THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire Jean Kuntzmann

Métamodèles de réseaux de neurones avec une nouvelle approche de monitoring pour accélérer les simulations multiphysiques

Surrogate Neural Network Models with a Novel Monitoring Approach to Accelerate Multi-Physics Simulations.

Présentée par :

Mohamed Bahi YAHIAOUI

Direction de thèse :

Julyan ARBEL — Directeur de thèse
CHARGE DE RECHERCHE HDR, CENTRE INRIA UNIVERSITE GRENOBLE ALPES

Geoffrey DANIEL — Co-encadrant de thèse
INGENIEUR DE RECHERCHE, CEA CENTRE DE PARIS-SACLAY

Jeremie BRUYELLE — Co-encadrant de thèse
INGENIEUR DE RECHERCHE, CEA CENTRE DE CADARACHE

Rapporteurs :

FABRICE GAMBOA
PROFESSEUR DES UNIVERSITES, UNIVERSITE TOULOUSE III - PAUL SABATIER

BADIH GHATTAS
PROFESSEUR DES UNIVERSITES, AIX-MARSEILLE UNIVERSITE

Thèse soutenue publiquement le **27 novembre 2025**, devant le jury composé de :

PIERRE PUDLO, — Président
PROFESSEUR DES UNIVERSITES, AIX-MARSEILLE UNIVERSITE

JULYAN ARBEL, — Directeur de thèse
CHARGE DE RECHERCHE HDR, CENTRE INRIA UNIVERSITE GRENOBLE ALPES

FABRICE GAMBOA, — Rapporteur
PROFESSEUR DES UNIVERSITES, UNIVERSITE TOULOUSE III - PAUL SABATIER

BADIH GHATTAS, — Rapporteur
PROFESSEUR DES UNIVERSITES, AIX-MARSEILLE UNIVERSITE

CLEMENTINE PRIEUR, — Examinatrice
PROFESSEURE DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES

GIANNI FRANCHI, — Examinateur
ASSISTANT PROFESSOR, INSTITUT POLYTECHNIQUE DE PARIS

Invités :

GEOFFREY DANIEL
INGENIEUR DE RECHERCHE, CEA CENTRE DE PARIS-SACLAY

ANTOINE BOULORE
INGENIEUR DOCTEUR, CENTRE CEA DE CADARACHE

Abstract

Neural-network surrogates promise substantial speedups for multiphysics simulation, but their reliability is constrained by training-distribution dependence. This thesis examines these issues in the context of pressurized water reactors and nuclear fuel performance, where local models are invoked at every mesh point and time step of the calculus. After introducing the reactor and simulation context, we outline general acceleration strategies, contrast numerical error guarantees with those of data-driven surrogates, and clarify the specific challenges of neural networks used as surrogates.

In the first part of this work, we develop a runtime verification and monitoring framework emphasizing out-of-model-scope detection, namely Bounding Box Anomaly Score (BBAS). At its core is a bounding box abstraction based method. We formalize the geometric structure of these boxes, define an anomaly score, and show how activation-pattern clustering supports multi-layer monitoring rather than relying solely on the pre-final embedding. The approach is evaluated on standard out-of-distribution detection benchmark and the results show its competitive performances compared to state-of-the-art methods.

The second part of this work is dedicated to the development of surrogate models applied for two multi-physics simulation components. On the one hand, the BBAS framework is instantiated in a coupled thermodiffusion setting by metamodeling OpenCalphad, including data generation, preprocessing, surrogate training, and deployment. The monitor delineates the surrogate operational domain and triggers fallback to the high-fidelity evaluator when out-of-model-scope inputs are detected. On the other hand, we address the complex problem of developping a surrogate for the computation code CARACAS. The work is focused on the design and assessment of trajectory-aware surrogates and their integration within ALCYONE, using a specific architecture that helps ensure the robustness of the model predictions along the simulation.

These contributions bring new methodologies to improve reliability of neural surrogates, establishing a geometry-aware monitoring paradigm and applying dedicated trajectory-aware models for enforcing stability in long simulations.

Résumé

Les métamodèles fondés sur des réseaux de neurones promettent des accélérations substantielles pour les simulations multiphysiques, mais leur fiabilité est limitée par la dépendance à la distribution d'entraînement. Cette thèse examine ces enjeux dans le contexte des réacteurs à eau sous pression (REP) et des performances du combustible nucléaire, où des modèles locaux sont invoqués à chaque nœud du maillage et à chaque pas de temps du calcul. Après avoir présenté le contexte du réacteur et de la simulation, nous exposons des stratégies générales d'accélération, mettons en contraste les garanties d'erreur numériques avec celles des métamodèles fondés sur les données, et clarifions les défis spécifiques des réseaux de neurones utilisés comme métamodèles.

Dans la première partie, nous développons un cadre de vérification et de surveillance à l'exécution mettant l'accent sur la détection hors du domaine du modèle, à savoir le Bounding Box Anomaly Score (BBAS). Au cœur de l'approche se trouve une méthode fondée sur une abstraction par boîtes englobantes. Nous formalisons la structure géométrique de ces boîtes, définissons un score d'anomalie et montrons comment le regroupement (clustering) des schémas d'activation permet une surveillance multi-couches plutôt que de s'appuyer uniquement sur la représentation de l'avant-dernière couche (pre-final embedding). L'approche est évaluée sur des benchmarks standard de détection hors distribution, et les résultats montrent des performances compétitives par rapport à l'état de l'art.

La seconde partie est consacrée au développement de métamodèles appliqués à deux composants de simulation multiphysique. D'une part, le cadre BBAS est instancié dans un contexte de thermodiffusion couplée via la métamodélisation d'OpenCalphad, incluant la génération de données, le prétraitement, l'entraînement du métamodèle et son déploiement. Le moniteur délimite le domaine opérationnel du métamodèle et déclenche un repli vers l'évaluateur haute fidélité lorsque des entrées hors domaine sont détectées. D'autre part, nous traitons le problème complexe du développement d'un métamodèle pour le code de calcul CARACAS. Le travail porte sur la conception et l'évaluation de métamodèles sensibles à la trajectoire et leur intégration au sein d'ALCYONE, au moyen d'une architecture spécifique qui contribue à assurer la robustesse des prédictions du modèle tout au long de la simulation.

Ces contributions proposent de nouvelles méthodologies pour améliorer la fiabilité des métamodèles neuronaux, en établissant un paradigme de surveillance conscient de la géométrie et en appliquant des modèles dédiés, sensibles à la trajectoire, afin d'imposer la stabilité dans les simulations longues.

Remerciements

Les pages qui suivent présentent une contribution que j'ai voulue aussi rigoureuse que possible, fruit de trois années de travail. Avant d'en venir au cœur de ce manuscrit, qu'il me soit permis d'exprimer ma gratitude à celles et ceux qui l'ont soutenu et accompagné.

Je souhaite tout d'abord remercier l'ensemble des membres du jury d'avoir accepté d'évaluer ce travail et de prendre part à la soutenance. Leur présence à cette étape importante, ainsi que le temps qu'ils y ont consacré, m'ont profondément honoré et ont beaucoup compté pour moi. J'adresse en particulier mes remerciements à Pierre PUDLO, président du jury, qui a accepté de présider la soutenance avec un préavis très court et qui en a assuré le bon déroulement. Ma gratitude va aussi à Badih GHATTAS, qui a accepté d'évaluer ce travail et en a suivi le développement au sein du comité de suivi individuel pendant toute la durée de la thèse. Je suis également reconnaissant à Fabrice GAMBOA pour l'examen attentif de ce manuscrit et pour le temps qu'il lui a consacré. Mes remerciements vont aussi à Clémentine PRIEUR et Gianni FRANCHI pour leur présence au sein du jury en qualité d'examinateurs et pour les échanges que nous avons eus lors de la soutenance.

