theta) = KL(q(\omega|\theta)||p(\omega)) - \mathbb{E}_{q(\omega|\theta)}[\log p(D|\omega)] \quad . \tag{2.33}$$

Variational free energy can be viewed as the sum of a data-dependent part, called the likelihood cost, and a prior dependent part, called the complexity cost ([99], [100]). Expanding the KL term from Eq. (2.33), the cost function can be rewritten as

$$\begin{aligned}\mathcal{F}(D,\theta) &= KL(q(\omega|\theta)||p(\omega)) - \mathbb{E}_{q(\omega|\theta)}[\log p(D|\omega)] \\ &= \mathbb{E}_{q(\omega|\theta)}[\log q(\omega|\theta)] - \mathbb{E}_{q(\omega|\theta)}[\log p(w)] - \mathbb{E}_{q(\omega|\theta)}[\log p(D|\omega)] \quad . \end{aligned} \tag{2.34}$$

All three of these terms are expectations with respect to the variational distribution, $q(\omega|\theta)$. This means the variational free energy can be written as proportional to,

$$\mathcal{F}(D,\theta) \propto \log q(\omega|\theta) - \log p(w) - \log p(D|\omega) \quad , \tag{2.35}$$

and it can be approximated by drawing samples from of $w^{(i)}$,

$$\mathcal{F}(D,\theta) \propto \frac{1}{N}\sum_{i=1}^{N}[\log q(\omega^{(i)}|\theta) - \log p(\omega^{(i)}) - \log p(D|\omega^{(i)})] \quad . \tag{2.36}$$

Blundell [101] describes an efficient algorithm named Bayes by backprop (BBB) that uses the gradients computed during normal backpropagation to compute the gradients with respect to the estimated means and standard deviations of the parameters of the neural network using the relationship in Eq. (2.36). During the forward pass, the weights are sampled from their variational distribution. Because the forward pass involves a stochastic sampling step, the so-called "reparametrization trick" from [102] is applied. This ensures that during the backward pass the gradients can be computed (i.e., gradients for purely sampled values cannot be computed with backpropagation).

The idea of the reparametrization trick is to use a random variable, $\epsilon \sim q(\epsilon)$, as a non-variational source of noise. The variational parameters, θ, are not sampled directly, but obtained via a deterministic transform $t(\epsilon,\theta)$ such that $\omega = t(\epsilon,\theta)$ follows $q_\theta(\omega)$. ϵ is sampled and thus changes at each iteration but can still be considered a constant with respect to other variables ([103]). This allows a regular backpropagation implementation to work as usual to converge on the variational parameters. In the BBB algorithm proposed by [101] (which assumes that θ describes a Gaussian distribution with mean μ and standard deviation ρ) the reparametrization trick is represented as,

$$t(\epsilon, \theta) = t(\epsilon, \mu, \rho) = \mu + \log(q + \exp(\rho)) \circ \epsilon \quad . \tag{2.37}$$

Next, to leverage Eq. (2.35) in deep learning so that weights and biases, and their distribution parameters θ, can be learned, [101] makes the critical note that if $q(\omega|\theta)d\omega = q(\epsilon)d\epsilon$, the derivative of an expectation for a function f with derivatives in ω can be expressed as the expectation of the derivative,

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(\omega|\theta)}[f(\omega|\theta)] = \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(\omega,\theta)}{\partial\omega}\frac{\partial\omega}{\partial\theta} + \frac{\partial f(\omega,\theta)}{\partial\theta}\right] \quad . \tag{2.38}$$

If we let f from Eq. (2.38) be the right hand side of Eq. (2.35), we note that the function f corresponds to an estimate of the ELBO from a single sample (i.e., Eq. (2.36) for one sample). This has the consequence that the estimate of the gradient will be noisy and the convergence graph of the loss versus the training epoch will be more noisy than in the case of a classic backpropagation [103]. However, training is able to progress successfully. This behavior is noted in the original research paper and is seen empirically during the work presented here.

2.3.2 Bayesian Neural Network Implementation

In the end, what this implies is that by using the regular backpropagation scheme utilized by deterministic neural networks, with augmented loss functions to represent the ELBO instead of a typical MSE (typically used in regression problems), the variational parameters of ω can be learned with standard backpropagation. Because this algorithm ultimately resembles that of a classical neural network training algorithm, optimizing deep learning VI using BBB is simple to do if the Python implementation of the Py-

Torch layers is adjusted to include appropriate changes in the loss terms. Specifically, the PyTorch implementation must include the KL-divergence computation in the loss term recast the weights and biases as probability distributions instead of point estimates. Algorithm 1 shows the final algorithm defined by BBB as it is implemented in this work.

Algorithm 1 Bayes by Backpropagation

```
$\omega = \omega_0$
**while** not converged **do**
    Sample $\epsilon \sim \mathcal{N}(0, I)$
    $\omega = \mu + \log(q + \exp(\rho)) \circ \epsilon$
    $f(\omega, \theta) = \log q(\omega|\theta) - \log p(\omega) - \log p(D|\omega)$
    $\Delta_\theta f = \text{backprop}_\theta f$
    $\omega = \omega - \alpha \Delta_\theta f$
**end while**
```

VI in deep learning doubles the number of trainable parameters required to optimize, since instead of a single weight and bias for a neural network, there are now a mean and standard deviation for each weight and bias. As an improvement, [104] proposed the so-called "flipout" method. This method decorrelates the gradients within a mini-batch by implicitly sampling pseudo-independent weight perturbations for each sample. [104] claims a number of benefits from this algorithmic improvement, including the ability to vectorize the optimization algorithm and the faster convergence of fully connected, recurrent and convolutional neural networks.

Finally, as another improvement, [105] proposed the MOdel Priors with Empirical Bayes using DNN (MOPED) method. This method chooses informed weight priors for Bayesian neural networks using a deterministic version of the Bayesian neural network of interest. Practically, the work herein saw similar performance with and without the MOPED method with slightly improved convergence times.

The Bayesian-torch python package [106] was used to implement all VI related procedures (BBB, flipout, reparametrization, and MOPED) during training and testing of neural networks for the studies in this document. The Bayesian-torch package is a library of neural network layers and utilities that extend the core libraries of PyTorch to enable Bayesian inference in deep learning models. It provides interfaces for both the

BBB with and without flipout, as well as a way to incorporate the MOPED algorithm when initializing the Bayesian neural network by providing a deterministically trained version of a neural network. As mentioned previously, Appendix B describes the Python tool Parody that was created to support the ROM research here. Parody is built on top of PyTorch Lightning, which itself is built on top of PyTorch and therefore can leverage the BNN tools provided by Bayesian-torch. Parody thus includes a simple "bnn" flag to invoke VI via the algorithms described above into the training loop. It works by first converting all layers of the deterministic neural network into their Bayesian counterparts and modifies the loss term used during the backpropagation step to include the requisite KL-divergence relationship so that BBB can proceed as described.

2.4 Summary

This chapter introduced the theoretical foundation for the work in subsequent chapters. Section 2.1 began with a discussion of the methods underlining the FOMs used by later chapters. This includes MPACT for neutron transport, and CTF for the fluid field solutions. Section 2.2 walked through the evolution of ROM methods in the last decade, beginning with POD methods and ending with fully non-intrusive neural network-based ROMs. In particular, it included discussions of CNN and operator based neural network ROMs. Finally, Section 2.3 introduced the concept of BNNs. It included a discussion on the VI algorithm that is used in the next sections to capture epistemic uncertainty for model predictions.

Chapters 3 and 4 apply the tools introduced in this chapter to two engineering application problems of interest. Chapter 3 creates a CNN and NIDS based ROM to model a RIA for a single assembly reactor model and compares their performance. Chapter 4 will extend these ideas to a quarter-core model for an isotopic depletion analysis. In both instances, we see that NIDS is capable of capturing the physics of interest better than the CNN ROMs, and encourages the further research of these concepts for application in reactor design and analysis.

CHAPTER 3

Single-Assembly Reactivity Insertion Accident

A transient single-assembly reactivity insertion accident (RIA) is used to demonstrate the viability of the convolutional neural network (CNN) and Non-Linear Independent Dual System (NIDS) based reduced-order model (ROM) algorithms. The physics analysis code for this study, Virtual Environment for Reactor Analysis (VERA), is invoked using two of its integrated solvers; Michigan Parallel Characteristics Transport Code (MPACT) for neutron transport and COBRA-TF (CTF) for thermal-hydraulics. See Section 2.1 for a complete discussion on the full order model (FOM)s used in this chapter. These codes are coupled together to capture the important interactions of neutronics and thermal-hydraulics. This coupling is important because local energy deposition is directly correlated with the neutron flux. And, the neutron flux is strongly correlated to the surrounding coolant temperatures, which in turn is dependent on the local energy deposition. The code used for this analysis is the VERA toolkit, which couples together MPACT and CTF as described in Section 2.1.2 and 2.1.3 respectively.

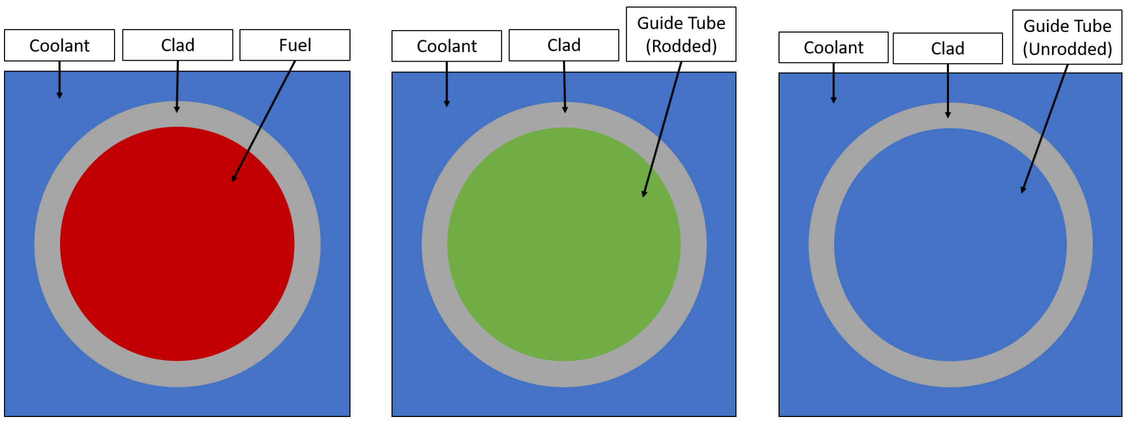
Section 3.1 describes the nuclear model and the specific transient that is modeled for this application problem. Section 3.2 discusses the two types of ROM algorithms (CNN and NIDS) that are tested against the application problem and basic neural network training details. Section 3.3 describes the two types of analysis that are used to assess the success of the ROM models - a uncertainty quantification (UQ) and input sensitivity study. Sections 3.4 and 3.5 discuss the results of the CNN and NIDS models when applied to this application problem. Finally, Section 3.6 summarizes key findings and results.

3.1 Nuclear Model Description

The model used for this work is a 1x1 fuel assembly configuration with reflecting boundary conditions on each side. This small-scale geometry allowed for easy testing of various ROM frameworks quickly compared to using a larger 3x3 assembly or full core configuration. The fuel assembly consists of a 17x17 grid of fuel elements with key geometric parameters described in Table 3.1. Each of the 17x17 grid locations contains either a fuel pin, a rodded guide tube containing a neutron poison, or an unrodded guide tube. Figure 3.1 shows the material definitions for each type of fuel pin configuration.

Parameter	Value	Unit
Axial height	406	cm
Pin pitch	1.26	cm
Clad material	zirc4	-
^{235}U Enrichment	2.5	%
Assembly pitch	21.5	cm

Table 3.1: MPACT single-assembly parameters.



(a) (b) (c)

Figure 3.1: Fuel pin (left), rodded guide tube (middle), and guide tube (right).

Each fuel pin is discretized azimuthally and radially. Radially, Figure 3.2a shows an image of the detailed computational mesh of a single pin for the neutronics solver.

The edits from this model come in the form of pin-averaged relative power, which is the quantity of interest (QOI) for this application problem. In other words, Fig. 3.2a is condensed to one output for each axial height. Figure 3.2b shows the computational mesh of a quarter assembly, with the colors indicating the relative thermal flux levels in each pin for a representative realization of the model. The predominantly blue cells contain neutron poisons that suppress the neutron flux. Figure 3.2c shows the entire assembly, with the colors indicating the relative power levels of each pin. The black regions indicate guide tubes where there is no fuel. All of these images are 2D slices along the xy plane.

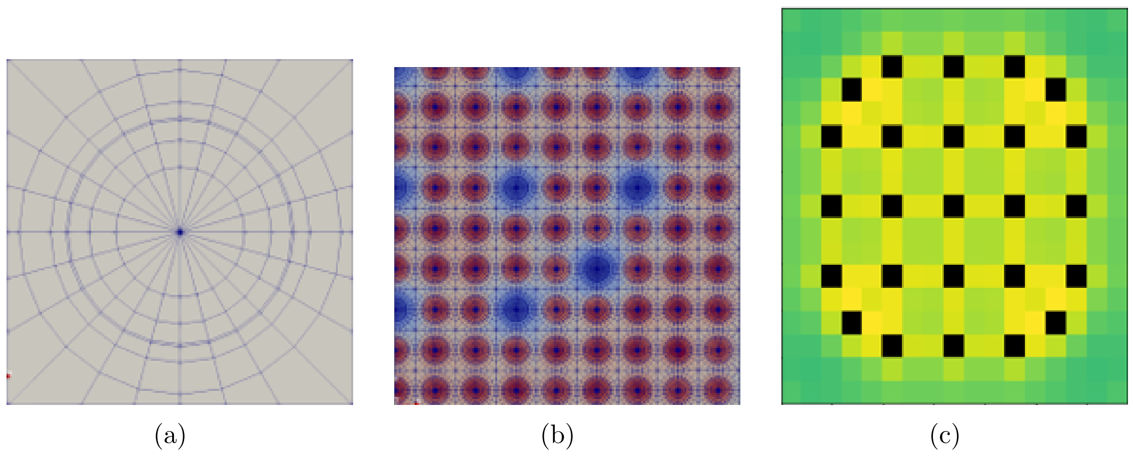

(a) (b) (c)

Figure 3.2: Various fidelities of the 1-assembly model used in this analysis. 3.2a shows the computational mesh of a single pin, 3.2b shows the computational mesh of a quarter assembly, and 3.2c shows the full assembly with each pin represented as one block.

Axially, the problem is broken up into 49 equally spaced regions. Therefore, the edits from this model are of dimension 17x17x49 (that is, there are approximately 14k spatial regions for each spatial solution). The transient is defined as 50 equally spaced time steps of 5 ms, during which the model experiences a rod ejection event removing them axially out of the model at a constant speed. The speed of the rod ejection is different on a case-by-case, spanning 280 to 320 cm/s. The total transient time is thus 0.250 s.

Compared to other problems requiring dimensionality reduction, this problem is relatively small. It would not be unreasonable to perform a full-scale UQ or design optimization analysis using the FOM. Each run takes approximately 1 hour to run on one node of the Idaho National Laboratory (INL) Sawtooth high-performance computing (HPC) machine. Individual compute nodes contain dual Xeon Platinum 8268 processors with 24 cores each and 196 GB of memory. However, this small problem allows for quick experimentation of many neural network architectures, and its non-linear and highly input sensitive nature, due to the complex interaction of the physics involved, make it an ideal benchmark problem for the purpose of assessing the deployment of ROMs in the nuclear field.

The transient of interest is a RIA. In this scenario, a control rod is simulated as being suddenly and quickly ejected from the model. The rod ejection causes a severe increase in reactivity, and therefore power, in the model. When the fuel temperature increases, the Doppler broadening of the resonance capture cross-section of fuel material inserts negative reactivity, which reduces power in the model. This is followed by an increase in temperature of the water moderator. This occurs because of the negative temperature coefficient of reactivity present in this model (and in most commercial power plants). As the temperature increases, the density of the coolant decreases. This decrease in density reduces the effectiveness of neutron moderation, causing a decrease in the effectiveness of the neutron's ability to cause fission, which causes quick decrease in power. However, the effects from the coolant temperature increase are negligible for transient models simulating less than 2-3 seconds, because the heat generated does not have time to conduct out of the pin into the coolant.

Figure 3.3 shows the model average power during a representative transient due to an ejected rod. Note the rapid increase followed by a rapid decrease in power over a short period of time.

The behavior of this power spike is of interest to a nuclear analyst on a global and local scale. On a global scale, analysts are often interested in understanding how much total power is introduced into the coolant loop to ensure the plant design has enough cooling capacity during this accident scenario. For a fast moving transient such as this though, an analyst is likely more interested in local scale energy deposition. It is important to

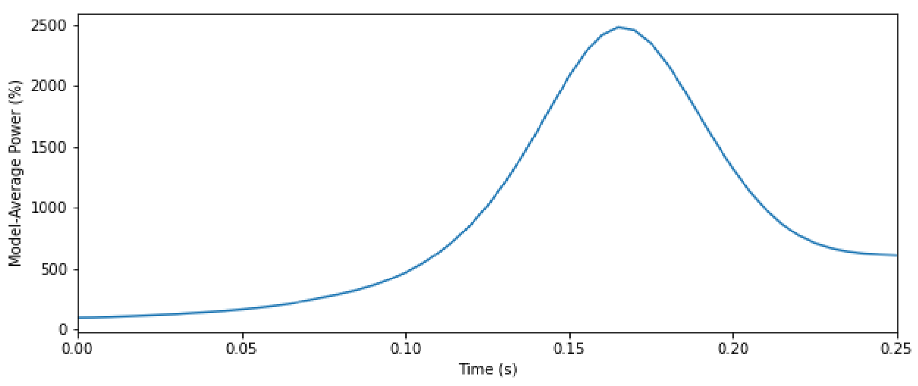


Figure 3.3: A representative power spike for the one-assembly rod ejection casualty.

understand where the power is peaking and by how much. As the analysis will show, some parts of the model have relative power values in excess of 2.0, which means that if the global power spike indicates an assembly average power of 2000%, those parts of the core will actually be greater than 4000%. Understanding the locations and magnitude of the local power peak is important from a fuel rupture and departure from nucleate boiling perspective. There are many analyses that seek to understand the important inputs for this type of scenario (see [107] and [108]).

From a licensing perspective this transient represents an important challenge for designers as well. The RIA is a postulated design basis accident for reactors and its analysis is required for licensed operation. General Design Criteria 28 of 10 CFR Part 50 Appendix A can be violated if this accident is not properly designed for. General Design Criteria 28 of 10 CFR Part 50 Appendix A requires that reactivity limits on the amount and rate of reactivity increase do not exceed certain thresholds, and an accident does not remove to the ability to adequately cool the core or result in a breach pressure boundaries. These considerations make this accident scenario represent both a difficult physics problem and a problem with a high importance in the regulatory landscape.

3.2 Surrogate Modeling Architectures

To assess the viability of ROM methodologies on this representative nuclear engineering application, two algorithms are explored. The first is the multistage CNN algorithm described in Section 2.2.3.4. The second is the NIDS algorithm described in Section 2.2.4. Each are qualitatively and quantitatively evaluated to determine its applicability to the previously described nuclear engineering application. Section 3.2.1 and 3.2.2 summarizes each network's key design features. The notation in each network's description section is assumed known throughout the discussions below. Finally, the general procedure for training the neural networks followed the procedure outlined in Appendix C.

3.2.1 Multi-Stage CNN Architecture

The first surrogate modeling method applied to this application problem is the multistage CNN architecture described in Section 2.2.3.4. The notation and terminology of Section 2.2.3.4 is assumed throughout this section. The general architecture for each level was found through the steps described in Section 3.2.

The convolutional autoencoder (CAE) encoder is made up of 4 consecutive 3D convolutional layers. The size of the kernel was $3 \times 3 \times 3$, and the number of filters ranged from 50 to 150 for all layers except the final layer, which had only one filter. The stride sizes are also fixed at 1, except for the third layer, which have a stride size of 2 to condense each spatial dimension by a factor of two. The stride sizes refer to the size of the step taken to move the convolutional kernel from one location to the next during a convolution across the entire input. A stride size of one means that each individual "pixel" in an input receives a direct convolutional calculation. After the convolutional layers, a flatten layer was used to rearrange the 3 dimensional tensor to one vector. This is followed by one densely connected 400-unit layer.

The CAE decoder consists of a densely connected layer followed by a reshape layer to reproduce a 3D tensor, one deconvolutional layer, and an upsampling layer to reconstruct the desired dimensions. Three final deconvolutional layers were used, with filter sizes of 75 or 150, to produce the final estimate of $\hat{q}$ (the state variable of interest). ReLu activation was used throughout the network except for the final layer of the decoder, which uses a linear activation. The most influential hyperparameters of the CAE are summarized in Table 3.2.

Layer	Type	Filters	Kernel Shape	Stride Length	Units
			Encoder		
1	3D conv	75	1x1x1	1x1x1	-
2	3D conv	75	3x3x3	1x1x1	-
3	3D conv	75	3x3x3	2x2x2	-
4	3D conv	1	3x3x3	1x1x1	-
5	flatten	-	-	-	-
6	dense	-	-	-	400
			Decoder		
7	dense	-	-	-	2025
8	3D deconv	75	3x3x3	-	-
9	upsample	-	2x2x2	-	-
10	3D deconv	75	2x2x2	-	-
11	3D deconv	150	1x1x1	-	-
12	3D deconv	1	1x1x1	-	-

Table 3.2: Summary of CAE hyper-parameters.

The temporal autoencoder (TCAE) encoder begins with a dense layer of 400 units. The size of this layer must match the size of the final layer of the CAE encoder. Six one-dimensional causal convolutional layers are then used with strides of length 2 to compress the temporal information. This is followed by a final dense layer of 600 units.

The TCAE decoder begins with a repeat vector to set the appropriate dimension to reproduce the correct number of q_s's. As a reminder, q_s is the code of the first CNN stage (the CAE) and the input of the second stage (the TCAE). See Section 2.2.3.4 for more details. The repeat vector is followed by a dense layer with 450 units, and 4 1-dimensional causal convolutions, followed by a dense layer of 400 units. ReLu activation units are used in all layers except for the final dense layer, where linear activation is used. The most influential hyperparameters are shown in Table 3.3.

Layer	Type	Filters	Kernel Shape	Stride Length	Units
			Encoder		
1	dense	-	-	-	400
2	1D conv	150	6	2	-
3	1D conv	500	2	2	-
4	1D conv	300	5	2	-
5	1D conv	550	1	2	-
6	1D conv	300	1	2	-
7	1D conv	150	1	2	-
8	flatten	-	-	-	-
9	dense	-	-	-	600
			Decoder		
10	repeat-vector	-	-	-	-
11	dense	-	-	-	450
12	1D deconv	950	25	6	-
13	1D deconv	950	25	5	-
14	1D deconv	950	12	1	-
15	1D deconv	1400	18	3	-
16	dense	-	-	-	400

Table 3.3: Summary of TCAE hyper-parameters.

The final stage of the multistage CNN ROM is a multi-layer perceptron (MLP). The performance of the ROM did not appear to be sensitive to the hyperparameters of the MLP. A hyperparameter optimization was performed, but little performance improvement was seen over the baseline model. The MLP consists of two densely connected layers, each with 400 nodes and ReLu activation layers.

A second MLP was used for this application problem to capture the magnitude of the average power of the model. The spatial edits of the MPACT model that describe the 3D distribution are relative powers, not absolute powers. In addition to spatially dependent edits, MPACT also outputs scalar edits. This includes the average power of the model, which is what this second MLP was fitted against. It is used to capture the transient behavior of the model average power. Let the time-dependent model average power be denoted by $X = [x(1), ..., x(n_t)]$, where x is the average power of the model at some time step. The goal of this MLP (call it Ω) is to map a set of inputs $\mu \in \mathbb{R}^{n_\mu}$ onto $\hat{X} \in \mathbb{R}^{n_t}$.

This MLP uses a set of 4 dense layers with 500 units in each layer. Although an MLP is used here, any other multidimensional fitting scheme could be used to attempt to capture the time dynamics of the average power of the model.

In summary, two systems are trained, consisting of 4 neural networks. The first is the model average power MLP, which maps μ to the time dynamics of the model average power. The second reproduces the 3D distributions of relative pin powers and is comprised of three levels - CAE, TCAE, and MLP. In order, these constituent levels act to compress the time dynamics of the 3-dimensional relative power distribution by first compressing each snapshot in the spatial dimension, compressing a series of snapshots in the time dimension, and finally mapping a set of inputs onto the fully compressed information. Ultimately, a new parameter prediction can be achieved by performing $\Psi_s(\Psi_l(P(\mu')))$ for some new set of input parameters μ'. The process of training and using this architecture is visually represented in Fig. 2.4.

The final architectures are then trained for the specific application problem described in Section 3.1. For a generic application problem different hyperparameters will be more effective (number of layers, stride length, hidden units, learning rates, etc.). However, for the application problem used herein, the specific architecture described above proved to be the most effective.

3.2.2 NIDS Neural Network Architecture

The second surrogate modeling method applied to this application problem is the NIDS method described in Section 2.2.4. The notation in Section 2.2.4 is assumed throughout the discussion of the network architecture. The general architecture for each level was found through the steps described in Section 3.2.

The spatial network and the parameter network are both dense neural networks with ReLU activation. After combining the output of the two networks, the final layer of the NIDS architecture uses a linear activation. The hyperparameters for the NIDS algorithm are much less complex than those for the CNN based ROM described in Section 2.2.3.4. Table 3.4 lists the main parameters defining the architecture of neural networks. For ease of reading, the image from [25] is reproduced in Figure 3.4

Parameter	Value
Param input dimensions	8
Param dense layers	6
Param units per layer	800
Latent space dimensions	800
Spatial input dimensions	3
Spatial dense layers	6
Spatial units per layer	800
Output dimensions	2

Table 3.4: NIDS neural network hyperparameters for the MPACT application.

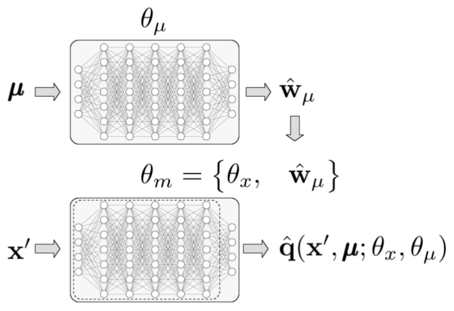

θ_μ
$\boldsymbol{\mu}$
$\hat{\mathbf{w}}_\mu$
$\theta_m = \{\theta_x, \quad \hat{\mathbf{w}}_\mu\}$
$\mathbf{x}'$
$\hat{\mathbf{q}}(\mathbf{x}', \boldsymbol{\mu}; \theta_x, \theta_\mu)$

Figure 3.4: Schematic diagram of a NIDS network ([25]).

3.3 Description of Modeling Scenarios and Figures of Merits

Each run of the RIA analysis problem is defined by several scalar inputs that determine how the transient progresses. The inputs used in this study are summarized in Table 3.5. Each parameter affects different parts of the transient in different ways (heat deposition rate, power peak magnitude, power peak suppression speed, time of maximum power, etc.). Table 3.5 indicates the allowable range of each parameter within the analysis. These ranges are intended to emulate a real application that would have uncertainty in its input parameters; however, these ranges are not representative of any specific core design. Finally, the last column in Table 3.5 shows how many individual evaluations are performed in the sensitivity study, which is discussed in more detail in this section.

Parameter	+/- Range	Units	Number of input sensitivity evaluations
Rod Ejection Speed	300 ± 20	cm/s	9
Starting Power	100 ± 3	%	7
Guide Tube Coefficient	0.5 ± 0.2	-	5
Gamma Heating Fraction	0.02 ± 0.005	%	5
Flow Rate	50 ± 1.5	%	7
Fuel Rod Gap Conductance	5500 ± 500	W/m^2/K	5
Inlet Temperature	565 ± 2	K	5

Table 3.5: Input parameters for uncertainty quantification and sensitivity studies.

Two studies are performed that represent typical analyses performed in the nuclear engineering field (and are consistent with many other engineering disciplines) - a UQ and input sensitivity study. These studies assess the efficacy and robustness of using a ROM in the context of an RIA analysis. These studies are also used to highlight the impact of using different training approaches when creating a ROM.

The UQ analysis was a simple forward UQ study where inputs to the VERA model are varied within their ranges, and the impact of the inputs on relative power distributions is observed. This analysis is primarily concerned with the ability of a ROM to capture the resulting variability in relative power from uncertain inputs. Specifically, a suite of 100 FOM MPACT runs are executed, sampling the input parameters described in Table 3.5. All sampling assumes a uniform distribution.

The sensitivity study holds all but one parameter constant, with that single parameter varied within its range in a uniform stepwise fashion. The constant parameters are held in the center of their allowable ranges. The sensitivity study assesses how well a ROM is able to capture the contribution of individual inputs to the overall transient. Again, the impact on relative power is observed.

Both studies represent a commonly encountered analysis in many engineering disciplines, and both types of analyses benefit when a cheaper ROM is available for analysis instead of having to rely on a computationally expensive FOM. For context, 1 FOM result takes 1 hour to execute, was computed using MPACT version 2.1.0, and was executed on the INL Sawtooth1 supercomputers on a compute node consisting of 48 CPUs.

3.4 CNN Results and Discussion

3.4.1 CNN Training and Qualitative Results

Once the set of hyperparameters was chosen for the neural network and the suite of 100 MPACT runs was executed, the multi-stage CNN ROM was trained. To explore how many FOM results are required for adequate performance, 6 neural networks were trained using varying numbers of FOM results. The set of training data sets and their validation losses at the various stages of the network are shown in Table 3.6.

Training occurred independently for each stage of the network until the neural network for that stage experienced a training plateau in its loss function. These neural networks used TensorFlow's standard callbacks to reduce the learning rate on a training plateau (patience of 8, factor of 0.2), and early stopping criteria with a patience of 14 to ensure efficient use of compute resources and training convergence. The time to train each neural network is also listed in Table 3.6, which scales mostly linearly with the training set size.

Figure 3.5 shows the expected result that increasing the number of training cases improves performance. Histograms of $q/\hat{q}$ for every spatial location for all 100 FOM realizations are shown. Table 3.7 shows these results in table format for each training case. Each value in Table 3.7 represents the percent agreement between ROM and FOM in various forms. The first two columns show the maximum model-average power error and

Case	Number of Training Inputs	Training Time (min)	CAE Validation Loss	TCAE Validation Loss	MLP Validation Loss
0	5	3	8.9×10^{-4}	2.0×10^{-4}	7.1×10^{-5}
1	10	5	4.9×10^{-4}	1.8×10^{-4}	3.5×10^{-4}
2	25	8	3.2×10^{-4}	2.3×10^{-4}	9.9×10^{-4}
3	50	16	2.3×10^{-4}	1.3×10^{-4}	3.8×10^{-4}
4	75	25	1.8×10^{-4}	1.3×10^{-4}	3.7×10^{-4}
5	100	29	1.1×10^{-4}	9.0×10^{-5}	8.1×10^{-5}

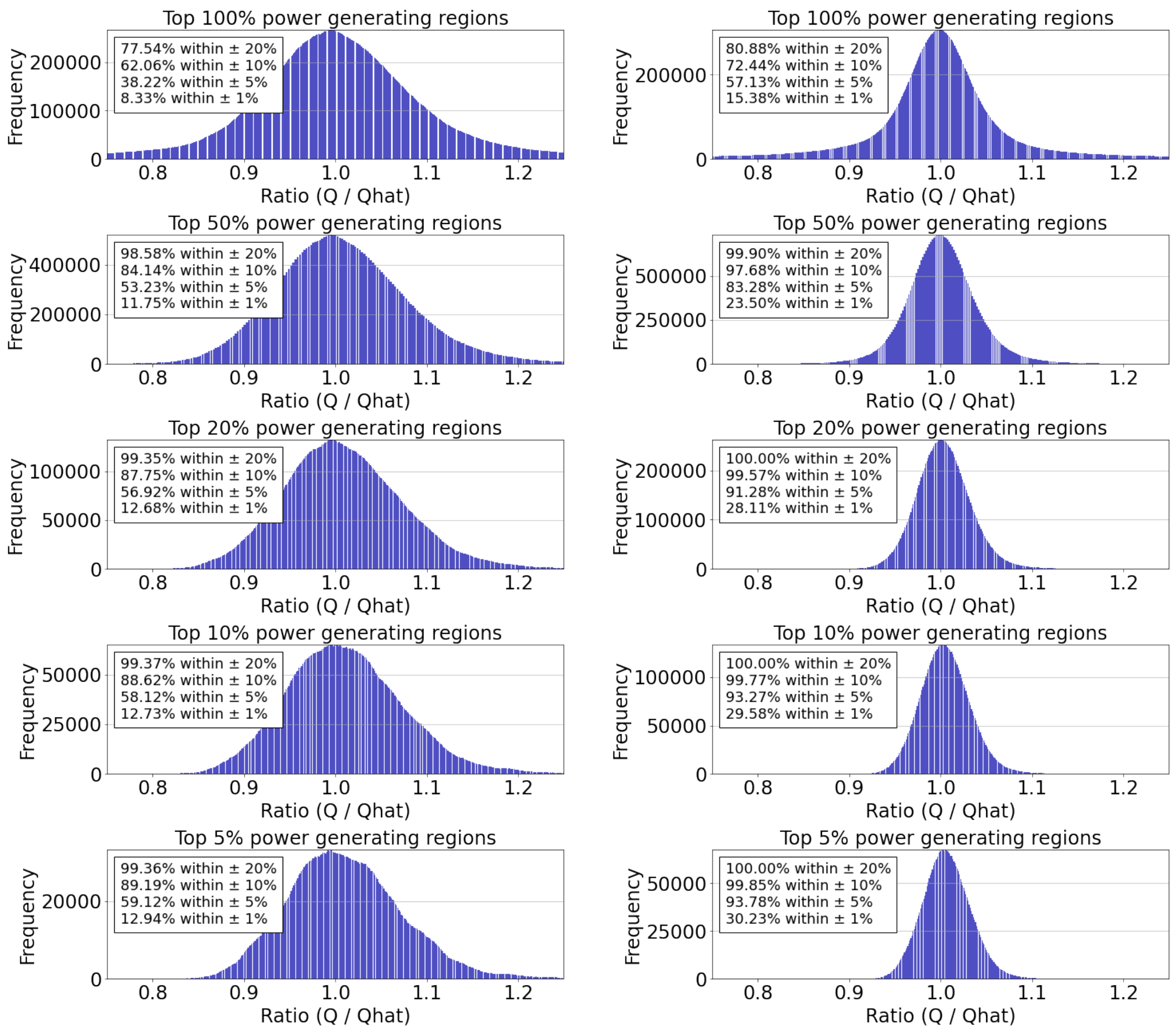
Table 3.6: Summary of model and training datasets.

the time-to-max power error. These metrics deal with global quantities captured by the model-average power MLP. The remaining columns represent the percentage of spatial locations for each reproduced MPACT run that falls within some band of agreement. These values are further filtered to determine how much power is generated in that region. For example, the rightmost column is filtered to include only the top 5% power generating regions in the model and represents what percentage of those regions have ROM results that fall within +/- 5% of the FOM result. These filtered metrics are important to keep in mind for nuclear applications, as designers are often interested only in regions reaching some power thresholds, as they are the locations that set performance limits.

Case	Average maximum power error (%)	Average time-to-max power error (%)	top 100% reg - +/-5	top 10% reg - +/-5	top 5% reg - +/-5
0	0.00	0.01	21.0	28.7	28.4
1	-0.06	-0.02	41.5	69.7	70.0
2	-0.11	-0.01	27.9	38.4	37.8
3	0.01	0.00	51.2	82.7	83.1
4	-0.01	0.00	54.6	90.1	90.8
5	-0.01	0.00	60.3	95.7	96.2

Table 3.7: Summary of training performance over 100 MPACT runs.

Performance of the CAE and TCAE is also demonstrated with qualitative visual checks. Starting with the CAE, Figure 3.6 shows the relative power condensed axially

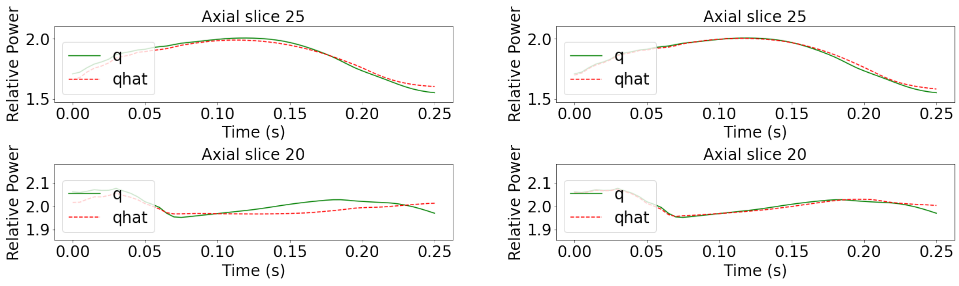


(a) (b)

Figure 3.5: Histogram of errors for the recreated suite of 100 cases for Case 0 (left column of plots) and 5 (right column of plots).

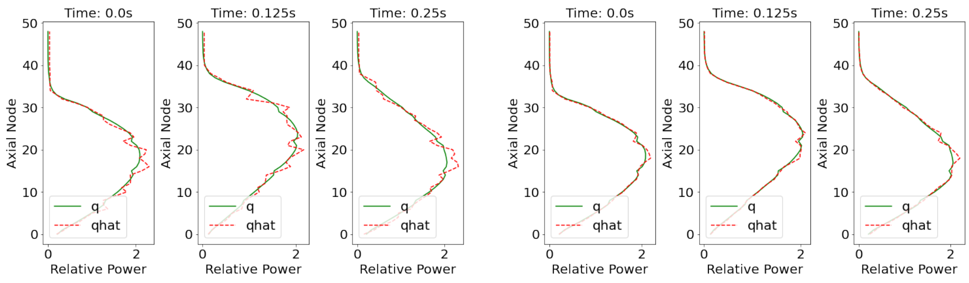
for selected axial slices as a function of the time step for Case 0 and Case 5. Figure 3.7 shows the relative power condensed radially for select transient times as a function of the axial height for Case 0 and Case 5.

However, the next stage, the TCAE, results in a significant loss in accuracy in power distribution reconstruction. Figures 3.8 and 3.9 illustrate the performance of the TCAE. Training the TCAE and then reconstructing the power distributions from the resulting code of the TCAE results in degraded performance. The axial slices with time appear to have stepwise features. It captures the general upward or downward trends of the power distributions, but so poorly that it requires large periodic course corrections, resulting in the observed stepwise features.



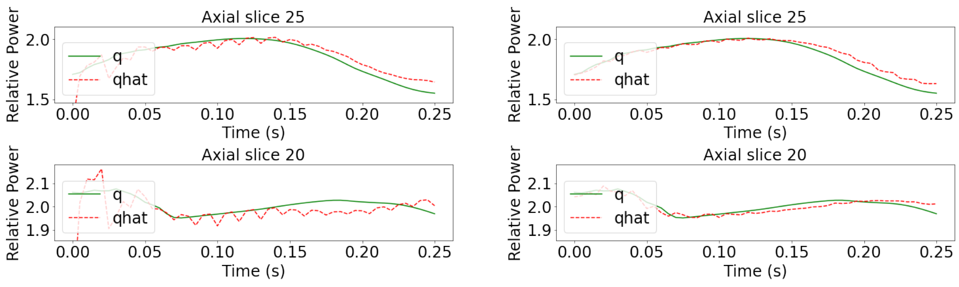
(a) (b)

Figure 3.6: CAE radially condensed relative power results for Case 0 and 5.



(a) (b)

Figure 3.7: CAE axially condensed relative power results for Case 0 and 5.

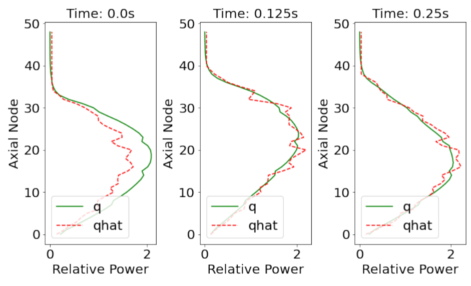


(a) (b)

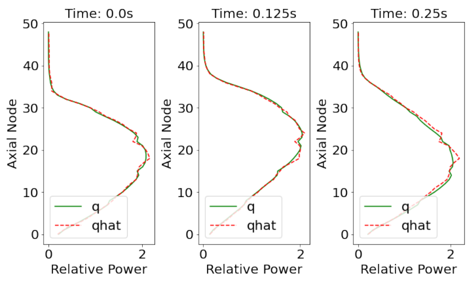
Figure 3.8: TCAE axially condensed relative power results for Case 0 and 5.

3.4.2 CNN Uncertainty Quantification Performance

Once the three-stage network is trained, each stage is combined to produce relative powers for individual pin and axial locations as a function of time. Specifically, this section assesses how well local relative powers agree between the FOM and ROM when performing a simple forward UQ analysis.



(a)



(b)

Figure 3.9: TCAE axially condensed relative power results for Case 0 and 5

Using the CNN model for uncertainty quantification reveals that although some qualitative metrics show reasonable performance, the ability of the neural network to accurately predict relative powers for new parameters is limited. Figure 3.10 shows the average and standard deviation of a representative local region within the model after 100 ROM realizations with randomly sampled inputs. This model uses the Case 0 training set, or just 5 FOM results. Notice how the model average relative power over all 100 realizations appears to look reasonable; however, the standard deviation in model average power due to uncertain inputs is not captured by the CNN ROM. The variation is much too large and does not exhibit the trait of smaller uncertainty at the beginning of the transient followed by a larger uncertainty near timestep 40.