Je souhaite également remercier l'ensemble de l'encadrement de cette thèse pour l'accompagnement scientifique dont j'ai bénéficié tout au long de ce travail. Leurs remarques, leurs conseils et leur disponibilité ont permis de faire évoluer ce projet et d'en améliorer la qualité. Au sein de cet encadrement, je tiens à remercier tout particulièrement mon directeur de thèse, Julyan ARBEL, pour la direction scientifique de ce travail et pour l'intérêt qu'il lui a porté. Ses remarques, ses conseils et son regard sur ce manuscrit ont contribué à orienter mes réflexions. Je lui suis également reconnaissant pour le temps qu'il a consacré à la lecture de ce travail, pour ses retours aux différentes étapes d'avancement et pour la confiance qu'il m'a accordée dans la conduite de ce projet. J'aimerais aussi exprimer une gratitude particulière envers Geoffrey DANIEL. Tout au long de ces années, il a été un point d'appui scientifique et humain. Ses remarques et ses questions lors de nos échanges ont joué un rôle important dans la manière dont les idées se sont structurées au fil du temps. Je le remercie sincèrement pour sa grande disponibilité, pour le temps consacré aux réunions, à la lecture des textes intermédiaires et du manuscrit, ainsi que pour les conseils donnés à des moments clés, qu'il s'agisse de choix méthodologiques, de l'orientation des travaux ou de décisions plus pratiques. Je tiens également à remercier chaleureusement Loïc GIRALDI, qui m'a encadré pendant la première année de la thèse et m'a beaucoup aidé à démarrer ce travail. Ses explications patientes, sa grande disponibilité et la confiance qu'il m'a accordée ont été essentielles pour poser les bases de ce projet et me permettre de trouver mes repères. Je souhaite aussi

remercier Jérémie BRUYELLE pour l'implication dont il a fait preuve tout au long de cette thèse. Il a suivi de près l'avancée du travail et a pris une part réelle dans sa réalisation. Je lui suis profondément reconnaissant pour la contribution qu'il a apportée à ce projet, pour le temps qu'il y a consacré et pour son engagement jusqu'à sa finalisation.

Je souhaite également remercier Clément INTROÏNI pour son aide précieuse au cours de cette thèse. Les développements qu'il a réalisés, notamment pour la partie thermodiffusion, ont constitué un support important pour les travaux présentés ici. Je lui suis reconnaissant pour le temps qu'il a consacré à ces développements et pour les échanges techniques qui les ont accompagnés. Ma gratitude va aussi à Antoine BOULORÉ, dont la connaissance approfondie du modèle CARACAS utilisé dans ce travail m'a été très précieuse. Les échanges que nous avons eus à son sujet, la manière dont il m'en a présenté la structure et les choix de modélisation ont rendu la découverte de CARACAS particulièrement enrichissante. Je le remercie pour le temps qu'il a consacré à répondre à mes questions et à expliquer en détail son fonctionnement. Je souhaite également remercier Elie DELPLACE pour les discussions qui m'ont permis de mieux comprendre le fonctionnement de CARACAS. Grâce à sa connaissance du code et à sa maîtrise de ses spécificités, il a pu répondre à de nombreuses questions techniques et m'aider à clarifier plusieurs points qui ont été importants pour les résultats obtenus dans ce travail.

Je souhaite également remercier le laboratoire LMCP et, plus largement, l'ensemble du SESC du CEA Cadarache pour le cadre dans lequel cette thèse a pu être menée, ainsi que pour le soutien apporté au projet. Le travail au quotidien, les échanges et la bonne ambiance ont beaucoup compté pendant ces années. Je pense en particulier à Marius, Ahmed, Carlo, Clément, Aurélien, Elie, Guillaume, Pedro, Alexandre, Solomon, Hela, Louis, Simon, Léo, Hanae et Mouhammad, avec qui les discussions, les cafés et les moments partagés ont été particulièrement précieux. Je tiens aussi à remercier tout spécialement Michel FREYSS, mon chef, pour l'attention qu'il a portée à cette thèse et pour le soutien qu'il m'a accordé tout au long de ces années.

Enfin, je voudrais également remercier ma famille, mes parents et mes frères. Leur soutien, leurs encouragements et le fait de savoir que je pouvais compter sur eux, même à distance, m'ont beaucoup aidé pendant ces années de thèse. Les messages, les appels, l'humour parfois au milieu des périodes les plus chargées et les moments passés ensemble ont été précieux pour garder un peu de recul et continuer à avancer.

Contents

1 Introduction **1**
- 1.1 Context . . . 2
  - 1.1.1 Pressurized water reactors . . . 2
  - 1.1.2 Nuclear fuel performance simulation . . . 3
  - 1.1.3 Thermochemical simulation code . . . 5
  - 1.1.4 Fission gas behavior simulation code . . . 6
- 1.2 Challenges of neural network based surrogate models . . . 7
  - 1.2.1 General simulation acceleration strategies . . . 7
  - 1.2.2 Error guarantees of surrogate models versus numerical models 8
  - 1.2.3 Neural network as surrogate models . . . 9
- 1.3 Objectives . . . 10
- 1.4 Thesis overview and structure . . . 10
- 1.5 Contributions . . . 11

2 Runtime verification and monitoring of neural networks surrogate models **13**
- 2.1 Physics and solver oriented monitoring . . . 14
- 2.2 Uncertainty quantification for monitoring . . . 15
  - 2.2.1 Uncertainty in neural networks . . . 16
  - 2.2.2 Epistemic uncertainty quantification and estimation . . . 18
  - 2.2.3 Limitations of epistemic uncertainty quantification . . . 19
- 2.3 Out-of-Distribution (OOD) detection . . . 20
- 2.4 Out-of-Model-Scope (OMS) detection . . . 21

3 Developing and validating a runtime monitoring approach based on bounding boxes abstraction and activation patterns **23**
- 3.1 Basic terminology and notation . . . 24
- 3.2 Bounding Box abstraction: foundations, advances, and impact . . . 25
  - 3.2.1 Interval abstraction: foundations and applications in deep learning . . . 25
  - 3.2.2 Bounding box abstraction . . . 26
  - 3.2.3 Literature review . . . 28
- 3.3 On the geometric structure of neural network bounding box . . . 29
  - 3.3.1 Bounding box over the first layer . . . 30
  - 3.3.2 Activation patterns . . . 31
- 3.4 Constructing a bounding box monitor using activation patterns . . . 34

3.4.1 Activation patterns as clustering features . . . 35
3.4.2 Bounding Box Anomaly Score . . . 39
3.4.3 Constructing a bounding box monitor . . . 40
3.4.4 Using the monitor in OMS detection . . . 42
3.5 Experiments and comparison using OpenOOD benchmark . . . 42
3.5.1 Extension to Convolutional Neural Networks . . . 42
3.5.2 Experimental setup . . . 44
3.5.3 Ablation study . . . 44
3.5.4 Clustering parameters . . . 46
3.5.5 Comparison with state-of-the-art methods . . . 46
3.5.6 Extension to transformer architectures . . . 48
3.5.7 Discussion on the choice of parameters . . . 49
3.6 Conclusion . . . 49

4 Metamodeling OpenCalphad 51
4.1 Coupled thermodiffusion model . . . 52
4.1.1 Description of the physical problem . . . 52
4.1.2 Partitioned two-stage simulation loop . . . 53
4.2 Surrogate model development . . . 54
4.2.1 Data generation . . . 54
4.2.2 Preprocessing . . . 55
4.2.3 Methodology . . . 56
4.2.4 Results . . . 57
4.3 Application of bounding box abstraction for Out-of-Model Scope (OMS) detection . . . 59
4.3.1 Operational domain of the neural network . . . 59
4.3.2 Data generation for OMS detection evaluation . . . 61
4.3.3 Implementation and evaluation of BBAS . . . 61
4.3.4 Comparison with deep-ensemble . . . 64
4.3.5 Determining the threshold for the anomaly score . . . 67
4.4 Conclusion and perspectives . . . 69