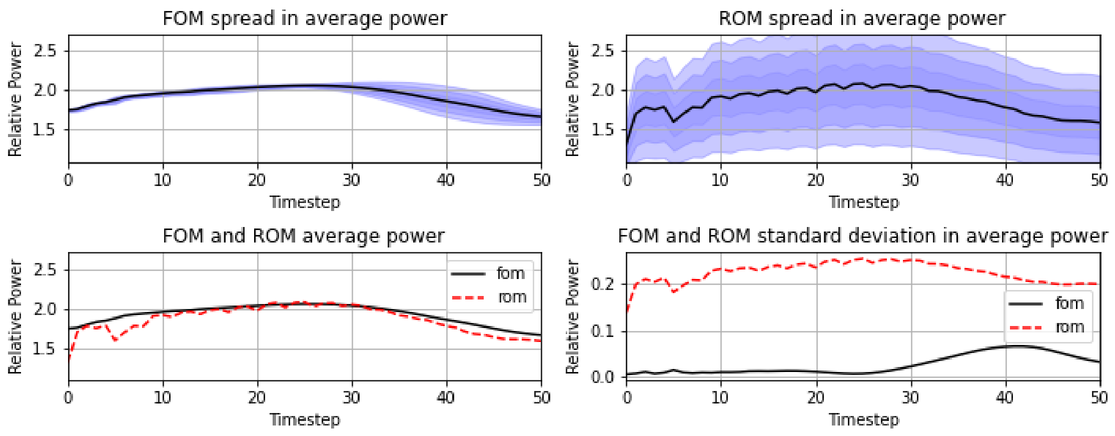


Figure 3.10: FOM and ROM forward uncertainty propagation for relative power at a single location in the core for Case 0. Each shade of blue represents one additional standard deviation of separation from the mean.

Adequate improvement is not seen even after increasing the dataset from 5 to 100 FOM realizations. Instead, the deficiencies of Case 0 results are exacerbated. Figure 3.11

shows the performance using the Case 5 training set. This amount of data is already out of the realm of reasonableness for a practical ROM (i.e., a ROM requiring 100 FOM realizations is of little value). Still, the performance is poor. The standard deviation in the average power shrinks but is more uncertain in the beginning of the transient than in the end, a complete reversal of the actual FOM standard deviation behavior. These results indicate that the CNN based ROM as implemented here is not capable of capturing the impact of individual input parameters on relative power and would be a poor candidate for deployment in a nuclear engineering environment.

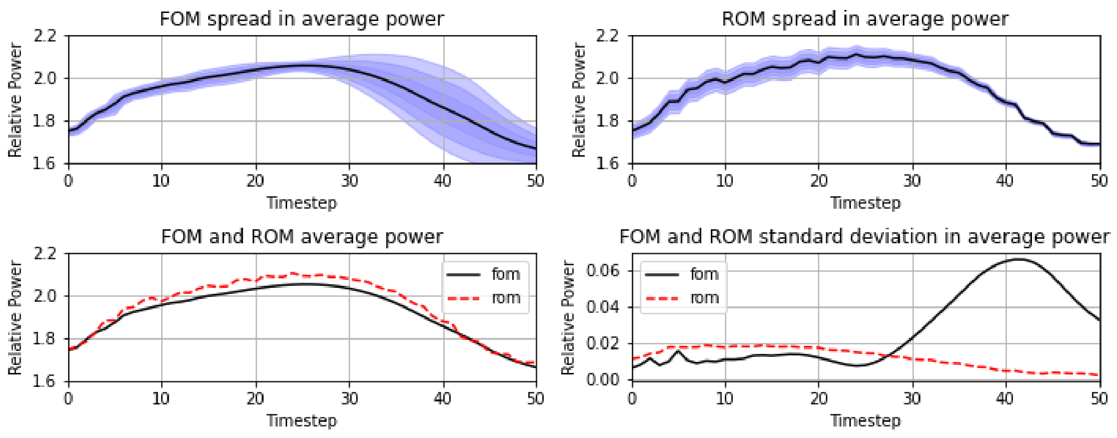


Figure 3.11: FOM and ROM forward uncertainty propagation for relative power at a single location in the core for Case 5. Each shade of blue represents one additional standard deviation of separation from the mean.

For comparison with other ROM methodologies, the metrics shown in Figures 3.10 and 3.11 are condensed to single figures for each training set. Table 3.8 shows the average agreement between the average power of the model over the UQ analysis of 100 realizations. In other words, it represents the average error between the red and black lines in the lower left image of Figure 3.11 for each location in the model. Table 3.8 also shows this condensed result for the agreement in standard deviations (that is, the plot on the lower right of Figure 3.11).

Table 3.8 shows an improvement in these metrics as the number of training samples increases, but only marginally. This table is misleading also because although the MSE of the standard deviation improves dramatically, Fig. 3.11 shows that the shape of the predicted standard deviation in relative power is not captured by the ROM. The improvement in Mean Squared Error (MSE) of the standard deviation is only due to the collapse of any variation in power, not to any increase in ability to capture the input-output relationships. Performance does not continue to increase even when including 50 or more FOM realizations in the training data set.

Case	MSE of average power	MSE of standard deviation
0	1.5×10^{-1}	1.7×10^{-2}
1	1.5×10^{-1}	4.5×10^{-4}
2	1.4×10^{-1}	2.6×10^{-4}
3	1.4×10^{-1}	4.7×10^{-4}
4	1.4×10^{-1}	5.6×10^{-4}
5	1.4×10^{-1}	4.3×10^{-4}

Table 3.8: Performance metrics for increasing training dataset sizes for the CNN.

3.4.3 CNN Sensitivity Analysis Performance

The previous section showed that within the CNN based ROM there is an inability to capture the finer details of relative power distributions, especially in the context of input impact variability. This lack of ability is evident due to the poor spread in model power in local regions of the model by the ROM when compared to the FOM. This section digs deeper and further highlights the CNN based ROMs inability to capture individual input impacts on relative power by performing the individual sensitivity analysis.

Figure 3.12 shows the results of the study of power sensitivity using the best available CNN ROM. In this figure, all input parameters are held constant, while the input, "starting power", is varied within is range according to Table 3.5 (97-103%). In essence, it is a simplified UQ analysis in that only one parameter is varied, but it is more informative in that it isolates the ROM's ability to capture the impact on the solution due only to one parameter. Notice in Fig. 3.12 that, much like the UQ analysis, there is good agreement of the average power between the FOM and ROM; however, the standard deviation in the results is very poor performing. The ROM always predicts convergence in relative power at the end of the transient, even though that is precisely where the FOM predicts the spread to be the greatest. What plots like this show is that the CNN ROM is not capturing input dependence, but simply finds the average output of the FOM.

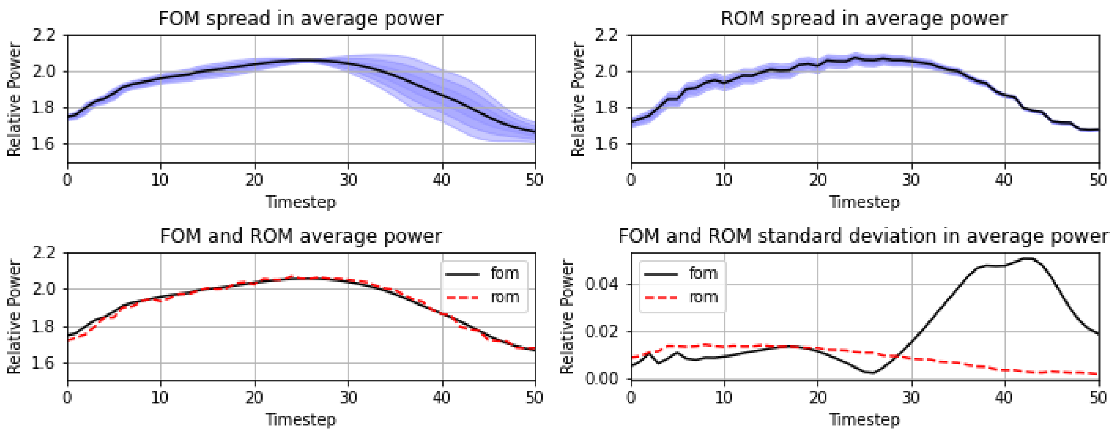


Figure 3.12: FOM and ROM initial power sensitivity for the multi-stage CNN for Case 5. Each shade of blue represents one additional standard deviation of separation from the mean.

For completeness, a sensitivity study for each input parameter from Table 3.5 is run with the FOM and ROM. The mean-square-error is summarized in Table 3.9. Much like the UQ analysis, once a certain threshold is reached in the size of the training set, the performance does not improve beyond what is shown in Table 3.9.

Parameter	MSE of average power	MSE of standard deviation
dhfrac	1.5×10^{-3}	5.6×10^{-5}
flow	1.6×10^{-3}	2.2×10^{-4}
guide tube coefficient	1.2×10^{-3}	3.6×10^{-4}
hgap	1.3×10^{-3}	3.1×10^{-4}
power	1.7×10^{-3}	3.3×10^{-4}
rodspeed	1.6×10^{-3}	8.5×10^{-5}
tinlet	1.5×10^{-3}	3.0×10^{-4}

Table 3.9: Comparison of UQ performance metrics when isolating input parameters.

3.5 NIDS Results and Discussion

3.5.1 NIDS Training and Qualitative Results

Once the set of hyperparameters was chosen for the NIDS model, a set of 100 MPACT runs was executed, sampling the input parameters according to Table 3.5. To explore how many FOM results are required for adequate performance for the NIDS model, 4 neural networks are trained using varying numbers of FOM results - much like the analysis performed in Section 3.4.1. Table 3.10 shows the cases, the number of FOM realizations in each case, the total training time and test metrics.

This process is identical to the CNN training regimen. In this case, however, the Parody tool (see Appendix B for details) is used for all ROM network implementations. Parody was built with efficient training in mind and incorporates common methods used today for training neural networks. Leveraging Pytorch-Lightning's intuitive interface ([109]), this includes callbacks to implement early stopping criteria (with patience of 20 epochs) and reduction of the learning rate on training plateaus (with patience of 8 epochs). This practice provides confidence that a given neural network is trained to its performance limit while not wasting resources on training epochs that no longer improve results. Additionally, more training data were not incorporated after 50 FOM realizations. As are shown below, 50 FOM realizations are enough to provide promising results for this class of ROM algorithm for this application problem.

Case	Number of Training Inputs	Training Time (min)	Test RMSE	TEST MAE
0	5	43	8.7×10^{-4}	4.2×10^{-4}
1	10	71	7.1×10^{-4}	3.2×10^{-4}
2	25	157	3.3×10^{-4}	1.6×10^{-4}
3	50	237	4.0×10^{-4}	2.7×10^{-4}

Table 3.10: Summary of NIDS model and training data sets.

Next, the figures of merits discussed in 3.3 are used to assess the ability of the trained networks to reproduce the results from 100 FOM realizations. Figure 3.13 shows how increasing the number of training cases improves performance. It is already clear that the NIDS framework significantly improves performance compared to similar plots of the CNN model (see Fig. 3.5).

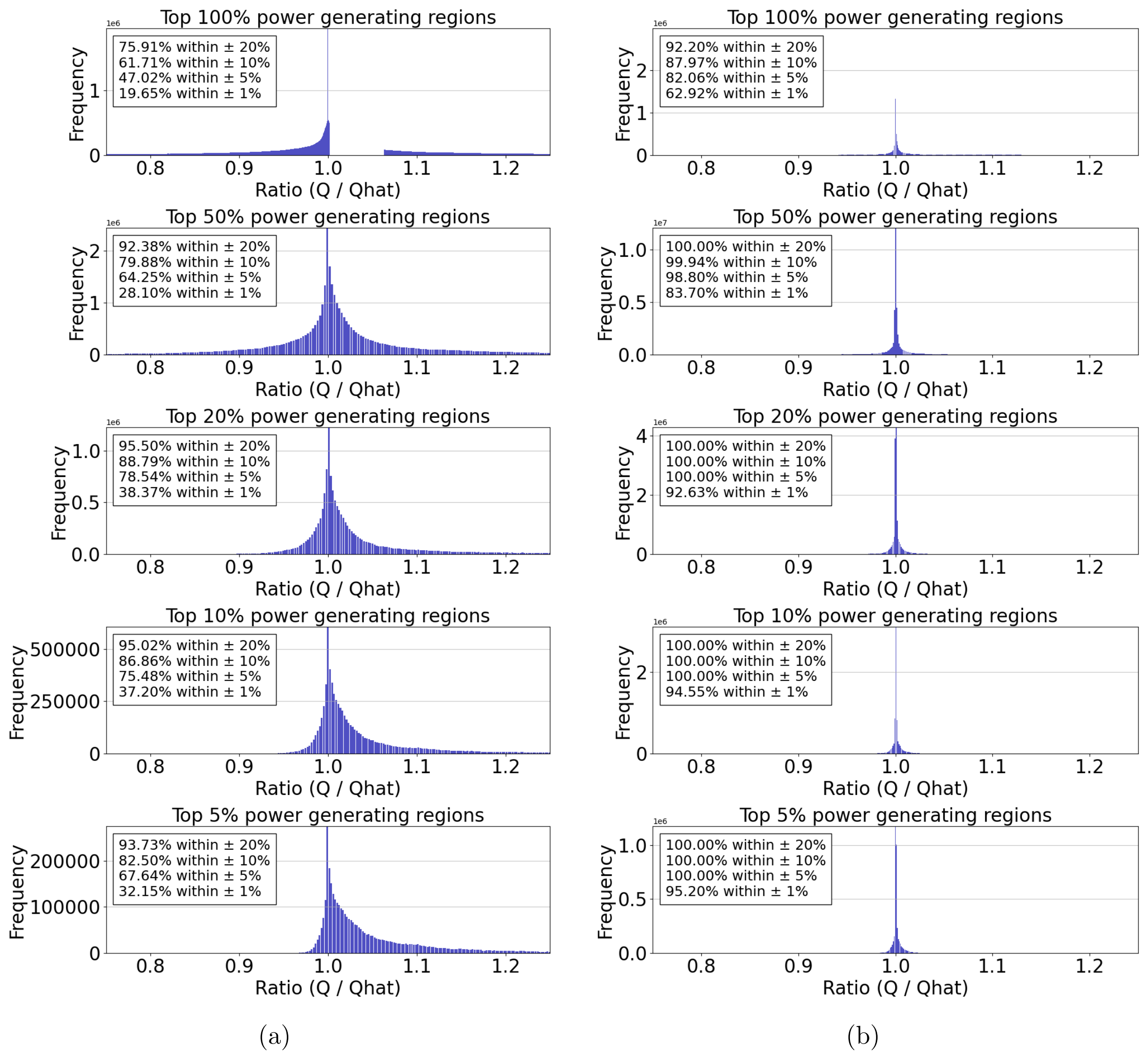


Figure 3.13: Histogram of ratios for the recreated suite of 100 cases. Figures 3.13a and 3.13b show Case 0 and 3 results respectively.

Unlike the CNN framework, there is no second stage where the results have the opportunity to be compressed and uncompressed for a second time, losing fidelity in the results. Section 3.4.1 discusses and shows, with axially condensed and radially condensed power trace plots, how this occurred for the multistage CNN based ROM and introduced a stepwise pattern in the graphs that capture time evolution. For the NIDS algorithm, the plots of the condensed relative power (axially and radially) are shown below, along with the cross sections of the relative power. Figures 3.14 and 3.15 show this for the Case 0 and Case 3 training sets for individual realizations of power distributions.

In Figures 3.14 and 3.15, each row represents an individual realization of the FOM and ROM. These figures have two columns of line plots. The first column shows the time evolution of the relative power for a single representative x-y-z location in the assembly. As a function of time, the NIDS model is able to capture the evolution quite well. The second column shows a single axial power profile for one point in that x-y plane. Again, this shows that axial power profiles are also captured quite well. The second and third columns show 2D distributions across high-power generating planes in the assembly for the FOM and ROM, with the fifth column showing the ratio between the two. Taken all together, these plots demonstrate a remarkable agreement between FOM and ROM.

Qualitatively, significantly improved performance is observed with the NIDS algorithm compared to the CNN framework. With the CNN model, even at the spatially condensed level, the plots showed deviation from truth and prediction. However, with NIDS, on a spatially condensed basis, the results are almost indistinguishable. Compare Figures 3.14 and 3.15 with Fig. 3.8.

The CNN based ROMs had reasonable performance at the CAE stage (average power traces, condensed on some spatial basis), but had degraded performance once the TCAE stage was complete. The NIDS based ROMs have good spatially condensed performance and individual location performance, and, as we will see later, this allows them to be more useful on both a UQ and sensitivity-based analysis.

3.5.2 NIDS Uncertainty Quantification Performance

The NIDS model is able to more accurately quantify uncertainties than the CNN model. Figures 3.16, 3.17, and 3.18 show the performance of the NIDS model for the Cases 0, 2 and 3 training sets, respectively. These plots show the results of a forward UQ analysis

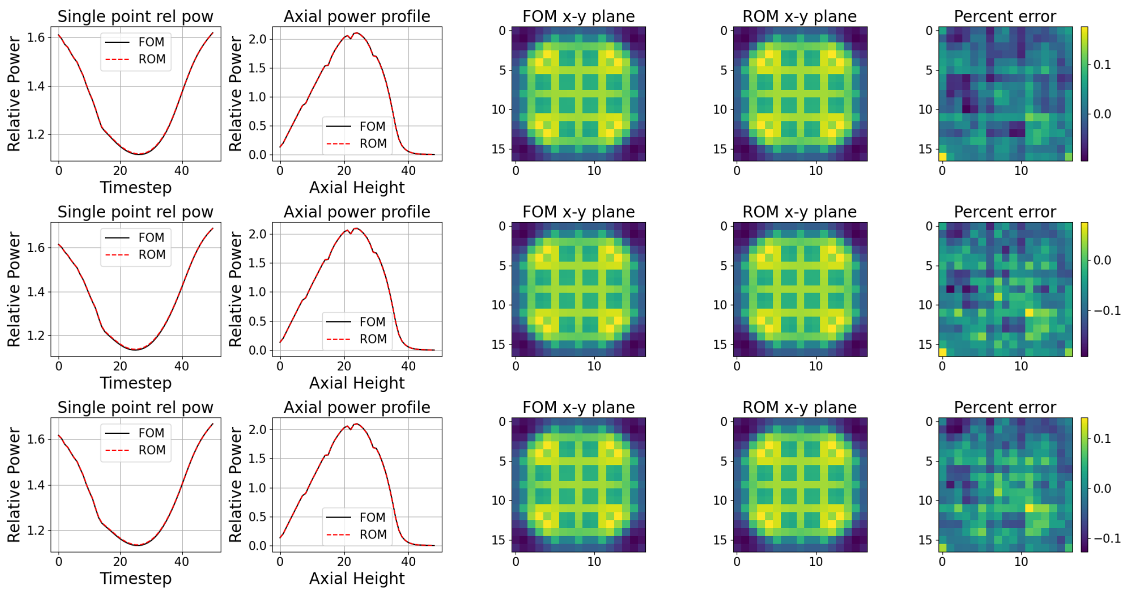


Figure 3.14: NIDS ROM and FOM realizations of the same inputs and associated error for Case 0.

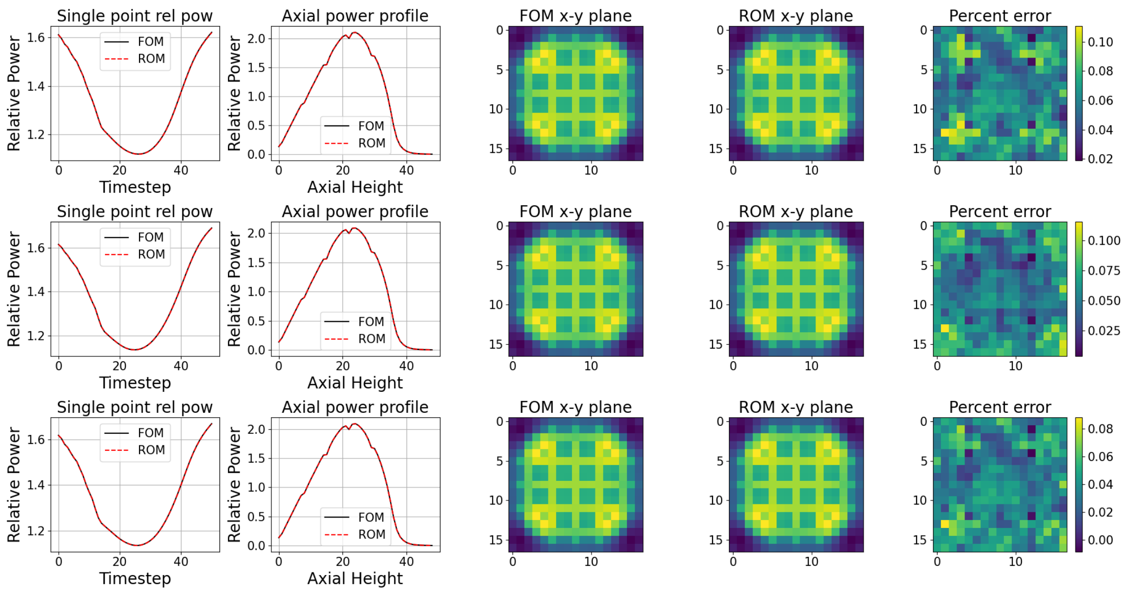


Figure 3.15: NIDS ROM and FOM realizations of the same inputs and associated error for Case 3.

at a single location in the core. Each shade of blue represents one additional standard deviation in relative power. The lower plots in each figure show a comparison between the predicted average model power and the standard deviation in the model power for 100 realizations of the FOM and ROM. Five FOM realizations are not enough to produce reasonable UQ results. However, depending on the application, 25 or 50 FOM realizations may be satisfactory for some nuclear engineering applications.

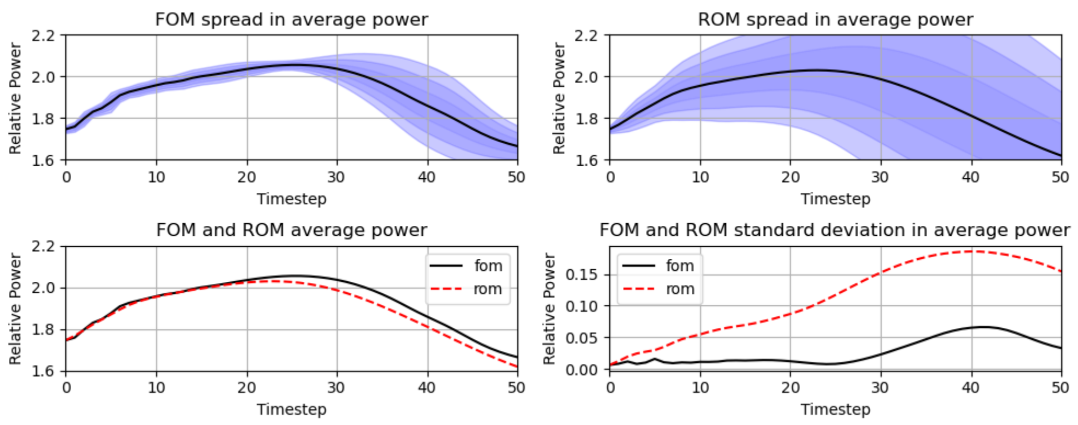


Figure 3.16: FOM and ROM forward uncertainty propagation for NIDS Case 0. Each shade of blue represents one additional standard deviation of separation from the mean.

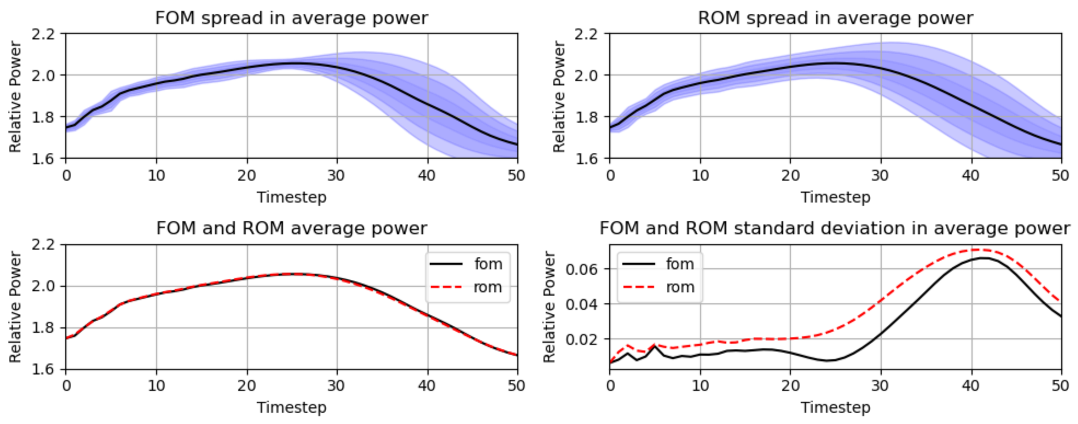


Figure 3.17: FOM and ROM forward uncertainty propagation for NIDS Case 2. Each shade of blue represents one additional standard deviation of separation from the mean.

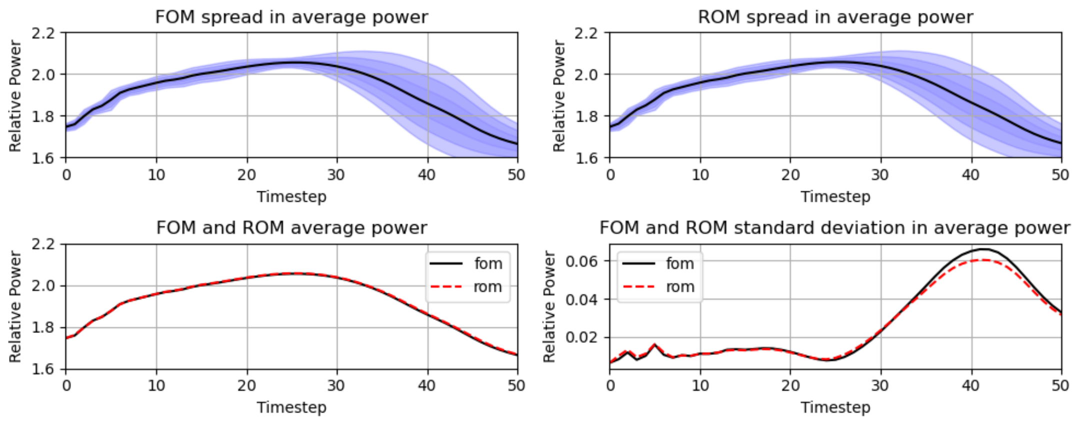


Figure 3.18: FOM and ROM forward uncertainty propagation for NIDS Case 3. Each shade of blue represents one additional standard deviation of separation from the mean.

For comparison with the CNN ROM method, the metrics shown in Figures 3.16, 3.17, and 3.18 are condensed into single figures for each training case. Table 3.11 shows the average agreement between model average power over the UQ analysis of 100 realizations. It represents the average error in an image such as the lower left plot of 3.18 for each location in the model for all time steps. Table 3.11 also shows this condensed result for the agreement in standard deviations (i.e., the lower right plot in Figure 3.18). Table 3.11 demonstrates the improvement in these metrics as the number of training samples increases, and shows plainly that the NIDS model is able to capture the individual contributions of the input parameters such that it more correctly captures the standard deviation in local power when compared against the CNN model by multiple orders of magnitude (see Table 3.8).

Case	MSE of average power	MSE of standard deviation
0	1.6×10^{-3}	1.1×10^{-2}
1	1.0×10^{-4}	3.0×10^{-3}
2	3.0×10^{-5}	3.3×10^{-4}
3	4.0×10^{-6}	8.9×10^{-6}

Table 3.11: NIDS UQ performance for increasing training dataset sizes.

Finally, we use the Wilks' formula value of 59 samples to obtain a 95/95 confidence interval on the worst performing sample (i.e., we are 95% confident that we have captured the top 95% worst performing case). Figure 3.19 shows the average relative power for the ROM and FOM for a random sampling of 59 cases for one location in the model as a function of time. The red shaded area in this image shows the maximum and minimum spread in value over all cases. The blue dotted line shows the maximum value as predicted by the. Here, it is clear that the NIDS ROM is able to predict any individual case to a high level of accuracy, including the cases which determine the most limiting out of a sample of 59. The MSE and Mean Absolute Error (MAE) for all 59 cases for the NIDS ROM are 0.01% and 0.38% respectively.

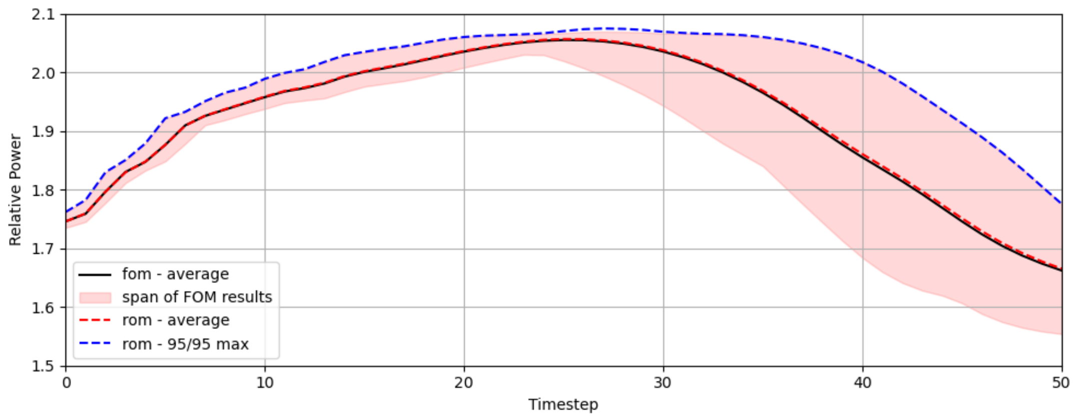


Figure 3.19: The spread in 59 cases as determined by Wilks' formula for a 95/95 confidence interval as predicted by the FOM and ROM. The red shaded region represents the full spread in FOM results.

3.5.3 NIDS Sensitivity Performance

The previous section showed an increase in performance compared to the CNN model for the UQ analysis. This section uses the sensitivity analysis to further illustrate the ability of NIDS to capture the dependence of input parameters on a local scale both in the average and standard deviation of local power. Figures 3.20, 3.21, and 3.22 show the sensitivity results for a representative location in the assembly when performing sensitivity analyses for the input rod speed, starting power, and inlet coolant temperature. All of these cases utilize the trained model of Case 3 (with 50 FOM realizations).

For a parameter such as the rod speed, which does not have a large impact on the overall power trace when varied over its range for this location in the assembly, the NIDS model is able to represent this lack of impact. The shape of the standard deviation of power is not precise, but it captures the order of magnitude of its impact.

On the contrary, starting power and input coolant temperature have a large impact on the spread of the power trace. In particular, the ends of the power spike seem to be heavily affected by these input parameters. Here, the NIDS model is able to capture this variability. It is encouraging and important that any ROM considered for deployment in a production environment can learn which input parameters have a large impact on the state variable of interest and which do not. These results suggest that NIDS is capable of capturing these dependencies.

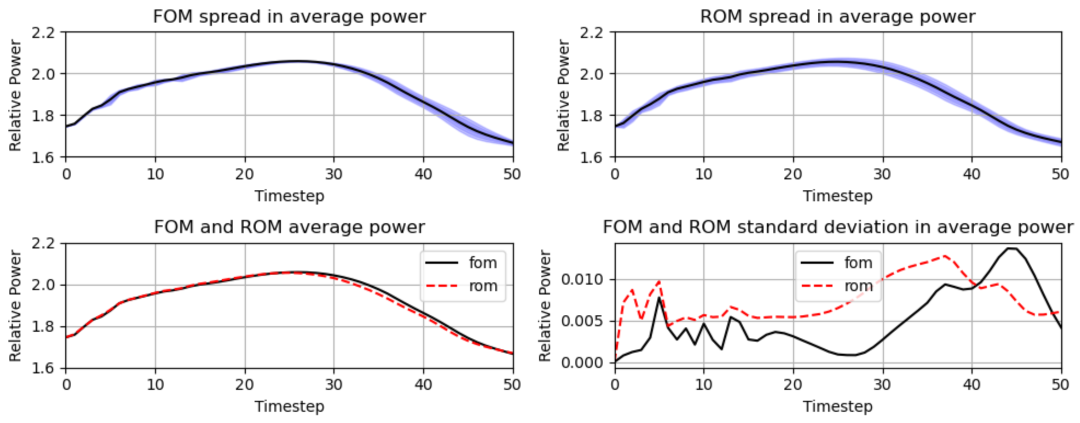


Figure 3.20: FOM and ROM rodspeed sensitivity for NIDS Case 3. Each shade of blue represents one additional standard deviation of separation from the mean.

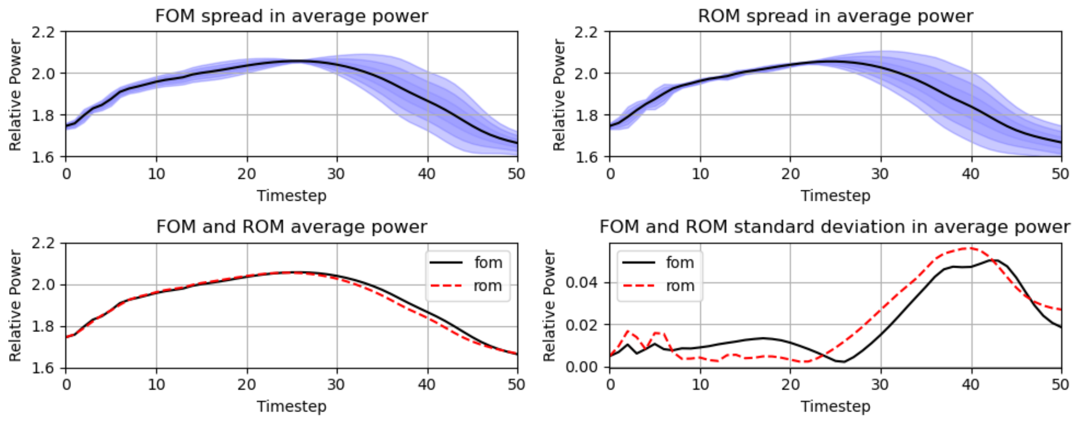


Figure 3.21: FOM and ROM initial power sensitivity for NIDS Case 3. Each shade of blue represents one additional standard deviation of separation from the mean.

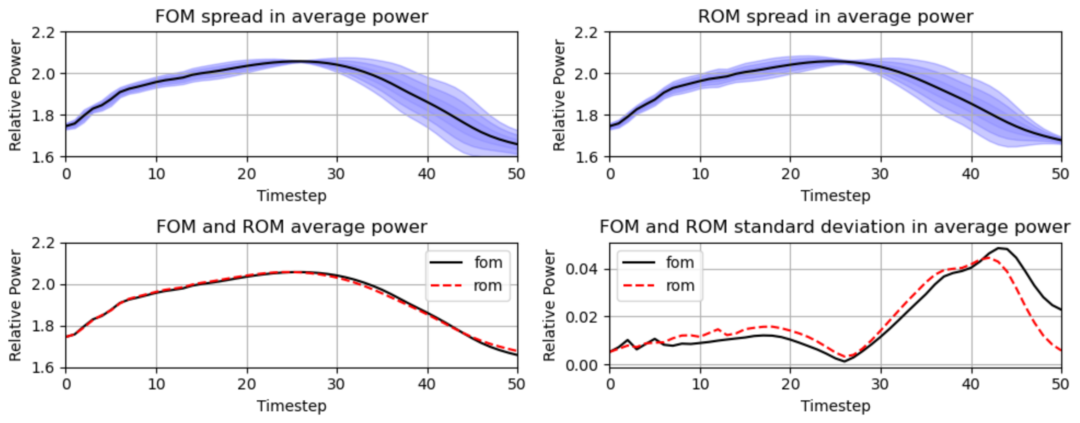


Figure 3.22: FOM and ROM coolant flow sensitivity for NIDS Case 3. Each shade of blue represents one additional standard deviation of separation from the mean.

When each of the four trained NIDS models was used to perform each sensitivity study, it is clear that the performance increased with larger training sets and that NIDS performed orders of magnitude better than the CNN ROM. These images are condensed into single metrics representing the performance of the sensitivity study by computing the MSE of the predicted average power and the standard deviation of the power. Tables 3.12 and 3.13 show the results for all input parameters and training sets. Here again we see the expected trend that as the number of training cases increases, so does agreement. However, unlike the CNN case, a plateau is not reached until much greater agreement is reached between FOM and FOM. Figure 3.23 visualizes these table values. Notice that case three only sees a marginal increase in average agreement for all but one input parameter (hgap - Figure 3.23 shows this result graphically), while the error in standard deviation shows a continuous improvement for all cases.

Case	dhfrac	flow	guide	hgap	power	rodspeed	tinlet
0	1.1×10^{-2}	9.6×10^{-3}	9.8×10^{-3}	5.7×10^{-3}	1.2×10^{-2}	1.3×10^{-2}	1.1×10^{-2}
1	4.9×10^{-3}	4.2×10^{-3}	2.9×10^{-3}	3.4×10^{-3}	4.0×10^{-3}	4.3×10^{-3}	3.5×10^{-3}
2	1.1×10^{-4}	1.5×10^{-4}	1.4×10^{-4}	4.2×10^{-4}	2.8×10^{-4}	1.5×10^{-4}	8.8×10^{-5}
3	1.2×10^{-4}	1.4×10^{-4}	6.5×10^{-5}	6.2×10^{-5}	2.4×10^{-4}	7.9×10^{-5}	7.2×10^{-5}

Table 3.12: NIDS sensitivity analysis FOM and ROM average predictions.

Case	dhfrac	flow	guide	hgap	power	rodspeed	tinlet
0	2.7×10^{-3}	4.2×10^{-4}	1.1×10^{-2}	5.7×10^{-3}	3.6×10^{-3}	2.8×10^{-3}	1.6×10^{-3}
1	4.1×10^{-4}	1.4×10^{-3}	3.1×10^{-3}	3.2×10^{-3}	1.4×10^{-4}	4.4×10^{-4}	2.1×10^{-3}
2	6.1×10^{-4}	4.1×10^{-4}	5.4×10^{-4}	1.5×10^{-4}	1.2×10^{-4}	1.6×10^{-4}	1.9×10^{-4}
3	2.7×10^{-5}	2.7×10^{-5}	1.2×10^{-4}	9.8×10^{-5}	5.7×10^{-5}	6.4×10^{-5}	1.7×10^{-5}

Table 3.13: NIDS sensitivity analysis FOM and ROM standard deviation error.

3.6 Summary

These studies demonstrate the relative effectiveness of the CNN and NIDS based ROMs for the single-assembly MPACT RIA analysis. The NIDS model performs much better

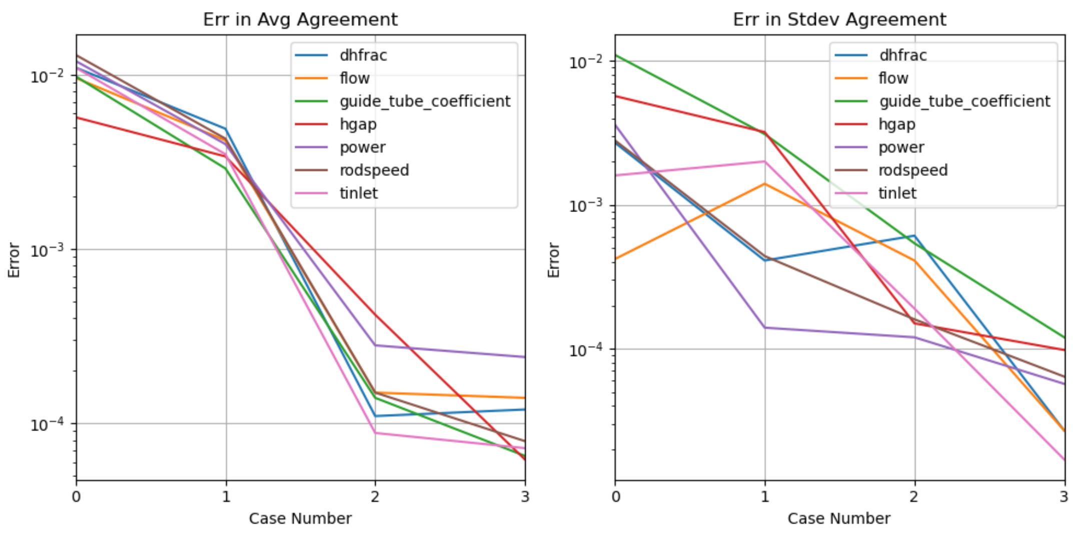


Figure 3.23: NIDS error in relative power prediction - average (left) standard deviation (right).

than the CNN based ROM. Improvements to the CNN model are available that could help it improve performance.

One way to improve performance would be to perform a more exhaustive hyperparameter search. There are a significant number of hyperparameters that describe the multistage CNN. Although a hyperparameter study was performed, a more exhaustive one could be warranted, which would include everything from latent space size to stride length to skip layer inclusion to batch normalization and kernel size/filter. These parameters were included during the hyperparameter search in this work; however, due to the large size of the search space, these parameters were searched piecemeal. In other words, not all parameters were allowed to change during each phase of tuning, only a subset. Then, those parameters were locked in place, and another set of hyperparameters were allowed to change. This was required due to the size of the search space, but it is possible that more compute time could find a more optimum combination of hyperparameters.

Another way to potentially find improvement would be to use a different time-stepping algorithm for the second stage of the architecture. In its current implementation, the entire code of the first stage is condensed into one temporal CNN. However, other work has shown to have good performance when using a time series prediction framework such as long short term memory (LSTM)s to progress the latent space of the CAE forward in time. With this improvement, the entire transient is not predicted in one shot; instead, the difference between the current and next timestep is predicted, much like how NIDS was adopted to produce transient results for this work.

Similarly, the NIDS algorithm could see an improvement by using the concepts of signed distance function (SDF)s for this application. Specifically, having each xy coordinate be accompanied by the distance to the closest control rod or poison pin in the model. This technique could help the model resolve more quickly and improve both training time and performance. This method was used in Chapter 4 and was able to improve performance.

A final idea to progress this research project involves a much more complex implementation but could be the subject of subsequent work in this field. The RIA analysis is itself a coupled simulation between a neutronic, heat conduction, and fluid solver. The results from this coupled model are the state variables of interest which are trained on and predicted in the work presented above. Instead of making the state variable of interest the power distribution of the converged solution, the ROM could be inserted into the convergence process itself. With an appropriate data set on which to train, one of or both intermediate (i.e., before convergence) neutronics and computational fluid dynamics (CFD) solutions could be the state variables of interest. During each multiphysics iteration, instead of converging one of the physics solutions from scratch (or from the previously converged), a ROM could be used to predict the next converged solution. The physics solver would then be used to converge the rest of the way for that iteration. This back and forth hand-off between the physics solvers and a collection of ROMs could significantly improve run time and negate the need for a fully non-intrusive ROM approach in the first place.

In its current form, the NIDS model could be considered for use to replace the back half of a UQ study. If an analysis requires 100 realizations of FOM to complete, the methods proposed here could be used to replace the last 50 executions with 50 realizations of the ROM. The results from Section 3.5.2 show this directly. Another context in which it could be used is design space exploration. Specifically, rather than exploring all combinations of input parameters in a search to find the most limiting combination for some metric (peak power, average power over some region, time to max power, etc.), a handful of FOM runs could be generated to train a ROM. Then, the ROM could feasibly be used to inform the designer of the anticipated top 10% limiting locations of the design space. At which point, FOM could be used to fully characterize those areas of the design space. This iterative process could significantly reduce the time it takes for these types of optimization studies.

The purpose of this study is to determine the viability of using ROM models in a design environment. The decision of whether either of these methods could be used in a true production level analysis is difficult to speculate on. This is due to the fact that each analysis, tool, regulatory requirement, analysis method, and application will have different constraints setting criteria for acceptability. Some application cases may require proof that the ROM agrees with the FOM for some benchmark cases representing the design space to within a certain degree of the relevant figure of merit for that analysis. Other types of analyses may not use results directly from the ROM and instead use the FOM to always run a final confirmatory case. In this situation, the ROM may just be used in a design space exploration context. However, these cases may require proof that the ROM is able to capture the contributions from each individual input of the FOM on the output of interest in a design space regime of interest. Still other applications may integrate the ROM with the FOM solver in such a way as to only improve convergence times. In these cases the users may desire the final results of the FOM to be within the FOM margin of error regardless if the ROM was used to speed up convergence or not.

At first glance, the CNN model appears to be out of contention as it does not capture the variability even when all input parameters are varied within their ranges, as shown in Section 3.4.2. And although the NIDS model captures this variability with orders of magnitude more agreement, each analysis would have its own criteria for how accurate a ROM tool needs to be in order to be used in a production environment. Putting these caveats aside, the NIDS model has at least shown itself as a viable candidate when evaluating ROM methods to be included in production-level design environments.

CHAPTER 4

Depletion Trajectory Sensitivities

A depletion trajectory application is used to assess the performance of a convolutional neural network (CNN) and Non-Linear Independent Dual System (NIDS) algorithm applied to a large-scale nuclear engineering application. The analysis code for this application is MC21 (see Section 2.1.4. This section extends the ideas of Chapter 3 in two ways. First, from a spatial size perspective (single assembly to multi-assembly, and two to three dimensions in space) and second from how many state variables of interest are present.

The first application problem in Chapter 3 predicts just one state variable - relative power peaking. This chapter demonstrates predictions of 190 state variables of interest. Specifically, this chapter demonstrates the prediction of isotopic number densities of all depletable isotopes in the model for two MC21 models. The first is a 2D truncated single-assembly application, and the second is a 3D quarter-core application. The role of this reduced-order model (ROM) would be to predict isotopic number densities for arbitrary power histories without the need of a full order model (FOM) for the depletion analysis. The number densities predicted by the ROM would then be loaded into a downstream nuclear solver in order to perform "branch calculations" at core configurations of interest.

Two neural network ROM approaches are explored in this application problem. The first is a CNN based approach (the latent space stepper architecture described in Section 2.2.3.5). The second is a NIDS model (Section 2.2.4). This section further demonstrates the primary shortcomings of a CNN based ROM for nuclear applications - its inability to scale to large problems and inaccuracies even with small-scale problems. Although it can be applied to the 2D truncated single assembly problem, its performance is shown to be much worse than that of the NIDS based ROM. However, although the CNN based ROM can be applied to the 2D model it is simply unable to scale to the 3D quarter-core application due to memory constraints whereas the NIDS based ROM performs almost as well on the 3D problem as it does for the 2D problem with no significant increase in memory footprint.

Section 4.1 describes the 2D and 3D nuclear models used for this analysis. Section 4.2 discusses the two types of ROM algorithms (CNN and NIDS) tested against application problems and basic network training details. Section 4.3 describes the analysis that is used to evaluate the performance of the ROM algorithms. Section 4.4 discusses the 2D planar results, while Section 4.5 discusses the 3D quarter-core result. Section 4.7 summarizes the key findings and results.

4.1 Nuclear Model Description

4.1.1 2D Plane

The first MC21 problem is a truncated two-dimensional assembly model that was used to test the mechanics and basic performance of the candidate ROM algorithms. The model consists of an 11 by 11 grid of fuel pins surrounded by water coolant. The fuel pins have a diameter of 3.575 cm. The inner and outer diameters of the cladding are 3.615 cm and 4.11 cm, respectively. In between the fuel and cladding is a ^{4}He filled gap. In one corner of the model there is no fuel pin, which was introduced to include some radial asymmetry in the neutron flux. Finally, all boundaries are reflective. None of these dimensions is reflective of a real reactor core design, and they were manually tuned to provide a model that was approximately critical and demonstrated variability in its depletion trajectories depending on the power history. However, the magnitudes are representative of values which might be found in a real core design (any matching dimensions are coincidental). Figure 4.1 shows an image of the 2D model, and Table 4.1 lists the definitions of the materials used in the model.

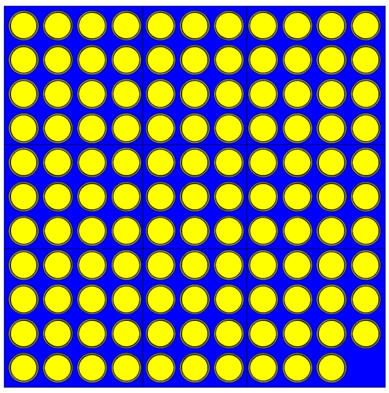


Figure 4.1: Truncated 2D single-assembly MC21 model

Isotope	Density
O16	4.58×10^{-2}
O17	1.741×10^{-5}
U234	1.3607×10^{-5}
U235	1.8161×10^{-4}
U238	2.5014×10^{-2}

(a) Fuel material

Isotope	Density
O	2.96×10^{-4}
Cr	7.6×10^{-5}
Fe	7.1×10^{-5}
Ni	3.4×10^{-5}
Zr	4.254×10^{-2}
Sn	4.65×10^{-4}
Hf	1.660×10^{-6}

(b) Clad material

Isotope	Density
O	2.790×10^{-2}
H	5.580×10^{-2}

(c) Coolant material

Table 4.1: Material definitions for 2D MC21 model. All units in (# / barn-cm). When the isotope is not specified, natural abundances are assumed.

In addition to the isotopes listed here, a selection of fission products are also modeled within the fuel. These isotopes are listed in Table 4.2 and constitute all isotopes modeled.

This model is quite small and is intended to be used to quickly test candidates for the ROM algorithm under various conditions, neural network architectures, and MC21 preprocessing scripts. All results presented using this 2D model took approximately 50-60 minutes per timestep to complete on 1 node (48 CPUs) of the Idaho National Laboratory (INL) Sawtooth1 supercomputer. Each timestep was executed using 400 batches, 10 discards, and 6E4 histories per batch. This run scheme achieved a 1-σ reactivity convergence of approximately 20 pcm. Analyses were performed using mostly

AG109	AG110M	AG111	AM241	AM242M
AM243	AS75	B10	BA134	BA135
BA136	BA137	BA138	BA140	BR81
CD110	CD111	CD112	CD113	CD114
CD115M	CD116	CE140	CE141	CE142
CE143	CE144	CM242	CM243	CM244
CS133	CS134	CS135	CS136	CS137
DY160	DY161	DY162	EU151	EU152
EU153	EU154	EU155	EU156	EU157
GA71	GD152	GD153	GD154	GD155
GD156	GD157	GD158	GD160	GE72
GE73	GE74	GE76	I127	I129
I130	I131	I135	IN115	KR82
KR83	KR84	KR85	KR86	LA139
LA140	MO100	MO95	MO96	MO97
MO98	MO99	NB95	ND142	ND143
ND144	ND145	ND146	ND147	ND148
ND150	NP237	O16	PD104	PD105
PD106	PD107	PD108	PD110	PM147
PM148	PM148M	PM149	PM151	PR141
PR142	PR143	PU238	PU239	PU240
PU241	PU242	RB85	RB87	RH103
RH105	RU100	RU101	RU102	RU103
RU104	RU105	RU106	RU99	SB121
SB123	SB124	SB125	SB126	SE76
SE77	SE78	SE79	SE80	SE82
SM147	SM148	SM149	SM150	SM151
SM152	SM153	SM154	SN115	SN116
SN117	SN118	SN119	SN120	SN122
SN123	SN124	SN125	SN126	SR88
SR89	SR90	TB159	TB160	TC99
TE122	TE123	TE124	TE125	TE126
TE127M	TE128	TE129M	TE130	TE132
U233	U234	U235	U236	U238
XE128	XE129	XE130	XE131	XE132
XE133	XE134	XE135	XE136	Y89
Y90	Y91	ZR90	ZR91	ZR92
ZR93	ZR94	ZR95	ZR96	-

Table 4.2: All isotopes within the 2D and 3D MC21 models.

30-timestep depletion runs, with one 60-timestep analysis included as a side study.

Consistent between the 2D and 3D models is an MC21 post-processing data pipeline, which was created to support this application problem. This pipeline reads the MC21 restart files and retrieves the model component names and isotopic number densities. As discussed previously, when constructing NIDS models one of the inputs is the spatial location of the variables of interest. Combining the information from the restart files with the MC21 mapping files, this spatial information was extracted by the post processing scripts.

At the end of the process, each MC21 run (consisting of 30 timesteps for the purpose of this discussion) produced a tuple of Python Numpy saved data arrays consistent with the expectations of the Parody tool (see Appendix B). This tuple had four Numpy array elements (see Appendix B for more information on what these terms refer to):

1. state_variable array of shape (1, 30, 11, 11, 190)
2. locs array of shape (1, 11, 11, 2)
3. params array of shape (1, 30, 3)
4. params_dist array of shape (1, 30, 11, 11, 1)

The state_variable array dimensions correspond to a single case with 30 timesteps on an 11x11 spatial grid for each of the 190 isotopes in the model. The locs array dimensions correspond to a single case on an 11x11 spatial grid, with the xy coordinate pair represented by the last dimension. The params array represents the three scalar input values for the model for each timestep - depletion power fraction, effective full power depletion timestep length, and cumulative depletion hours. Note that each depletion timestep is exactly 48 hours long. The effective full power depletion timestep length corresponds to the depletion power fraction multiplied by 48 hours.

Finally, an important step in feature engineering is normalizing the concentration of each isotope by the average concentration for just that isotope. This was required because of the varying magnitudes of all of the isotopes during depletion, which span many orders of magnitude (ranging from 10^{-2} to 10^{-15}). Stochastic gradient descent, the algorithm used to train neural networks, often benefits from normalized inputs to avoid having large variance in the scales of input dimensions.

Finally, the params_dist array corresponds to each spatial location's closest distance to a pin cell grid location without a fuel pin. Using the concept discussed in Section 2.2.4 on the signed distance function (SDF), additional geometric information is encoded as input into the ROM models. Figure 4.2 shows what the x-y distribution of the SDF looks like for the 2D model. In the color bar the whole numbers are used to represent the number of grid points away each location is from the bottom right corner where no fuel is present. These numbers are scaled within Parody during training.

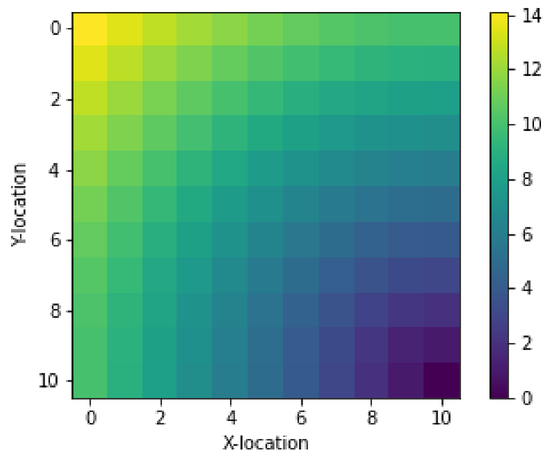


Figure 4.2: Signed distance function representing distance to closest non-fuel grid location for the 2D model.

This params_dist array construct allows for unique inputs to be applied to each location (and at each timestep—this functionality is not leveraged in this work but is supported with Parody). When discussing spatial scales on the order of a neutron's mean-free path in a lattice, neutron flux is heavily dependent on proximity to strong absorbers or reflectors. Rather than creating a larger neural network and having it learn the relationship between neutron flux/flux gradient and proximity to absorbers and reflectors, this feature engineering step allows us to encode this information as part of the input. With this additional input, the ROM converged faster and with greater precision. The benefits were minimal for the small-scale 2D case, but significant for the 3D case.

See Section 4.6 for an analysis on the impact of the SDF for the 2D and 3D case. Note that this SDF approach is important when predicting values that are directly related to the proximity of fuel or poison pins. In the current case, because this is a thermal reactor and it is dominated by thermal fission, the SDF offers significant benefits. However, for other systems such as a fast reactor (not explored in this work) where there is a higher spatial coupling the SDF may not be as impactful.

These four Python Numpy arrays (state variables, locations, scalar inputs, and distributed parameter inputs) were concatenated and entered into the Parody tool (see Appendix B). Specifically, the inputs to the resulting NIDS model are the scalar and distributed parameter values, and the outputs are the difference in isotopic concentrations for each isotope. Parody is designed to accept arrays of this shape and handle all of the scaling, data reshaping and mapping, data splitting, and GPU-CPU handoffs during training, as well as the scaling and inverse transformations required for inference.

4.1.2 3D Full Core

The second MC21 problem is a 3-dimensional model that is used to demonstrate the ability of NIDS to scale to a more realistic and representative real-world problem. The model is a quarter-core representation of a 37-assembly core. Each assembly consists of a grid of 17x17 pins. Each assembly also has a regular pattern of poison pins or coolant pins. There are reflective boundaries on the lines of symmetry and vacuum boundaries on the outside of the water reflectors and axially.

Figures 4.3 and 4.4 show a top-down and axial view of the geometry. Once again, this application problem does not represent a real reactor core design. Its dimensions and concentrations of beginning of life (BOL) isotopics were artificially tuned to provide an approximately critical configuration and demonstrate variability in its depletion trajectories with power history. A real reactor core design would likely include variability in its radial fuel and poison loading based on pin and assembly location. However, as will be demonstrated in subsequent sections, the geometry analyzed for this work still produces adequate neutron flux and isotopic concentration gradients in the radial dimension such that the NIDS ROM capabilities are stressed.

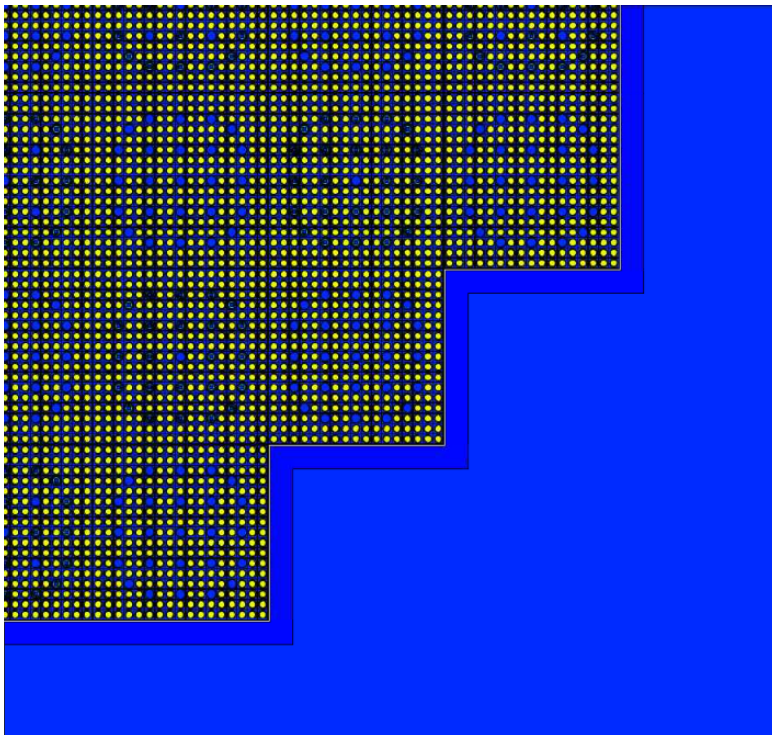


Figure 4.3: Top-down view of the 3D quarter-core geometry along the x-y plane

Although difficult to see in the quarter-core image in Figure 4.3, the geometry of this 3D model has more constituent elements than the 2D model. There are three types of combinations of pin geometry materials that are present in the model and two types of assemblies, each consisting of a 17x17 grid of fuel pins. Figure 4.5 shows the three types of pin cells in the model. There is a fuel pin, poison pin, and coolant pin. The fuel pin contains the fuel, the poison pin contains a boron-based poison to act as a reactivity control substance, and the coolant pins contain a zirconium pin filled with coolant water.

Table 4.3 describes the geometric and material definitions for each of the pin cell types in Figure 4.5. Table 4.4 describes the axial mesh used for depletion in the model. The material names referred to are defined in Table 4.5. The distance from one centroid of a pin to the next in the x- and y-directions is 1.26 cm. Note that a ^{4}He filler was used to fill the space for all locations of the pin geometry as is not specifically identified. The number density of ^{4}He is largely unimportant due to it being virtually invisible from a neutronics perspective (i.e., the cross section for ^{4}He is negligible).

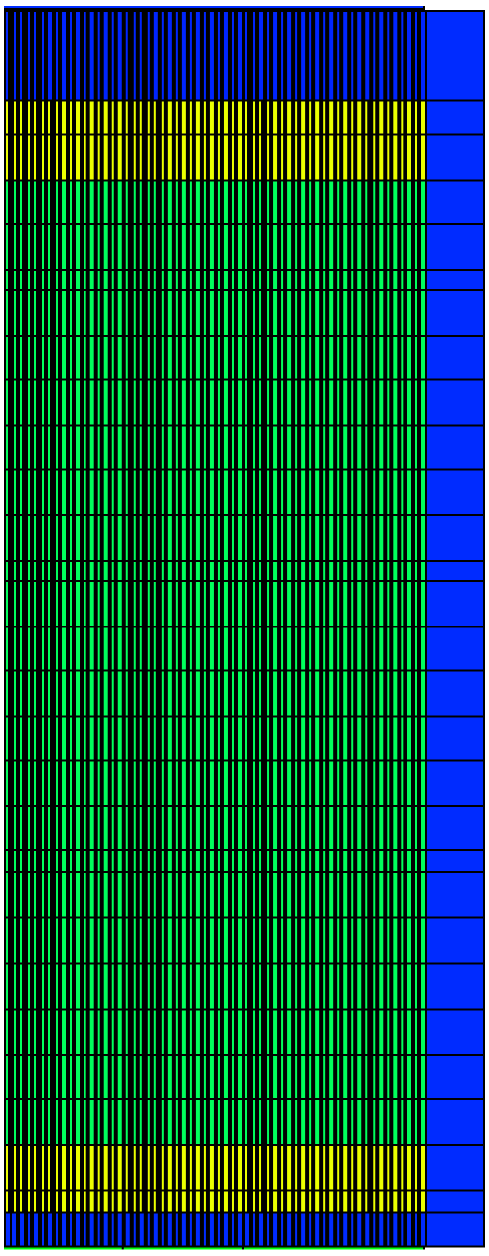


Figure 4.4: Axial view of the 3D quarter-core geometry along the x-z plane

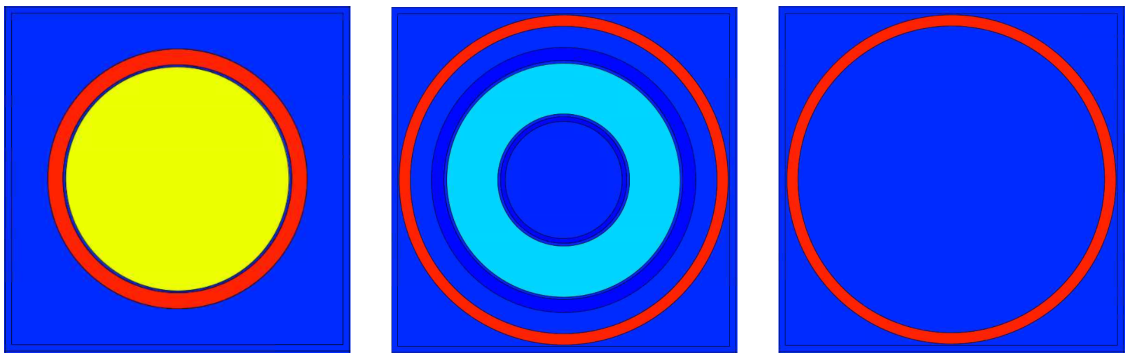

(a) (b) (c)

Figure 4.5: Fuel pins present in the 3D model - fuel (left), poison (middle), coolant (right).

Characteristic	Inner diameter	Outer diameter	Material
Fuel Pin			
Fuel	-	0.8192	U18
Clad	0.836	0.95	ZIRC4
Coolant	0.95	-	COOLANT
Poison Pin			
Inner pyrex container	0.428	0.462	SS
Pyrex	0.482	0.854	PYREX
Outer pyrex container	0.874	0.968	SS
Coolant	0.968	1.122	COOLANT
Clad	1.122	1.204	ZIRC4
Coolant	1.204	-	COOLANT
Coolant pin			
Coolant	-	1.122	COOLANT
Clad	1.122	1.204	ZIRC4
Coolant	1.204	-	COOLANT

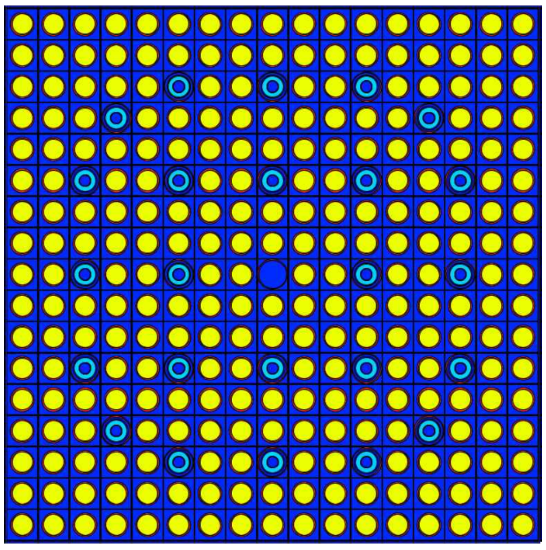
Table 4.3: Fuel pin characteristics in cm, with material definitions in Table 4.5.

Finally, the assemblies are placed in a checkerboard pattern for all 37 assemblies in the model. Note that this is a quarter-core symmetric model, so only one quarter of the core is represented in the model input files (as seen in Fig. 4.3), but it behaves as a full core model would due to the symmetry of the design. Figure 4.6 shows a closeup of the two types of assemblies. The distance from the center of any assembly to its neighbor in the x and y directions is 21.504 cm. There are 15 total non-fuel pin cells in each type of assembly in a regular pattern.

Table 4.5 defines the materials that are seen in all the geometric figures that represent the 3D model. These materials make up the constituents shown in Fig. 4.5, with one addition of the "LOWERNOZZLE" material, which is used for the structural material above and below the core. This structural material is seen in Fig. 4.4 as the yellow regions at the top and bottom of the core. There are several other structural materials not represented in Table 4.5; however, they are not included because of their negligible importance from a neutronic perspective and because they are not present in large quantities from a volumetric or mass perspective.

Node	Axial Height (cm)
1	11.951
2	15.817
3	24.028
4	32.239
5	40.450
6	48.662
7	56.873
8	65.084
9	73.295
10	77.105
11	85.170
12	93.235
13	101.300
14	109.365
15	117.430
16	125.495
17	129.305
18	137.370
19	145.435
20	153.500
21	161.565
22	169.630
23	177.695
24	181.505
25	189.570
26	197.635
27	205.700
28	211.951

Table 4.4: Axial mesh break points used for depletion.


(a)

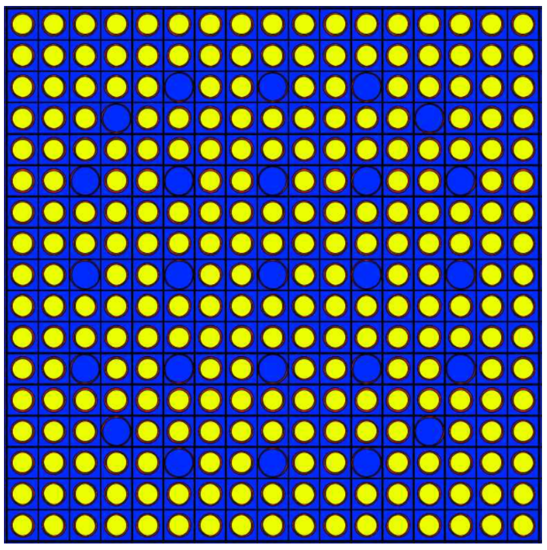

(b)

Figure 4.6: Assembly types present in the 3D MC21 model - the left assembly contains the poison pins, while the right assembly contains the coolant pins.

Isotope	Density
O16	0.0457591
U234	4.04814×10^{-6}
U235	4.9368901×10^{-4}
U236	2.23756×10^{-6}
U238	0.0223844

(a) U18

Isotope	Density
B10	9.61468×10^{-4}
B11	0.00389444
O16	0.0466888
SI28	0.0181641
SI29	9.22749×10^{-4}
SI30	6.08994×10^{-4}

(b) PYREX

Isotope	Density
B10	1.05835×10^{-5}
B11	4.25999×10^{-5}
H-H2O	0.0496231
O16	0.0248116

(c) COOLANT

Isotope	Density
CR50	3.30121×10^{-6}
CR52	6.36606×10^{-5}
CR53	7.2186×10^{-6}
CR54	1.79686×10^{-6}
FE54	8.68307×10^{-6}
FE56	1.36306×10^{-4}
FE57	3.14789×10^{-6}
FE58	4.18926×10^{-7}
HF174	3.54138E-9
HF176	1.16423×10^{-7}
HF177	4.11686×10^{-7}
HF178	6.03806×10^{-7}
HF179	3.0146×10^{-7}
HF180	7.76449×10^{-7}
SN112	4.68066×10^{-6}
SN114	3.18478×10^{-6}
SN115	1.64064×10^{-6}
SN116	7.01616×10^{-5}
SN117	3.70592×10^{-5}
SN118	1.16872×10^{-4}
SN119	4.14504×10^{-5}
SN120	1.57212×10^{-4}
SN122	2.23417×10^{-5}
SN124	2.79392×10^{-5}
ZR90	0.0218865
ZR91	0.00477292
ZR92	0.00729551
ZR94	0.00739335
ZR96	0.0011911

(d) ZIRC4

Isotope	Density
C	3.20895×10^{-4}
CR50	7.64915×10^{-4}
CR52	0.0147506
CR53	0.0016726
CR54	4.16346×10^{-4}
FE54	0.00344776
FE56	0.0541225
FE57	0.00124992
FE58	1.66342×10^{-4}
MN55	0.00175387
NI58	0.00530854
NI60	0.00204484
NI61	8.88879×10^{-5}
NI62	2.83413×10^{-4}
NI64	7.2177×10^{-5}
P31	6.99938×10^{-5}
SI28	0.00158197
SI29	8.03653×10^{-5}
SI30	5.30394×10^{-5}

(e) SS

Isotope	Density
B10	7.61305×10^{-6}
B11	3.06435×10^{-5}
C	8.96008×10^{-5}
CR50	2.13581×10^{-4}
CR52	0.00411869
CR53	4.67027×10^{-4}
CR54	1.16253×10^{-4}
FE54	9.6269×10^{-4}
FE56	0.0151122
FE57	3.49006×10^{-4}
FE58	4.64463×10^{-5}
H-H2O	0.0357666
MN55	4.89719×10^{-4}
NI58	0.00148226
NI60	5.70964×10^{-4}
NI61	2.48194×10^{-5}
NI62	7.91351×10^{-5}
NI64	2.01534×10^{-5}
O16	0.017883
P31	1.95438×10^{-5}
SI28	4.4172×10^{-4}
SI29	2.24397×10^{-5}
SI30	1.48097×10^{-5}

(f) LOWERNOZZLE

Table 4.5: Material isotopic definitions for zirconium, stainless steel, and lower-nozzle regions. All units in (# / barn-cm).

In addition to the isotopes listed here, fission products are modeled. These isotopes are the same as was present for the 2D model, and are listed in Table 4.2 and constitute all isotopes modeled.

This model is much larger and is intended to be used as a final test for the NIDS ROM algorithm in a real-world context. The scope of its analysis is much smaller than that of the 2D model due to its more narrow goals of stressing the size of the ROM, memory requirements, and to assess if NIDS can capture a more realistic fuel-poison loading pattern than an 11 by 11 truncated assembly. All results presented using this model took approximately 260-275 minutes per timestep to complete on 10 nodes (480 CPUs) of the INL Sawtooth1 supercomputer. Each timestep was executed using 1500 batches with 500 discards and $500,000$ neutron histories per batch. This run scheme achieved a one 1-σ reactivity convergence of approximately 7 pcm. It is more important for this application to have a more tightly converged solution than that of the 2D case because the more complicated flux shape resulting from a more converged solution will produce more complicated isotopic depletion distributions later in core life.

Like the 2D model, the same post-processing infrastructure was used for the 3D model results to produce the input required by Parody. For completeness, the dimensionality of the arrays associated with the 3D model are listed below, however, no additional steps or manipulations to the data were made to the 3D model compared to the 2D model other than the addition of a spatial dimension. At the end of the process, each MC21 run (consisting of 10 timesteps for the 3D models) produced a tuple of Python Numpy saved data arrays consistent with the expectations of the Parody ROM tool (see Appendix B). This tuple had four Numpy array elements:

1. state_variable array of shape (1, 10, 60, 60, 24, 189)
2. locs array of shape (1, 60, 60, 24, 3)
3. params array of shape (1, 10, 3)
4. params_dist array of shape (1, 10, 60, 60, 24, 1)

The dimensions of the state_variable array correspond to a single case with 10 timesteps on a 60x60x24 spatial grid for each of the 189 isotopes in the model. There was one less isotope present in this model compared to the 2D model due to the materials used in the 3D MC21 model. The locs array dimensions correspond to the single case on a 60x60x24 spatial grid, with the x-y-z coordinate pair represented by the last dimension. The params array represents the three scalar input values for the model for each timestep - depletion power fraction, effective full power depletion timestep length, and cumulative depletion hours. And, much like the 2D case, the params_dist array corresponds to each spatial location's closest distance to a pin cell grid location without a fuel pin. The distance for the 3D case corresponds to the distance in the x-y plane, as any pin cell without a fuel element will contain no fuel along the entire z-dimension.

This again uses the SDF concept detailed in Section 2.2.4. Specifically, additional geometric information is encoded as input into the ROM models. Figure 4.7 shows what the x-y distribution of the SDF looks like for the 3D model.

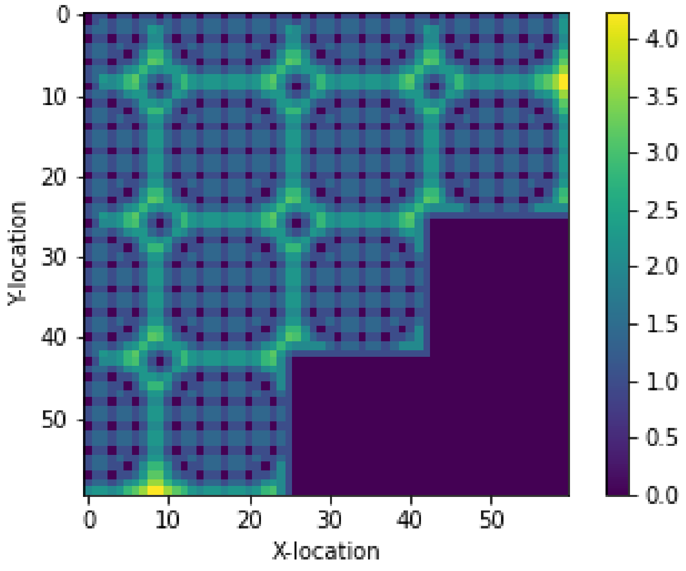


Figure 4.7: Signed distance function representing distance to closest non-fuel grid location for the 3D model.

This params_dist array construct allows for unique inputs to be applied to each location (and at each timestep - this functionality is not leveraged in this work but is supported with Parody). When discussing spatial scales on the order of a neutron's mean-free path in a lattice, neutron flux is heavily dependent on proximity to strong absorbers or reflectors. Rather than creating a larger neural network and having it learn the relationship between neutron flux/flux gradient and proximity to absorbers and reflectors, this feature engineering step allows us to encode this information as part of the input. With this additional input, the ROMs were found to converge faster and with greater accuracy. The benefits were minimal for the small-scale 2D case, but significant for the 3D case.

These four Python Numpy arrays (state variables, locations, scalar inputs, and distributed parameter inputs) were concatenated and entered into the Parody tool (see Appendix B). Parody is designed to accept arrays of this shape and handle all of the scaling, data splitting, and GPU-CPU handoffs during training, as well as the scaling and inverse transformations required for inference.

4.2 Surrogate Modeling Architecture

To assess the viability of ROM methodologies on this representative nuclear engineering application, two algorithms are explored. The first is the CNN latent space stepper algorithm are qualitatively and quantitatively evaluated to determine its applicability to the previously described nuclear engineering application. Section 4.2.1 and 4.2.2 will summarize each networks' key design features. The notation in each network's description section is assumed to be known throughout the discussion below. In general, the procedure for designing the neural networks and selecting their hyperparameters followed the procedure outlined in Appendix C.

4.2.1 CNN Latent Space Stepper Architecture

The first surrogate modeling method applied to this application problem is the CNN latent space stepper described in Section 2.2.3.5. The terminology in Section 2.2.3.5 is assumed to be known for this discussion. The general architecture was found by the steps described in Appendix C. All activation layers were ReLU except the final layer. For ease of reading, the image from 2.2.3.5 is reproduced in Fig. 4.8.

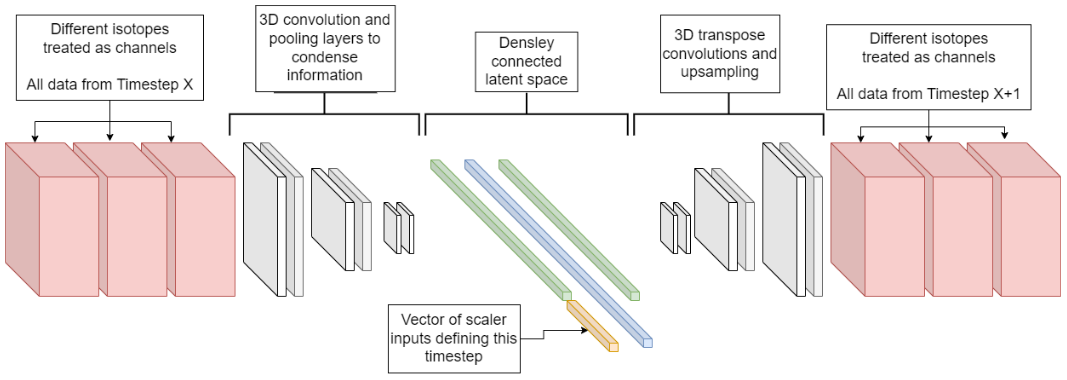

Different isotopes treated as channels
All data from Timestep X
3D convolution and pooling layers to condense information
Densley connected latent space
3D transpose convolutions and upsampling
Different isotopes treated as channels
All data from Timestep X+1
Vector of scaler inputs defining this timestep

Figure 4.8: Illustration of the convolutional autoencoder and latent space modifying neural network architecture.

Figure 4.9 shows a more detailed depiction of the model used. Note that skip connections are used in this model. These skip layers were found to improve performance and were inspired from a similar application to predict the CRUD deposit distributions with neural networks ([38]). Although the use of skip layers has been shown to help deep neural networks train much faster, common critiques of skip connections include the fact that the precise mechanism of their action is not well understood ([110]). Another critique noted in this work is that their architecture is harder to optimize as it is more difficult to design automatic architecture search heuristics that account for the added complexity introduced by skip connections. The reason for the added complexity is because any neural network layer receiving input from both a skip connection layer and its own parent layer would require a dedicated concatenation intermediate step to combine the inputs from both layers.

However, due to their success in many different applications (such as vision, natural language processing, reinforcement learning, and others - [111], [112], [113]), they were used here to improve performance. One potential reason for the improvement is that skip connections make neural networks behave like an ensemble of networks, which are often found to perform better than their independent constituents. Figure 4.9 provides the basic architecture of this ROM.

In Fig. 4.9, each node in the graph represents one layer of the neural network. The inputs and outputs correspond to the inputs and outputs of that layer, with the dimensions noted in parentheses. The names of the layers during creation are also noted on the left-hand side of each of the nodes of the graph, along with the type of Tensorflow layer being used. The edges of the graph correspond to where the output of some node is input. For example, the layer "conv2d_21: Conv2D" tells us that the layer is a 2D convolutional layer with inputs of dimension $(11 \times 11 \times 3)$, and outputs of $(11 \times 11 \times 200)$. From this information, we can surmise that the number of filters in this layer is 200, as that is how large the last dimension is. Reading Fig. 4.9 in this way allows for this network to be reproduced by any program capable of making neural networks (PyTorch, TensorFlow, Keras, PyTorch Lightning, etc.).