5 Surrogate modeling of fission gas behavior 70
5.1 CARACAS inner workings and integration within ALCYONE . . . 71
5.1.1 CARACAS physical model . . . 71
5.1.2 Integration within ALCYONE . . . 72
5.1.3 CARACAS numerical model . . . 73
5.2 Data generation and preprocessing . . . 75
5.2.1 Data generation . . . 75
5.2.2 Preprocessing . . . 75
5.3 Methodology . . . 76
5.3.1 Direct approach: next-step prediction . . . 76
5.3.2 Proposed approach: trajectory prediction . . . 78
5.3.3 Model components . . . 80
5.3.4 Training losses . . . 81
5.4 Results . . . 81
5.4.1 Single-step evaluation . . . 81

5.4.2 Trajectory evaluation 82
5.4.3 In simulation evaluation 86
5.4.4 Comment on the application of BBAS monitoring for OMS Detection 89
5.5 Conclusion and perspectives 91

6 Conclusion and perspectives **93**

A Detailed experiments results for Chapter 3 **96**
A.1 Full CIFAR-10 results 96
A.2 Full CIFAR-100 results 98
A.2.1 Full ImageNet-200 results: 100
A.3 Full ImageNet-1k results 102

B Extra results for Chapter 4 **108**

List of Figures

1.1 PWR Plant Schematic: Coolant Loop and Steam Cycle. Source: CEA. 3
1.2 PWR fuel-rod and fuel-assembly schematics (Guillet and Guérin, 2009). 4
1.3 Fuel pellets (Guillet and Guérin, 2009). 4
1.4 ALCYONE generic computational scheme (Michel et al., 2021). . . . 5
1.5 Nuclear fission (Parisot, 2015). 6

2.1 Sources of epistemic uncertainty. 17
2.2 An example of model misspecification using a Gaussian Process (GP) fitted on the blue points. The blue points represent the result of a Ground truth function for some inputs in $[0, 0.5] \cup [1.5, 2]$. The fitted GP grossly underestimate the uncertainty over $[0.5, 1.5]$ (Lahlou et al., 2021). 19
2.3 Neural network runtime monitoring is a research area aiming to characterize safe data instances, in order to reject unsafe predictions during inference (Guérin et al., 2023) 21
2.4 Differences between OOD and OMS detection (Guérin et al., 2023). . 22

3.1 Bounding box of the first layer, level sets in blue represent the upper bounds and in red lower bounds over the whole dataset. 31
3.2 Neural Network bounding boxes of the other layers, level sets in violet represent the lower bounds and in red the upper bounds over the whole dataset. 32
3.3 Activation regions of the neural network, on each partition of space the activation pattern is constant. 33
3.4 Bounding boxes defined over points according to their activation patterns. Activation boundaries shown in black. 39
3.5 Comparison between the aggregated anomaly map (left) and the standard anomaly score (AS) map (right) on the two moons dataset. 41
3.6 Complete BBAS pipeline processing: from constructing the monitor to runtime monitoring. 41
3.7 AUROC scores (%) for OOD identification performance on CIFAR-10 and CIFAR-100 averaged over four OOD datasets, where BBAS demonstrate very strong performance. See Section A.1 and A.2 for details. 43
3.8 AUROC scores on CIFAR-10 (top) and CIFAR-100 (bottom), averaged over all FarOOD datasets, depending on the number of clusters and linkage method. 46

4.1 One-dimensional radial discretization of the fuel pellet, illustrating the mesh of spatial nodes used to model diffusion from the center to the periphery (Introïni et al., 2021). 53
4.2 Schematic overview of the simulation loop. 54
4.3 Example of a temperature profile used for a simulation: temperature ($T[K]$) in terms of the pellet radius ($r[m]$). 55
4.4 Evolution of the oxygen molar quantity (N_O) vs pellet radius ($r[m]$). 58
4.5 Evolution of oxygen chemical potential (μ_O) vs pellet radius ($r[m]$). . 58
4.6 Evolution of oxygen mobility (M_O) vs pellet radius ($r[m]$). 58
4.7 Histogram of $\log_{10}$ of the loss values over the training dataset. 60
4.8 Histogram of $\log_{10}$ of the loss values over the extended test dataset. . 62
4.9 Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\text{anomaly score} + 1)$ on held-out test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$). . . . 63
4.10 Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\text{anomaly score} + 1)$ on extended test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$). . . . 64
4.11 Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\sigma^2)$ on held-out test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$). 65
4.12 Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\sigma^2)$ on extended test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$). 66

5.1 Left: Representative grain schematic with fission-gas populations. Right: Representative grain schematic with high burnup structure. Source: CEA. 71
5.2 Schematic representation of the ALCYONE multiphysics computational scheme with the different levels of description (Introïni et al., 2024). 73
5.3 Comparison of reference data and the neural net next-step predictor model across time (days) in simulation for example variables. 79
5.4 Neural network predictions vs. reference simulation for multiple variables over a full trajectory at two spatial locations close to the pellet center (left column) and close to the pellet periphery (right column). 84
5.5 Neural network predictions vs. reference simulation for $Y^{(15)}$ and $Y^{(13)}$ over mesh points at different times. 85
5.6 Neural network predictions vs. reference simulation for multiple variables over a completed simulation integration case at two spatial locations close to the pellet center (left column) and close to the pellet periphery (right column). 87
5.7 Neural network predictions vs. reference simulation for multiple variables over an incomplete simulation integration case at two spatial locations close to the pellet center (left column) and close to the pellet periphery (right column). 88
5.8 Neural network predictions compared with reference simulations for $Y^{(9)}$ across three runs: (top) the trajectory evolution example from Section 5.4.2, (middle) the completed simulation integration case, and (bottom) the incomplete integration case. 90

B.1 Neural network and reference simulation for multiple variables evolution vs pellet radius ($r[m]$). 108

List of Tables

3.1 Ablation study on layer, channel statistics and anomaly score choices. We report AUROC (%) and FPR@95 (%), averaged over FarOOD datasets for both CIFAR-10 and CIFAR-100. **Bold** values represent the best metric for each ablation study. 45
3.2 Average performances on FarOOD benchmarks using ResNet architectures. Best performance is in **bold**, second best is underlined. We indicate a − when the result is not provided in the original paper of the method nor in the OpenOOD leader board. 47
3.3 Average performances on NearOOD benchmarks using ResNet architectures. Best performance is in **bold**, second best is underlined. We indicate a − when the result is not provided in the original paper of the method nor in the OpenOOD leader board. 48
3.4 Results for Transformer tests on ImageNet-1k (AUROC). Best performance is in **bold**, second best is underlined. 48

4.1 Accuracy metrics for neural-network predicted chemical potentials, diffusion coefficients, derived chemical potential difference and interdiffusion coefficients. 57
4.2 Accuracy metrics for neural-network predicted chemical potentials, diffusion coefficients, derived chemical potential difference and interdiffusion coefficients for the extended test dataset. 62
4.3 Performance comparison among the two methods for out-of-model scope data detection (bold for best performance). Computation time is given with reference to the execution of a single neural network model. 66
4.4 Conformal thresholding at level $\alpha = 0.05$ with two calibration sources. Metrics are computed on the remaining extended test set: in-scope FPR on its in-scope subset, OMS TPR on its OMS subset. 68

5.1 Accuracy metrics for neural-network predicted normal gas population metrics computed when the values are active. 77
5.2 Accuracy metrics for the neural-network predicted local fraction of HBS ($Y^{(15)}$) and HBS gas population metrics computed when the values are active. 77
5.3 Individual time steps accuracy metrics for neural-network predicted normal gas population metrics computed when the values are active. . 82
5.4 Individual time steps accuracy metrics for neural-network predicted local fraction of HBS ($Y^{(15)}$) and HBS gas population metrics computed when the values are active. 82

5.5 Trajectory accuracy metrics for neural-network predicted normal gas population metrics computed when the values are active. 83
5.6 Trajectory accuracy metrics for the neural-network predicted local fraction of HBS ($Y^{(15)}$) and HBS gas population metrics computed when the values are active. 83