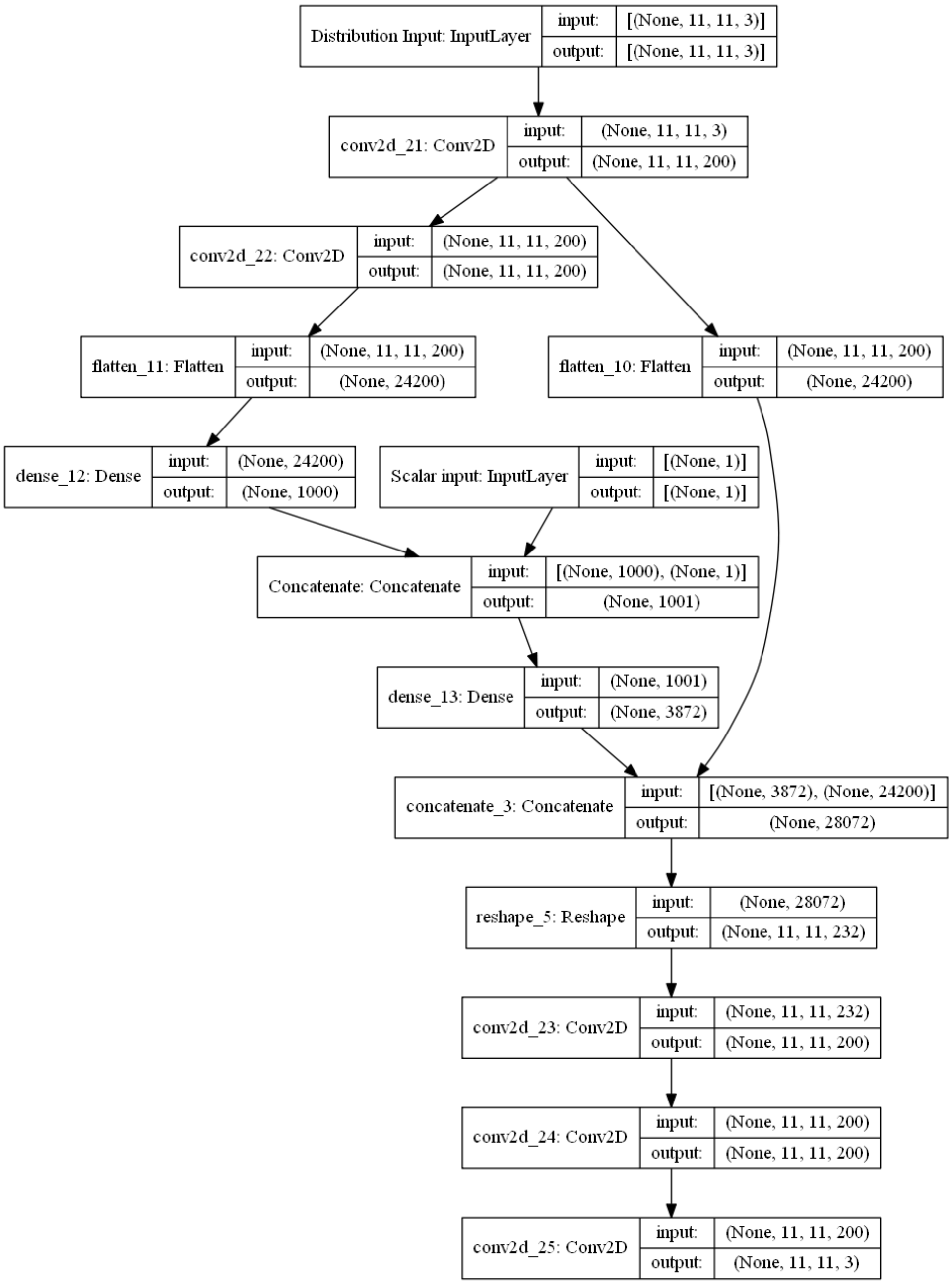

Distribution Input: InputLayer | input: [(None, 11, 11, 3)] | output: [(None, 11, 11, 3)]
conv2d_21: Conv2D | input: (None, 11, 11, 3) | output: (None, 11, 11, 200)
conv2d_22: Conv2D | input: (None, 11, 11, 200) | output: (None, 11, 11, 200)
flatten_11: Flatten | input: (None, 11, 11, 200) | output: (None, 24200)
flatten_10: Flatten | input: (None, 11, 11, 200) | output: (None, 24200)
dense_12: Dense | input: (None, 24200) | output: (None, 1000)
Scalar input: InputLayer | input: [(None, 1)] | output: [(None, 1)]
Concatenate: Concatenate | input: [(None, 1000), (None, 1)] | output: (None, 1001)
dense_13: Dense | input: (None, 1001) | output: (None, 3872)
concatenate_3: Concatenate | input: [(None, 3872), (None, 24200)] | output: (None, 28072)
reshape_5: Reshape | input: (None, 28072) | output: (None, 11, 11, 232)
conv2d_23: Conv2D | input: (None, 11, 11, 232) | output: (None, 11, 11, 200)
conv2d_24: Conv2D | input: (None, 11, 11, 200) | output: (None, 11, 11, 200)
conv2d_25: Conv2D | input: (None, 11, 11, 200) | output: (None, 11, 11, 3)

Figure 4.9: Visualization of the CNN latent space stepper model architecture.

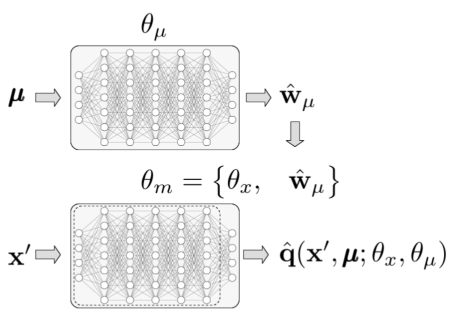
4.2.2 NIDS Neural Network Architecture

The second surrogate modeling method applied to this application problem is the NIDS method described in Section 2.2.4. The general architecture for each of the spatial and parameter networks was found through the steps described in Appendix C.

The spatial network and the parameter network were both dense neural networks with ReLU activation. After combining the output of the two networks, the final layer of the NIDS architecture used a linear activation. The hyperparameters for the NIDS algorithm are much less complex than those for the CNN based ROM described in Section 4.2.1. Table 4.6 lists the main parameters defining the architecture of neural networks. For ease of reading, the image from [25] is reproduced in Figure 4.10

Parameter	Value
Param input dimensions	4
Param dense layers	4
Param units per layer	500
Latent space dimensions	400
Spatial input dimensions	3
Spatial dense layers	4
Spatial units per layer	500
Output dimensions	189

Table 4.6: NIDS hyperparameters for the MC21 application problem.



θ_μ
$\boldsymbol{\mu}$
$\hat{\mathbf{w}}_\mu$
$\theta_m = \{\theta_x, \quad \hat{\mathbf{w}}_\mu\}$
$\mathbf{x}'$
$\hat{\mathbf{q}}(\mathbf{x}', \boldsymbol{\mu}; \theta_x, \theta_\mu)$

Figure 4.10: Schematic diagram of a NIDS network ([25]).

4.3 Description of Modeling Scenarios and Figures of Merits

This section describes a representative depletion analysis that has value in the nuclear reactor design community to be able to do quickly and is hindered by the current high cost of FOMs. Subsequent studies based on this analysis present an assessment of the viability of deploying the previously described ROMs in a realistic nuclear engineering context. Specifically, the application problems for this full-scale analysis produce a ROM that can capture the time-dependent trajectory of isotopic concentrations both in number of isotopes at each timestep and in the 3D distribution of isotopic concentration for each isotope modeled.

An MC21 model which is used for fission depletion analyses typically has on the order of 200 isotopes. Although the constituent elements that define a nuclear reactor's fuel, coolant, and structural materials do not require this many isotopes, the process of nuclear fission creates many other isotopes. However, only a portion of all possible isotopes are included. The decision to include isotopes or not is usually dependent on how much they impact the neutron flux distribution, with isotopes having larger cross sections typically being included. This decision also has to balance memory limitations on modern high-performance computing (HPC)s, as including all possible isotopes would create too large a memory footprint for the model, while having too few isotopes results in inaccurate depletion behavior. Getting this balance right is important because isotopes with non-negligible nuclear cross sections will impact the steady-state neutron population distribution (i.e., the neutron flux distribution), and thus the power distribution, which will in turn affect how many fission events will occur and therefore impact how much the isotopic concentrations will change for the next timestep. See Section 2.1.5 for a discussion on the governing equations describing isotopic depletion.

The following sections will attempt to produce an accurate ROM that can be used to deplete a nuclear model experiencing an arbitrary power history. A power history is defined by two quantities, the timestep length and the power level at which the timestep is depleted. This analysis will focus only on varying the power level and withholding the variation in timestep length for a future study. These analyses will explore what type of training data set is required to encode the proper information within the ROM to be able to execute arbitrary power history depletion trajectories. Specifically, the application in mind should be an expensive multiphysics calculation where it is too expensive to deplete the core fully. In this scenario, we are interested in how many FOM timesteps are required in the training dataset before a ROM can adequately replace the functionality of the FOM and perform depletion steps into the future using only the ROM.

The following metrics will be used to quantify the effectiveness of these studies. These are the Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Mean Squared Error (MSE). Eq. (2.25) shows the MAE, which represents a metric corresponding to the expected value of the L1-norm loss. Eq. (2.24) shows the MSE, which is the expected value of the squared error between prediction and truth. Finally, Eq. (2.27) shows the MAPE, which normalizes MAE by the maximum value in the collection. This last metric highlights relative error performance, which will be useful in subsequent sections because average isotopic concentrations can range from 10^{-2} to 10^{-15} $\frac{\#}{\text{barn-cm}}$. The MAPE provides a way to capture how well individual isotopes are predicted compared to each other.

Because there are 190 isotopes of varying importance estimated by the ROM, only the relevant ones are displayed in the following plots. Some isotopes have small half-lives and/or small number densities, with small cross sections. In other words, they do not impact the neutronics of the model to any appreciable degree, while other isotopes can be very important to the depletion trajectory, such as isotopes defining the fuel or poison pin cells. Although all isotopes are modeled and predicted by the ROM and FOM, only a subset are presented in the results.

^{234}U, ^{235}U, ^{238}U, ^{238}Pu, and ^{239}Pu are included. These isotopes represent isotopes that are important in nuclear engineering because they are fissile material or isotopes that can be transmuted into fissile material by neutron capture. SR90 and CS137 are included, as they are commonly created after nuclear fission. CS137 and SR90 both have a relatively long half-life of about 30 years and produce relatively high energy radiation. Because of these features, they are often the focus trace nuclides that are used to help benchmark spent nuclear fuel isotopic quantities or are tracked due to their potentially negative health consequences. ^{10}B is also included due to its common use as a poison in reactor design. ^{10}B has a large thermal neutron cross section and tends to be depleted substantially throughout core life. Because of this its impact on the neutron trajectory changes with life and it represents both an important and potentially challenging isotope to include.

Finally, ^{135}Xe and ^{135}I are included due to their often significant importance in nuclear transients on time scales of interest to thermal reactor operators. ^{135}Xe has such a high neutron cross section that it is directly accounted for as a dimension of analysis for reactor analysis. In other words, when converging on neutron flux distribution, it can be included in the iterative convergence process alongside the power and thermal distributions. ^{135}I decays into ^{135}Xe with a slightly shorter half-life, giving ^{135}Xe an overall difficult quality that its concentration will peak hours after shutdown and can exert influence on the power distribution in ways other isotopes cannot.

The ROMs are trained on all 190 isotopes, and in general the performance of these nine focal isotopes are representative of all isotopes in the model. For the reasons above, these 9 focal isotopes will be shown in plots and listed in metrics as the focal point of the analyses below, while other isotopes, although still predicted by the ROM, will be neglected in the plotted results. However, the performance of all isotopes are included in any global metrics of performance, such as MSE for a particular case.

4.4 2D Planar Results

In the subsequent plots, there are a few different ways that time could be represented. However, a simple "timestep" metric is used in all x-axes. It would also be appropriate to illustrate this information in the form of "effective full power days", or "calendar time". Timestep was ultimately chosen so that comparisons between each cases' power history could be made on a consistent basis. To choose to use "effective full power days" would result in plots having different scales for the different power history cases. Likewise, calendar time is directly related to timestep. However, its use may cause confusion as the models shown here are not intended to represent a real core design and are not reflective of what may be found when applying these methods to other designs. So, to avoid confusion from these perspectives, a simple “timestep” metric was chosen to represent time in all x-axes for this study.

4.4.1 Training and Qualitative Results

With these metrics and scenarios in mind, the MC21 models were created and executed. For the 2D planar model described in Section 4.1.1, a series of 60 FOM runs was executed. Consistent with common machine learning best practices, the datasets are always divided into train, validation, and test datasets. For the 2D case, the last 10 FOM results are withheld as the test dataset while 15% of the remaining dataset was reserved as a validation dataset. Each FOM solution consisted of 30 time steps of randomly sampled power levels for each case and timestep ranging from 0% power to 100% power.

As previously discussed, these data are pre-processed in the data pipeline, helping convert the MC21 output restart, mapping, and MC21 input files into the format required by Parody to create the ROMs. Once the FOM dataset was created, the set of hyperparameters was chosen for the neural networks, and the suite of 60 MC21 runs was executed, NIDS and CNN models were trained.

Training occurred until the neural network experienced a training plateau in its loss function. The CNN network and the NIDS networks used Tensorflow and PyTorch Lightning (with Parody as executor), respectively, to handle training. Both libraries offer standard callbacks to reduce the learning rate on a training plateau (patience of 10, factor of 0.2) and early stopping criteria (patience of 20) to ensure efficient use of compute resources and adequate training convergence.

First, we review the qualitative performance of the CNN and NIDS ROMs before citing their performance via the metrics discussed above for our isotopes of interest (^{234}U, ^{235}U, ^{238}U, ^{238}Pu, ^{239}Pu, SR90, CS137, ^{10}B, ^{135}Xe and ^{135}I). For the first two models shown (the

CNN and the first NIDS model results) the models are trained using all 30 timesteps and the data associated with 50 FOM realizations. For predictions and comparisons, the test input power history represents a power history that is previously unseen by the model. And the only input the ROM has is the isotopic depletions in the first timestep. In other words, these comparisons will show how well the models can do when the isotopes are depleted from a "fresh core" distribution through 30 new timesteps.

Figure 4.11 shows comparisons of isotopic depletion for the nine focal isotopes for the CNN model. Each row is a separate isotope, and each column shows a different perspective on performance. The first column shows traces with time of the isotopic concentrations for the FOM and ROM at 3 locations in the model. These locations show two corners and a centrally located region. The second column shows the planar average MAPE for each timestep for that isotope as a function of time to illustrate how the error changes with depletion. This quantity is averaged over all locations and therefore looks much tighter than some of the plots in the first column. Finally, the third column shows a distribution of MAE for the last timestep in the trajectory.

Figure 4.12 shows similar qualitative performance. Each row in Fig. 4.12 is a different isotope, and each column is a different arbitrary power history. These are power histories that are not present in the training or validation training sets and represent performance on unseen data from the perspective of the model/hyperparameter tuning processes. Here, we see more of the same patterns that Figure 4.11 shows. Specifically, the order of magnitude and trends are correct, but overall agreement is poor.

In general, Figures 4.11 and 4.12 show promising performance in that the order of magnitudes of the isotopes (which have a wide range through the 190 isotopes and within the nine focal isotopes) are captured. In addition, their general trajectories are correct (isotopes that accumulate through life and isotopes that diminish through life do so in the ROM reconstructions). This is true for all focal isotopes except ^{135}Xe and ^{135}I. As will be discussed later, this is practically of no consequence for the way in which a ROM would be deployed for this type of analysis. However, much like in the analysis in Chapter

3, the CNN models do not perform as well as the NIDS models.

Better performance could be found with more appropriate architectures, combinations of skip layers, and hyperparameters. However, in light of how well the NIDS models performed for this model and due to the fact that the CNN models have much higher memory requirements which would prevent them from being effective for a full-scale depletion analysis, these paths were not pursued.

One pattern to note, and one that will not be improved even by the NIDS architecture, is that the short-lived isotopes (such as ^{135}Xe and ^{135}I) vary significantly in their concentrations in the FOM. This trend is consistent with expectations due to their shorter half-life. The agreement for these isotopes is typically lower for the ROMs explored here. Although these two isotopes are important to be able to describe in a FOM to obtain the correct power distributions, the ability of a ROM to capture their dynamics for a depletion study is of little actual practical consequence for a nuclear designer. The reason is because typically, in a design context, a depletion is first performed to capture the isotopic depletion trajectories. Then, for conditions of interest (i.e., for some temperature and pressure combination) those isotopic number densities will be reloaded alongside the new core condition, and a "branch calculation" is performed, which finds the proper equilibrium concentration distributions for these short-lived isotopes under different operating conditions.

Fig. 4.13 shows comparisons of isotopic depletion for all focal isotopes. Again, each row is a separate isotope, and each column shows a different perspective on performance. Notice the much better performance for all isotopes except ^{135}Xe and ^{135}I, which show essentially indistinguishable results between the FOM and ROM. The errors are relatively constant with timestep, with the exception of SR90 and CS137, which see a slight increase in error for the second half of the depletion.

Figure 4.14 shows similar qualitative performance. Each row in Fig. 4.14 is a different isotope, and each column is a different arbitrary power history. Here again we see markedly improved performance over the CNN model. The ability to capture ^{135}Xe and

^{135}I is still lacking; however, as previously discussed, these isotopes have little consequence for a nuclear designer if the ROM would be used in this manner.

Of the isotopes that perform poorly, there are several contributing factors that may explain why they are not well captured by the ROM.

1. **Isotopes that are not depleted by the model (i.e., O17)**. The NIDS model is unable to learn that some isotopes are not changing and ends up predicting an arbitrary trajectory that causes a large error. This is easily fixed by neglecting these isotopes in the training and assuming they remain constant during the predictions. Figure 4.15 shows what the NIDS model from above predicts for a constant isotope before this processing step is performed.

2. **Isotopes with short half-lives.** Short half-lives will make their depletion trajectories more erratic and often cause concentrations to be quite small during low-power timesteps. This may make them span a large range of orders of magnitude in concentration and cause difficulty in the training process (despite all isotopes being normalized to their own average concentrations). An example of this type of isotope is ^{135}I. Because this is one of the highlighted isotopes, another isotope of this nature is shown in Fig. 4.16 for RU105, which has a half-life of 4.4 hours. As you can see, this type of isotope looks just like ^{135}I in its agreement between ROM and FOM.

3. **Isotopes with large neutron absorption cross sections** which do not contribute to the fission process, even if they have a large half life. A large neutron absorption cross section will cause an isotope to transmute quickly into other isotopes, and it has a similar effect as that of a short half-life. However, the model seems to be able to recognize isotopes which are important to the fission process, such as fissile material. This is because they are of the greatest importance in producing all other isotope concentration growth and are thus recognized during the stochastic gradient descent process as large contributors to the training error.

This effect is harder to illustrate because neutron cross sections are energy dependent, and correlating some collapsed cross section value to half-life and error is not straightforward.

4. Isotopes that are part of active and complicated decay chains, in that their dependence is reliant on difficult to predict isotopic concentrations which make their own concentrations more difficult to predict.

4.4.2 Quantitative Results

This section provides quantitative results to compare the types of models and training methods and perform the study described in Section 4.3. First, we compare the CNN and NIDS ROMs. Table 4.7 shows the average error metrics for the CNN model when predicting all 30 timesteps and all 10 test cases. One additional metric is shown below (and will be in all error tables) - the "MAE-norm". This metric is simply the MAE divided by the average mass of the isotope for the depletion trajectory. MSE is given for completeness and because it is a common regression metric. However, for isotopes with different magnitudes of mass, this metric is misleading. The MAE is useful because it gives an absolute difference, but it also suffers from the same issue in that comparing the MAE between isotopes is not a fair comparison. The MAPE and MAE-norm offer normalized results. The former normalizes to the maximum value in the dataset and the latter normalizes to the average value in the dataset.

Table 4.8 shows the same data for the NIDS model. Notice how the CNN model actually performs better than the NIDS model for the two more erratically behaved isotopes, ^{135}Xe and ^{135}I. However, this is misleading and mirrors what was seen in Chapter 3 in that the CNN model appears to simply average the values over all timesteps and does not capture the step-by-step variation in concentration. Figures 4.14 and 4.12 show this behavior. Other than that, the CNN model performs orders of magnitude worse in all other isotopes and is not considered going forward in the representative nuclear

Isotope	MAPE	MAE	MAE-norm	MSE
U234	0.0401	4.65e-08	0.0398	5.32e-15
U235	0.0845	9.71e-06	0.0804	1.79e-10
U238	0.0398	9.89e-04	0.0398	2.59e-06
PU238	0.4365	1.08e-08	0.2304	4.00e-16
PU239	0.1929	8.58e-06	0.1118	1.22e-10
SR90	0.1888	4.99e-07	0.1163	5.12e-13
CS137	0.1905	8.73e-07	0.1262	1.71e-12
XE135	0.0952	4.64e-10	0.0910	3.87e-19
I135	1.9080	4.38e-08	0.4744	2.60e-15

Table 4.7: Summary of CNN error metrics.

engineering application as we evaluate the viability of a NIDS model being used in this application.

Isotope	MAPE	MAE	MAE-norm	MSE
U234	0.0003	3.86e-10	0.0003	2.75e-19
U235	0.0013	1.42e-07	0.0012	3.67e-14
U238	0.0000	2.29e-07	0.0000	1.10e-13
PU238	0.0971	3.52e-10	0.0075	2.53e-19
PU239	0.0026	1.89e-07	0.0025	6.51e-14
SR90	0.0014	4.70e-09	0.0011	3.75e-17
CS137	0.0011	6.19e-09	0.0009	8.16e-17
XE135	0.1705	9.35e-10	0.1834	1.61e-18
I135	5.1459	1.85e-07	2.0059	6.42e-14

Table 4.8: Summary of NIDS error metrics.

Continuing to the analysis described in Section 4.3, the NIDS model is evaluated to see how many timesteps are needed within the training dataset to complete a full 30-timestep depletion analysis. The models are trained using the first 5, 15 and 20 time steps in the 30-timestep depletion training case. Figures 4.17 through 4.19 show the ROM and FOM results for just the ^{235}U and ^{238}Pu isotopes. All the isotopes are not shown for readability, but the agreement for these two isotopes is representative of the behavior for most isotopes. Notice how the agreement slowly diverges shortly after the 10th timestep for the 5 timestep case, and shortly after the 20th timestep for the 15 timestep case. The 20-timestep case appears to have enough data so that the neural network is able to learn the trajectory up to 30 timesteps.

This procedure is repeated so that the models were trained with 5, 10, 15, 20 and 30 time steps. The MAPE error averaged throughout the 2D plane for each time step for the 10 test cases was calculated for each training set. Figure 4.20 shows how the errors for each case evolve over time. The patterns are not completely reliable (i.e., ^{135}Xe shows similar behavior for all training datasets, ^{239}Pu shows the 5 case dataset performing better than the 10 case dataset late in life); however, in general performance for the isotopes degrades more quickly for the training datasets which have less information late in life. This aligns with expectations - the more that the neural network has to extrapolate, the less high quality the predictions will be.

These results raise two natural questions to help assess the viability of this class of ROM algorithm for this class of nuclear engineering problem.

1. Does the 30-timestep case contain enough information to encode how depletions will progress indefinitely? Or will it also deteriorate shortly after it begins to extrapolate, as the other cases demonstrated?

2. How well would a mixed dataset perform? In other words, a training dataset with some 5-timestep and some 30-timestep data.

For item 1, Fig. 4.21 shows clearly that the limitation in the models ability to extrapolate still exists even for the 30 timestep case. This study used a unique 60 timestep FOM result to compare against, and extrapolated the previously trained neural networks (based on 30 timestep training datasets) well past their 30 timestep dataset. This result suggests that all models seem to begin to deteriorate after around 5-10 timesteps outside of its training dataset range (this varies from isotope to isotope). This is a common finding in neural networks in general. Their ability to extrapolate outside of their training dataset is often limiting. Because the Bateman equations defining isotopic concentrations are linear one might expect neural networks to be able to extrapolate more successfully.

However, as previously discussed the evolution of the flux determining the spatial distributions of isotopic concentrations is based upon non-linear phenomena, making this a difficult problem to extrapolate.

Item 2 is more complicated in that there are a large number of ways this strategy may be explored. For the purpose of this work, an approach was used which held the total amount of MC21 run time constant. Put another way, it kept the total amount of training data constant. This assesses the situation where there is a fixed amount of computing resource and the practitioner needs to decide how best to spend it to create the best training dataset.

To explore this concept, the 5-timestep dataset is adjusted. For the studies mentioned previously, 50 FOM realizations were included in the training dataset, each having 5 timesteps. This constitutes running ($50 \times 5 = 250$) MC21 timesteps. Instead of only depleting 5 timesteps for each realization, a designer could deplete a mix of 5 and 30 timestep datasets while still remaining at the 250 timestep budget. Table 4.9 shows 9 dataset combinations which include increasing amounts of 30-timestep training data. Each row in Table 4.9 contains exactly 250 MC21 timesteps.

dataset number	5-timestep realiza-tions	30-timestep realiza-tions
0	50	0
1	44	1
2	38	2
3	32	3
4	26	4
5	20	5
6	14	6
7	8	7
8	2	8

Table 4.9: Dataset definitions for constant resource budget study.

Using some of the "run-time budget" dedicated to execute depletions to completion greatly improves performance when extrapolating past the first 5 timesteps. Figures 4.22 and 4.23 show performance when using a dataset comprised of 50 5-timestep and 0 30-timestep FOM realizations and 26 5-timestep and 4 30-timestep FOM realizations, respectively. Notice that including just 4 30-timestep realizations causes extrapolations past the first 5 timesteps to significantly improve.

In practice, there are clearly many ways this time of "constant run-time budget" experiment could be executed and assessed. The constraints would be application-specific. However, this study at least demonstrates that this way of considering using the available computational resource budget is impactful and deserves attention.

Figure 4.24 shows the MAE-norm error for the ROM realizations as a function of how many 30-timestep datasets are included in the training dataset. Note that all it takes is one 30-timestep depletion within the training dataset to get markedly improved results.

4.4.3 Variational Inference Results

Section 2.3.1 discusses the primary building blocks for the specific Bayesian context through which this section views the NIDS ROMs. In summary, this section will utilize the variational inference (VI) approach to Bayesian neural networks.

VI was chosen for this work because its implementation lends itself to large neural networks, the algorithm is able to be implemented directly into the Parody framework without significant refactoring, and its training time is small compared to other more accurate methods, such as a brute-force Monte Carlo approach. A key concept of VI, is that we replace the unknown posterior with a probability distribution that we know, such as a Gaussian distribution, and reformulate the loss function to include the Kullback-Leibler divergence (KL-divergence) to allow the network to make an estimate for the probability distribution parameters of the model weights.

Other important concepts leveraged for this work include use of the Bayes by backprop (BBB) algorithm with "flipout" and the reparametrization trick to take advantage of the stochastic gradient descent that already occurs within a neural network training loop. The MOdel Priors with Empirical Bayes using DNN (MOPED) approach is also used to improve the convergence speed of neural network training. See Section 2.3.1 for a more detailed review of these algorithms and how they were implemented within the Parody tool.

For this study, we test the implementation of VI within Parody and demonstrate the expected behavior that increasing the sizes of the training data set used for training reduces the estimates of epistemic uncertainty in the neural network. This study also highlights the risk with this methodology and cautions a future practitioner from trusting uncertainty estimates without question.

A subset of isotopic information is shown below to illustrate these conclusions. ^{235}U, ^{235}U, and ^{135}Xe were chosen. ^{135}Xe was chosen because of how difficult it is to capture in general for the ROM models generated for this work and represents the lowest performance bound expected for isotopic depletion predictions. ^{235}U and ^{235}U were chosen as representatives of the other main heavy isotopes in the focal isotope group of the previous section.

Once a Bayesian model is trained ($p(\omega|D)$) it was sampled 100 times to get a distribution of its outputs at each location in the model. The outputs are assumed to follow a Gaussian distribution. These values are plotted below in Figures 4.25 through 4.28 which show the estimates of epistemic uncertainty for the models for all timesteps for a representative location in the model. These figures show results when training with datasets containing 5, 15, 20, and 30 timesteps, respectively. The remaining time steps for each case are extrapolated. Each shade of blue represents one standard deviation of distance away from the model predicted average.

This study demonstrates the ability to capture epistemic uncertainty in a deep learning context for the isotopic depletion application problem on the 2D model. Epistemic uncertainty is notoriously difficult to benchmark against, as it requires us to answer the question: "how uncertain <u>should</u> the model results be?" However, figures 4.25 through 4.28 show the expected conclusions. First, the uncertainty for ^{135}Xe should be higher than that for other, more well-behaved isotopes. This is because, as shown in Section 4.4, this isotope is not captured well with the NIDS ROM. Second, training with more data reduces the uncertainty bands for all isotopes. This is an expected trait of neural networks and epistemic uncertainty in general. The cost of increased training time is shown to be a trade-off in the associated uncertainty of the model output. Finally, the more timesteps that are extrapolated, the larger the uncertainty bands.

Importantly, a practitioner may want to ensure that their neural network is safe to extrapolate a certain number of timesteps. What these plots show is what was suggested by the results in Section 4.4: extrapolating the past 5-10 timesteps is a risk. Notice in Fig. 4.26 that although the uncertainty bands are relatively small in timesteps 25-30, the uncertainty bands do not encompass the true FOM result. When performing any sort of extrapolation, Bayesian neural networks (BNN) approaches can help provide insight into how many timesteps a model can extrapolate before being outside the calculated uncertainty bands. Future practitioners of these methods in a nuclear engineering context should be aware of this limitation and perform similar studies to ensure that they do not trust their models or uncertainty bands without a questioning attitude.

4.5 3D Quarter Core Results

To demonstrate the scalability of the algorithm to a more realistic nuclear engineering problem in terms of size/degrees of freedom, the same NIDS approach is applied to a 3D case. For the 3D model as described in Section 4.1.2, a collection of 20 FOM runs was executed. Consistent with common machine learning best practices, training datasets were always divided into training, validation, and test datasets. For the 3D case, the last 10 FOM results were withheld as the test dataset and 85% of the remaining dataset was used as the training dataset, with the remaining used for validation. Each FOM solution consisted of 10 time steps of randomly sampled power levels for each case and timestep ranging from 0% to 100% power.

As before, the data were preprocessed in the data pipeline that converted the MC21 output restart, mapping, and MC21 input files into the format required by Parody to create the ROMs. Once the suite of 20 MC21 runs was executed, a NIDS model was created. In this case, the CNN model was unable to produce sensible results before encountering memory problems with the hardware used for this analysis. This, in addition to the poor relative performance of the CNN ROM for the 2D case, gives confidence that the NIDS approach is the most promising of the two approaches. The 3D model is substantially larger than the 2D model ($60 \times 60 \times 24$ spatial locations), resulting in much more expensive and computationally limiting files at each intermediate stage in the data pipeline. For comparison, a single MC21 timestep file for the 2D model is $\tilde{2}.5$ MB, while one timestep for the 3D model is $\tilde{3}00$ MB.

These file sizes may seem small until one considers that for this demonstration problem, with 10 FOM results that each contain 10 timesteps, this equates to 90GB. For a typical analysis you may have upwards of 50-100 timesteps, for 10s or 100s of FOM results. Considering this potential for the overall file footprint to balloon so easily, for larger application problems efficient memory management (loading data onto and off of training GPUs, for instance) will become required. This drastic difference in disk space and in memory requirements, as well as substantially higher computational cost (approximately 2160 CPU hours for the 3D case versus approximately 44 CPU hours for the 2D case), means that the scope of this analysis will be smaller and will serve primarily as a proof of concept of this training pipeline rather than an opportunity to explore the best training strategies and analyzing isotopic performance as closely as was performed for the 2D case.

Training occurred until the neural network experienced a plateau in its loss function, using the same callback inputs as the 2D model (a patience of 10 and factor of 0.2 for the reduction of the learning rate on a plateau, and a patience of 20 for early stopping criteria) to ensure efficient use of compute resources and adequate training convergence. As before, we first review the qualitative performance of the NIDS ROM before citing its performance using the metrics from Section 4.3. Also consistent with the 2D model, we will observe the focal isotopes of ^{234}U, ^{235}U, ^{238}U, ^{235}U, ^{239}Pu, SR90, ^{135}Xe, and ^{135}I. The ^{10}B isotope is also shown here in the 3D model, which was not present in the 2D model.

Figure 4.29 shows comparisons of isotopic depletion for all focal isotopes. This model is trained with all 10 timesteps worth of data. Each row is a separate isotope, and each column shows a different perspective on performance. The first column shows traces with time of isotopic concentrations at various x-y-z locations. These locations are intended to show performance for regions with differing properties (proximity to the radial reflector, axial reflector, and centroid of the core). The second column shows the core average MAPE for each timestep for that isotope as a function of time to illustrate how the error changes with depletion. This quantity is averaged over all locations and therefore looks much tighter than some of the plots in the first column. Finally, the third column shows a distribution of MAE for the last timestep in the trajectory for a representative z-slice.

Figure 4.30 shows similar qualitative performance. Each row in Fig. 4.30 is a different isotope, and each column is a different arbitrary power history for a single location in the core. The ability to capture ^{135}Xe and ^{135}I is lacking here, as was the case in the 2D model; however, as previously discussed, these isotopes have little consequence for a nuclear designer if the ROM were used in this way.

The timestep study is performed again for the 3D model. This time, the first 2, 5, and 10 timesteps were included. Results similar to those of the 2D study were observed. Quickly after the model begins extrapolating to lifetimes not included in the training dataset, performance deteriorates. Because the 2D and 3D models performed qualitatively similarly, it is likely a safe assumption that the 3D model would perform similarly if using a constant budget training dataset as shown in Section 4.4. We leave it to future analyses and applications to explore which type of training dataset and regiment are most appropriate for a 3D model of interest.

Figures 4.31 through 4.33 show the performance of the model, which includes the first 2, 5 and 10 time steps for ^{235}U and ^{235}U. Table 4.10 shows the results for the 10 timestep case for comparison.

Isotope	MAPE	MAE	MAE-norm	MSE
U234	0.0431	5.34e-06	0.0181	6.38e-11
U235	0.0159	8.68e-07	0.0162	1.63e-12
U238	0.1397	1.92e-09	0.0258	1.33e-17
PU238	0.0255	2.52e-06	0.0266	1.46e-11
PU239	0.0344	7.46e-07	0.0250	1.31e-12
SR90	0.0315	9.35e-10	0.0173	2.92e-18
CS137	0.0138	3.07e-07	0.0104	2.43e-13
XE135	0.0154	6.37e-07	0.0109	1.05e-12
I135	1.8221	2.17e-08	1.0059	1.12e-15

Table 4.10: Summary of NIDS error metrics for the 10-timestep case.

4.6 Impact of SDF on Performance

As mentioned in Section 4.1.1, when discussing spatial scales on the order of a neutron's mean-free path in a lattice, neutron flux is heavily dependent on proximity to strong absorbers or reflectors. To capture this behavior with a surrogate model, either a larger neural network could be constructed, which gives the model the flexibility to capture these dynamics, or we must do some level of feature engineering/augmentation to provide the model more information on the problem at hand. The SDF was used for both 2D and 3D models, with varying degrees of impact. This feature engineering step allows us to encode important geometric information as part of the input. With this additional input, the ROM converged with greater precision. The benefits were minimal for the small-scale 2D case, but significant for the 3D case.

To assess the impact of the SDF on model performance, new models are trained for both the 2D and 3D cases. For each case, all available training data was used. This includes all training cases (30 and 10 cases for the 2D and 3D cases respectively) and available timesteps (30 and 10 timesteps for the 2D and 3D cases, respectively). Figure 4.34 shows the impact of including the SDF as part of the parameter input array. The error metric shown is the MAPE. Notice that the performance increases for some isotopes, but it also decreases for others. Generally, the changes are marginal. It is expected that the 2D results would improve less than the 3D results because the 2D results do not have a complicated lattice structure. There is only one non-fuel pin, and it resides in the corner. So, the distance to the single non-fuel element follows a simple linear pattern as the fuel pins move farther from the corner of the model. This relationship is likely easy for the neural network to capture and thus including the SDF does not result in large improvements.

Contrast this with the impact of the SDF on the 3D case in Figure 4.35. Here we see that each isotope sees a significant improvement. Notice that the y-axis is log scale for readability between isotopes. Performance increases by as much as 35% for ^{239}Pu, and on average improves the isotopic errors by 17%. Further improvements to the way in which the SDF is implemented will be discussed in Section 4.7.

The SDF also improves the spatial distribution of the isotopes. While the previous metrics look at global performance, observing plots of spatially dependent error reveals the absolute necessity of including an SDF for the 3D model. Figures 4.36 and 4.37 show the performance of NIDS for a few uranium isotopes. Of interest is the column on the right, which shows the spatial distribution of error. These plots demonstrate the importance of the SDF. Notice that without the SDF, there are two undesirable features. First, there is a clear in-to-out dependence on the error. In other words, there is a clear error gradient when moving from the center (the lower right corner) to the reflector. This likely reflects the inability to capture the dependence of isotopic evolution on proximity to the reflector. Second, the pattern of pins, where there is no fuel, is more clearly seen in Figure 4.37. This reflects the fact that the model including the SDF is better able to capture how these isotopes should be evolving when close to non-fuel elements.

4.7 Summary

These studies demonstrate the relative effectiveness of the CNN and NIDS based ROMs for a nuclear model depletion study. Specifically, this analysis looked at a small-scale truncated assembly 2D model, as well as a 3D quarter-core model meant to simulate a 37 assembly core. As with the reactivity insertion accident (RIA) analysis, the CNN ROM under-perform when compared to the NIDS ROM. This study extended the ideas of chapter 3 by applying them to a more complicated nuclear model (37 versus 1 assembly), predicting a higher dimensional output (190 isotopic distributions versus only relative power distributions) and incorporated an improvement to the NIDS algorithm in the inclusion of the SDF, which provides additional geometric information in the form of distance to the nearest non-fuel region. In general, a good agreement was observed between the NIDS ROM and the FOM.

This work represents a first-of-a-kind approach to using ROMs in the context of prediction of depletion trajectory. It leaves open a number of paths to explore both in improving the results shown herein and extending this work to make it more appealing for deployment in a production environment.

1. Augment the inputs with isotopic half-lives.

   There appeared to be some dependence on performance of individual isotopes and that isotope's half-life. A simple follow-up study would incorporate the isotope half-lives as an input to the model. When performing machine learning at any scale for any set of inputs, if the practitioner is able to do feature engineering to any degree, it often leads to a benefit in performance. This is because giving the model a stronger signal via providing it with known relationships (i.e., half-lives) to direct its learning during stochastic gradient descent, rather than depending on the weights and biases within the model to learn those relationships, inevitably results in more effective (and in some cases even possible) training. It is expected that including this additional information would allow the neural network to better predict some isotopes that were not captured well due to short half-lives.

2. Explore the impact of the variable timestep length.

   The primary input to this study was the power level at which to deplete the model.

Another realistic input that is varied during these types of studies in a production environment is the timestep length. It is expected that the NIDS ROM can learn the complicated relationship between timestep length and isotopic depletion, however it was not shown in this work. This study would be a natural next step to extend the work herein to bring it one step closer to being deployable in a production context for studies with arbitrary depletion trajectories.

3. Augment SDF parameters to include a dimension for each type of "non-fuel" region.

   Currently, the SDF input has one dimension outside of time and location. This single dimension contains the distance to the closest non-fuel region. However, there are a few ways that a location in the core can be "non-fuel", and this difference is likely important for improving performance. For example, as seen in the poison assembly type (repeated below in Figure 4.38 for convenience), there are both the coolant (in the center) and the poison pin types (all other non-fuel regions). Also, the reflector represents another type of non-fuel region that is important to the behavior of nearby neutrons. Put another way, if the SDF were 3 dimensional instead of 1, the performance could be improved. The three dimensions are proximity to poisons, coolant pins, and the reflector.

Each of these side studies represents a way to improve performance or explore what unique enhancements to the modeling scheme shown here that are required for specific applications of nuclear engineering.

The purpose of a study like this is to determine the viability of using ROM models in a design environment for some application. However, as discussed in this work, there are many ways these ROM models could be applied, trained, and evaluated. Using this work as evidence, the NIDS model seems much more likely to be deployable in a production environment than the CNN model. However, even the NIDS model lacks in certain metrics which may be important for some applications. For example, some analysis types are specifically concerned with the behavior and impacts of xenon transients, which we have seen are not well captured in this incarnation of the ROM. Likewise, other applications may not have as many isotopes, may have more controlled power histories (i.e., not ranging from 0% to 100%, but perhaps something more manageable like 70% to 100%), and may take much larger steps in time. All of this would likely improve the metrics of interest.

Similarly to Chapter 3, it is clear that the decision of whether a ROM can be applied to some application is difficult to speculate on without knowing the specific interests of the application or how it would be deployed. However, this work has demonstrated that ROMs constructed in this manner are at least a viable option and worthy of future research.

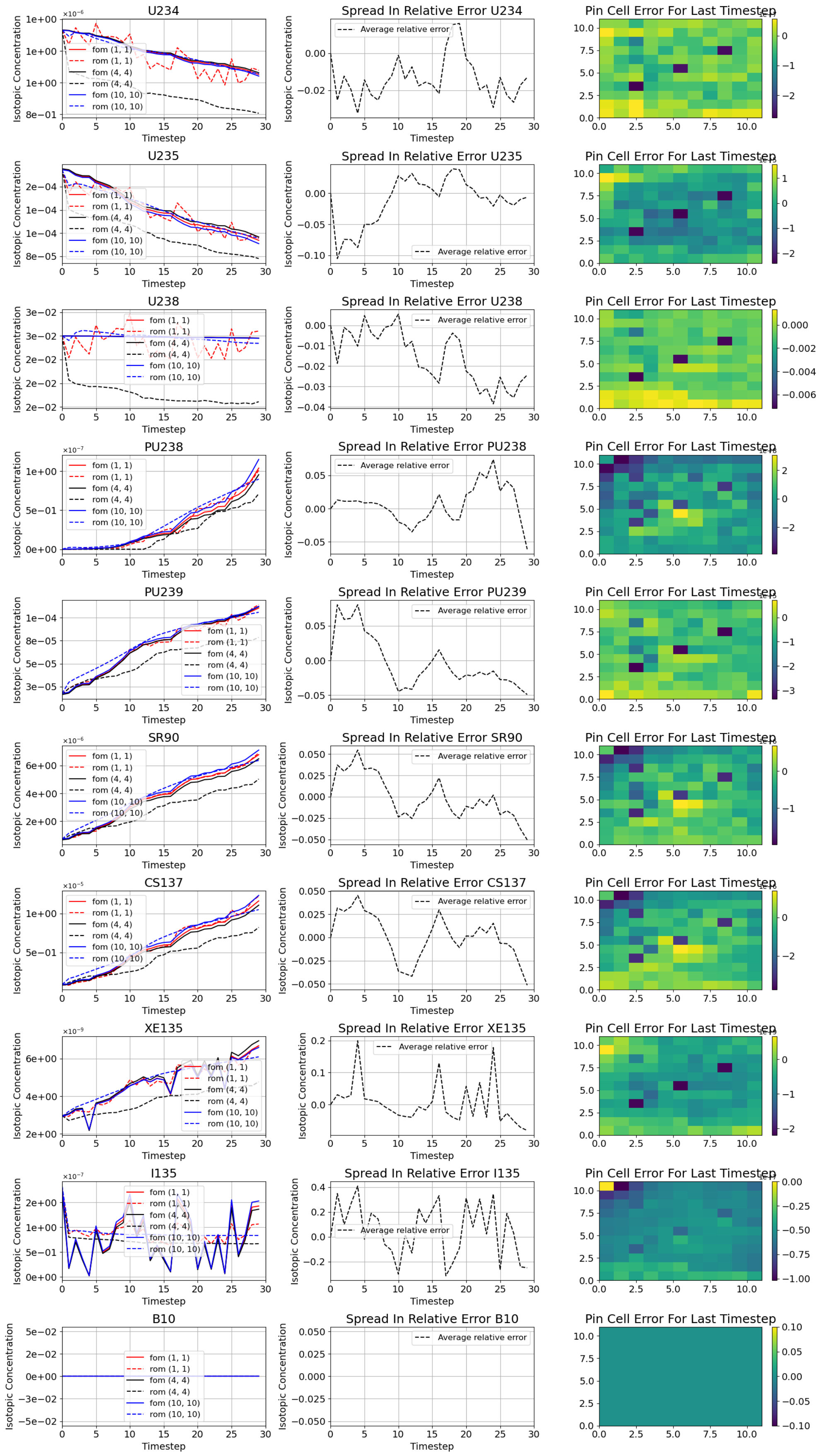


Figure 4.11: FOM and ROM (CNN) depletion trajectories for an arbitrary power history at various locations for all focal isotopes. All 50 realizations used during training.

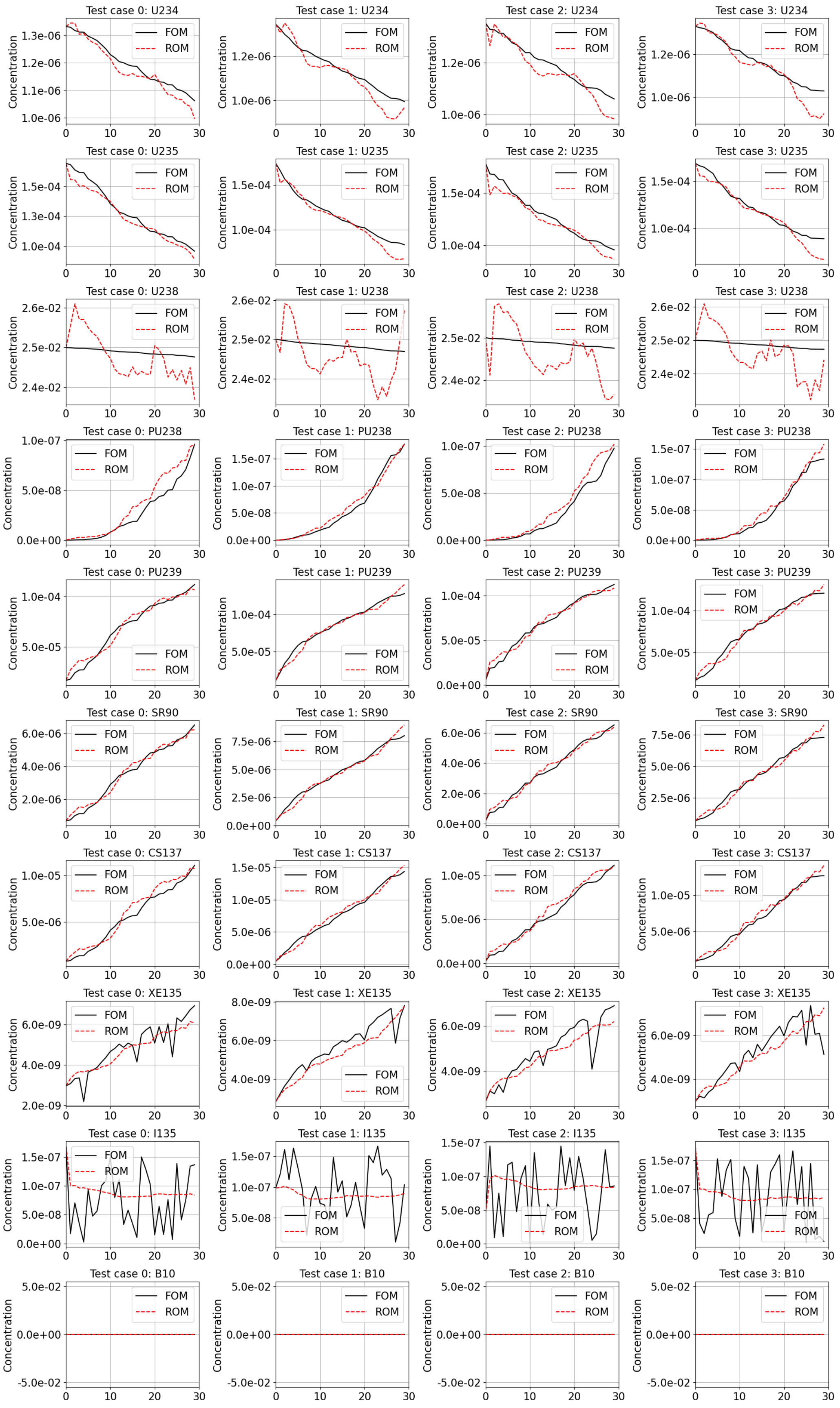


Figure 4.12: FOM and ROM (CNN) depletion trajectories for an arbitrary power history for 4 test cases for the focal isotopes. All 50 realizations used during training.

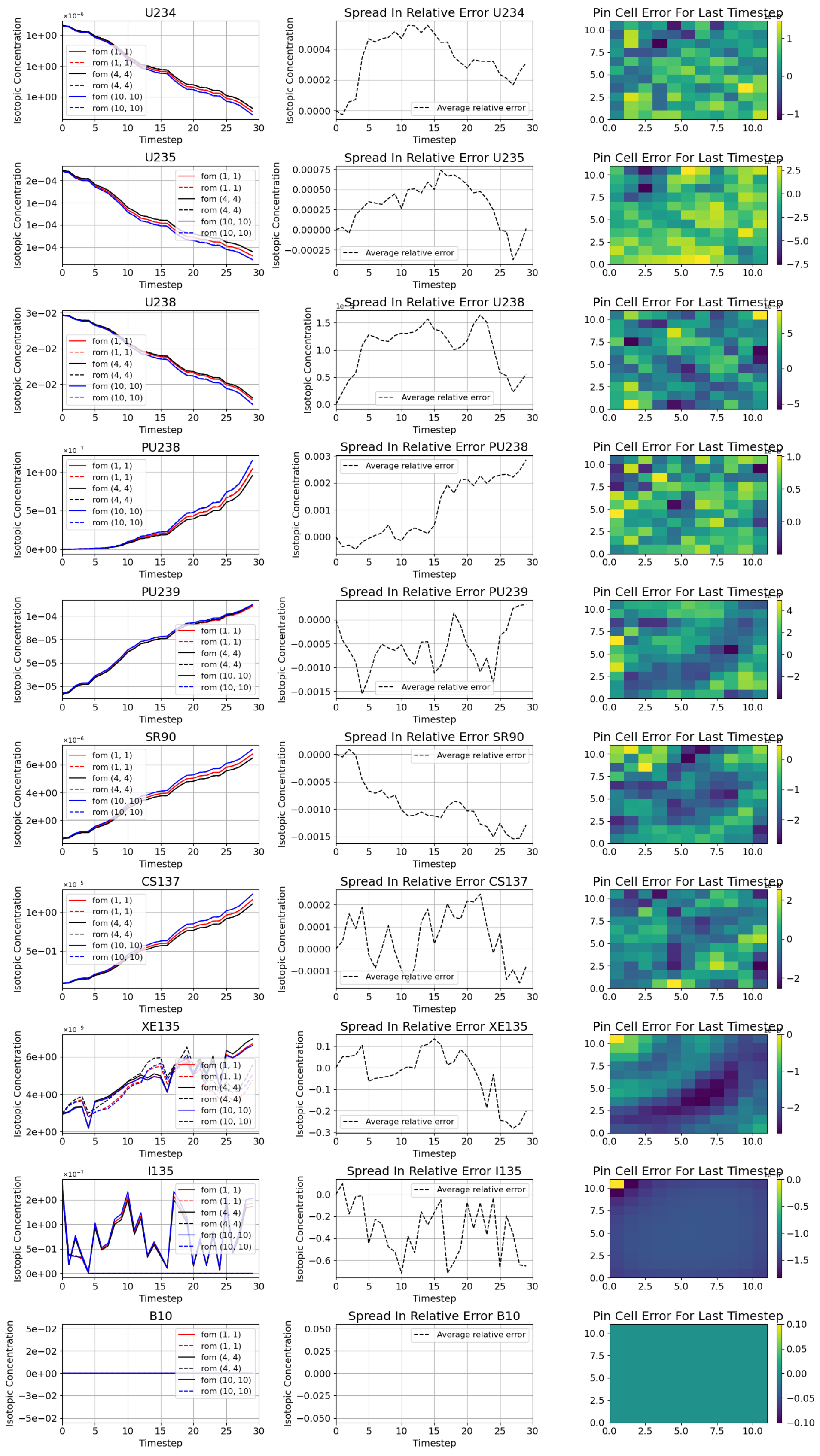


Figure 4.13: FOM and ROM (NIDS) 2D depletion trajectories for an arbitrary power history at various locations for all focal isotopes.

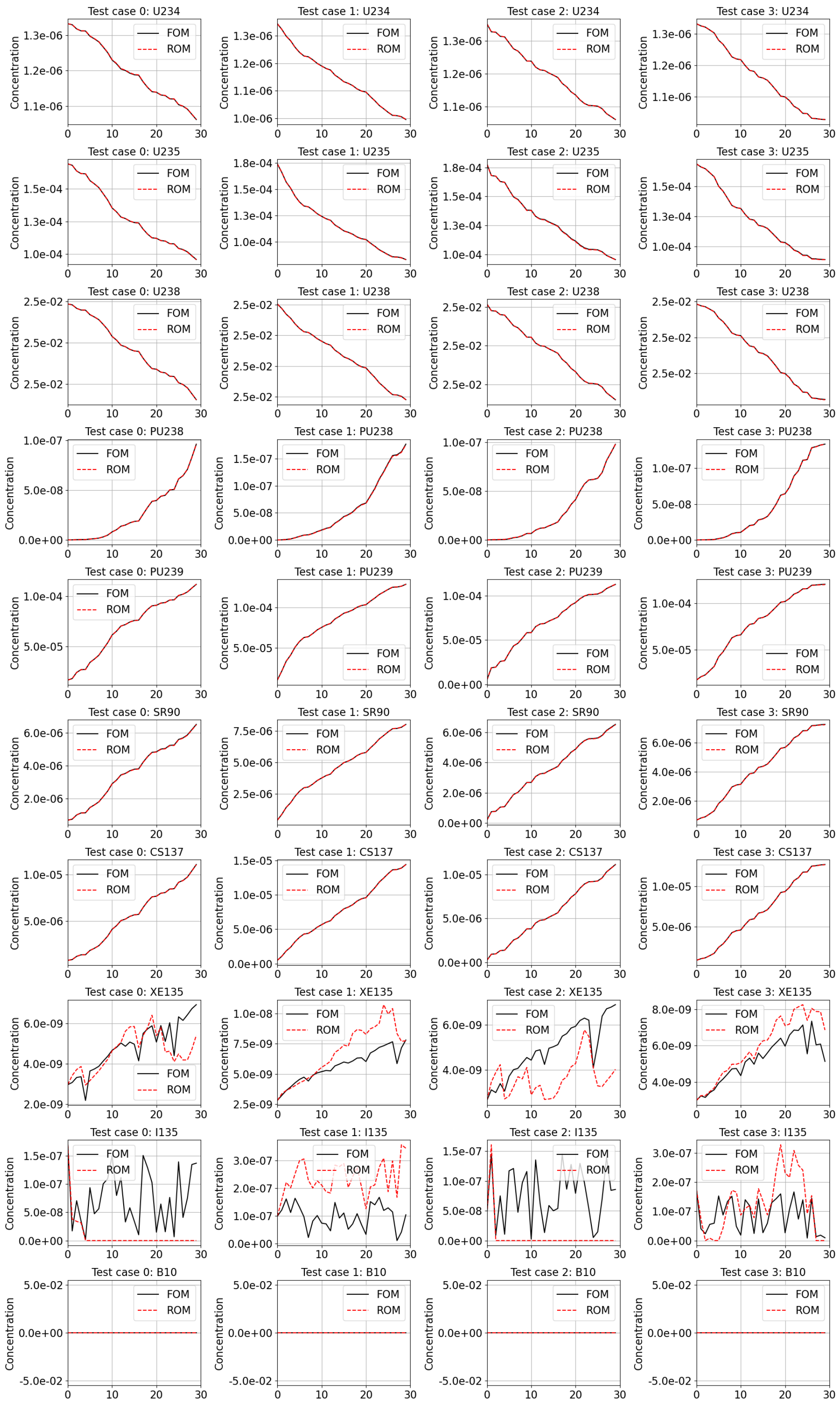


Figure 4.14: FOM and ROM (NIDS) 2D depletion trajectories for arbitrary power histories for 4 test cases for the focal isotopes.

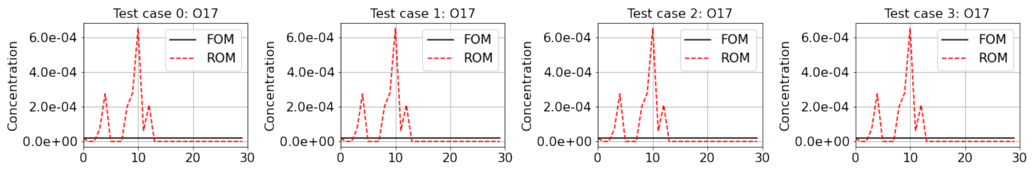


Figure 4.15: O17 depletion trajectory predictions.

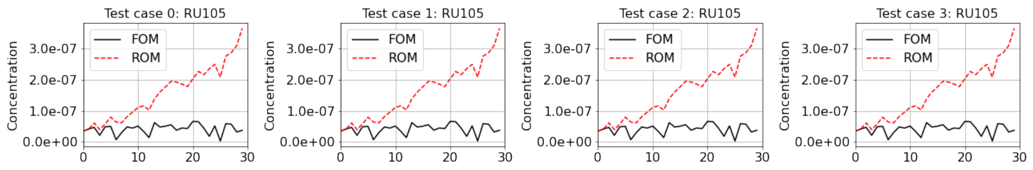


Figure 4.16: RU105 depletion trajectory predictions.

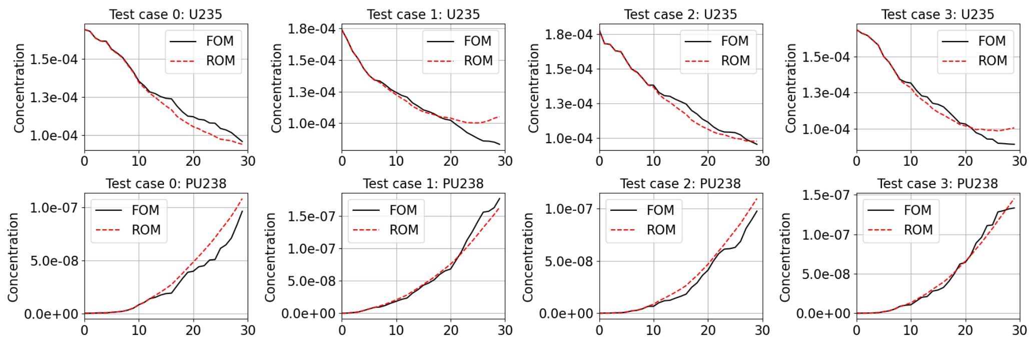


Figure 4.17: FOM and ROM agreement when training with first 5 timesteps.

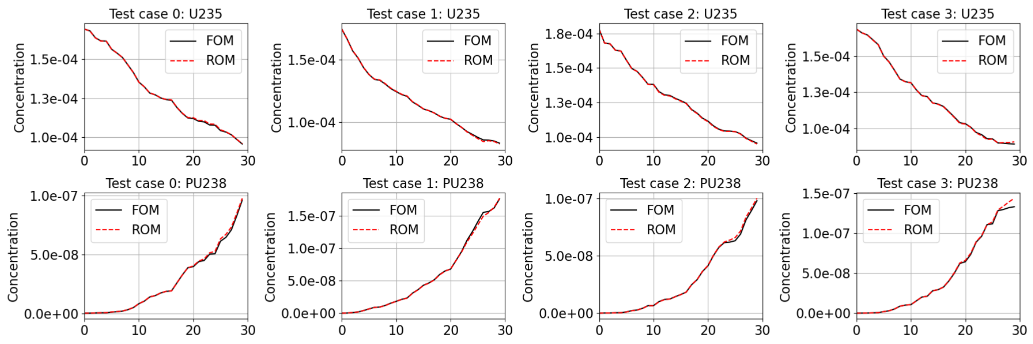


Figure 4.18: FOM and ROM agreement when training with first 15 timesteps.

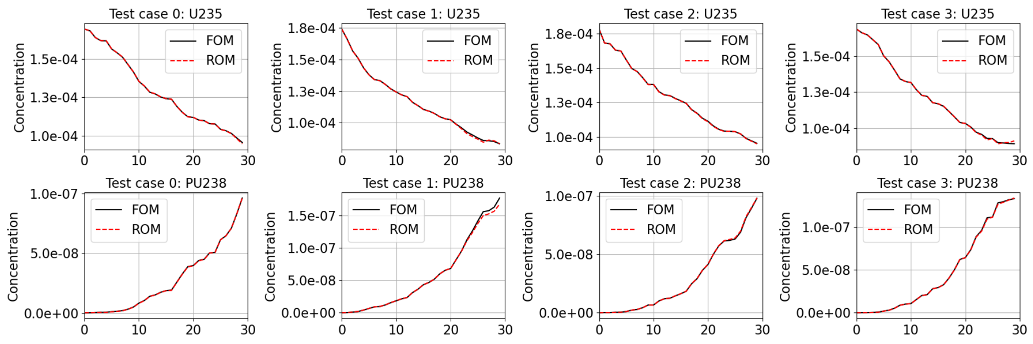


Figure 4.19: FOM and ROM agreement when training with first 20 timesteps.

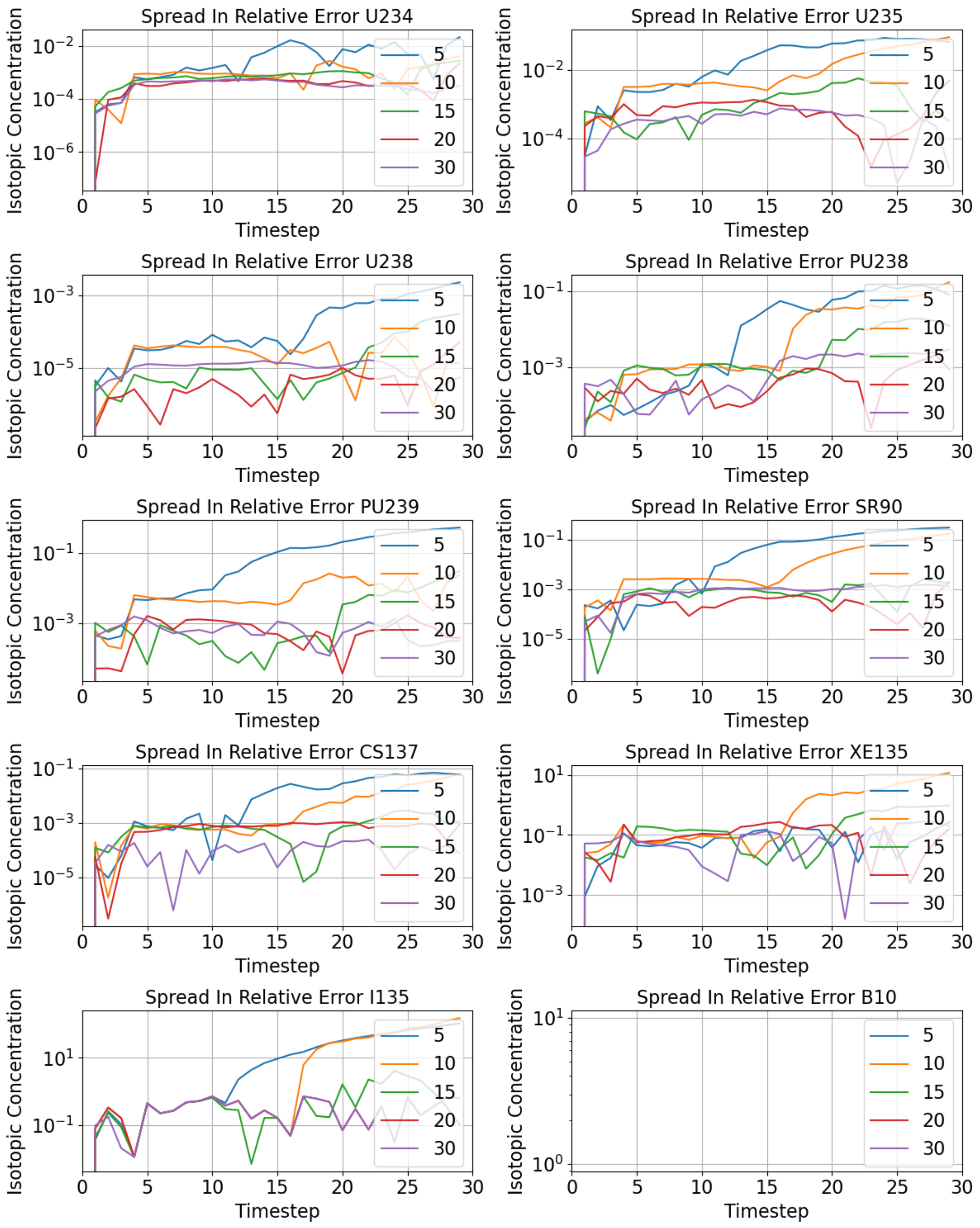


Figure 4.20: FOM and ROM agreement for each training dataset.

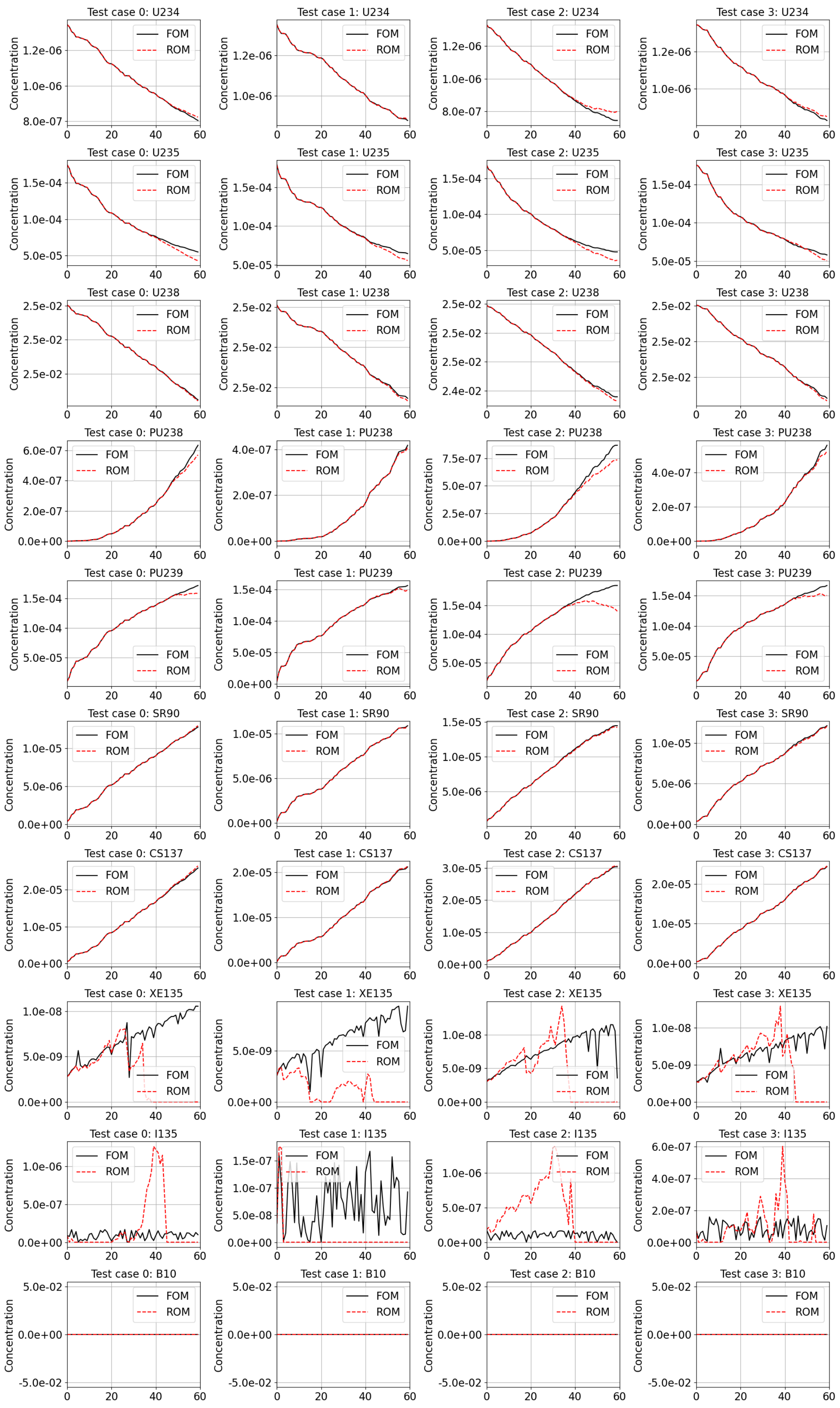


Figure 4.21: FOM and ROM depletion trajectories for a 60-timestep depletion using dataset containing 30 timesteps.

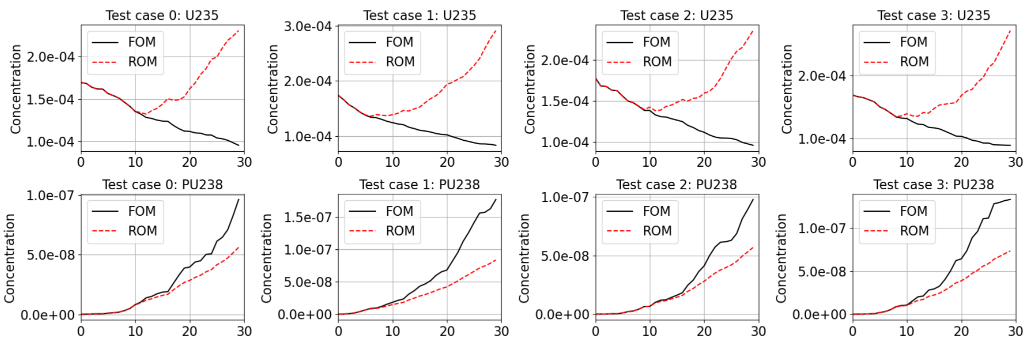


Figure 4.22: FOM and ROM agreement using 50 5-timestep realizations with 0 30-timestep realizations.

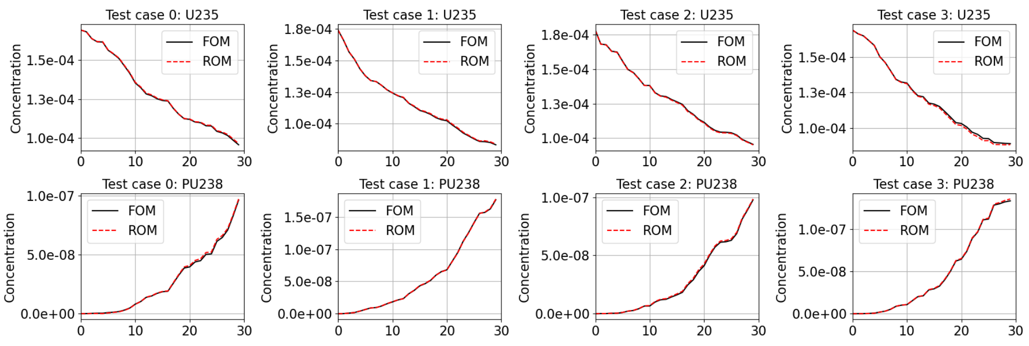


Figure 4.23: FOM and ROM agreement using 26 5-timestep realizations with 4 30-timestep realizations.

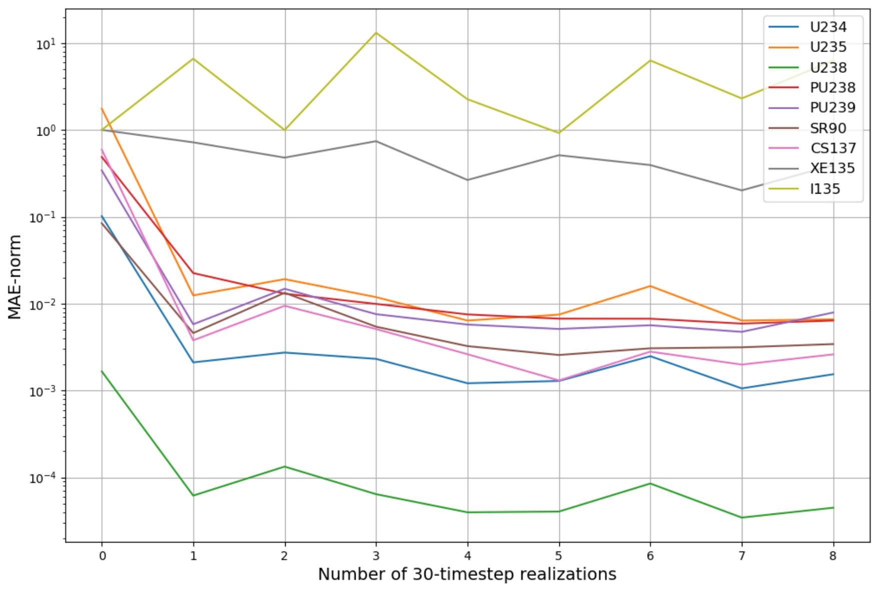


Figure 4.24: FOM and ROM agreement as a function of number of 30-timestep datasets used in training.

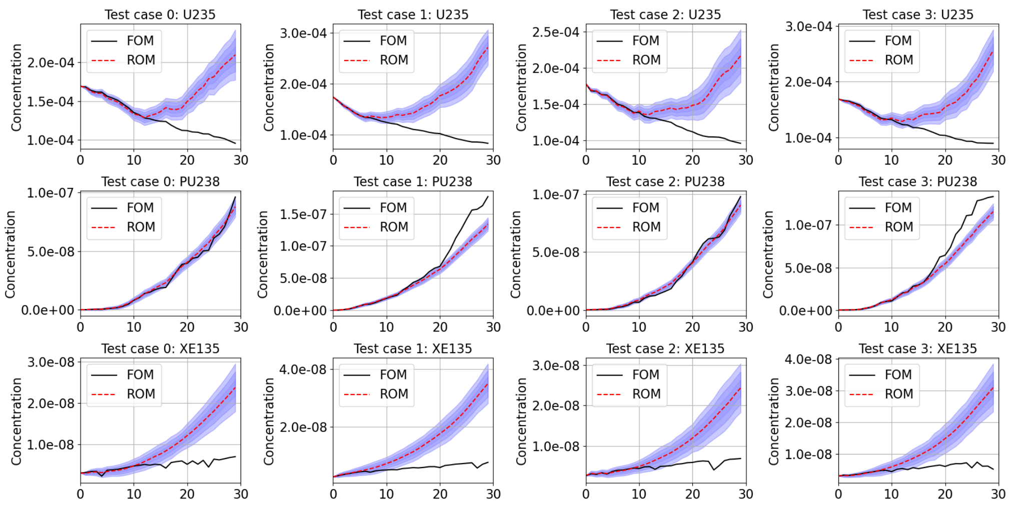


Figure 4.25: FOM and ROM (NIDS) VI depletion trajectories for an arbitrary power history at one location in the 2D plane. This model used 5 timesteps in its training dataset and extrapolated the remaining timesteps.

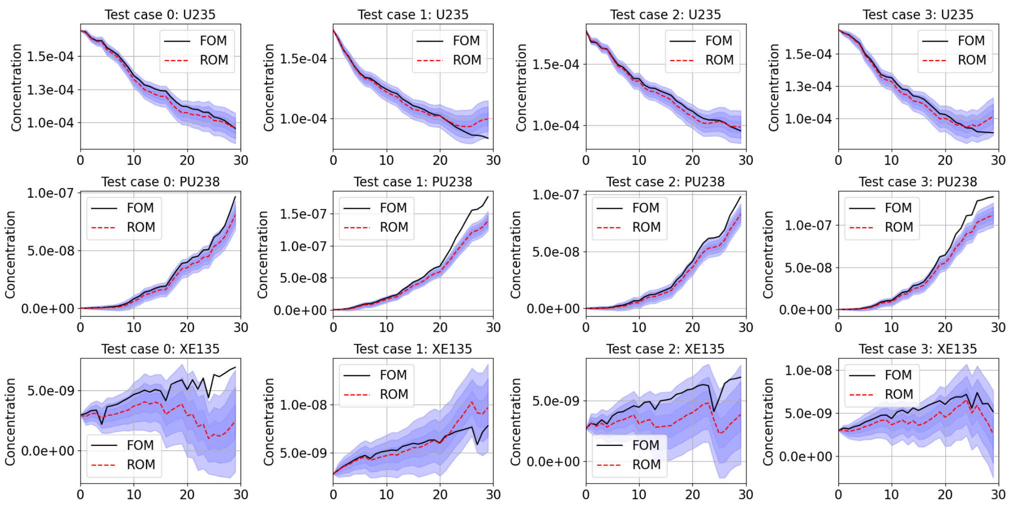


Figure 4.26: FOM and ROM (NIDS) VI depletion trajectories for an arbitrary power history at one location in the 2D plane. This model used 15 timesteps in its training dataset and extrapolated the remaining timesteps.

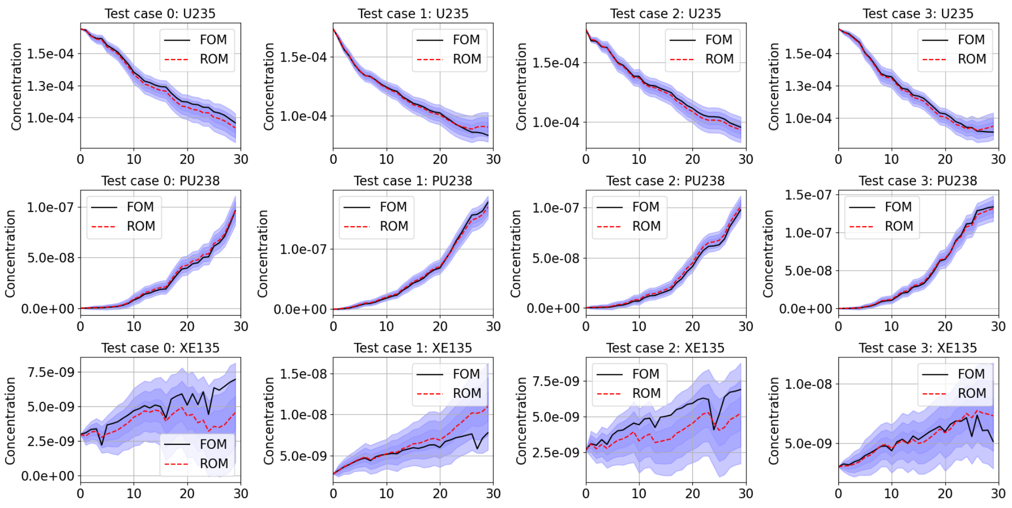


Figure 4.27: FOM and ROM (NIDS) VI depletion trajectories for an arbitrary power history at one location in the 2D plane. This model used 20 timesteps in its training dataset and extrapolated the remaining timesteps.

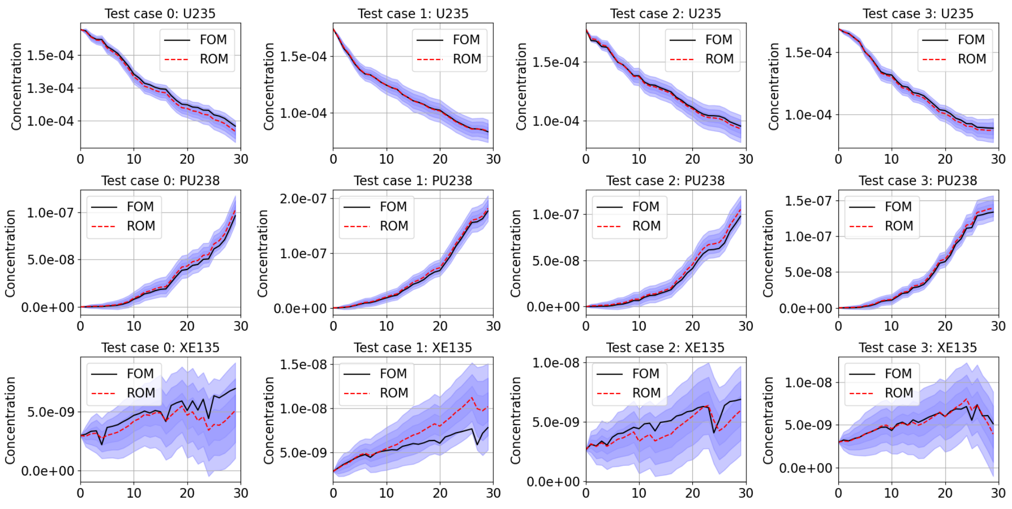


Figure 4.28: FOM and ROM (NIDS) VI depletion trajectories for an arbitrary power history at one location in the 2D plane. This model used 30 timesteps in its training dataset.

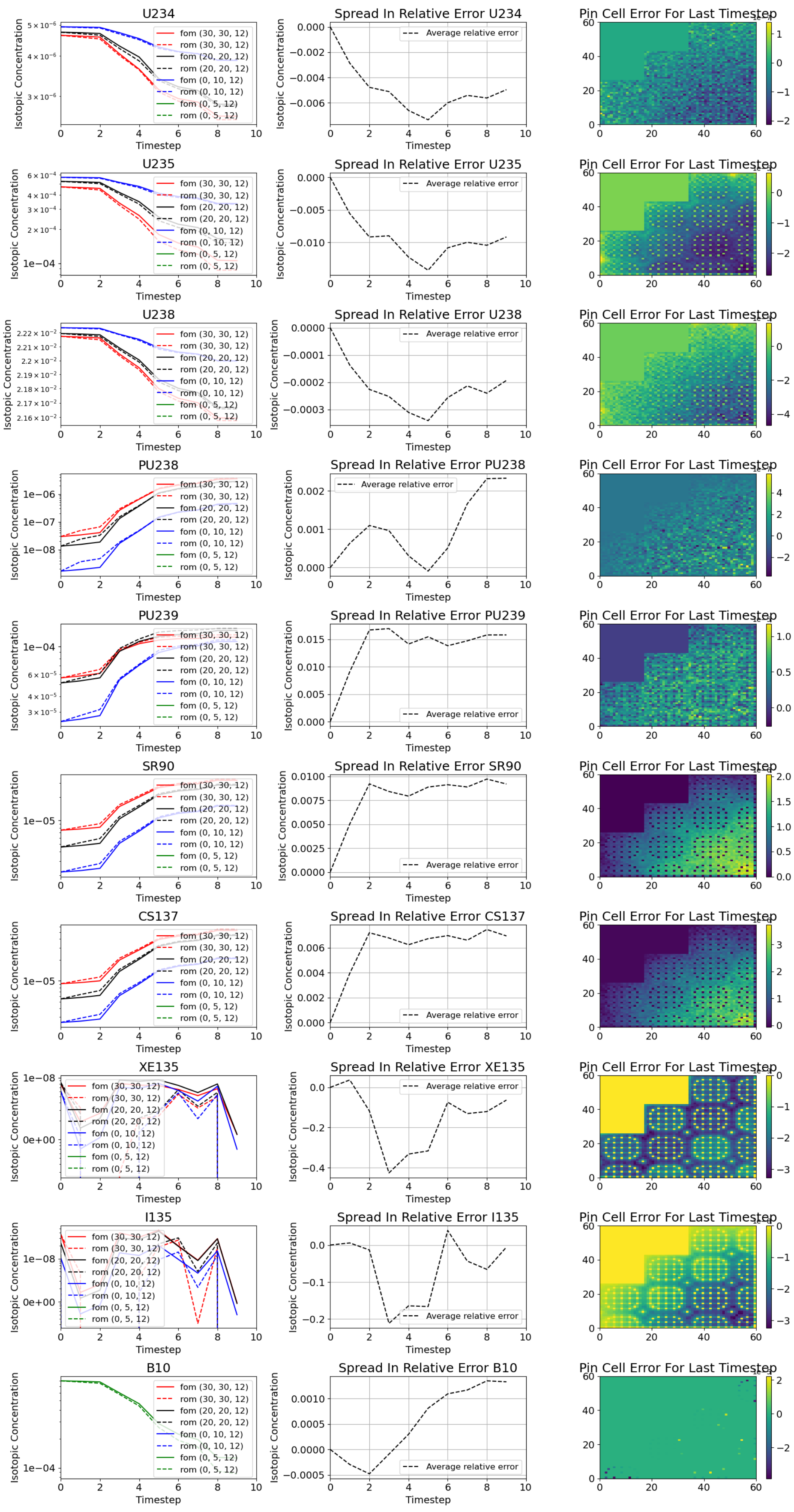


Figure 4.29: FOM and ROM (NIDS) 3D depletion trajectories for an arbitrary power history at various locations for all focal isotopes.