A.1 CIFAR-10 FarOOD Benchmark results on the ResNet18 architecture. 97
A.2 CIFAR-10 NearOOD Benchmark results on the ResNet18 architecture. 97
A.3 CIFAR-100 FarOOD Benchmark results on the ResNet18 architecture. 98
A.4 CIFAR-100 NearOOD Benchmark results on the ResNet18 architecture. 99
A.5 ImageNet-200 FarOOD Benchmark results on the ResNet18 architecture. 100
A.6 ImageNet-200 NearOOD Benchmark results on the ResNet18 architecture. 101
A.7 ImageNet-1k FarOOD Benchmark results on the ResNet50 architecture. 102
A.8 ImageNet-1k NearOOD Benchmark results on the ResNet50 architecture. 103
A.9 ImageNet-1k FarOOD benchmark results on the ViT-B/16 architecture. 104
A.10 ImageNet-1k NearOOD benchmark results on the ViT-B/16 architecture. 105
A.11 ImageNet-1k FarOOD benchmark results on the Swin-T architecture. 106
A.12 ImageNet-1k NearOOD benchmark results on the Swin-T architecture. 107

Chapter 1

Introduction

Numerical simulations unfold known causal relationships into a concrete description of system dynamics. Developers of numerical models apply these relationships, whether derived from first or empirically established principles, to specific physical configurations to predict their behavior under various conditions. Indeed, numerical models are virtual imitations of systems with some level of accuracy that can be analyzed and experimented upon in silico. Computer simulation plays a central role in modern engineering and scientific research, enabling the resolution of complex, coupled physical phenomena (mechanical, thermal, fluidic, electromagnetic, chemical) across a wide range of spatial and temporal scales, capturing details from grain-level plasticity in alloys (Ravaji and Joshi, 2021) to eddy-scale turbulence in arterial blood flow (Paul et al., 2009), such information being inaccessible in real world experiments. Also, simulations offer a significant financial upside through reduced scrap, shorter design loops, and predictive maintenance, as organisations use them to test ideas and designs way before building any physical prototype, which is also interesting from an environmental point of view, given the carbon and material cost of such experiments.

The scope of numerical simulation frequently involves multiphysics and multiscale aspects into one cohesive computational workflow that improves the predictive accuracy of simulations. However, it comes at the cost of significantly increased computational size and complexity (Keyes et al., 2013). Each additional physics coupling or finer scale resolution multiplies the computational workload, resulting in simulation models that can easily have millions or even billions of degrees of freedom to solve. The growing complexity and fidelity of simulations have made High-Performance Computing (HPC) an integral part of the simulation process, and many state-of-the-art simulations push the limits of what even modern supercomputers can handle. The reliance on supercomputing is underscored by large-scale public and private investment in petascale and exascale supercomputers (Attig et al., 2011), which has created a non-trivial environmental footprint. Some supercomputers draw continuous electrical loads in the order of megawatts, and individual multi-day simulation campaigns can consume several megawatt-hours of energy. As jurisdictions tighten absolute energy-use ceilings and carbon budgets (Suarez et al., 2025), these operational footprints are becoming binding constraints on numerical simulation itself, improvements that shorten runs and reduce resource contention also tend

to moderate operational footprints, reinforcing the motivation to make simulation workflows more efficient. Relying only on the hardware performance to obtain fast and accurate simulations is therefore not sustainable and other strategies must be explored to reduce the computation cost.

In light of these challenges, neural networks are increasingly viewed as a promising avenue for accelerating simulations (Karniadakis et al., 2021). Their ability to approximate complex mappings and capture patterns from data makes them an attractive complement to traditional numerical methods, particularly where repeated or large-scale computations are prohibitive. In this thesis, we examine the potential of neural network approaches to reduce the computational burden of high-fidelity simulations, with the goal of achieving faster predictions while maintaining the level of accuracy required for scientific and engineering applications. We focus on computation codes related to the simulation of elements in pressurized water reactors, a critical application that requires to guarantee the reliability of each code component.

1.1 Context

Nuclear energy is central to France electricity supply, providing roughly two-thirds of the country electricity and maintaining a stable, low-carbon energy mix. France primarily operates pressurized water reactors (PWRs), directing significant research toward enhancing their fuel performance and safety, particularly by extending fuel burnup, optimizing fuel cycles, and ensuring reliable behavior under accidental conditions.

1.1.1 Pressurized water reactors

A PWR, illustrated in Figure 1.1, is a type of light-water reactor using ordinary water as both coolant and neutron moderator. Its distinctive dual-loop cooling system comprises a primary loop, circulating water through the reactor core under high pressure (about 155 bar) to remain liquid above 300 °C, and a secondary loop, where heat is transferred through a steam generator boils water into steam, powering turbines to generate electricity. The two-loop design ensures that the potentially radioactive primary coolant does not mix with the secondary steam that goes to the turbine. In PWRs, water serves as the moderator, slowing down neutrons to sustain the fission chain reaction. Importantly, water also provides a negative feedback: if the reactor heats up excessively, the water in the core expands and becomes less dense, reducing its moderating capability and thus slowing the reaction.

At the reactor core, fuel rods, as represented in Figure 1.2a, produce heat via nuclear fission. Each fuel rod is a sealed metal tube (approximately 4 meters long) made of a zirconium alloy, known as zircaloy cladding, chosen for its strength, corrosion resistance, and low neutron absorption (i.e. they don't significantly absorb the neutrons needed to sustain fission). Inside each cladding tube is a stack of cylindrical uranium dioxide UO_2 ceramic pellets. A photo of such pellets is shown in Figure 1.3. Uranium dioxide is a refractory material with a very high melting point ($\sim$ 2800 °C) and can remain stable under intense radiation. The uranium

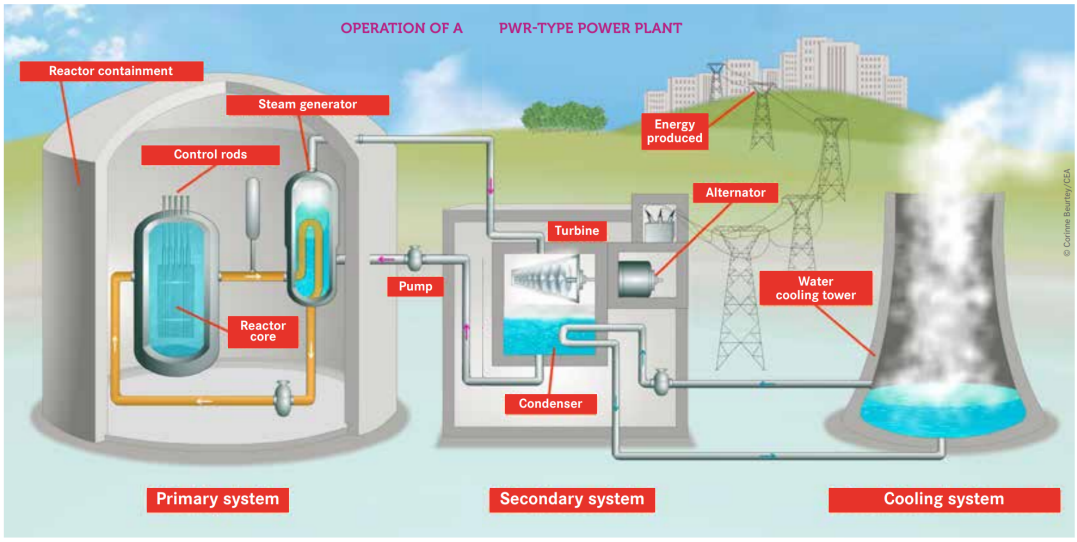


Figure 1.1: PWR Plant Schematic: Coolant Loop and Steam Cycle. Source: CEA.