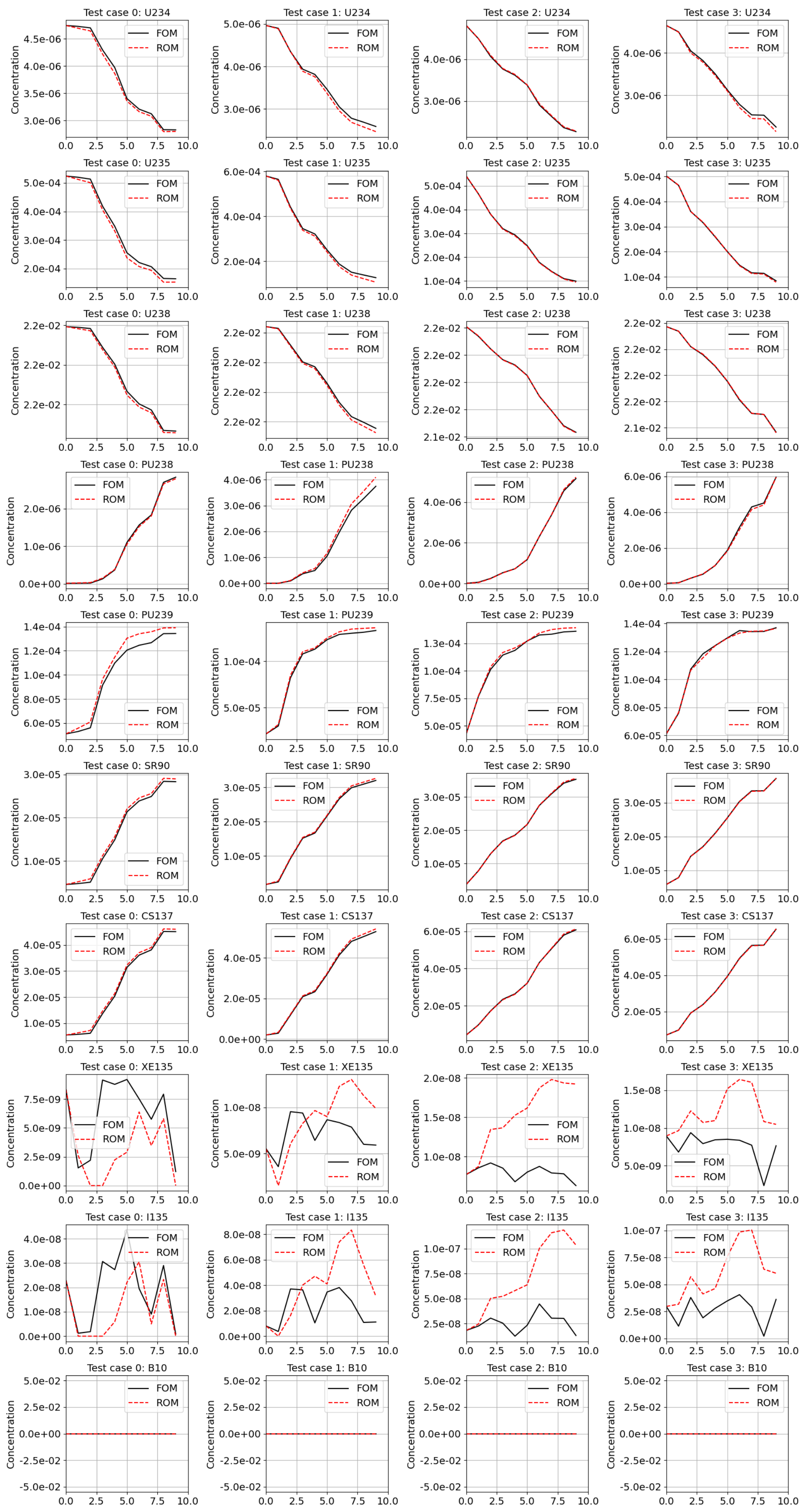


Figure 4.30: FOM and ROM (NIDS) 3D depletion trajectories for arbitrary power histories for 4 test cases for the focal isotopes.

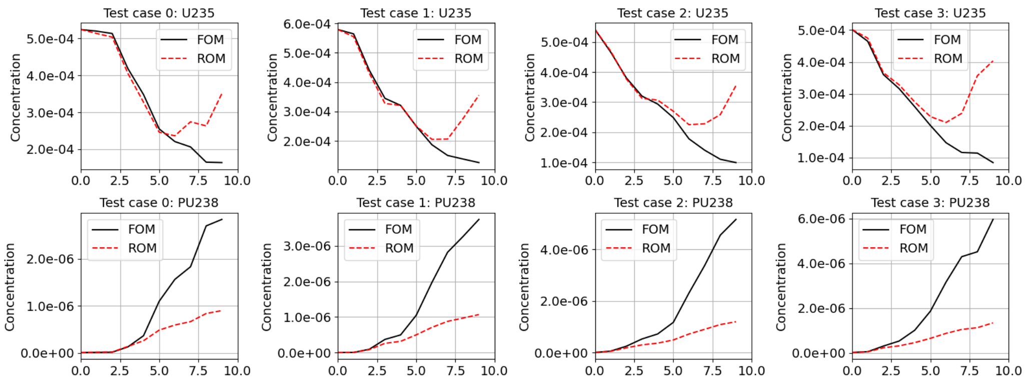


Figure 4.31: FOM and ROM agreement when training with first 2 timesteps.

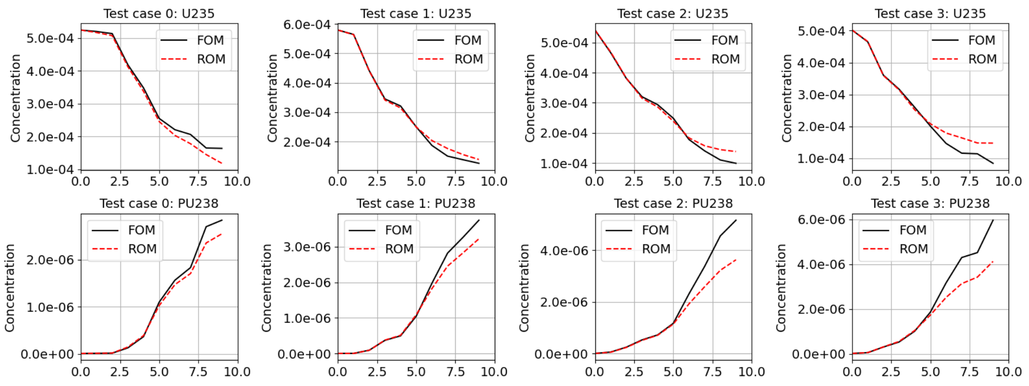


Figure 4.32: FOM and ROM agreement when training with first 5 timesteps.

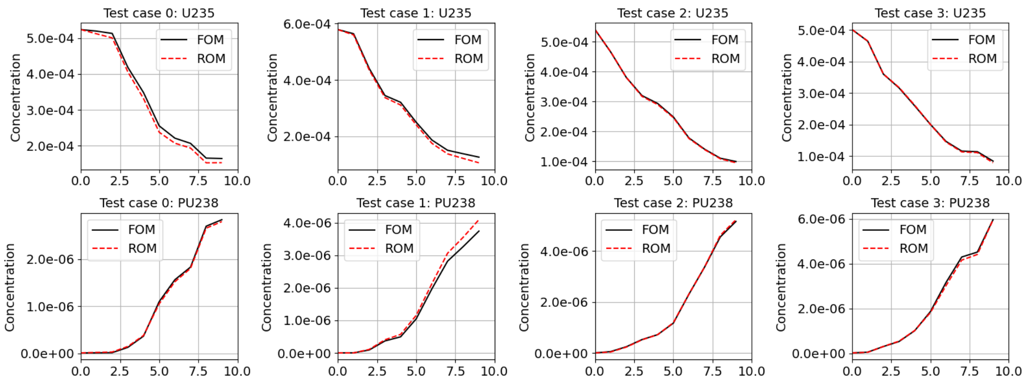


Figure 4.33: FOM and ROM agreement when training with all 10 timesteps.

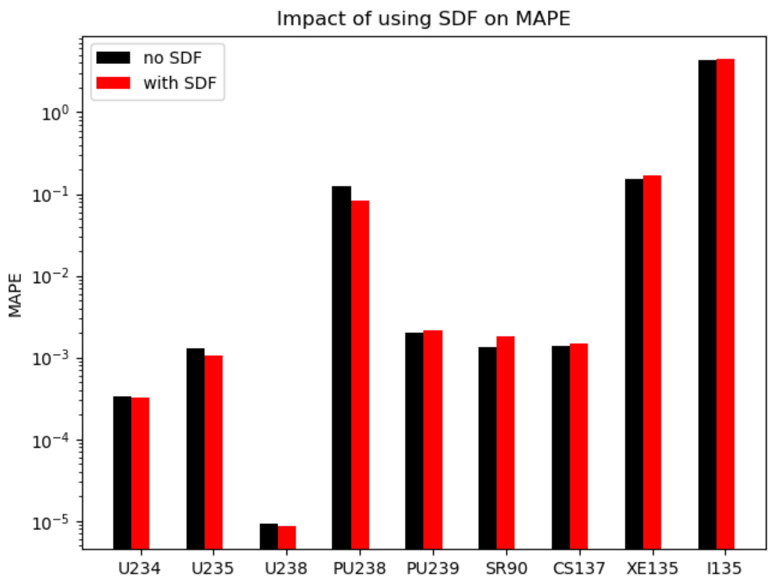


Figure 4.34: SDF impact on MAPE for 2D case.

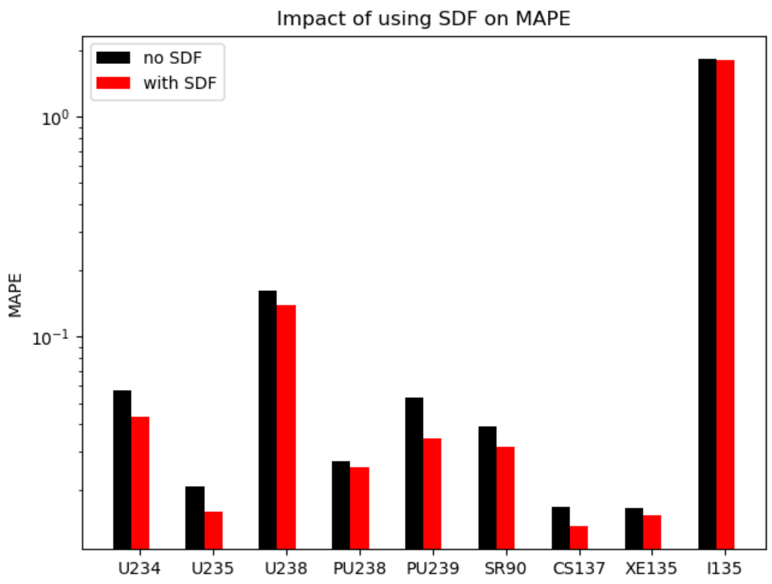


Figure 4.35: SDF impact on MAPE for 3D case.

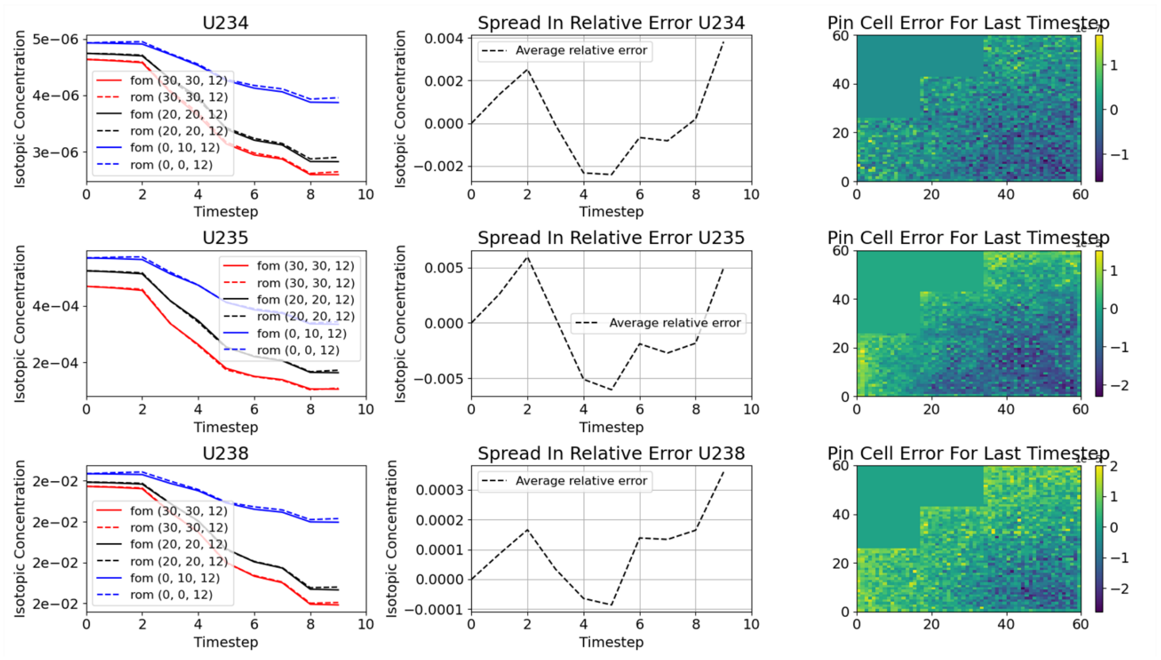


Figure 4.36: NIDS performance for important Uranium isotopes with no SDF.

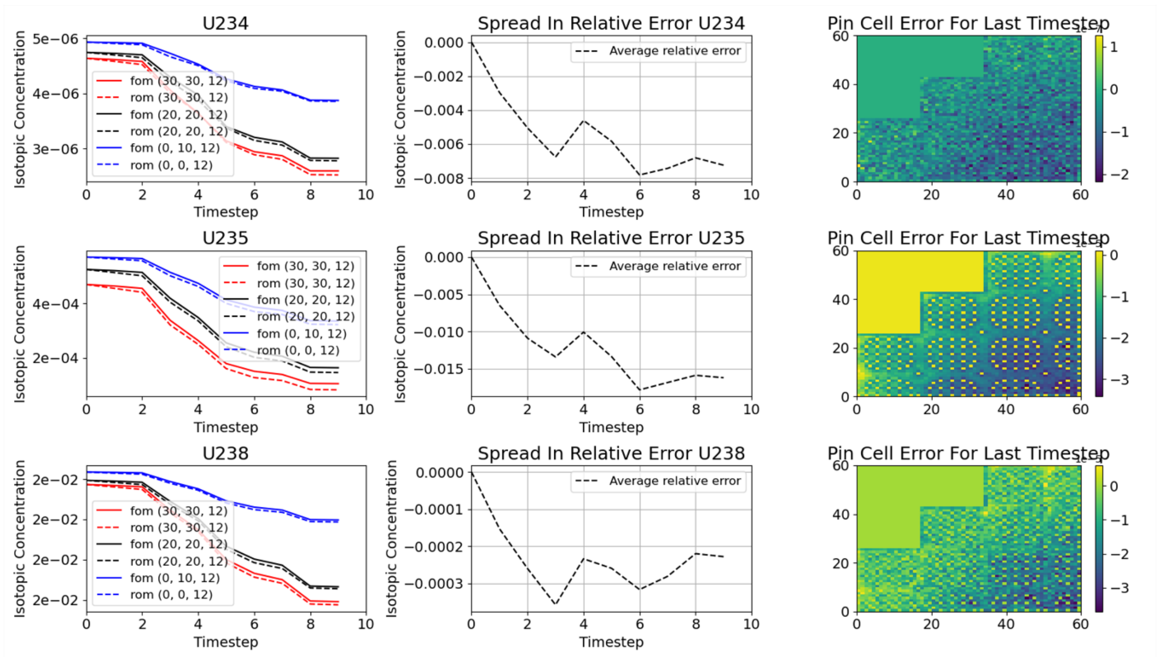


Figure 4.37: NIDS performance for important Uranium isotopes with an SDF.

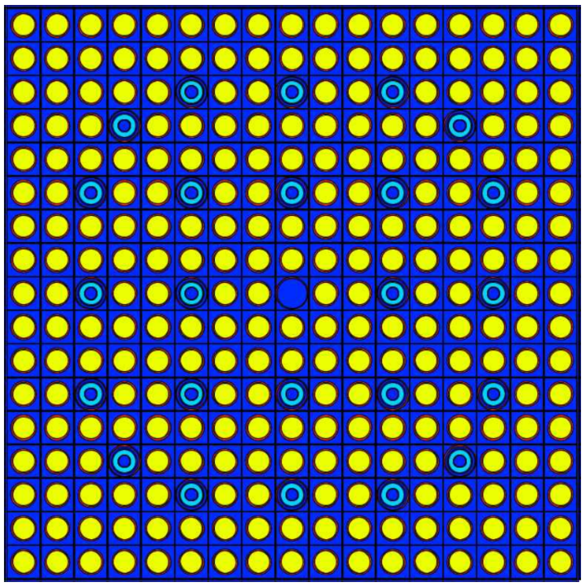


Figure 4.38: The poison assembly in the 3D model.

CHAPTER 5

Conclusion

5.1 Conclusions

The work presented here constitutes a first-of-a-kind deployment of the neural network-based reduced-order model (ROM) construct applied to two representative nuclear engineering applications. Work has been done in the ROM community to address the need to reduce the dimensionality of physics problems, but the nuclear engineering community has not yet deployed these methods outside of research contexts. The nuclear reactor design and engineering field has a strong need for coupled thermal hydraulics/neutronics multiphysics simulations because of the tight interaction between thermal-hydraulic and neutron transport, which poses an additional layer of complexity when converging solutions. This work also introduces the Parody tool (see Appendix B) to provide an abstract interface for ROM research and deployment. Overall, the work helps to take the application of these ROM methods one step closer to deployment in a production nuclear engineering environment. Specifically, two classes of ROMs are applied to two classes of nuclear engineering problems.

The first class are two types of convolutional neural network (CNN) based ROMs, which used convolutional layers to compress and expand the state variables of interest into lower-dimensional subspaces. These ROMs then step forward in time in this lower-dimensional subspace and used inverse convolutional layers to expand the latent space back into the full-order subspace. By mapping the inputs to these compressed latent spaces, it becomes possible to predict the state variable distributions for new combinations of input parameters. The second class of ROM is the Non-Linear Independent Dual System (NIDS) algorithm. This approach can be thought of as a non-linear analog to the classic Proper Orthogonal Decomposition (POD) algorithm. NIDS exists within a class of "hypernetwork" approaches that seek to train neural networks that produce the weights and biases of another network that is ultimately used for predictions. The main feature of NIDS is that it builds a continuous field representation where the inputs are local quantities. Specifically, NIDS produces the weights and biases for a single-layer neural network. NIDS trains two networks, one to encode geometric information and another to encode input parameter information. The output of the parameter networks becomes the weights and biases, while the output of the spatial networks become the latent space of the final layer of the "hypernetwork". Furthermore, shown in Chapter 4, the memory requirements for NIDS are significantly lower than those of typical CNN based ROMs. Coupled with the signed distance function (SDF) encoding of geometric information on fuel layout, these characteristics make it an excellent candidate for full field ROMs in the nuclear reactor design field.

The first application problem is a single assembly reactivity insertion accident (RIA) transient. It represents a commonly performed class of analysis in nuclear engineering: a non-linear multiphysics transient with complex physics, feedback mechanisms, and high input sensitivity. To assess whether this RIA analysis could be easily enhanced with ROMs, it is observed in two contexts. In the first, a traditional uncertainty quantification analysis is performed where inputs to the problem were sampled according to their allowed distributions and the impacts to the output state variable of interest (relative power) are observed. In the second context, all input variables except one are kept constant, while the singled-out input variables varied throughout their allowable range. This analysis demonstrates how well each ROM approach is able to tease out the contribution to the state variable of interest by each input parameter. In both contexts, NIDS ROMs performs significantly better than CNN ROM and represents a more promising direction for future research.

The second application problem is a depletion analysis. It represents another commonly performed class of analysis, a lifetime study of a core design to observe how its nuclear characteristics change as fuel is depleted and fission products build up. This application problem also represents a test of the ROM's ability to scale to a realistically sized problem. The first application is a single-assembly model, while the second application is a reflectively symmetric quarter-core model intended to represent a 37-assembly core with a nonhomogeneous fuel and poison arrangement of pins. Additionally, the smaller model has roughly 14k spatial locations, while the larger 3D model has roughly 86k spatial locations.

As a test to assess viability, the isotopic depletion trajectories are chosen as the state variable of interest. This also represents a scaling in output dimensionality. The first application problem has 1 state variable of interest, while this second application problem has 190. Again, both CNN and NIDS ROM algorithms are applied and again the NIDS algorithm shows the most promising results.

NIDS ROMs consistently perform better than CNN ROMs and represent the best path forward as an option for non-intrusive ROM approaches. This is due to both accuracy and hardware limitations. The second application problem in particular is too large for a CNN model to fit on the hardware used for this analysis (a single NVIDIA 3090 GPU). In production environments where much more capable hardware is available, a CNN model may indeed prove to be not only possible but provide competitive results; however, this option was not explored in this work due to the positive performance and light weight nature of the NIDS algorithm. Additionally, CNN models require more complexity to describe via their more numerous hyperparameters and dimensionality constraints (particularly, the act of compression and expansion while conserving final dimensionality requirements can be a fragile exercise). Although not insurmountable, it represents a challenge that is not encountered with the more simplistic NIDS neural network architecture hyperparameter requirements.

Overall, the agreement between NIDS ROMs and the full order model (FOM) is promising. One of the primary purposes of this work is to determine the viability of using ROM models in realistic design environments often encountered in nuclear engineering contexts. The decision of whether these methods can be used in a true production level analysis is difficult to conclude without caveat. The requirements for the analyses are different. For the first application, a designer may be interested in the duration of the power peak, how accurate the peak is at certain thermally limiting locations, the integrated relative power over some time or spatial region, the accuracy at the inlet or outlet, the ROM's behavior in a uncertainty quantification (UQ) analysis, or any other number of unique concerns. For the second application, a designer may be interested in certain isotopes over others, the total mass of isotopes in individual pins, assemblies, or an integrated core, or just the global reactivity value predicted using the ROM tool. Each analysis would require specific metrics, hyperparameters, training regimens, and unique considerations. However, this work shows that these ROMs should be considered as a viable research paths to serve as supplements, replacements, or guides in nuclear engineering analyses at the production level.

5.2 Future Work

This work opens the path to many potential future projects. These paths fall into one of three categories. The first category includes production-level workflows which could benefit from the types of NIDS ROM introduced exactly as presented here. These involve more engineering challenges than theoretical challenges. The second category includes features that improve the generic ROM tool Parody. Finally, the third includes algorithmic extensions that would likely improve performance and allow the application of these methods in novel ways.

5.2.1 Existing workflow integration

There is a spectrum along which a ROM could be used in production design calculations. On the one hand, designers could completely eliminate the need for a FOM (other than to generate training data for a ROM) and use a ROM to replace its role in a whole analysis workflow. On the other side of the spectrum, it could be used as a design space exploration tool, serving as a guide to where in the design space a FOM should be used. The ideas below represent three locations along that spectrum that should be considered when assessing the role a ROM should play in a reactor design context.

1. Improve initial guesses for transient neutronic calculations.

Transient neutronics calculations involve simulating the time-dependent behavior of nuclear reactors under different conditions. These calculations often require numerous iteration steps between solvers at each timestep to obtain accurate results. However, this can be computationally expensive, especially when complex non-linear physics interactions occur, which creates a need for multiple iterations to converge.

A potential solution to this problem is to use the NIDS ROM algorithm. This algorithm can help reduce the number of iteration steps needed, particularly when dealing with complex non-linear physics interactions. Currently, a common approach is to use the last timestep solution as the initial guess for the next timestep. However, using a ROM model to provide a better initial guess for the next timestep could further improve the computational efficiency of the simulations, without replacing the use of FOM models in their role as providing the final converged solution.

To use a ROM model in this context, the state variables of interest (such as relative power distributions, coolant temperature, pressure distribution, etc.) would be used as inputs to the model. The model would then provide the next timestep's state variables of interest as outputs. Relevant physics solvers would use these outputs as a starting point for convergence for that timestep, which would help speed up the simulation process. This would require a robust coding framework to handle interaction between the ROM and FOM and must be compatible with the existing FOM workflow.

This approach could also be used in the context of coupled physics simulations. Instead of converging on solutions using only FOM calculations, ROM calculations could be used to provide better initial guesses for each physics of interest. Like the previous suggestion, the state variables of interest for a coupled neutronics-thermal hydraulics analysis would be the neutronics solution for the thermal hydraulics solver, and the thermal solution for the neutronics solver. This would improve convergence times without impacting the final results, as all results would ultimately come from FOM solutions.

2. Replace the FOM solver every N timesteps.

Some depletion or transient analyses may have a large number of timesteps. One way to insert a ROM into an analysis workflow would be to have it provide state variables at some regular interval, rather than replace the FOM entirely. As in the previous research idea, this would require a robust coding framework to handle the interaction between ROM and FOM. This is reminiscent of other methods currently used in nuclear engineering, where alternating diffusion and transport solutions are used to step forward in time. This idea is different from the previous idea in that the ROM predictions would be used in place of the FOM results instead of providing them with a better initial guess during convergence.

3. Use the ROM tool as a “first-cut” design space exploration tool.

Another method would be to use a ROM as a purely design space exploration tool. Often a designer is interested in finding the most limiting combinations of inputs for some physics model. A combination of FOM results, first principle analysis, and engineering intuition is usually used to justify final conclusions (since it is often intractable to fully survey a high-dimensional design space completely for reactor design applications). The most optimal way to insert a ROM into this procedure would be an interesting study. A ROM could not be deployed until sufficient training data is available. Therefore, an iterative approach involving a FOM to generate training data to feed a ROM to suggest the limiting locations in the design space in which to run the FOM (which would then improve the next iteration of predictions from the ROM) could be valuable. The goal of such a research project would be to find the most optimal balance between compute time and optimized design with the FOM runs and ROM training time representing more accuracy but higher cost.

5.2.2 Programmatic improvements to Parody

The Parody tool (see Appendix B) was developed to facilitate quick turnaround of experimentation with ROM algorithms, particularly the NIDS algorithm. It was generated with inline documentation, testing, and Python package distribution in mind; however, many traits and features that were not developed would preclude it from being used by a wider audience. With proper investment, a tool like Parody could become something that other researchers or industry ROM practitioners could use to help them in their work. These ideas represent considerations for transforming Parody into an attractive tool for such a user base.

1. Develop an abstraction paradigm for data consumption.

Currently, Parody requires data to be in monolithic structures that represent entire training datasets. This is not sustainable if it were to be implemented on larger scales with large datasets. Furthermore, the first two ideas in the previous section would require a smooth transition between FOM and ROM input and output, which would benefit from an abstracted data layer. Work must be done to provide a higher level of abstraction for training data consumption and inference. The abstract classes which Parody is built on top of (PyTorch Lightning's DataModule class) support this type of work, but Parody does not yet allow for this level of abstraction out of the box. As a few examples: a user may want to specify file locations of FOM results files, filter by metadata on which results to include, create interfaces for FOM results fetching, and define metadata on its outputs. A user may also want an application program interface (API) with which to integrate Parody into production FOM tools or workflow tools to facilitate the handoff of FOM and ROM data in a larger pipeline.

2. Expand Parody to include other ROMs.

Parody currently includes only the NIDS and Proper Orthogonal Decomposition neural network (POD-NN) ROM algorithms. It does not include the CNN based algorithms explored in this work, or the many other ROM approaches which exist in the literature. An obvious project to improve Parody's usefulness would be to make it a library of ROMs. Such a work effort would flesh out the areas of improvements needed in its design to allow it to support the various classes of algorithms. This work effort would also force code contributors to consider and implement the requisite level of abstraction to accommodate the needs of varying ROM algorithms. Once a library of ROMs is available, then they could all quickly be compared against one another for different applications of interest.

3. Improve Parody quality according to best practices in software engineering.

Parody currently has basic Python tests, Sphinx documentation of its Python API, example Jupyter Notebooks, and other features common in open-source Python packages. However, it has not undergone development at a production level, which may flesh out use cases not tested, documentation needs not covered, and a robust build and distribution system. An overhaul to improve its professionalism does not represent a research project, but a required software development endeavor if it were to be distributed to a wider user audience. The ROM community would benefit from efforts in this vein. This would allow researchers to not need to create the workflows required by this complicated work for each research group. Instead, work could be spent on exploring different methods rather than establishing the programmatic framework itself. Much like the first suggestion in this subsection, this would require the right level of abstraction. Research would likely be needed to converge on the correct level of abstraction and the correct way to programmatically express it to users. A balance of abstraction and usability/ease of adoption is always a challenge and would require deep planning and expertise in the ROM research field as well as software engineering.

5.2.3 Theoretical improvements

These improvement ideas refer to ways to extend or improve upon the underlying ROM algorithms. The methods suggested would allow the deployment of NIDS in other types of nuclear engineering applications. In its current implementation, it is limited in its breadth of application potential. But with a few small improvements, its applicability widens drastically to include other classes of nuclear engineering problems.

1. Use SDF for core optimization.

The SDF construct allows the practitioner to embed geometric information into the ROM in ways that drastically speed up the accuracy and convergence of the neural networks. In the work presented herein, we saw how it was used to represent the distance to the nearest non-fuel element. One way this idea could be expanded is by instead using the SDF as a one-hot encoder to describe the type of fuel pin at each location. Once a training data database is created with enough variety in fuel pin loadout, the ROM could theoretically be used to test new fuel loading patterns. This type of analysis may be of interest for refueling load plans, core optimization, or other applications concerned with accurate 3D distributions of relative power or depletion isotopics.

The size of the training database and how far outside the training dataset the neural network could extrapolate would be the subject of a potential research project. Additionally, other ways of embedding the neighboring pin information as well as what type of pin the target pin is, for each fuel pin other than one-hot encoding, could be explored. It may be, like the natural language processing field, that other encoding forms exist which are much more efficient than one-hot encoding. It may also be that instead of encoding physical distances, encoding information about the mean free path distance to larger absorbers or reflector regions.

2. Explore other NIDS variants

In [114], Duvall et al. discuss other types of hypernetwork approaches that are related to the NIDS architecture. These include the DV-Hnet models. Whereas NIDS uses two neural networks to produce the weights and biases of a single-layered neural network, the DV-Hnet model uses two networks to produce the weights and biases of a multi-layered neural network. The authors saw much better performance from the DV-Hnet architecture than that for NIDS. The application of the DV-Hnet approach to nuclear engineering applications may further improve the performance of the application problems explored in this work. A future research project could be dedicated to applying this and other ROM methodologies aimed at creating surrogates for physics fields, which has become a very active field in the last few years.

3. Explore physics-informed loss terms

Other work has shown that including physics-derived terms in the loss function of neural networks can improve the accuracy and convergence speed. By embedding information specific to the neutron transport equation that constrains the weights and bias search space, coupled with the SDF construct to encode geometric information, future projects should be able to improve performance for the application problems discussed herein. Physics terms related to boundary conditions of the problem and neutron conservation across boundary conditions could help improve convergence time and accuracy at difficult spatial locations. Particularly, areas that are next to strong absorbers, near the reflector, or corner locations where two assemblies meet next to a reflector represent difficult areas to model neutronically. These areas are likely the most susceptible to a neural network-based ROM generalizing poorly, as they represent a small part of the dataset.

Similarly, in the depletion analysis certain isotopes behaved poorly compared to others. It may be that a loss term constructed intentionally to focus on certain isotopes would yield better results tailored to an application of interest. For example, if only fissile material, or only material related to high energy gamma decay modes concerning public health are of interest, then having a model that captures xenon-135 is of less importance. A loss term targeting these isotopes' performance may be favorable. Research that understands the best way to contribute to weight loss from certain isotopes would be beneficial.

4. Explore other nuclear engineering applications

All of the work presented here contains ideas related to the application of ROMs in critical core configuration analysis in nuclear engineering. There are other classes of problems that a nuclear design team may consider during the optimization and analysis of reactor cores. These include shielding applications, plant transient analysis, and spent fuel analysis. Other work (see [36] and [37]) has explored the application of ROMs in nuclear engineering in other contexts. A future research project could be dedicated to exploring these other application areas and finding solutions to the unique challenges present in these topics. These could include shielding applications, spent fuel criticality optimization, or real time nuclear simulations.

Shielding applications offer difficulties in that their quantity of interest (QOI)s vary by many orders of magnitudes and could require unique solutions to deal with this problem. Previous work has explored solutions to this problem using POD based approaches [36]. Critical configuration approaches involve potentially very large design spaces, and often incur large uncertainties to deal with the inability to explore them fully. Finally, real time nuclear simulations would have obvious benefits from being able to rely on higher fidelity neutronics solutions for certain casualty simulations, particularly asymmetric ones like a single rod ejection. Simplified neutronics methods are typically used in these applications, however ROMs could instead be used to reduce the amount of accuracy lost when finding models which perform fast enough for real time simulations.

For example, some analyses require looking at global and/or local reactivity coefficients. The application of full field ROMs, such as NIDS would only be as useful as their ability to predict spatially condensed quantities of interest, such as assembly-wise k-infinity values. Like the previous future work item, work involving honing on in the correct loss terms that represent impacts to spatially condensed quantities of interest could improve performance.

5. Explore other nuclear engineering applications

This work explored implementing a Bayesian neural networks (BNN) for ROM generation. Specifically, variational inference (VI) was used. This process could be explored more in depth to create a suite of benchmark dataset results that various BNN approaches could be applied to. Each could be tested against the dataset for calibration accuracy. In other words, quantifying how many of the true results fall within the model predicted ranges. Methods such as MCMC, model ensembles, or MC dropout could be compared and contrasted to one another for neutron transport applications.

All of these ideas represent areas that would further advance the field of nuclear engineering towards the stated goal of this work: the deployment of ROM methods in production-level nuclear engineering applications. To assist future projects that may use this work as a reference, the Appendix A documents the source code, models, and scripts used to generate the results herein for any future researcher or engineer looking to build on this work and push it further.

APPENDIX A

A. Supporting Code

This work represents the capabilities of multiple repositories, codes, and nuclear models created and maintained for this project. For reproducibility and to assist potential future work in this field, a description of each repository, where it can be found, and additional considerations are provided in Table A.1. For convenience, one Git repository, which can be found at git@github.com:bdlafleur/phd.git, contains all of these repositories as Git submodules. Refer to the tag "dissertation" in this repository to checkout the version of all repositories representing their state at the time of this issuance.

For future reproducibility, MC21 input files were generated with PUMA version 9.1.1-INL-b5 and were run with MC21 version 9.00.02-UNCL (Build 1) for all MC21 results here. MPACT version 2.1.0 (rev. 6) was used for all MPACT results in this document. For versions of all Python packages, see each repository's Conda environment Yaml definition files as well as any associated outputs from Python scripts/Jupyter Notebooks to obtain other Python package version constraints.

Code	Description
mc21-modeling (mc21-modeling.git)	Contains the 3D and 2D MC21 models (as Git submodules) and scripts used to generate MC21 input template folders, perform template substitution to create final MC21 input files, and facilitate input file submission to the Idaho National Laboratory (INL) high-performance computing (HPC)s. All results in Chapter 4 were generated using these models.
mc21-post (mc21-post.git)	Contains post processing scripts used to generate MC21 based ROM results in Chapter 4. This includes all scripts used to go from MC21 output files to the input arrays used as training datasets for neural network training, the neural network definitions themselves, and supplementary Jupyter Notebooks which summarize and test the MC21 supporting scripts.
mpact-modeling (mpact-modeling.git)	Contains instructions, templates, and submission scripts for Michigan Parallel Characteristics Transport Code (MPACT) models used to generate Chapter 3 nuclear model results.
mpact-post (mpact-post.git)	Contains bash submit scripts for MPACT related neural network training for Chapter 3. This repository contains two Jupyter Notebooks which produce all MPACT related result plots in this work - results_cnn.ipynb and results_nids.ipynb. It also contains parsing scripts to generate the input arrays from MPACT output files.
multi-stage CNN (multi-stage-cnn-rom.git)	Contains the TensorFlow scripts defining the multi-stage CNN from Section 2.2.3.4 and associated inputs which were used to generate results in Chapter 3.
Parody (parody.git)	Parody provides an abstract layer over ROM methodologies explored in this work. Specifically, it was used to generate all NIDS results herein. See Appendix B for more details on, and demo Jupyter Notebooks using, Parody. Parody was used to generate NIDS ROM results in Chapters 3 and 4.
Thesis (thesis.git)	This repository contains the LaTeX source code and associated images/supporting scripts for creating this document.

Table A.1: Programs and scripts used to create this work (repository name in parentheses). GitHub repositories found at git@github.com/bdlafleur/phd.git.

APPENDIX B

B. Parody: A Python Framework for ROM Research

B.1 Design Philosophy and User Interface

During the exploration of ROM methodologies, the Python tool Parody was created to improve efficiency, robustness, and reproducibility of results and ROM research. Parody provides an abstract layer over the ROM methodologies explored in this thesis. It is built on the Python package, PyTorch Lightning ([109]) (which is itself built on PyTorch), which allows for quick experimentation of machine learning algorithms. PyTorch Lightning was designed to be an extensible, lightweight, and high-performing framework that decouples research from coding implementation with the goal of making machine learning research more efficient. In a similar spirit, Parody was created to do the same for ROM research. Parody separates the research of ROM methodologies from the sometimes complicated implementation requirements of building a machine learning coding framework to demonstrate or implement ROM methods. This includes data manipulation and new-parameter predictions.

Because Parody was built on top of PyTorch Lightning, it experiences the same benefits of using PyTorch Lightning. Specifically, it leverages the Pytorch Lightning LightningModule and LightningDataModule constructs. Each of these assists in leveraging GPUs easily, optimizing batch sizes, hyperparameter tuning, keeping the data pipeline organized and reproducible, and other organizational benefits which reduce the programming burden of the machine learning practitioner. See the documentation for more details on the benefits gained from utilizing PyTorch Lightning ([109]).

Parody's interface is simple. The user should have prepared a collection of FOM results and the associated input parameters defining each FOM result. Some ROM algorithms require other classes of input, such as a collection of spatial locations for each FOM result, as required by the NIDS algorithm. Once these are prepared, they must be arranged according to the convention defined by Parody.

The FOM results must be in a Numpy array of the following shape: $(N_{cases}, t, x, ..., N_{st})$. N_{cases} is the number of FOM cases generated, t is the number of timesteps (if required), $x, ...,$ represents the number of dimensions in the x direction, with allowance for 2 or 3 dimensions as required, and N_{st} represents the number of state variables in the FOM result. Spatial location information must be in a Numpy array of the same shape as the FOM results, with the exception that the last dimension should be as large as the number of spatial dimensions in the problem. As an example, if there are 10 FOM realizations of a steady-state 2D problem discretized 100 by 100 in the x and y dimensions, with 5 state variables of interest, the state variable and spatial location arrays would have shapes of $(10, 100, 100, 5)$ and $(10, 100, 100, 2)$ respectively.

To define the parameterization of the FOM realizations, the parameters can be defined in two ways. The first would associate one set of input parameters with each FOM realization. In this case, the dimensionality would be (N_{cases}, N_p), where N_p is the number of parameters defining each realization. The second would be to associate each spatial location with its own set of unique parameters. In this case the dimensionality would be $(N_{cases}, x, ..., N_p)$. A problem must have one or both of these types of parameters associated with it. Additionally, in the case where the application problem is a transient Parody allows each time step, t, to have uniquely defined parameters. In this case one more dimension would be added to the parameter array. For example, if there are 10 FOM realizations of a transient with 20 timesteps for a 2D problem discretized 100 by 100 in the x and y dimensions, with 5 state variables of interest, the state variable and spatial location arrays would have shapes of $(10, 20, 100, 100, 5)$ and $(10, 100, 100, 2)$ respectively. The parameter array could have dimensions of either $(10, 20, N_p)$ or $(10, 20, 100, 100, N_p)$ depending on if the parameters are global inputs or spatially dependent.

B.2 Parody Design and Demo Toy Problems

Parody was designed to make exploring ROM frameworks simple. Each ROM is encapsulated in a package that contains the core ROM algorithm implemented with PyTorch (defined as an extension of the PyTorch Lightning LightningModule class), the definition of the forward pass and the data preprocessing utilities for the supported dimensions of that ROM. In addition to a class and utility script for each algorithm, a ROMDataModule class extends the Pytorch Lightning DataModule class and encapsulates the functionality required to transform the raw Numpy arrays into the format required for a given algorithm. Figure B.1 highlights these key components of Parody and shows two algorithms implemented for illustration. Note that other functionality and utility scripts are required; Figure B.1 is intended to highlight only the key components.