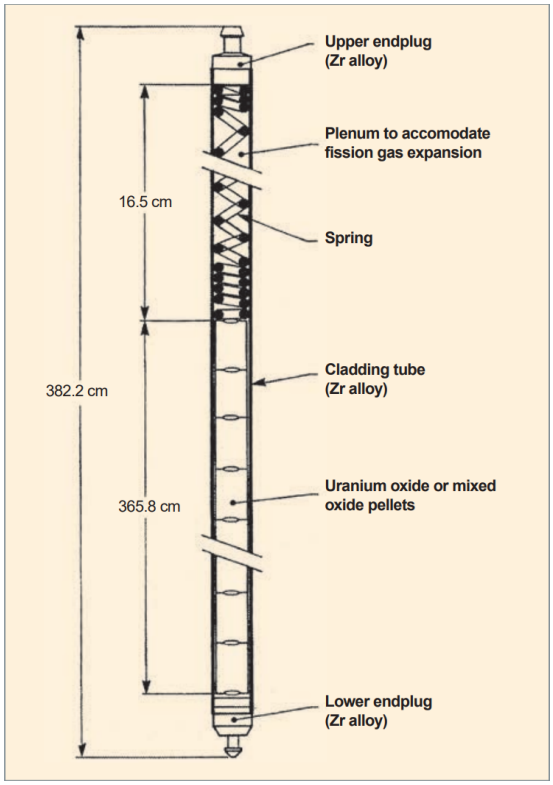
in these pellets is slightly enriched (commonly 3-5% uranium-235) to support a steady fission chain reaction in the light-water moderated environment. A typical fuel pellet is about 8 mm in diameter and 1 to 1.5 cm tall, weighing roughly 5-7 grams. Hundreds of these pellets are stacked within a single fuel rod cladding tube, separated by a small gap. This gap is filled with pressurized helium gas during fabrication, which serves two purposes: it improves heat transfer from the hot fuel pellets to the cladding, and it provides a space to accommodate pellet expansion and the release of fission gases. Fuel rods are grouped into fuel assemblies which are rigid bundles holding anywhere from 100 to 300 rods each, depending on the reactor design. Figure 1.2b shows a schematic of an assembly. A large PWR core will contain on the order of 150-200 such assemblies, totaling tens of thousands of fuel rods and on the order of 10-18 million fuel pellets in the core at once.

During operation, the fuel undergoes intense neutron irradiation and high temperatures. Over a typical cycle of 3 to 5 years, the pellets gradually change: they swell, develop microcracks, and accumulate fission products. The cladding also experiences corrosion, creep, and irradiation hardening. Maintaining the structural integrity of the fuel rod under these evolving conditions is critical for both operational safety and economic efficiency, and this effort relies heavily on advanced fuel performance modeling and simulation.

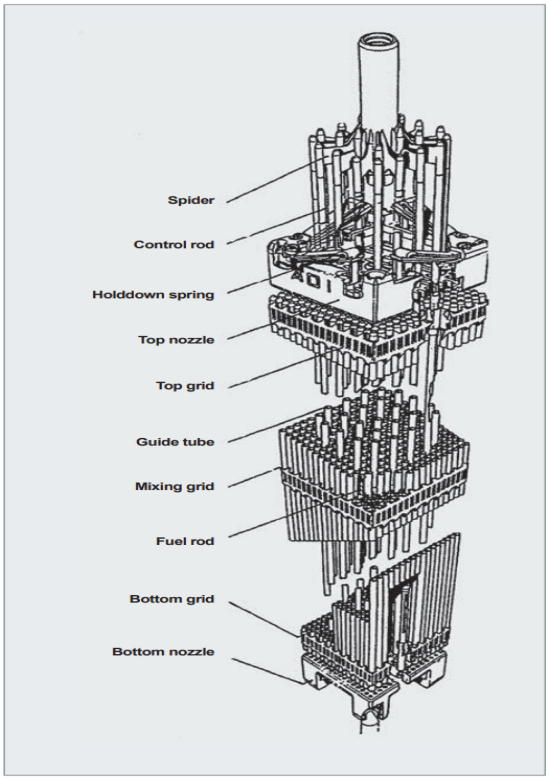
1.1.2 Nuclear fuel performance simulation

The CEA (Commissariat à l'Énergie Atomique et aux Énergies Alternatives) has developed a digital platform called PLEIADES, dedicated to simulating nuclear fuel performance. It aims to simulate their behaviors for all nuclear sectors, addressing industrial challenges and exploratory or R&D studies. The scientific computing tools of the platform can be parallel, multi-physics, and multi-scale and require considerable computing power to represent the behavior of a fuel pellet realistically.

ALCYONE (Introïni et al., 2024) is one of the platform simulation codes. It is a



(a) Schematic of a fuel rod.



(b) Schematic of a 17 x 17 PWR fuel assembly and control bundle.

Figure 1.2: PWR fuel-rod and fuel-assembly schematics (Guillet and Guérin, 2009).

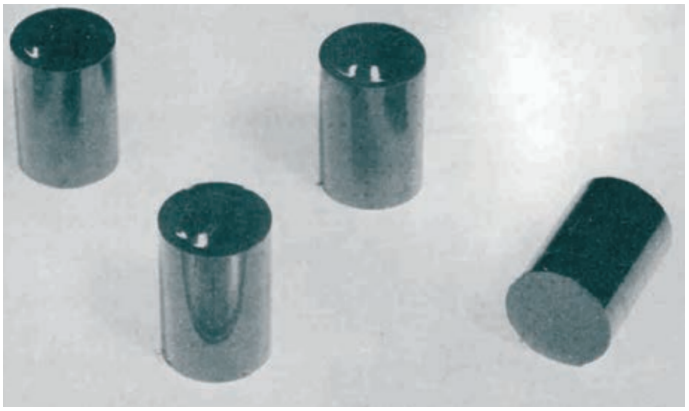


Figure 1.3: Fuel pellets (Guillet and Guérin, 2009).

state-of-the-art fuel performance code developed by the CEA in partnership with EDF and Framatome, dedicated to the simulation of PWR fuel rods behavior. In practice, ALCYONE can model an entire fuel rod along its length (with a 1D axial scheme coupled to radial resolution in each slice), or zoom in on a half-pellet 2D cross-section ($r-\theta$ plane) or even a 3D quarter-pellet fragment to capture asymmetric effects like pellet cracking (Sercombe et al., 2013).

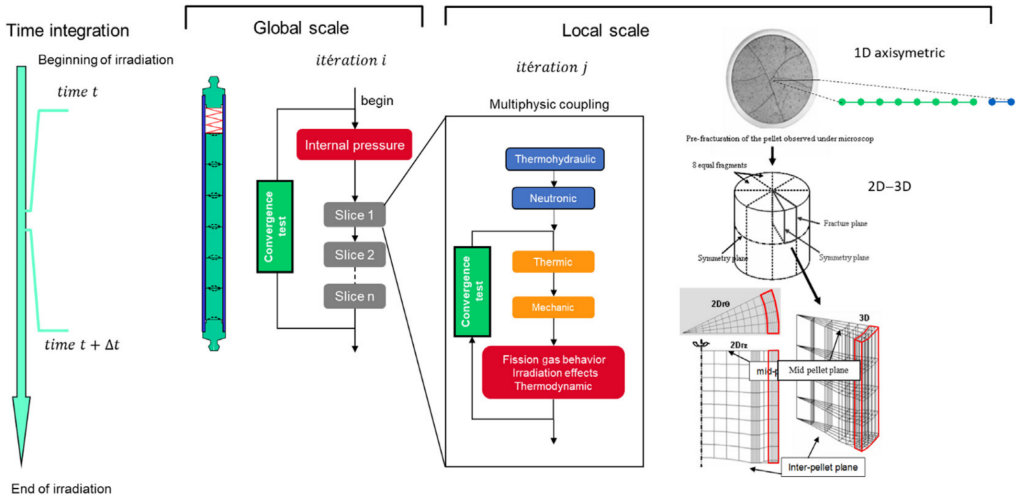


Figure 1.4: ALCYONE generic computational scheme (Michel et al., 2021).

All these schemes share a common finite-element core, the Cast3M FE (Finite Elements) solver, to solve the coupled thermal-mechanical equations of the pellet-gap-cladding system at each time step. The code architecture couples heat conduction, mechanical deformation, contact/friction, irradiation effects, fission gas behavior, and even basic coolant thermal-hydraulics in transient scenarios. A key feature of ALCYONE is its multiscale, triple-loop architecture that couples global, mesoscale, and local simulations, as illustrated in Figure 1.4. In practice, this means the code solves the macroscopic fuel rod behavior (thermal and mechanical equations for the pellet-cladding system) while simultaneously embedding microscopic models for grain-level phenomena. This heterogeneous simulation enables understanding, via numerical simulations, what happens at lower (meso- and microscopic) scales within the fuel, beyond what a purely homogeneous (averaged) model would reveal.

1.1.3 Thermochemical simulation code

To describe the thermochemical properties of the fuel, ALCYONE can couple to a CALPHAD-based thermochemical solver, OpenCalphad (Sundman et al., 2015), to evaluate local chemical equilibria consistently with the code thermo-mechanical and transport models. During a time step, selected integration points in the fuel pellet and at the fuel–clad interface are mapped to independent equilibrium problems defined by the current temperature, pressure, and elemental inventory: U–O and relevant fission products. OpenCalphad minimizes the total Gibbs energy for the user-specified phase set and constraints, and returns phase assemblages, phase fractions,

site fractions, activities, and chemical potentials. These quantities close several loops in ALCYONE: oxygen chemistry is driven by the oxygen potential and resulting O/M, fission-product speciation governs the formation of compounds and metallic inclusions, and solidus/liquidus information feeds melting criteria and thermophysical properties used in the energy balance.

1.1.4 Fission gas behavior simulation code

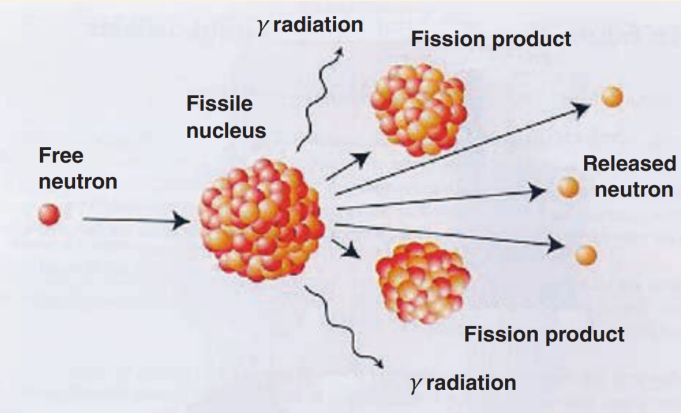


Figure 1.5: Nuclear fission (Parisot, 2015).

Fission gas behavior in nuclear fuel is an important factor for PWR modeling and simulation. During irradiation of uranium oxide UO_2, fission reactions, represented in Figure 1.5, produce noble gases, primarily xenon and krypton, that have low solubility in the fuel matrix. These gases tend to accumulate in the fuel as atomic defects or in bubbles, affecting fuel rod performance through phenomena such as fission gas release into the rod free volume, retention of gas within the fuel, swelling of the fuel pellets, and changes in thermal conductivity. ALCYONE employs the CARACAS (Jomard et al., 2014) model to simulate fission gas behavior within the UO_2 fuel grains. CARACAS is an advanced physico-mechanical model developed by CEA that handles fission gas creation and evolution at the grain scale. It tracks how xenon and krypton gas atoms produced by fission diffuse and precipitate to form gas bubbles both within the grains (intragranular) and along grain boundaries (intergranular) and eventually the release of gas to the free volume once grain-boundary bubbles interconnect. This local model feeds back into the simulation by providing the fission gas release source term and swelling strain to the global mechanical model.

One important computational aspect of ALCYONE design is that the CARACAS local model is invoked frequently during a simulation. Every material point in the fuel pellet model that requires fission gas evaluation will call CARACAS at each sub step of the global simulation. Consequently, CARACAS is solved hundreds of thousands to potentially millions of times per fuel rod simulation, depending on the mesh and number of time steps. This triple-loop structure (global-meso-local) ensures accuracy but also incurs a high computational cost. In fact, the fission gas behavior calculation is often one of the most CPU-intensive parts of the fuel performance modeling. To

mitigate this cost, the developers have introduced parallelism to ALCYONE 2.0 which can run multiple axial slices in parallel to speed up simulations. Still, running a full fuel rod simulation for many scenarios, such as different reactor transients, fuel designs, burnup levels, etc., can be time-consuming and impractical for real-time or iterative applications. This is where surrogate modeling comes in.

Surrogate models are data-driven models that emulate the behavior of a simulation code at a fraction of the cost. The idea of this thesis is to train a neural network on a detailed fuel simulation code to build a fast general-purpose and reliable surrogate model to predict fission gas behavior with reasonable accuracy.

1.2 Challenges of neural network based surrogate models

1.2.1 General simulation acceleration strategies

Modern simulation workflows employ various acceleration strategies operating at multiple levels, each with distinct challenges and trade-offs. Recent advances in hardware (multicore CPUs, GPUs, and high-bandwidth memory accelerators) allowed for significant acceleration without loss of fidelity. The only limitation is hardware cost and the overhead needed for code refactoring or optimization. In tandem with hardware, algorithmic accelerators (e.g., algebraic multigrid methods and communication-avoiding Krylov solvers) improve the efficiency of the numerical methods themselves, with the bonus of offering users the flexibility to balance fidelity and performance (Xu and Zikatanov, 2017; Hoemmen, 2010). Another approach is simplifying physics or model geometry by reducing dimensionality from full 3D models to 1D or 2D approximations, employing coarse grids, or adopting simplifying hypotheses, which accelerates simulations at the expense of fidelity. In order to maintain acceptable accuracy, these models are frequently calibrated against high-fidelity benchmarks. All these approaches try to accelerate simulations either by using more computing power or reducing the fidelity of the simulation. These approaches result in a tension between fidelity and computational cost, which becomes particularly acute in studies that require a large number of simulations, such as uncertainty propagation, sensitivity analysis, design optimization, or parameter space exploration.

Among acceleration strategies, surrogate modeling has gained prominence as the demand grows for rapid predictive tools in science and engineering. A surrogate models, also known as metamodels, are approximations of an expensive simulation. They act as computationally efficient replacements for time-consuming or resource-intensive simulations. Surrogate models replace an input-output map representing the computation code f:

$$f : \mathcal{X} \to \mathcal{Y}, \qquad y = f(x),$$

with a cheaper approximation $\hat{f}$ trained on a finite dataset $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$, where typically $y_i = f(x_i)$. The goal is not to solve the governing equations directly, but to predict y at new inputs x quickly. In essence, surrogate models trade offline

effort (data generation and training) for online speed, enabling previously infeasible design and inference loops. Main surrogate model approaches include, among others, Gaussian processes (Rasmussen and Williams, 2006; Marrel et al., 2015), spectral methods such as polynomial chaos (Marzouk and Najm, 2009), reduced order models (Kadeethum et al., 2022) and more recently neural networks (Antonello et al., 2023; Sobes et al., 2021).

1.2.2 Error guarantees of surrogate models versus numerical models

Data-driven surrogates are often built to accelerate studies that require a large number of evaluations by replacing the costly solver with a learned approximation of its input-output map over a prescribed parameter range drawn from some data-generating process $\mathcal{D}$. The surrogate is trained on simulation data sampled from that process, which means its performance guarantees are statistical.

Error guarantees of surrogate models. For a surrogate $\hat{f}$ trained on data, performance on unseen inputs is expressed via the expected generalization error:

$$R(\hat{f}) \;\coloneqq\; \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\, L\Big(\hat{f}(x), y\Big)\,\Big], \tag{1.1}$$

for a task-appropriate loss L.

Since $\mathcal{D}$ is often unknown and the expectation cannot be analytically computed in general, the empirical validation error $\hat{R}_n(\hat{f})$ is used as a proxy for $R(\hat{f})$:

$$\hat{R}_n(\hat{f}) \;\coloneqq\; \frac{1}{n}\sum_{i=1}^{n} L\Big(\hat{f}(x_i), y_i\Big), \tag{1.2}$$

where (x_i, y_i) is a collection of points that were not previously used to build the surrogate model $\hat{f}$.