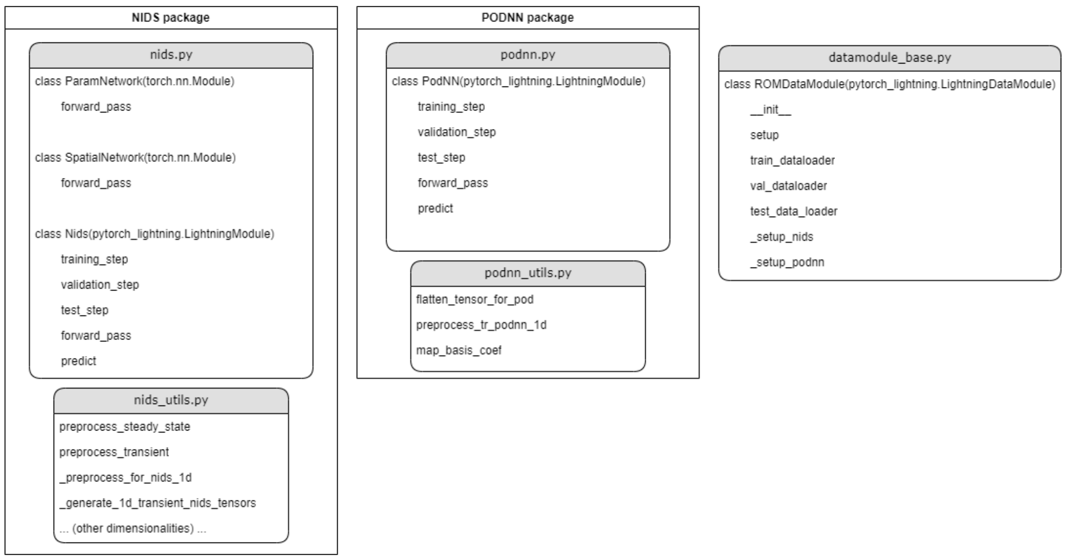


Figure B.1: Key components of Parody code infrastructure.

This framework allows any algorithm within Parody to be trained quickly and tested against a FOM dataset. It also allows new ROM algorithms to be quickly added, tested, and compared to other algorithms. To include a new algorithm, preprocessing scripts would be created that would take the standardized Numpy array format and reshape it into the expected shape by the new algorithm, and a PyTorch defined forward pass would be created. Finally, support for that algorithm in the ROMDataModule class would also be required. Overall, this process is lightweight and makes for quick testing and comparison of ROM algorithms.

To demonstrate the simple interface provided by Parody, toy problems were implemented within Parody for algorithm and functionality testing in the parody.demo_utils package. Each demo toy problem includes an interface that produces Numpy arrays in the format required for Parody. The following toy problems are included in Parody at the time of this writing:

- Burgers equation (1D transient)
- Multivariate Gaussian (2D steady-state)
- Sine wave (1D steady-state)

The Jupyter notebooks in the following sections illustrate how a user would invoke the demo tools within Parody to create the requisite Numpy arrays and interact with the different ROM Parody algorithms. These notebooks also fully define each of the toy problems.

nids_1d_transient

March 16, 2023

1 NIDS 1D transient (Burgers equation)

This notebook demonstrates non-linear independent dual systems (NIDS), a reduced order modeling (ROM) algorithm proposed by Duvall et al (2021) in https://arxiv.org/abs/2109.07018, for a 1D transient application.

[ ]:
```
import os

import matplotlib.pyplot as plt
import numpy as np
import pytorch_lightning as pl
import torch

from parody import datamodule_base
from parody.demo_utils import burgers
from parody.nids import nids
from parody.nids import nids_utils
from parody.utils import rom_utils

import warnings
warnings.filterwarnings("ignore", ".*does not have many workers.*")

# Select GPU for training if available
ACC = 'gpu' if torch.cuda.is_available() else 'cpu'

# Change nmber of epochs to 1 if running nbmake
NUM_EPOCHS = 200 if os.getenv('NBTEST') is None else 1
NUM_X = 1024 if os.getenv('NBTEST') is None else 100
NUM_TS = 101 if os.getenv('NBTEST') is None else 10
```

1.1 Problem description

This toy problem is the 1D transient Burgers equation, defined below.

$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{1}{Re}\frac{\partial^2 u}{\partial x^2}$

The first step with any ROM problem application is to generate an offline collection of full order model (FOM) results. This is often referred to as a `snapshot matrix`, or a collection of `state variables`.

To define a collection of FOM results, the equation above is assessed over the domains $X \in [0,1]$ and $t \in [0,1]$. This equation can be solved exactly to obtain an analytical formulation for the time evolution of the field via the following equation:

$u(x,t) = \frac{\frac{x}{t+1}}{1+\sqrt{\frac{t+1}{t_0}} exp(Re \frac{x2}{4t+4})}$

The FOM solutions are obtained upon 1024 equidistant spatial coordinates along X and 101 temporal coordinates along t. The snapshot matrix contains runs of 10 equally spaces Reynolds numbers, $Re \in [175, 225]$.

```
[ ]: inp_params = np.array(
         [(i,) for i in [50, 75, 100, 125, 150, 175, 200, 225, 250, 275]]
     )
     print('--- Training data design space --')
     print(f'{inp_params.shape = }')

     state_vars, params, locs = burgers.generate_data(inp_params, NUM_X, NUM_TS)

     print('\n--- Training datasets ---')

     state_vars = np.swapaxes(state_vars, 1, 2)
     print(f'{state_vars.shape = }')

     params = np.expand_dims(params, 1)
     params = np.repeat(params, NUM_TS, 1)
     param_t = np.linspace(0, 1, NUM_TS)
     param_t = np.expand_dims(param_t, -1)
     param_t = np.expand_dims(param_t, 0)
     param_t = np.repeat(param_t, params.shape[0], 0)
     params = np.concatenate((params, param_t), -1)
     print(f'{params.shape = }')

     print(f'{locs.shape = }')
```

```
--- Training data design space --
inp_params.shape = (10, 1)

--- Training datasets ---
state_vars.shape = (10, 101, 1024, 1)
params.shape = (10, 101, 2)
locs.shape = (10, 1024, 1)
```

1.2 NIDS model

The image below is taken from the original work in https://arxiv.org/abs/2109.07018. See that reference for full technical details. One way to conceptualize this approach is that it can be viewed as a nonlinear analogy to Proper Orthogonal Decomposition (POD). With NIDS, there are two networks. One for the parameters defining your problem and the other for the spacial coordinates. The output of the parameter network becomes the weights and biases of the final layer of the spatial network. POD likewise uses spatial modes combined with and basis coefficients which capture the dynamics of the linear combination of those spatial modes. However in POD, the spatial modes are determined using singular value decomposition.

There are many hyperparameters for this architecture. For instance, how may hidden nodes just before the final output layer, and the structure of the parameter and spatial networks. Each are encapsulated in the `parody.nids.nids.Nids` class.

The following code uses Parody to invoke the NIDS algorithm to create a ROM for the 2D Gaussian toy problem.

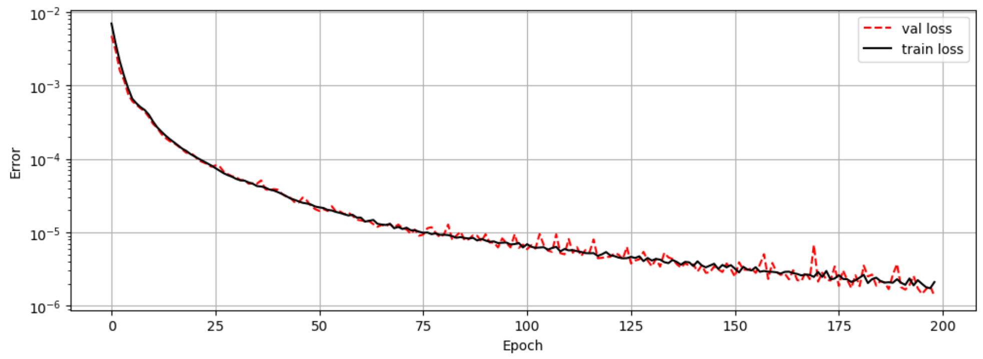
[ ]:
```
%%time

# Set hyperparameters for NIDS algorithm
config = {
    'transient': True,
    'param_dims': 2,
    'param_layers': 4,
    'param_units': 200,

    'latent_space': 100,

    'spat_dims': 1,
    'spat_layers': 4,
    'spat_units': 200,

    'output_dims': 1,
}

# Create NIDS model object and training dataset
model = nids.Nids(config)
datamodule = datamodule_base.ROMDataModule(
    model=model, locs=locs, state_vars=state_vars, params=params,
    batch_size=2**12, num_workers=1)

# Create callback
early_stopping = \
    pl.callbacks.early_stopping.EarlyStopping(
        monitor='val_loss', min_delta=0, patience=10, mode='min')
lr_monitor = pl.callbacks.LearningRateMonitor()
metrics = rom_utils.MetricTracker()

# Create pl trainer and fit
trainer = pl.Trainer(
    max_epochs=NUM_EPOCHS, accelerator=ACC, devices=1,
    callbacks=[lr_monitor, early_stopping, metrics])

trainer.fit(model=model, datamodule=datamodule)
```

```python
# Calculate performance metrics on test dataset
test_metrics = trainer.test(model=model, datamodule=datamodule)

# Plot training metrics
val_loss = [m.cpu() for m in metrics.collection['val_loss_epoch']]
train_loss = [m.cpu() for m in metrics.collection['train_loss_epoch']]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(12, 4))
axes.semilogy(val_loss[1:-1], '--r', label='val loss')
axes.semilogy(train_loss, '-k', label='train loss')
axes.legend()
axes.grid()
axes.set_ylabel('Error')
axes.set_xlabel('Epoch')
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1]

  | Name            | Type           | Params
---------------------------------------------------
0 | param_network   | ParamNetwork   | 141 K
1 | spatial_network | SpatialNetwork | 141 K
---------------------------------------------------
282 K     Trainable params
0         Non-trainable params
282 K     Total params
1.130     Total estimated model params size (MB)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1]

Testing: 0it [00:00, ?it/s]


       Test metric             DataLoader 0


           MAE            0.0011400313815101981
          RMSE            0.001571514061652124
```

```
CPU times: total: 17min 17s
Wall time: 29min 57s
```

```
[ ]: Text(0.5, 0, 'Epoch')
```



1.3 Model results

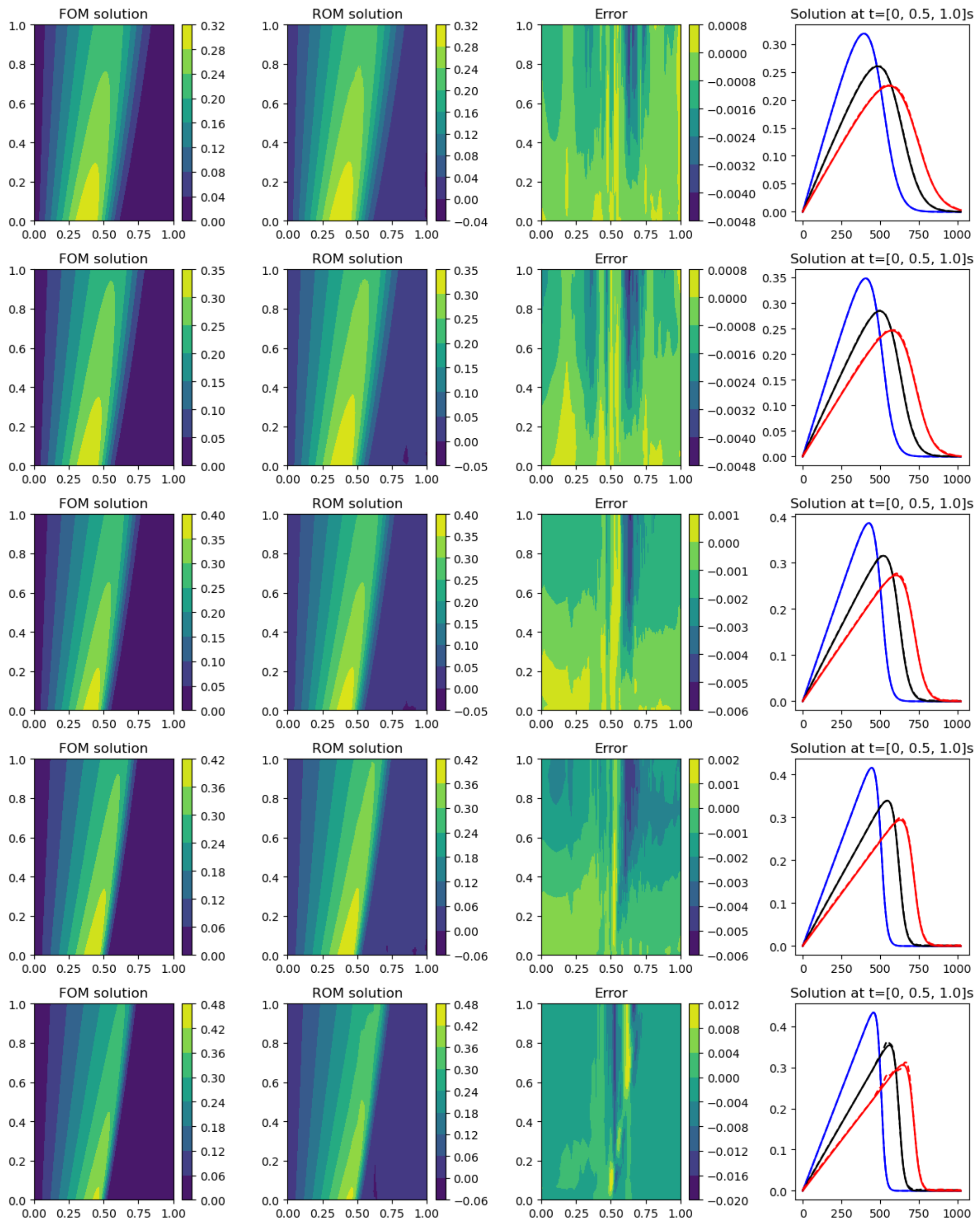
Shown below is the performance of the NIDS ROM over a sampling of Reynolds numbers which represent the design space captured by the training data. Reynolds numbers of 100, 150, and 225 are included (which are contained by the training data). Additional runs at Reynolds numbers of 75 and 300 are also included to illustrate performance of the ROM outside of the training data domain. In other words, the additional cases show that it extrapolates well, at least near the boundaries of the training data.

```
[ ]: ts = np.linspace(0, 1, NUM_TS)
     xs = np.linspace(0, 1, NUM_X)
     re_list = [75, 100, 150, 225, 300]
     fig, axes = plt.subplots(nrows=len(re_list), ncols=4, figsize=(12,␣
      ↪3*len(re_list)))

     for i, re in enumerate(re_list):
         # --- Get true solution
         u_true = np.zeros((NUM_X, NUM_TS))
         for j, ti in enumerate(ts):
             u_true[0:NUM_X, j] = burgers.get_soln_at_t(xs, ti, re)
         u_true = np.expand_dims(u_true.T, -1)

         # --- Get ROM solution
         u_init = np.array([[u_true[0, :, :]]])
         res = np.expand_dims(np.ones((len(ts), 1))*re, 0)
         param_t = np.expand_dims(np.expand_dims(ts, 0), -1)
         params = np.concatenate((res, param_t), -1)
```

5

```
    u_hat = model.clean_predict(
        datamodule, locs=locs, params=params, state_var_init=u_init)[0]
    # --- Plot results

    X, T = np.meshgrid(xs, param_t)
    surf = axes[i][0].contourf(X, T, u_true.squeeze(), antialiased=False)
    axes[i][0].set_title('FOM solution')
    plt.colorbar(surf, ax=axes[i][0])

    surf = axes[i][1].contourf(X, T, u_hat.squeeze(), antialiased=False)
    axes[i][1].set_title('ROM solution')
    plt.colorbar(surf, ax=axes[i][1])

    surf = axes[i][2].contourf(X, T, (u_true.squeeze() - u_hat.squeeze()),␣
  ↪antialiased=False)
    axes[i][2].set_title('Error')
    plt.colorbar(surf, ax=axes[i][2])

    axes[i][3].plot(u_true[0, :, 0], 'b', label='truth')
    axes[i][3].plot(u_hat[0, :, 0], '--b', label='pred')
    axes[i][3].plot(u_true[int(NUM_TS/2), :, 0], 'k', label='truth')
    axes[i][3].plot(u_hat[int(NUM_TS/2), :, 0], '--k', label='pred')
    axes[i][3].plot(u_true[-1, :, 0], 'r', label='truth')
    axes[i][3].plot(u_hat[-1, :, 0], '--r', label='pred')
    axes[i][3].set_title('Solution at t=[0, 0.5, 1.0]s')

fig.tight_layout()
```

```
100%|      | 100/100 [00:00<00:00, 112.36it/s]
100%|      | 100/100 [00:00<00:00, 117.51it/s]
100%|      | 100/100 [00:00<00:00, 114.16it/s]
100%|      | 100/100 [00:00<00:00, 108.87it/s]
100%|      | 100/100 [00:00<00:00, 119.69it/s]
```


FOM solution
ROM solution
Error
Solution at t=[0, 0.5, 1.0]s

nids_2d_steady_state

March 16, 2023

1 NIDS 2D steady state (multivariate gaussian)

This notebook demonstrates non-linear independent dual systems (NIDS), a reduced order modeling (ROM) algorithm proposed by Duvall et al (2021) in https://arxiv.org/abs/2109.07018, for a 2D steady state application.

[ ]:
```
import os
import random

import matplotlib.pyplot as plt
import numpy as np
import pytorch_lightning as pl
import torch

from parody import datamodule_base
from parody.demo_utils import gauss_2d
from parody.nids import nids
from parody.nids import nids_utils
from parody.utils import rom_utils

import warnings
warnings.filterwarnings("ignore", ".*does not have many workers.*")

# Select GPU for training if available
ACC = 'gpu' if torch.cuda.is_available() else 'cpu'

# Change nmber of epochs to 1 if running nbmake
NUM_EPOCHS = 50 if os.getenv('NBTEST') is None else 1
```

1.1 Problem description

This toy problem is a 2D steady state multivariate gaussian, defined below.

$z = \frac{1}{2\pi\sigma_x\sigma_y \times exp(\frac{x2}{2\sigma_x^2} + \frac{y2}{2\sigma_y^2})}$

The first step with any ROM problem application is to generate an offline collection of full order model (FOM) results. This is often referred to as a `snapshot matrix`, or a collection of `state variables`.

To define the collection of FOM results, σ_x and σ_y are sampled randomly from 1 to 4, and the above equation is evaluated on a 10×10 grid discretized by 100 points in each direction, with $X \in [-10, 10]$ and $Y \in [-10, 10]$. The input parameter, μ, is defined as a tuple of σ_x and σ_y. The training dataset consists of 50 sets of μ.

In the NIDS framework, the training set consists of $50 \times (100 \times 100) = 500,000$ training data points, where each data point consists of the x and y position, σ_x and σ_y, and the corresponding z value associated with those inputs.

```
[ ]: inp_params = np.array(
         [(random.random()*3+1, random.random()*3+1) for _ in range(50)]
     )
     print('--- Training data design space ---')
     print(f'{inp_params.shape = }')

     state_vars, locs, params = gauss_2d.generate_data(inp_params)

     print('\n--- Training datasets ---')
     print(f'{state_vars.shape = }')
     print(f'{locs.shape = }')
     print(f'{params.shape = }')
```

```
--- Training data design space ---
inp_params.shape = (50, 2)

--- Training datasets ---
state_vars.shape = (50, 100, 100, 1)
locs.shape = (50, 100, 100, 2)
params.shape = (50, 2)
```

1.2 NIDS model

The image below is taken from the original work in https://arxiv.org/abs/2109.07018. See that reference for full technical details. One way to conceptualize this approach is that it can be viewed as a nonlinear analogy to Proper Orthogonal Decomposition (POD). With NIDS, there are two networks. One for the parameters defining your problem and the other for the spacial coordinates. The output of the parameter network becomes the weights and biases of the final layer of the spatial network. POD likewise uses spatial modes combined with and basis coefficients which capture the dynamics of the linear combination of those spatial modes. However in POD, the spatial modes are determined using singular value decomposition.

There are many hyperparameters for this architecture. For instance, how may hidden nodes just before the final output layer, and the structure of the parameter and spatial networks. Each are encapsulated in the `parody.nids.nids.Nids` class.

The following code uses Parody to invoke the NIDS algorithm to create a ROM for the 2D Gaussian toy problem.

```
[ ]: %%time
```

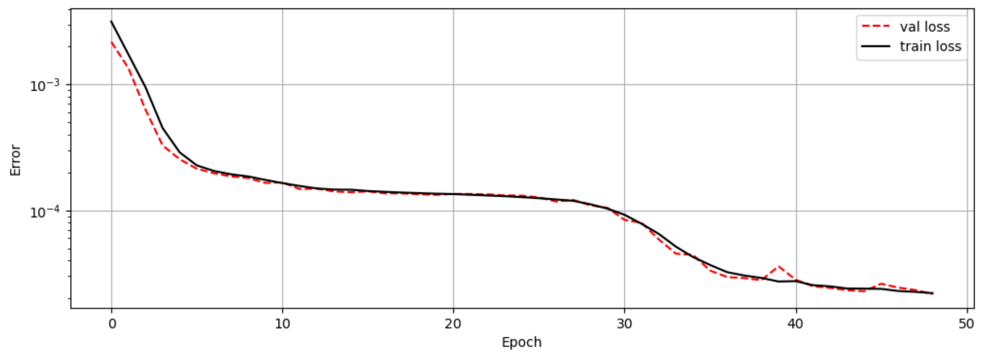
```python
# Set hyperparameters for NIDS algorithm
config = {
    'transient': False,
    'param_dims': 2,
    'param_layers': 4,
    'param_units': 100,

    'latent_space': 25,

    'spat_dims': 2,
    'spat_layers': 4,
    'spat_units': 100,

    'output_dims': 1,
}

# Create NIDS model object and training dataset
model = nids.Nids(config)
datamodule = datamodule_base.ROMDataModule(
    model=model, state_vars=state_vars, locs=locs, params=params,
    batch_size=2048, num_workers=1)

# Create callback
early_stopping = \
    pl.callbacks.early_stopping.EarlyStopping(
        monitor='val_loss', min_delta=0, patience=5, mode='min')
lr_monitor = pl.callbacks.LearningRateMonitor()
metrics = rom_utils.MetricTracker()

# Create pl trainer and fit
trainer = pl.Trainer(
    max_epochs=NUM_EPOCHS, accelerator=ACC, devices=1,
    callbacks=[lr_monitor, early_stopping, metrics])
trainer.fit(model=model, datamodule=datamodule)

# Calculate performance metrics on test dataset
test_metrics = trainer.test(model=model, datamodule=datamodule)

# Plot training metrics
val_loss = [m.cpu() for m in metrics.collection['val_loss_epoch']]
train_loss = [m.cpu() for m in metrics.collection['train_loss_epoch']]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(12, 4))
axes.semilogy(val_loss[1:-1], '--r', label='val loss')
axes.semilogy(train_loss, '-k', label='train loss')
axes.legend()
axes.grid()
axes.set_ylabel('Error')
```

3

```
axes.set_xlabel('Epoch')
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1]

  | Name            | Type           | Params
---------------------------------------------------
0 | param_network   | ParamNetwork   | 33.2 K
1 | spatial_network | SpatialNetwork | 33.1 K
---------------------------------------------------
66.4 K    Trainable params
0         Non-trainable params
66.4 K    Total params
0.265     Total estimated model params size (MB)

Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1]

Testing: 0it [00:00, ?it/s]


       Test metric             DataLoader 0


           MAE             0.003011499997228384
          RMSE              0.00483113806694746


CPU times: total: 3min 35s
Wall time: 4min 22s
```

```
[ ]: Text(0.5, 0, 'Epoch')
```



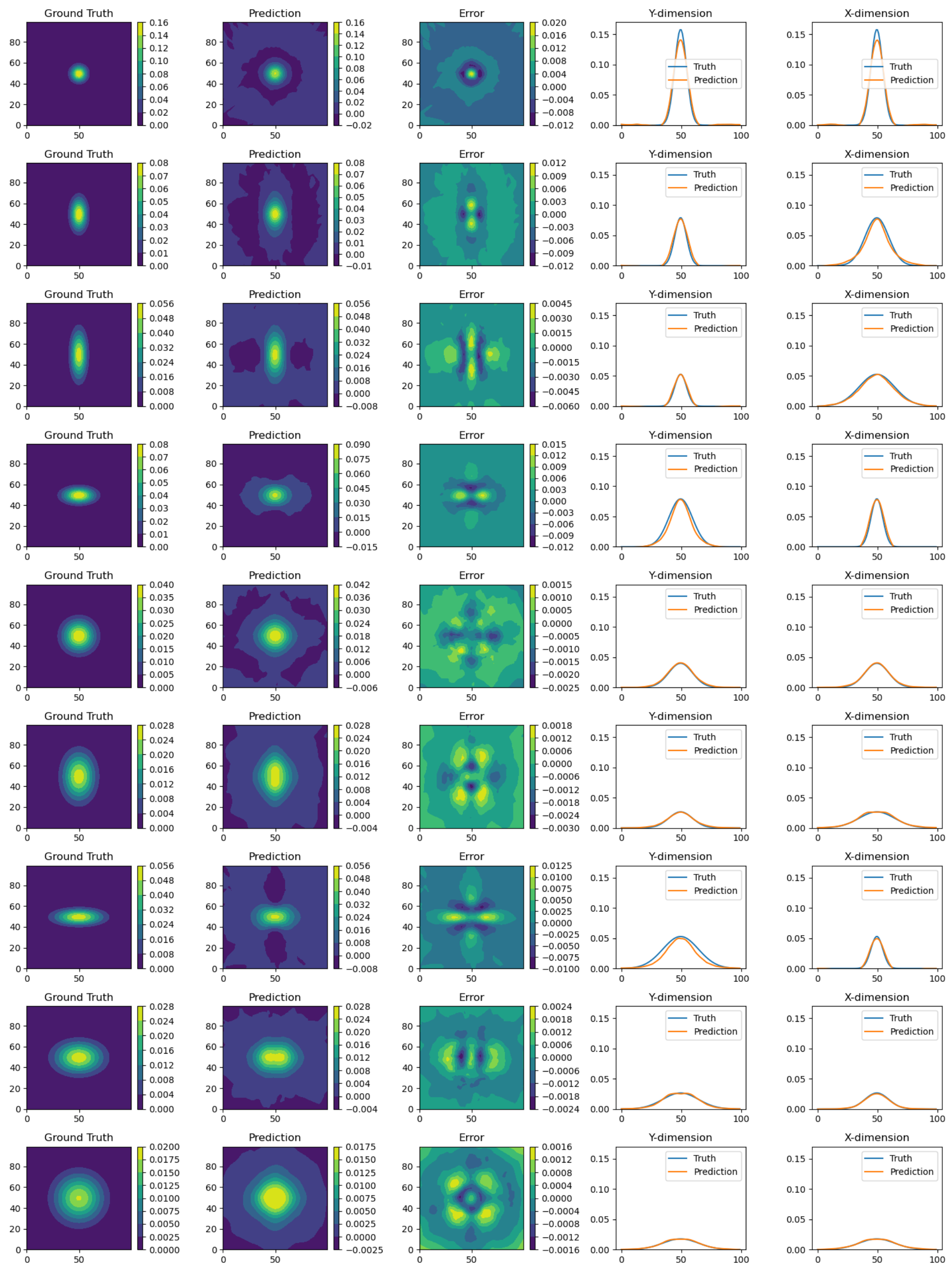
1.3 Model results

Shown below is the performance of the NIDS ROM over a sampling of μ which represents the design space captured by the training data. In the plots below, each row represents a new μ case. Across each row are the 2D images of the FOM results, the NIDS ROM predictions, and the associated errors. The two line plots are slices through the middle of the X and Y dimensions.

[ ]:
```
# Setup results for a sweep through input space
test_inp_params = np.array([
    [1, 1], [1, 2], [1, 3],
    [2, 1], [2, 2], [2, 3],
    [3, 1], [3, 2], [3, 3],
])

i = 0
fig, axes = plt.subplots(nrows=9, ncols=5, figsize=(15, 20))
axes = axes.flatten()
for ax, case in zip(axes, test_inp_params):

    # Generate truth.
    state_vars, locs, params = \
        gauss_2d.generate_data(np.array([case]))

    u_hats = model.clean_predict(
        datamodule, locs=locs, params=params
    )
    u_hat = u_hats.squeeze()

    # Format pretty plots
    state_var = np.reshape(state_vars, (100, 100))
    ind = [i*5+j for j in range(5)]
    cm0 = axes[ind[0]].contourf(state_var)
    axes[ind[0]].set_title('Ground Truth')
```

5

```
    fig.colorbar(cm0, ax=axes[ind[0]])

    cm1 = axes[ind[1]].contourf(u_hat)
    axes[ind[1]].set_title('Prediction')
    fig.colorbar(cm1, ax=axes[ind[1]])

    cm2 = axes[ind[2]].contourf((state_var-u_hat))
    axes[ind[2]].set_title('Error')
    fig.colorbar(cm2, ax=axes[ind[2]])
    fig.tight_layout()

    axes[ind[3]].plot(state_var[50, :], label='Truth')
    axes[ind[3]].plot(u_hat[50, :], label='Prediction')
    axes[ind[3]].legend()
    axes[ind[3]].set_ylim((0, 0.17))
    axes[ind[3]].set_title('Y-dimension')

    axes[ind[4]].plot(state_var[:, 50], label='Truth')
    axes[ind[4]].plot(u_hat[:, 50], label='Prediction')
    axes[ind[4]].legend()
    axes[ind[4]].set_ylim((0, 0.17))
    axes[ind[4]].set_title('X-dimension')
    i = i + 1

fig.tight_layout()
```


Ground Truth
Prediction
Error
Y-dimension
X-dimension
Truth
Prediction

podnn_1d_transient

October 11, 2023

1 PODNN 1D transient (Burgers equation)

This notebook demonstrates Proper Orthogonal Decomposition via neural network (PODNN), a non-intrusive reduced order modeling (ROM) algorithm applied to a 1D transient application. It closely follows the method outlined in https://www.sciencedirect.com/science/article/abs/pii/S1007570419301364?via%3Dihub for a 1D transient application.

[ ]:
```
import os

import matplotlib.pyplot as plt
import numpy as np
import pytorch_lightning as pl
import torch

from parody import datamodule_base
from parody.demo_utils import burgers
from parody.podnn import podnn
from parody.podnn import podnn_utils
from parody.utils import rom_utils

import warnings
warnings.filterwarnings("ignore", ".*does not have many workers.*")

# Select GPU for training if available
ACC = 'gpu' if torch.cuda.is_available() else 'cpu'

# Change nmber of epochs to 1 if running nbmake
NUM_EPOCHS = 100 if os.getenv('NBTEST') is None else 1
NUM_X = 1024 if os.getenv('NBTEST') is None else 100
NUM_TS = 101 if os.getenv('NBTEST') is None else 10
```

1.1 Problem description

This toy problem is the 1D transient Burgers equation, defined below.

$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{1}{Re}\frac{\partial^2 u}{\partial x^2}$

The first step with any ROM problem application is to generate an offline collection of full order model (FOM) results. This is often referred to as a `snapshot matrix`, or a collection of `state variables`.

To define a collection of FOM results, the equation above is assessed over the domains $X \in [0, 1]$ and $t \in [0, 1]$. This equation can be solved exactly to obtain an analytical formulation for the time evolution of the field via the following equation:

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

The FOM solutions are obtained upon 1024 equidistant spatial coordinates along X and 101 temporal coordinates along t. The snapshot matrix contains runs of 10 equally spaces Reynolds numbers, $Re \in [175, 225]$.

[ ]:
```
inp_params = np.array(
    [(i,) for i in [50, 75, 100, 125, 150, 175, 200, 225, 250, 275]]
)
print('--- Training data design space --')
print(f'{inp_params.shape = }')

state_vars, params, _ = burgers.generate_data(inp_params, NUM_X, NUM_TS)

print('\n--- Training datasets ---')
print(f'{state_vars.shape = }')
print(f'{params.shape = }')
```

```
--- Training data design space --
inp_params.shape = (10, 1)

--- Training datasets ---
state_vars.shape = (10, 1024, 101, 1)
params.shape = (10, 1)
```

1.2 PODNN model

The image below is taken from the original work in [11]. See that reference for full technical details. This method can be viewed as a non-intrusive, neural network based implementation of POD. With vanilla POD, the spatial modes are computed via singular value decomposition (SVD). It is assumed that the state vector of interest can be computed as a time-evolving linear combination of these modes. The FOM equations are recast using these assumptions, and the time dynamics of the spatial modes are computed (sometimes referred to as basis coefficients). In PODNN, the spatial modes are still computed via SVD, however a simple dense neural network is used to describe their dynamics. The image below shows this idea.

Notice that the timestep and the Reynolds number are used as input to the network.

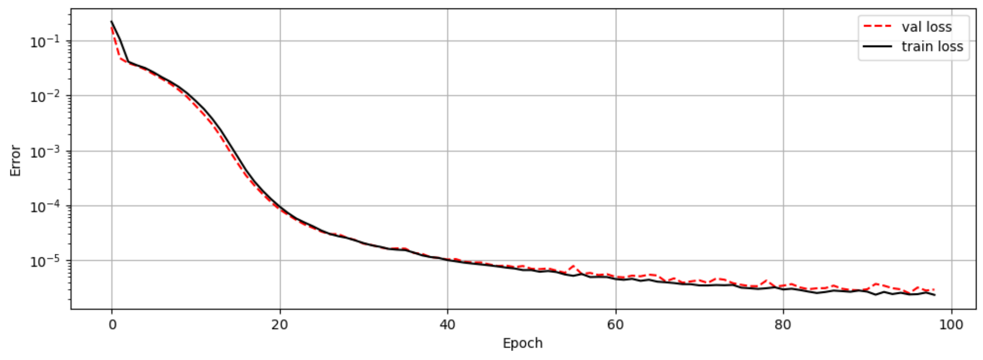
The key hyperparameters of this algorithm include the architecture of the neural network as well as how many spatial modes to keep from the SVD. These are encapsulated in the `parody.podnn.podnn.PodNN` class.

```
[ ]: %%time

# Set hyperparameters for PODNN algorithm
config = {
    'transient': True,
    'modes': 8,
    'hidden_units': 300,
    'layers': 4,
    'dimensions': params.shape[1],
    'cases': len(state_vars),
}

# Create PODNN model object and training dataset
model = podnn.PodNN(config)
datamodule = datamodule_base.ROMDataModule(
    model=model, state_vars=state_vars, params=params,
    batch_size=32, num_workers=1)

# Create callback
early_stopping = \
    pl.callbacks.early_stopping.EarlyStopping(
        monitor='val_loss', min_delta=0, patience=15, mode='min')
lr_monitor = pl.callbacks.LearningRateMonitor()
metrics = rom_utils.MetricTracker()

# Create pl trainer and fit
trainer = pl.Trainer(
    max_epochs=NUM_EPOCHS, accelerator=ACC, devices=1,
    callbacks=[lr_monitor, early_stopping, metrics])
trainer.fit(model=model, datamodule=datamodule)

# Calculate performance metrics on test dataset
test_metrics = trainer.test(model=model, datamodule=datamodule)

# Plot training metrics
val_loss = [m.cpu() for m in metrics.collection['val_loss_epoch']]
train_loss = [m.cpu() for m in metrics.collection['train_loss_epoch']]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(12, 4))
axes.semilogy(val_loss[1:-1], '--r', label='val loss')
axes.semilogy(train_loss, '-k', label='train loss')
axes.legend()
axes.grid()
axes.set_ylabel('Error')
axes.set_xlabel('Epoch')
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
```

```
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1]

  | Name        | Type       | Params
-------------------------------------------
0 | _input      | Linear     | 3.0 K
1 | _layer_list | ModuleList | 273 K
-------------------------------------------
276 K     Trainable params
0         Non-trainable params
276 K     Total params
1.105     Total estimated model params size (MB)

Sanity Checking: 0it [00:00, ?it/s]

c:\Users\brand\anaconda3\envs\phd\lib\site-
packages\pytorch_lightning\trainer\trainer.py:1933: PossibleUserWarning: The
number of training batches (24) is smaller than the logging interval
Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you
want to see logs for the training epoch.
  rank_zero_warn(

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1]

Testing: 0it [00:00, ?it/s]


       Test metric             DataLoader 0


           MAE            0.0011266579385846853
          RMSE            0.001462490065023303


CPU times: total: 34.1 s
Wall time: 3min 38s
```

```
[ ]: Text(0.5, 0, 'Epoch')
```



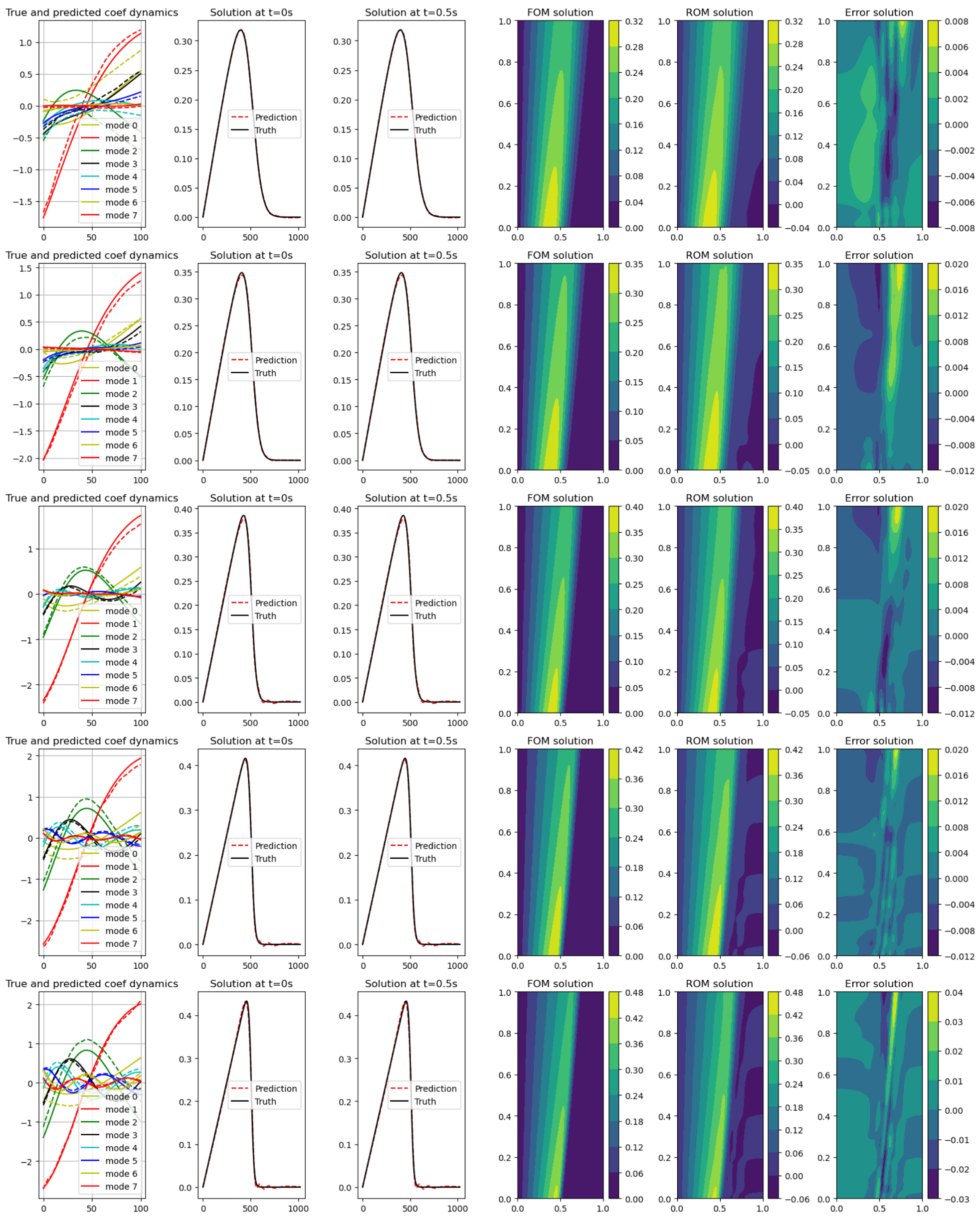
1.3 Model results

Shown below is a visualization of a test case at a Reynolds number of 200. Also shown are the true coefficient dynamics and the predicted coefficient dynamics for this case.

```
[ ]: test_re_list = [75, 100, 150, 225, 300]
     xs = np.linspace(0, 1, NUM_X)
     ts = np.linspace(0, 1, NUM_TS)
     fig, axes = plt.subplots(nrows=len(test_re_list), ncols=6, figsize=(16,␣
      ↪4*len(test_re_list)))

     # Setup results for single Reynolds number
     # Make true solution and get true coefs
     for i, test_re in enumerate(test_re_list):
         test_inp_params = np.array([(test_re,)])
         test_state_vars, test_params, _= \
             burgers.generate_data(test_inp_params, NUM_X, NUM_TS)

         # --- Get true coefficient dynamics
         pod_state_vars = podnn_utils.flatten_for_pod(
             test_state_vars, test_params)
         u_bar = np.mean(pod_state_vars[0:NUM_X], 1)
         coefs_true = np.zeros((config['modes'], NUM_TS))
         for j in range(0, NUM_TS):
             u_n = test_state_vars[0, :, j]
             # prjnt(f'{datamodule.modes.shape = }')
             a_n = np.dot(u_n[:, 0] - u_bar, datamodule.modes)
             # prjnt(f'{u_n.shape = }')
             # prjnt(f'{a_n.shape = }')
             coefs_true[:, j] = a_n

         # Compute ROM solution and ROM coefs
         init_cond = np.expand_dims(test_state_vars[:, :, 0, :], 2)
```

5

```
    u_hat, coefs_hat = model.clean_predict(
        datamodule=datamodule, params=test_inp_params, num_ts=NUM_TS,
        state_var_init=init_cond)

    # Plot metrics to confirm algorithm is working
    sty = {0: 'y', 1: 'r', 2: 'g', 3: 'k', 4: 'c', 5: 'b', 6: 'y', 7: 'r'}
    for j in range(0, coefs_hat.shape[0],):
        axes[i][0].plot(coefs_true[j], sty[j], label=f'mode {j}')
        axes[i][0].plot(coefs_hat[j], f'{sty[j]}--')
    axes[i][0].grid()
    axes[i][0].legend()
    axes[i][0].set_title('True and predicted coef dynamics')

    axes[i][1].plot(u_hat[:, 0], 'r--', label='Prediction')
    axes[i][1].plot(pod_state_vars[0:len(xs), 0:NUM_TS][:, 0], 'k',␣
↪label='Truth')
    axes[i][1].set_title('Solution at t=0s')
    axes[i][1].legend()
    axes[i][2].plot(u_hat[:, 0], 'r--', label='Prediction')
    axes[i][2].plot(pod_state_vars[0:len(xs), 0:NUM_TS][:, 0], 'k',␣
↪label='Truth')
    axes[i][2].set_title('Solution at t=0.5s')
    axes[i][2].legend()

    X, T = np.meshgrid(xs, ts)
    surf = axes[i][3].contourf(X, T, test_state_vars[0, :, :, 0].T,␣
↪antialiased=False)
    axes[i][3].set_title('FOM solution')
    plt.colorbar(surf, ax=axes[i][3])

    surf = axes[i][4].contourf(X, T, u_hat.T, antialiased=False)
    axes[i][4].set_title('ROM solution')
    plt.colorbar(surf, ax=axes[i][4])

    surf = axes[i][5].contourf(X, T, (test_state_vars[0, :, :, 0].T - u_hat.T),␣
↪antialiased=False)
    axes[i][5].set_title('Error solution')
    plt.colorbar(surf, ax=axes[i][5])
fig.tight_layout()
```


True and predicted coef dynamics
Solution at t=0s
Solution at t=0.5s
FOM solution
ROM solution
Error solution
mode 0
mode 1
mode 2
mode 3
mode 4
mode 5
mode 6
mode 7
Prediction
Truth

7

APPENDIX C

C. Data Science Tools and Best Practices

C.1 General Neural Network Training

Because many neural networks were trained to assess the viability of ROM methodologies in representative nuclear engineering applications, a brief discussion of some best practices that were followed is warranted. These best practices come from common approaches found in the data science industry as well as lessons learned from performing this work. The general process of converging on an architecture was the same for all neural networks.