The resulting generalization bounds, cross-validation error, and uncertainty estimates hold under hypotheses such as i.i.d. sampling, stationarity, and support coverage of the training distribution.

These assurances are weaker than the deterministic error bounds available for classical numerical schemes, where convergence rates and tolerances can be enforced by mesh, timestep, or polynomial order refinement.

Error guarantees of numerical models. For a numerical method with discrete solution u_h to a continuous target u the guarantees are typically deterministic. Two standard forms are:

$$\text{A priori (before computation):} \quad \|u - u_h\|_X \;\leq\; C\, h^p, \tag{1.3}$$

$$\text{A posteriori (after computation):} \quad \|u - u_h\|_X \;\leq\; C_{\text{est}}\, \eta(u_h), \tag{1.4}$$

where $\|.\|_X$ is an appropriate norm, h is a mesh size (or time step), p the method order, and $\eta(u_h)$ is a computable residual-based estimator that can be localized (per element) for adaptivity. Numerical-model bounds rely on analytical properties of the

underlying problem. Once these hold, the guarantees apply broadly to all admissible inputs.

Thus the comparative picture is clear. Numerical models can be computationally expensive but the error guarantees are generally strong and apply for all new admissible inputs. On the contrary, surrogate models deliver considerable speedups for studies by their reduced computational cost, but their error guarantees are statistical and specific to the generating distribution. This last point represents a drawback for ensuring the reliability of the models.

1.2.3 Neural network as surrogate models

Traditional surrogate model approaches face fundamental challenges when dealing with high-dimensional input spaces or broad parameter ranges. The curse of dimensionality, where the computational cost and data requirements grow exponentially with the number of input parameters, is a significant limitation of traditional surrogate models. For example, a simple full grid sampling with m points per dimension would require m^d simulations in d dimensions, an intractable scaling for large d. As a result, conventional surrogate modeling methods, like polynomial chaos expansions or standard Gaussian processes, become impractical or inaccurate in very high-dimensional settings (Zhou et al., 2020). Another difficulty arises when the study covers an extensive range for each parameter (broad input domains). Surrogates that perform well in a confined region may degrade in accuracy if asked to cover a much broader domain with potentially heterogeneous behaviors (Vargas-Fajardo et al., 2025).

Neural networks have emerged as powerful surrogate models capable of handling high-dimensional and highly nonlinear relationships. Modern neural network based surrogates (deep neural networks, convolutional or transformer-based models, etc.) often excel in high-dimensional or large-data scenarios where traditional surrogates struggle (Semenova, 2025). However, neural network surrogates come with significant caveats that limit their general usability beyond the immediate study domain. Chief among these is the black-box nature of neural networks. They generally lack transparency and interpretability. The complex, multi-layered structure of a deep network makes it difficult to understand how inputs are being mapped to outputs. Thus, it is not easy to detect when the neural network makes a mistake or is operating outside its valid conditions (Hassija et al., 2024). In contrast to traditional surrogate models which have clear validity limits, a neural network might silently extrapolate in regions with sparse or no training data, with no obvious warning sign to the user.

The practical consequence is that a neural network surrogate is, in effect, a one-time artifact tied to the hypotheses and sampling policy used to generate its data. If the governing distribution changes, through a new parameter range, altered boundary or source terms, different operating regimes, or any shift that moves queries outside the original support, then the learned approximation no longer enjoys its reported error properties and must be rebuilt or at least revalidated (Quiñonero-Candela et al., 2022). Indeed, extrapolation has no enforceable bound, and even small covariate or concept shifts can invalidate risk estimates (Ovadia et al., 2019). This character also means that design of experiments, sampling density near regions of interest,

and coverage of rare but consequential behaviors dominate reliability. The surrogate minimizes expected error under the training distribution not worst-case error, so optimization or uncertainty quantification loops can be misled if they lead into sparsely sampled or unseen zones.

1.3 Objectives

High-fidelity multiphysics, multiscale simulations offer clear economic and environmental value but are constrained by computational cost, especially when submodels are invoked at every time step and mesh location. In particular, the fission-gas behaviour model in nuclear fuel-rod analysis is indispensable for fidelity yet contributes disproportionately to wall-clock time. Traditional surrogates can accelerate such simulations, but in practice they are commonly engineered ad hoc for a single study and parameter range, with little reuse of either the high-fidelity training data or the surrogate itself, an approach that is inefficient and hinders cumulative progress (von Rueden et al., 2020). Neural networks promise broader applicability by handling high-dimensional inputs and wide parameter domains with strong approximation capacity. However, their reliability remains conditioned on the training distribution, and silent failure modes under distribution shift motivate conservative, study-specific deployment. This thesis aims to contribute to breaking that impasse. The objective is to develop and validate a general-use neural-network surrogate that:

- **Accelerate high-fidelity simulations.** Materially reduce wall-clock cost of multiphysics, multiscale simulations by replacing the costly fission-gas behaviour model CARACAS with a neural-network surrogate while preserving system-level accuracy and stability.
- **Enable general-use surrogates beyond ad hoc practice.** Develop a reusable neural-network surrogate that remains valid across closely related studies and wider parameter ranges, addressing the current pattern of one-off, study-specific surrogates (von Rueden et al., 2020).
- **Provide a runtime operational-domain monitor.** Design and integrate an online mechanism to detect departures from the training support in the surrogate input space, thereby defining and enforcing the model operational domain.
- **Establish protective safeguards.** When out-of-domain use is detected, trigger safe behaviours such as abstention with fallback to the high-fidelity model, and data acquisition to expand coverage of future models.
- **Validate the approach in end-to-end simulations.** Demonstrate the combined methodology on a thermodiffusion simulation using OpenCalphad.

1.4 Thesis overview and structure

In this manuscript, I present the methodology developed to address the research objectives and the results obtained:

Chapter 2 presents the monitoring and runtime verification paradigm for neural networks and the methods used to monitor neural networks from general-purpose methods, such as methods for uncertainty quantification and out-of-distribution (OOD) detection, as well as methods used to monitor neural network surrogate models in simulation specifically.

Chapter 3 develops and validates a runtime monitoring approach, namely BBAS, grounded in bounding-box abstraction to flag OOD or Out-of-Model-Scope (OMS) inputs. After fixing terminology and positioning the method within interval abstractions and follow-ups, we analyze the geometry of neural-network-induced boxes and their link to activation patterns. Then, we present the construction of the monitor including clustering and a bounding-box anomaly score as implementable pipeline, illustrate it on the two-moon dataset, and extend the approach to CNNs and Transformers. Finally, we provide comprehensive experiments with ablations, clustering/parameter studies, and state-of-the-art comparisons.

Chapter 4 describes a metamodel of OpenCalphad for coupled thermodiffusion. We formalize the coupled thermodiffusion problem and introduce a partitioned two-stage simulation loop around OpenCalphad. We then develop a surrogate model covering data generation, preprocessing, methodology, and results. Next, we apply bounding-box abstraction to OMS detection and compare it against deep ensembles.

Chapter 5 builds a neural-network surrogate for fission-gas behaviour in fuel-rod simulations. It first characterizes CARACAS and its integration within ALCYONE, then presents the training dataset that it pre-processes for learning. The methodology involves the use of a specific trajectory-predictive surrogate model. Results show strong single-step and long-horizon accuracy, and stable in-simulation behaviour.

1.5 Contributions

Publication in Peer-Reviewed Conference Proceedings

- Yahiaoui, M., Giraldi, L., Daniel, G., Introïni, C., and Arbel, J. (2025b). Reliable neural network model for accelerating coupled thermodiffusion simulation. In *International Congress on Advances in Nuclear Power Plants (ICAPP)*

Submission in International Peer-Reviewed Journals

- Yahiaoui, M., Daniel, G., Giraldi, L., Bruyelle, J., and Arbel, J. (2025a). Bounding box anomaly scoring for simple and efficient out-of-distribution detection

  In final preparation for submission

Oral Presentations

- Yahiaoui, M.B, Daniel, G., Giraldi, L., Bruyelle, J., and Arbel, J., 2025.