1. The general architecture for any neural network is found using trial and error, beginning with any available previous work's architecture. A condensed dataset is always used to facilitate a quick turnaround. Typically, about 5% of the available data is used during this step.

2. Once an architecture was found that could capture the general dynamics of the problem, the data set is expanded to confirm that the network could still capture these dynamics when including all input feature variability of the problem.

3. A hyperparameter optimization is performed using a subset of the hyperparameter sets. In this first hyperparameter stage, only those parameters which have large impacts on training trajectories are explored. This includes hyperparameters such as learning rate, regularization options, and loss functions.

4. Once the primary hyperparameters are optimized, a second hyperparameter optimization is performed using the remaining architecture detail hyperparameters. This includes hyperparameters such as number of layers in a dense network, the number of filters in a convolutional layer, or the number of latent spaces in a NIDS model. For the CNN model, this step was usually divided into multiple stages of optimization due to the high number of hyperparameters.

Choi [115] and Simard [116] each provide a write up summarizing some of the best practices for neural network training, with Simard focusing on convolutional neural networks. During training, the standard data science practice of splitting input data sets into training, validation, and test data sets is utilized. The training data set is the data set that informs the gradient updates during training. The validation data set is used during the analysis iterations to fine-tune the hyperparameters. It does not directly inform gradient updates during training, so it represents a first test of how the model may perform with previously unseen data. However, it does represent some information leakage into the model as it impacts which hyperparameters are used; therefore, it does not perfectly capture how well the model has generalized. In this context information leakage refers to the process by which information from datasets not intended to impact the parameter updates informs how these parameters are trained. Put another way, if the practitioner is making architectural decisions based on the validation dataset, then the final model parameters are being influenced based on decision influenced by the validation dataset. Finally, The test data set is run only at the end of the process, as it is the best representation of how the model will perform against unseen data. By restricting this dataset until the last step, information leakage is prevented and the best estimate of the models capability on unseen data is observed.

In the process described above for architecture and hyperparameter search, Step 3 usually sees greater variability in performance during training. Step 4 typically does not observe significant performance improvements during execution outside of marginal improvements. For the context and reproducibility of the studies described throughout this work, each neural network was locally trained on a single NVIDIA-3090 GPU.

C.2 Data Science Tools

For hyperparameter tuning, multiple hyperparameter frameworks, such as Optuna and Keras-Tuner, were tried. However, ultimately Ray-Tune was used ([117]). Ray-Tune is a Python library for data science experiment execution and hyperparameter tuning. It was chosen because it supports all major machine learning frameworks, including PyTorch and Keras (the two frameworks used in this work). Other hyperparameter frameworks can also be configured to support these frameworks, however it is the opinion of the author that Ray-Tune provided the most approachable interface with convenient features to integrate hyperparameter tuning into the training process. Ray-Tune contains within it the full implementations of many major hyperparameter optimization algorithms. See [118] for a review of common algorithms used today, or the Ray-Tune documentation available on their GitHub page. Method classes such as grid search, random search, Bayesian optimization, and its derivatives are available.

A note on best practices: Due to the complexity of dealing with a multistage data pipeline, the training of many separate neural networks for the results herein, and generating plots and metrics consistent between each analysis iteration, the tool Data Version Control (DVC) was used ([119]) and is recommended for similar work. DVC is a data science tool that takes advantage of existing software engineering tools such as Git, VS Code, and cloud storage. It helps machine learning projects manage large datasets, make projects reproducible, and improve collaboration. Specifically, in this work, the pipelining and data versioning features of the data from DVC were used.

DVC allows for arbitrary definitions of a directed acyclic graph (DAG) for defining and re-running machine learning pipelines. The number of models trained using similar source files with different inputs, and the desire for reproducible results, made DVC and its declarative DAG yaml syntax very useful to form the infrastructure of this analysis. Although DVC is advertised as a facilitator of collaboration in large-scale machine learning projects involving numerous participants, as a best practice, this author recommends exploring DVC for small, solo projects as well.

Another best practice employed in this work was the use of mixed precision for all neural network training. PyTorch Lightning [109], which Parody was built on top of, offers 16-bit (and lower) floating-point training. This enables the training and deployment of larger neural networks than would otherwise be possible because they require less memory and enhance data transfer operations. This essentially allows much faster operations on GPUs than full-precision training. Mixed precision combines the use of both 32- and 16-bit floating point operations to reduce memory footprints during training. All training for Parody-based models (i.e., not the CNN based ROMs) used 16-bit mixed precision during training. Negligible degradation in model performance was seen in addition to a 2X speed-up, allowing for a faster turnaround time during experimentation. See [120] for an introduction to the use of mixed precision training in neural networks.

BIBLIOGRAPHY

[1] R. Sanchez and N. J. McCormick. "A Review of Neutron Transport Approximations". In: *Nuclear Science and Engineering* 80.4 (Apr. 1, 1982). Publisher: Taylor & Francis _eprint: https://doi.org/10.13182/NSE80-04-481, pp. 481–535. ISSN: 0029-5639. DOI: `10.13182/NSE80-04-481`. URL: `https://doi.org/10.13182/NSE80-04-481` (visited on 04/24/2023).

[2] Edward Larsen et al. "Advances in Discrete-Ordinates Methodology". In: *Nuclear Computational Science: A Century in Review*. Journal Abbreviation: Nuclear Computational Science: A Century in Review. Apr. 15, 2010, pp. 1–84. ISBN: 978-90-481-3410-6. DOI: `10.1007/978-90-481-3411-3_1`.

[3] R. D. Lawrence. "Progress in nodal methods for the solution of the neutron diffusion and transport equations". In: *Progress in Nuclear Energy* 17.3 (Jan. 1, 1986), pp. 271–301. ISSN: 0149-1970. DOI: `10.1016/0149-1970(86)90034-X`. URL: `https://www.sciencedirect.com/science/article/pii/014919708690034X` (visited on 04/22/2023).

[4] Nam Zin Cho. "Fundamentals and recent developments of reactor physics methods". In: *Nuclear Engineering and Technology* 37.1 (2005). Place: Korea, Republic of INIS Reference Number: 37056374, pp. 25–78. ISSN: 1738-5733.

[5] Peter Benner, Serkan Gugercin, and Karen Willcox. *A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems — SIAM Review*. 2015. URL: `https://epubs.siam.org/doi/10.1137/130932715` (visited on 04/18/2023).

[6] Muruhan Rathinam and Linda Petzold. "A New Look at Proper Orthogonal Decomposition". In: *SIAM J. Numerical Analysis* 41 (Jan. 1, 2003), pp. 1893–1925. DOI: `10.1137/S0036142901389049`.

[7] K. Willcox and J. Peraire. "Balanced Model Reduction via the Proper Orthogonal Decomposition". In: *Aiaa Journal - AIAA J* 40 (Nov. 1, 2002), pp. 2323–2330. DOI: `10.2514/2.1570`.

[8] Kevin Carlberg, Matthew Barone, and Harbir Antil. "Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction". In: *Journal of Computational Physics* 330 (Feb. 1, 2017), pp. 693–734. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2016.10.033`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999116305319` (visited on 04/18/2023).

[9] Eric J. Parish, Christopher R. Wentland, and Karthik Duraisamy. "The Adjoint Petrov–Galerkin method for non-linear model reduction". In: *Computer Methods in Applied Mechanics and Engineering* 365 (June 15, 2020), p. 112991. ISSN: 0045-7825. DOI: 10.1016/j.cma.2020.112991. URL: https://www.sciencedirect.com/science/article/pii/S0045782520301754 (visited on 04/16/2023).

[10] F. Ghavamian, P. Tiso, and A. Simone. "POD–DEIM model order reduction for strain-softening viscoplasticity". In: *Computer Methods in Applied Mechanics and Engineering* 317 (Apr. 15, 2017), pp. 458–479. ISSN: 0045-7825. DOI: 10.1016/j.cma.2016.11.025. URL: https://www.sciencedirect.com/science/article/pii/S0045782516304054 (visited on 04/16/2023).

[11] Kookjin Lee and Kevin T. Carlberg. "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders". In: *Journal of Computational Physics* 404 (Mar. 1, 2020), p. 108973. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.108973. URL: https://www.sciencedirect.com/science/article/pii/S0021999119306783 (visited on 04/18/2023).

[12] Omer San, Romit Maulik, and Mansoor Ahmed. "An artificial neural network framework for reduced order modeling of transient flows". In: *Communications in Nonlinear Science and Numerical Simulation* 77 (Oct. 1, 2019), pp. 271–287. ISSN: 1007-5704. DOI: 10.1016/j.cnsns.2019.04.025. URL: https://www.sciencedirect.com/science/article/pii/S1007570419301364 (visited on 04/18/2023).

[13] Omer San and Romit Maulik. "Neural network closures for nonlinear model order reduction". In: *Advances in Computational Mathematics* 44 (Dec. 1, 2018). DOI: 10.1007/s10444-018-9590-z.

[14] J. S. Hesthaven and S. Ubbiali. "Non-intrusive reduced order modeling of nonlinear problems using neural networks". In: *Journal of Computational Physics* 363 (June 15, 2018), pp. 55–78. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.02.037. URL: https://www.sciencedirect.com/science/article/pii/S0021999118301190 (visited on 04/18/2023).

[15] Jiayang Xu and Karthik Duraisamy. "Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics". In: *Computer Methods in Applied Mechanics and Engineering* 372 (Dec. 1, 2020), p. 113379. ISSN: 0045-7825. DOI: 10.1016/j.cma.2020.113379. URL: https://www.sciencedirect.com/science/article/pii/S0045782520305648 (visited on 04/18/2023).

[16] Saakaar Bhatnagar et al. "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics* 64.2 (Aug. 1, 2019), pp. 525–545. ISSN: 1432-0924. DOI: 10.1007/s00466-019-01740-0. URL: https://doi.org/10.1007/s00466-019-01740-0 (visited on 04/18/2023).

[17] Xiaoxiao Guo, Wei Li, and Francesco Iorio. "Convolutional Neural Networks for Steady Flow Approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 13, 2016, pp. 481–490. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939738. URL: https://doi.org/10.1145/2939672.2939738 (visited on 04/18/2023).

[18] Kazuto Hasegawa et al. "Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes". In: *Theoretical and Computational Fluid Dynamics* 34 (Aug. 1, 2020). DOI: 10.1007/s00162-020-00528-w.

[19] Tianping Chen and Hong Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its applications to dynamic systems". In: *Neural Networks, IEEE Transactions on* (Aug. 1, 1995), pp. 911–917. DOI: 10.1109/72.392253.

[20] Lu Lu et al. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3 (Mar. 1, 2021), pp. 218–229. DOI: 10.1038/s42256-021-00302-5.

[21] Zongyi Li et al. *Multipole Graph Neural Operator for Parametric Partial Differential Equations*. Oct. 19, 2020. DOI: 10.48550/arXiv.2006.09535. arXiv: 2006.09535[cs,math,stat]. URL: http://arxiv.org/abs/2006.09535 (visited on 04/18/2023).

[22] Sifan Wang, Hanwen Wang, and Paris Perdikaris. "Learning the solution operator of parametric partial differential equations with physics-informed DeepONets". In: *Science Advances* 7.40 (Sept. 29, 2021). Publisher: American Association for the Advancement of Science, eabi8605. DOI: 10.1126/sciadv.abi8605. URL: https://www.science.org/doi/full/10.1126/sciadv.abi8605 (visited on 04/19/2023).

[23] Shengze Cai et al. "DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks". In: *Journal of Computational Physics* 436 (July 1, 2021), p. 110296. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2021.110296. URL: https://www.sciencedirect.com/science/article/pii/S0021999121001911 (visited on 04/19/2023).

[24] Zongyi Li et al. *Neural Operator: Graph Kernel Network for Partial Differential Equations*. Mar. 6, 2020. DOI: 10.48550/arXiv.2003.03485. arXiv: 2003.03485[cs,math,stat]. URL: http://arxiv.org/abs/2003.03485 (visited on 04/19/2023).

[25] James Duvall, Karthik Duraisamy, and Shaowu Pan. *Non-linear Independent Dual System (NIDS) for Discretization-independent Surrogate Modeling over Complex Geometries*. Sept. 14, 2021.

[26] Zachary Prince and Jean Ragusa. *Parametric Uncertainty Quantification Using Proper Generalized Decomposition applied to Neutron Diffusion*. Jan. 2, 2019.

[27] L. Gilli et al. "Uncertainty quantification for criticality problems using non-intrusive and adaptive Polynomial Chaos techniques". In: *Annals of Nuclear Energy* 56 (June 1, 2013), pp. 71–80. ISSN: 0306-4549. DOI: 10.1016/j.anucene.2013.01.009. URL: https://www.sciencedirect.com/science/article/pii/S0306454913000261 (visited on 04/19/2023).

[28] Rabie Abu Saleem, Majdi I. Radaideh, and Tomasz Kozlowski. "Application of deep neural networks for high-dimensional large BWR core neutronics". In: *Nuclear Engineering and Technology* 52.12 (Dec. 1, 2020), pp. 2709–2716. ISSN: 1738-5733. DOI: 10.1016/j.net.2020.05.010. URL: https://www.sciencedirect.com/science/article/pii/S1738573320301637 (visited on 04/19/2023).

[29] Akio Yamamoto. "Application of Neural Network for Loading Pattern Screening of In-Core Optimization Calculations". In: *Nuclear Technology* 144 (Oct. 1, 2003), pp. 63–75. DOI: 10.13182/NT03-A3429.

[30] A. Meneses, A. Lima, and R. Schirru. “Artificial Intelligence Methods Applied to the In-Core Fuel Management Optimization”. In: 2018. URL: `https://api.semanticscholar.org/CorpusID:59438711` (visited on 04/19/2023).

[31] T. Ikonen and V. Tulkki. “The importance of input interactions in the uncertainty and sensitivity analysis of nuclear fuel behavior”. In: *Nuclear Engineering and Design* 275 (Aug. 1, 2014), pp. 229–241. ISSN: 0029-5493. DOI: `10.1016/j.nucengdes.2014.05.015`. URL: `https://www.sciencedirect.com/science/article/pii/S0029549314002908` (visited on 04/18/2023).

[32] C. S. Brown and Hongbin Zhang. “Uncertainty quantification and sensitivity analysis with CASL Core Simulator VERA-CS”. In: *Annals of Nuclear Energy* 95 (Sept. 1, 2016), pp. 188–201. ISSN: 0306-4549. DOI: `10.1016/j.anucene.2016.05.016`. URL: `https://www.sciencedirect.com/science/article/pii/S0306454916302894` (visited on 04/20/2023).

[33] Péter German and Jean C. Ragusa. “Reduced-order modeling of parameterized multi-group diffusion k-eigenvalue problems”. In: *Annals of Nuclear Energy* 134 (Dec. 2019), pp. 144–157. ISSN: 03064549. DOI: `10.1016/j.anucene.2019.05.049`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0306454919303020` (visited on 07/12/2023).

[34] Péter German et al. “GeN-ROM—An OpenFOAM®-based multiphysics reduced-order modeling framework for the analysis of Molten Salt Reactors”. In: *Progress in Nuclear Energy* 146 (Apr. 1, 2022), p. 104148. ISSN: 0149-1970. DOI: `10.1016/j.pnucene.2022.104148`. URL: `https://www.sciencedirect.com/science/article/pii/S0149197022000282` (visited on 07/12/2023).

[35] Mauricio E. Tano and Jean C. Ragusa. “Sweep-Net: An Artificial Neural Network for radiation transport solves”. In: *Journal of Computational Physics* 426 (Feb. 2021), p. 109757. ISSN: 00219991. DOI: `10.1016/j.jcp.2020.109757`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0021999120305313` (visited on 07/12/2023).

[36] Patrick Behne and Jean C. Ragusa. “Parametric model-order reduction for radiation transport using multi-resolution proper orthogonal decomposition”. In: *Annals of Nuclear Energy* 180 (Jan. 2023), p. 109432. ISSN: 03064549. DOI: `10.1016/j.anucene.2022.109432`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0306454922004625` (visited on 07/12/2023).

[37] Ian Halvic and Jean C. Ragusa. “Non-intrusive model order reduction for parametric radiation transport simulations”. In: *Journal of Computational Physics* 492 (Nov. 1, 2023), p. 112385. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2023.112385`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999123004801` (visited on 11/14/2023).

[38] Brian Anderson. “A Machine Learning Based Approach to Minimize Crud Induced Effects in Pressurized Water Reactors”. PhD thesis. NC State, Apr. 13, 2021. URL: `https://repository.lib.ncsu.edu/handle/1840.20/38656?show=full`.

[39] B. Kochunas et al. “Validation and Application of the 3 D Neutron Transport MPACT Code within CASL VERA-CS”. In: 2015. URL: `https://api.semanticscholar.org/CorpusID:199367949` (visited on 04/20/2023).

[40] B. Kochunas et al. *Overview of development and design of MPACT: Michigan parallel characteristics transport code.* American Nuclear Society - ANS; La Grange Park (United States), July 1, 2013. URL: `https://www.osti.gov/biblio/22212692` (visited on 04/20/2023).

[41] A. Godfrey et al. "VERA Benchmarking Results for Watts Bar Nuclear Plant Unit 1 Cycles 1-12". In: 2016. URL: `https://www.semanticscholar.org/paper/VERA-Benchmarking-Results-for-Watts-Bar-Nuclear-1-Godfrey-Collins/ade52a6ab6fcbaac34f4d28aff528f69050d79cb` (visited on 04/20/2023).

[42] Benjamin Collins, Brendan Kochunas, and Shane Stimpson. "MPACT Theory Manual". In: *Department of Energy* (Nov. 1, 2019).

[43] Benjamin Collins et al. "Simulation of the BEAVRS benchmark using VERA". In: *Annals of Nuclear Energy* 145 (Sept. 15, 2020), p. 107602. ISSN: 0306-4549. DOI: `10.1016/j.anucene.2020.107602`. URL: `https://www.sciencedirect.com/science/article/pii/S0306454920303005` (visited on 04/20/2023).

[44] Tengfei Zhang and Zhipeng Li. "Variational nodal methods for neutron transport: 40 years in review". In: *Nuclear Engineering and Technology* 54.9 (Sept. 1, 2022), pp. 3181–3204. ISSN: 1738-5733. DOI: `10.1016/j.net.2022.04.012`. URL: `https://www.sciencedirect.com/science/article/pii/S1738573322002157` (visited on 04/24/2023).

[45] Nam-Zin Cho and Jonghwa Chang. "Some outstanding problems in neutron transport computation". In: *Nuclear Engineering and Technology* 41 (May 31, 2009). DOI: `10.5516/NET.2009.41.4.381`.

[46] L. A. Semenza, E. E. Lewis, and E. C. Rossow. "The Application of the Finite Element Method to the Multigroup Neutron Diffusion Equation". In: *Nuclear Science and Engineering* 47.3 (Mar. 1, 1972). Publisher: Taylor & Francis _eprint: https://doi.org/10.13182/NSE72-A22416, pp. 302–310. ISSN: 0029-5639. DOI: `10.13182/NSE72-A22416`. URL: `https://doi.org/10.13182/NSE72-A22416` (visited on 04/24/2023).

[47] Robert Salko et al. "Development of COBRA-TF for modeling full-core, reactor operating cycles". In: *Advances in Nuclear Fuel Management V (ANFM 2015)* (2015). Publisher: Hilton Head South Carolina.

[48] R Salko et al. "CTF Theory Manual". In: *US Department of Energy* (Nov. 8, 2019).

[49] D. P. Griesheimer et al. "MC21 v.6.0 – A continuous-energy Monte Carlo particle transport code with integrated reactor feedback capabilities". In: *Annals of Nuclear Energy.* Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms 82 (Aug. 1, 2015), pp. 29–40. ISSN: 0306-4549. DOI: `10.1016/j.anucene.2014.08.020`. URL: `https://www.sciencedirect.com/science/article/pii/S0306454914004058` (visited on 04/20/2023).

[50] Brian N. Aviles et al. “MC21/COBRA-IE and VERA-CS multiphysics solutions to VERA core physics benchmark problem #6”. In: *Progress in Nuclear Energy*. Special Issue on the Physics of Reactors International Conference PHYSOR 2016: Unifying Theory and Experiments in the 21st Century 101 (Nov. 1, 2017), pp. 338–351. ISSN: 0149-1970. DOI: `10.1016/j.pnucene.2017.05.017`. URL: `https://www.sciencedirect.com/science/article/pii/S0149197017301221` (visited on 04/20/2023).

[51] Daniel J. Kelly et al. “MC21/CTF and VERA multiphysics solutions to VERA core physics benchmark progression problems 6 and 7”. In: *Nuclear Engineering and Technology*. Special Issue on International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering 2017 (M&C 2017) 49.6 (Sept. 1, 2017), pp. 1326–1338. ISSN: 1738-5733. DOI: `10.1016/j.net.2017.07.016`. URL: `https://www.sciencedirect.com/science/article/pii/S1738573317302978` (visited on 04/20/2023).

[52] “Chapter 5 - Fission Product Release and Transport”. In: *Nuclear Safety in Light Water Reactors*. Ed. by Bal Raj Sehgal. Boston: Academic Press, Jan. 1, 2012, pp. 425–517. ISBN: 978-0-12-388446-6. DOI: `10.1016/B978-0-12-388446-6.00005-8`. URL: `https://www.sciencedirect.com/science/article/pii/B9780123884466000058` (visited on 11/05/2023).

[53] Bateman H. “The solution of a system of differential equations occurring in the theory of radioactive transformations”. In: *Proc. Cambridge Philos. Soc.* 15 (1910), pp. 423–427. URL: `https://cir.nii.ac.jp/crid/1571135649245258368` (visited on 11/05/2023).

[54] Marco Antonio Cardoso. “Development and Application of Reduced-Order Modeling Procedures for Reservoir Simulation”. PhD thesis. Standford University.

[55] Zhaojun Bai. “Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems”. In: *Applied Numerical Mathematics*. 19th Dundee Biennial Conference on Numerical Analysis 43.1 (Oct. 1, 2002), pp. 9–44. ISSN: 0168-9274. DOI: `10.1016/S0168-9274(02)00116-2`. URL: `https://www.sciencedirect.com/science/article/pii/S0168927402001162` (visited on 07/20/2023).

[56] Serkan Gugercin and Athanasios C. Antoulas. “A Survey of Model Reduction by Balanced Truncation and Some New Results”. In: *International Journal of Control* 77.8 (May 20, 2004), pp. 748–766. ISSN: 0020-7179. DOI: `10.1080/00207170410001713448`. URL: `https://doi.org/10.1080/00207170410001713448` (visited on 07/18/2023).

[57] Elnaz Rezaian and Karthik Duraisamy. *Data-driven Balanced Truncation for Predictive Model Order Reduction of Aeroacoustic Response*. May 30, 2023. DOI: `10.48550/arXiv.2304.13900`. arXiv: `2304.13900[physics]`. URL: `http://arxiv.org/abs/2304.13900` (visited on 07/18/2023).

[58] A. C. Antoulas. “Approximation of Large-Scale Dynamical Systems: An Overview”. In: *IFAC Proceedings Volumes*. 10th IFAC/IFORS/IMACS/IFIP Symposium on Large Scale Systems 2004: Theory and Applications, Osaka, Japan, 26-28 July, 2004 37.11 (July 1, 2004), pp. 19–28. ISSN: 1474-6670. DOI: `10.1016/S1474-6670(17)31584-7`. URL: `https://www.sciencedirect.com/science/article/pii/S1474667017315847` (visited on 07/20/2023).

[59] Kuan Lu et al. “A Review of Model Order Reduction Methods for Large-Scale Structure Systems”. In: *Shock and Vibration* 2021 (May 8, 2021). Ed. by Zeqi Lu, pp. 1–19. ISSN: 1875-9203, 1070-9622. DOI: 10.1155/2021/6631180. URL: https://www.hindawi.com/journals/sv/2021/6631180/ (visited on 07/20/2023).

[60] Peter Yichen Chen et al. “CROM: Continuous Reduced Order Modeling of PDEs using Implicit Neural Representations”. In: International Conference on Learning Representations. Kigali Rwanda, 2023.

[61] Fahad Alsayyari et al. “A nonintrusive adaptive reduced order modeling approach for a molten salt reactor system”. In: *Annals of Nuclear Energy* 141 (June 15, 2020), p. 107321. ISSN: 0306-4549. DOI: 10.1016/j.anucene.2020.107321. URL: https://www.sciencedirect.com/science/article/pii/S0306454920300190 (visited on 05/06/2023).

[62] Sk Mashfiqur Rahman et al. “Nonintrusive reduced order modeling framework for quasigeostrophic turbulence”. In: *Physical Review E* 100 (Nov. 12, 2019). DOI: 10.1103/PhysRevE.100.053306.

[63] Patrick Behne, Jan Vermaak, and Jean Ragusa. “Parametric Model-Order Reduction for Radiation Transport Simulations Based on an Affine Decomposition of the Operators”. In: *Nuclear Science and Engineering* 197.2 (Feb. 1, 2023), pp. 233–261. ISSN: 0029-5639, 1943-748X. DOI: 10.1080/00295639.2022.2112901. URL: https://www.tandfonline.com/doi/full/10.1080/00295639.2022.2112901 (visited on 07/12/2023).

[64] ROBERT Hecht-nielsen. “III.3 - Theory of the Backpropagation Neural Network**Based on “nonindent” by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE.” In: *Neural Networks for Perception*. Ed. by Harry Wechsler. Academic Press, Jan. 1, 1992, pp. 65–93. ISBN: 978-0-12-741252-8. DOI: 10.1016/B978-0-12-741252-8.50010-8. URL: https://www.sciencedirect.com/science/article/pii/B9780127412528500108 (visited on 04/22/2023).

[65] Barry J. Wythoff. “Backpropagation neural networks: A tutorial”. In: *Chemometrics and Intelligent Laboratory Systems* 18.2 (Feb. 1, 1993), pp. 115–155. ISSN: 0169-7439. DOI: 10.1016/0169-7439(93)80052-J. URL: https://www.sciencedirect.com/science/article/pii/016974399380052J (visited on 04/22/2023).

[66] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980[cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 07/27/2023).

[67] D. Ballard. “Modular Learning in Neural Networks”. In: AAAI Conference on Artificial Intelligence. July 13, 1987. URL: https://api.semanticscholar.org/CorpusID:38968420 (visited on 04/19/2023).

[68] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Neural Information Processing Systems* 25 (Jan. 1, 2012). DOI: 10.1145/3065386.

[69] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological Cybernetics* 36.4 (Apr. 1, 1980), pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251. URL: https://doi.org/10.1007/BF00344251 (visited on 04/19/2023).

[70] Yann Lecun, Fu Huang, and L. Bottou. "Learning methods for generic object recognition with invariance to pose and lighting". In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR. Vol. 2. Jan. 1, 2004, pp. II–97. ISBN: 978-0-7695-2158-9. DOI: 10.1109/CVPR.2004.1315150.

[71] Kevin Jarrett et al. "What is the Best Multi-Stage Architecture for Object Recognition?" In: In Proc Intl Conf on Comput Vis. Vol. 12. Sept. 1, 2009. DOI: 10.1109/ICCV.2009.5459469.

[72] Honglak Lee et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". In: Proceedings of the 26th International Conference On Machine Learning, ICML 2009. June 14, 2009, p. 77. DOI: 10.1145/1553374.1553453.

[73] Christian Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. June 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[74] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 4, 2014).

[75] Vincent Sitzmann et al. *Implicit Neural Representations with Periodic Activation Functions*. June 17, 2020. DOI: 10.48550/arXiv.2006.09661. arXiv: 2006.09661[cs,eess]. URL: http://arxiv.org/abs/2006.09661 (visited on 04/20/2023).

[76] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (Feb. 1, 2019), pp. 686–707. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125 (visited on 04/20/2023).

[77] Isaac Lagaris, Aristidis Likas, and Dimitrios Papageorgiou. "Neural-network methods for boundary value problems with irregular boundaries". In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 11 (Feb. 1, 2000), pp. 1041–9. DOI: 10.1109/72.870037.

[78] Yinhao Zhu et al. "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data". In: *Journal of Computational Physics* 394 (Oct. 1, 2019), pp. 56–81. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.05.024. URL: https://www.sciencedirect.com/science/article/pii/S0021999119303559 (visited on 04/20/2023).

[79] Jeong Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: June 1, 2019, pp. 165–174. DOI: 10.1109/CVPR.2019.00025.

[80] Kaustubh Tangsali, Vinayak R. Krishnamurthy, and Zohaib Hasnain. "Generalizability of Convolutional Encoder–Decoder Networks for Aerodynamic Flow-Field Prediction Across Geometric and Physical-Fluidic Variations". In: *Journal of Mechanical Design* 143.5 (Nov. 13, 2020). ISSN: 1050-0472. DOI: 10.1115/1.4048221. URL: https://doi.org/10.1115/1.4048221 (visited on 04/20/2023).

[81] David Ha, Andrew Dai, and Quoc V. Le. *HyperNetworks*. version: 4. Dec. 1, 2016. DOI: 10.48550/arXiv.1609.09106. arXiv: 1609.09106[cs]. URL: http://arxiv.org/abs/1609.09106 (visited on 04/20/2023).

[82] Moloud Abdar et al. "A review of uncertainty quantification in deep learning: Techniques, applications and challenges". In: *Information Fusion* 76 (Dec. 2021), pp. 243–297. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.05.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S1566253521001081 (visited on 04/16/2023).

[83] N.W. Porter. "Wilks' formula applied to computational tools: A practical discussion and verification". In: *Annals of Nuclear Energy* 133 (Nov. 2019), pp. 129–137. ISSN: 03064549. DOI: 10.1016/j.anucene.2019.05.012. URL: https://linkinghub.elsevier.com/retrieve/pii/S0306454919302543 (visited on 07/20/2023).

[84] Seung Wook Lee et al. "Analysis of uncertainty quantification method by comparing Monte-Carlo method and Wilks'formula". In: *Nuclear Engineering and Technology* 46 (Aug. 25, 2014), pp. 481–488. DOI: 10.5516/NET.02.2013.047.

[85] Eyke Hüllermeier and Willem Waegeman. "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods". In: *Machine Learning* 110.3 (Mar. 1, 2021), pp. 457–506. ISSN: 1573-0565. DOI: 10.1007/s10994-021-05946-3. URL: https://doi.org/10.1007/s10994-021-05946-3 (visited on 04/16/2023).

[86] Radford M. Neal. *Bayesian Learning for Neural Networks*. Google-Books-ID: LHHrBwAAQBAJ. Springer Science & Business Media, Dec. 6, 2012. 194 pp. ISBN: 978-1-4612-0745-0.

[87] Rohitash Chandra, Royce Chen, and Joshua Simmons. *Bayesian neural networks via MCMC: a Python-based tutorial*. Apr. 1, 2023. DOI: 10.48550/arXiv.2304.02595. arXiv: 2304.02595[cs,stat]. URL: http://arxiv.org/abs/2304.02595 (visited on 11/04/2023).

[88] Jonas Gregor Wiese et al. *Towards Efficient MCMC Sampling in Bayesian Neural Networks by Exploiting Symmetry*. Apr. 6, 2023. DOI: 10.48550/arXiv.2304.02902. arXiv: 2304.02902[cs,stat]. URL: http://arxiv.org/abs/2304.02902 (visited on 11/04/2023).

[89] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 04/25/2023).

[90] Guotai Wang et al. "Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks". In: *Neurocomputing* 338 (Apr. 21, 2019), pp. 34–45. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2019.01.103. URL: https://www.sciencedirect.com/science/article/pii/S0925231219301961 (visited on 04/25/2023).

[91] Ruihan Hu et al. "The MBPEP: a deep ensemble pruning algorithm providing high quality uncertainty prediction". In: *Applied Intelligence* 49.8 (Aug. 1, 2019), pp. 2942–2955. ISSN: 1573-7497. DOI: 10.1007/s10489-019-01421-8. URL: https://doi.org/10.1007/s10489-019-01421-8 (visited on 04/27/2023).

[92] Yeming Wen, Dustin Tran, and Jimmy Ba. *BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning.* Feb. 19, 2020. DOI: 10.48550/arXiv.2002.06715. arXiv: 2002.06715[cs,stat]. URL: http://arxiv.org/abs/2002.06715 (visited on 04/27/2023).

[93] Kurtland Chua et al. *Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models.* Nov. 2, 2018. DOI: 10.48550/arXiv.1805.12114. arXiv: 1805.12114[cs,stat]. URL: http://arxiv.org/abs/1805.12114 (visited on 04/27/2023).

[94] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms.* USA: Cambridge University Press, May 2002. ISBN: 978-0-521-64298-9. URL: https://www.inference.org.uk/itprnn/book.pdf (visited on 11/04/2023).

[95] Samuel Stanton et al. *Kernel Interpolation for Scalable Online Gaussian Processes.* Mar. 1, 2021. DOI: 10.48550/arXiv.2103.01454. arXiv: 2103.01454[cs,stat]. URL: http://arxiv.org/abs/2103.01454 (visited on 04/27/2023).

[96] Andreas Damianou and Neil D. Lawrence. "Deep Gaussian Processes". In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics.* Artificial Intelligence and Statistics. ISSN: 1938-7228. PMLR, Apr. 29, 2013, pp. 207–215. URL: https://proceedings.mlr.press/v31/damianou13a.html (visited on 11/04/2023).

[97] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (Mar. 1951). Publisher: Institute of Mathematical Statistics, pp. 79–86. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729694. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-1/On-Information-and-Sufficiency/10.1214/aoms/1177729694.full (visited on 11/04/2023).

[98] Karthik Duraisamy. *Variational Encoders and Autoencoders : Information-theoretic Inference and Closed-form Solutions.* Jan. 27, 2021.

[99] Geoffrey E. Hinton and Drew Van Camp. "Keeping the neural networks simple by minimizing the description length of the weights". In: *Proceedings of the sixth annual conference on Computational learning theory - COLT '93.* the sixth annual conference. Santa Cruz, California, United States: ACM Press, 1993, pp. 5–13. ISBN: 978-0-89791-611-0. DOI: 10.1145/168304.168306. URL: http://portal.acm.org/citation.cfm?doid=168304.168306 (visited on 04/26/2023).

[100] Alex Graves. "Practical Variational Inference for Neural Networks". In: *Advances in Neural Information Processing Systems.* Vol. 24. Curran Associates, Inc., 2011. (Visited on 04/26/2023).

[101] Charles Blundell et al. "Weight uncertainty in neural networks". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37.* ICML'15. Lille, France: JMLR.org, July 6, 2015, pp. 1613–1622. (Visited on 04/16/2023).

[102] Durk P Kingma, Tim Salimans, and Max Welling. "Variational Dropout and the Local Reparameterization Trick". In: *Advances in Neural Information Processing Systems.* Vol. 28. Curran Associates, Inc., 2015. URL: https://papers.nips.cc/paper_files/paper/2015/hash/bc7316929fe1545bf0b98d114ee3ecb8-Abstract.html (visited on 04/26/2023).

[103] Laurent Valentin Jospin et al. “Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users”. In: *IEEE Computational Intelligence Magazine* 17.2 (May 2022), pp. 29–48. ISSN: 1556-603X, 1556-6048. DOI: 10.1109/MCI.2022.3155327. arXiv: 2007.06823[cs,stat]. URL: http://arxiv.org/abs/2007.06823 (visited on 04/16/2023).

[104] Yeming Wen et al. “Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches”. In: (Mar. 12, 2018).

[105] Ranganath Krishnan, Mahesh Subedar, and Omesh Tickoo. “Specifying Weight Priors in Bayesian Deep Neural Networks with Empirical Bayes”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.4 (Apr. 3, 2020). Number: 04, pp. 4477–4484. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i04.5875. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5875 (visited on 04/16/2023).

[106] Ranganath Krishnan, Pi Esposito, and Mahesh Subedar. *Bayesian-Torch: Bayesian neural network layers for uncertainty estimation*. Version v0.2.0. Jan. 27, 2022. DOI: 10.5281/ZENODO.5908307. URL: https://zenodo.org/record/5908307 (visited on 04/16/2023).

[107] G.-K. Delipei et al. “Uncertainty analysis methodology for multi-physics coupled rod ejection accident”. In: International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2019). Aug. 25, 2019. URL: https://hal.science/hal-02907458 (visited on 04/20/2023).

[108] D J Diamond, B P Bromley, and A L Aronson. “Studies of the Rod Ejection Accident in a PWR”. In: *Nuclear Regulatory Commission* (2002).

[109] William Falcon and The PyTorch Lightning Team. *PyTorch Lightning*. URL: https://www.pytorchlightning.ai.

[110] James Martens et al. *Rapid training of deep neural networks without skip connections or normalization layers using Deep Kernel Shaping*. Oct. 4, 2021. DOI: 10.48550/arXiv.2110.01765. arXiv: 2110.01765[cs]. URL: http://arxiv.org/abs/2110.01765 (visited on 04/20/2023).

[111] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[112] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. (Visited on 04/20/2023).

[113] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. Dec. 5, 2017. DOI: 10.48550/arXiv.1712.01815. arXiv: 1712.01815[cs]. URL: http://arxiv.org/abs/1712.01815 (visited on 04/20/2023).

[114] James Duvall, Karthik Duraisamy, and Shaowu Pan. *Discretization-independent surrogate modeling over complex geometries using hypernetworks and implicit representations*. May 17, 2022. DOI: 10.48550/arXiv.2109.07018. arXiv: 2109.07018[physics]. URL: http://arxiv.org/abs/2109.07018 (visited on 05/07/2023).

[115] Rene Y. Choi et al. “Introduction to Machine Learning, Neural Networks, and Deep Learning”. In: *Translational Vision Science & Technology* 9.2 (Feb. 27, 2020), p. 14. ISSN: 2164-2591. DOI: `10.1167/tvst.9.2.14`. URL: `https://doi.org/10.1167/tvst.9.2.14` (visited on 11/04/2023).

[116] Patrice Simard, Dave Steinkraus, and John Platt. “Best Practices for Convolutional Neural Networks”. In: (Sept. 27, 2003).

[117] Richard Liaw et al. *Tune: A Research Platform for Distributed Model Selection and Training*. July 13, 2018.

[118] Tong Yu and Hong Zhu. “Hyper-Parameter Optimization: A Review of Algorithms and Applications”. In: *ArXiv* (Mar. 12, 2020). URL: `https://www.semanticscholar.org/paper/Hyper-Parameter-Optimization%3A-A-Review-of-and-Yu-Zhu/a48fd38cc34f8ffe9bcf043eafe11289627dd91a` (visited on 04/20/2023).

[119] Ruslan Kuprieiev et al. *DVC: Data Version Control - Git for Data & Models*. Apr. 12, 2023. DOI: `10.5281/zenodo.7823624`. URL: `https://zenodo.org/record/7823624` (visited on 04/16/2023).

[120] Paulius Micikevicius et al. *Mixed Precision Training*. Feb. 15, 2018. DOI: `10.48550/arXiv.1710.03740`. arXiv: `1710.03740[cs,stat]`. URL: `http://arxiv.org/abs/1710.03740` (visited on 04/20/2023).