  "Development and monitoring of a neural-network based surrogate model to accelerate fission gas simulation"

  International Congress on Advances in Nuclear Power Plants (ICAPP), 17-19 September 2025 , Antibes, France

- Yahiaoui, M.B, Daniel, G., Giraldi, L., Bruyelle, J., and Arbel, J., 2025.

  "Development and monitoring of a neural-network based surrogate model to accelerate fission gas simulation"

  Café Thésard du Département d'Etudes des Combustibles (DEC) du CEA du centre Cadarache, 11 juillet 2025 , Saint-Paul-lez-Durance, France

- Yahiaoui, M.B, Daniel, G., Giraldi, L., Introïni, C., and Arbel, J., 2024.

  "Reliable Neural Network Model for Accelerating Coupled Thermodiffusion Simulations"

  CEA-List Séminaire REVEAL 2024 | Sûreté des IA complexes, 12 Juillet 2024, Gif-sur-Yvette, France

- Yahiaoui, M.B, Daniel, G., Giraldi, L., Introïni, C., and Arbel, J., 2024.

  "Modèle de réseau de neurones fiable pour accélérer les simulations couplées de thermodiffusion"

  55ièmes Journées de Statistique de la Société Française de Statistique (SFdS), 27-31 mai 2024, Bordeaux, France.

- Yahiaoui, M.B, Daniel, G., Giraldi, L., Introïni, C., and Arbel, J., 2024.

  "Neural Network emulation of OpenCalphad for thermodiffusion simulation: Construction, challenges and uncertainties of Neural Network predictions"

  Séminaire UQ, rencontre DAM/DES, 02 avril 2024 , Saint-Paul-lez-Durance, France.

Chapter 2

Runtime verification and monitoring of neural networks surrogate models

Monitoring and runtime verification are techniques to ensure that systems behave safely and correctly when deployed in real-world applications. Monitoring in this context refers to continuously observing the inputs, outputs, and the internal states of a system during its operation, in order to detect anomalies, check compliance with requirements, or trigger safeguards. Runtime verification is a more formal approach where execution traces of the system are checked against specified properties or requirements during runtime. In other words, a runtime verification framework uses a monitor that watches the system execution and determines, after each observation, whether the behavior satisfies a desired property. In technical terms, the monitor provides a verdict, either satisfaction or violation, when the system reaches a terminal state, i.e. a state that, once assigned, never changes. Conceptually, runtime verification can be thought of as attempting proof by counter-example: the monitor cannot conclusively prove that a system always adheres to the specification. Instead, it only confirms violations or observes a finite compliance scenario.

Correctness properties. Monitors have two correctness properties. The first one is soundness: a monitor is sound if it never incorrectly issues verdicts about satisfaction or violation. Second is completeness: it guarantees that if the system violates a specification, the monitor will detect this violation within a finite execution. Completeness is tied to monitorability, a concept describing whether a property can be determined from a finite execution trace. Since monitors only have access to finite runtime traces, properties must be monitorable, meaning a finite prefix of the system execution must suffice to determine compliance or violation. Another essential concept related to runtime verification effectiveness is coverage, meaning how comprehensively the monitored properties capture the critical aspects of the system behavior. In practice, coverage determines whether the set of runtime specifications effectively captures potential system faults. Runtime verification techniques are considered "lightweight" verification, as they do not attempt to prove correctness upfront, but rather provide dynamic checks and assurances as the system runs

(Henzinger et al., 2023).

Runtime verification for deep learning. The use of runtime verification in deep learning is both promising and challenging. The wide use of deep learning component prompts a lot of questions about their reliability and safety. Runtime verification offers a principled yet pragmatic framework to improve safety and reliability of systems relying on deep learning. Deep learning systems, unlike traditional software, are data driven and lack explicit formally defined specification beyond statistical accuracy. The internal decision process of neural networks is often opaque and notoriously difficult to interpret or formalize into clear logical properties. Due to these inherent complexities, runtime verification approaches for deep learning typically do not attempt absolute correctness verification. Instead, deep learning runtime monitors aim to identify scenarios where models are likely unreliable, suspicious, or known to fail. For example, a monitor might check if the network outputs stay within safe bounds, or if the network confidence in its predictions is high enough. If a violation or anomaly is detected, the system can raise an alarm or activate a fallback mechanism. The main difficulty lies in the development of such a reliable monitor.

Applications and challenges. The field of monitoring neural networks is broad and multifaceted. In autonomy and robotics, runtime monitors supervise perception pipelines to catch failures and inconsistencies, surveys and recent methods span input/output checks, temporal logic monitors, and fault detection across cameras and LiDAR (Rahman et al., 2021; Antonante et al., 2023). In safety-critical control, formal runtime-verification frameworks monitor temporal-logic properties during operation (Schumann et al., 2015), while "shielding" enforces safety for reinforcement-learning agents by intercepting unsafe actions at runtime (Alshiekh et al., 2018).

Focus of this chapter. In this chapter, we narrow our focus to the monitoring of neural networks when they serve as surrogate models within physical simulations. Here, the challenges differ from those in perception or decision-making systems. The emphasis is on detecting extrapolation beyond training regimes, ensuring numerical stability in iterative solvers, and verifying adherence to conservation laws and physical constraints.

2.1 Physics and solver oriented monitoring

A fundamental problem with monitoring neural networks, particularly when they are used as surrogates in physical simulations, is the lack of a precise specification against which to verify their behavior. Traditional runtime verification relies on well-defined properties (e.g., "the system must respond within 2 seconds" or "the output must remain within these bounds"). However, for a neural surrogate, the very property of interest is often its accuracy with respect to the actual physics. This objective results in a paradox: one cannot simply specify "the model error must remain below some threshold," since the ground truth error is what is unknown during

deployment. Unlike classical solvers, where numerical accuracy and conservation laws can be guaranteed to a level, neural networks provide no such intrinsic assurances.

A way to address the absence of a direct error specification at runtime is to monitor practically verifiable system properties, analogous to the invariants and stability checks routinely enforced in conventional numerical solvers, namely:

- **Range checks:** Detect outputs that lie outside physically meaningful bounds and flag or clip them immediately.
- **Numerical instability:** Monitor the simulation state for indicators of divergence or abnormal growth (e.g., exploding norms, rapidly increasing step-to-step differences).
- **Physical invariants and conservation checks:** Compute conserved or invariant quantities (e.g., mass, energy, probability) from the surrogate output and verify that deviations remain within a prescribed tolerance.
- **Comparative monitors:** Periodically compare the surrogate predictions to a baseline or reference model evaluated at a lower frequency or resolution.

In all of these approaches, the monitor has a prescribed tolerance, and its response can range from halting the simulation upon violation to adjusting the surrogate output or switching to a high-fidelity model. Despite these strategies, important limitations remain when using neural network as surrogate models. Neural network surrogates are often designed to enforce physical invariances, conservation laws (Sturm and Wexler, 2022), or variable ranges directly through their architecture or training process. However, even when these conditions are satisfied, whether by construction or by monitoring, it does not necessarily guarantee that the simulation is accurate. A model may conserve mass or energy globally while still producing locally nonphysical fields, distorted phase relationships, or incorrect dynamics, meaning that adherence to invariants is necessary but far from sufficient for ensuring overall fidelity (Krishnapriyan et al., 2021; Gruver et al., 2022).

In summary, such monitors help prevent local errors from cascading into global instabilities and ensure that surrogate outputs remain physically plausible. Nevertheless, these methods are inherently reactive and limited in scope: passing a conservation or range check does not imply overall accuracy. As a result, these monitors should be seen as complementary safety layers rather than guarantees of correctness. They form the first line of defense in maintaining simulation integrity, while other techniques are needed to address data-driven risks and will be discussed in the following sections.

2.2 Uncertainty quantification for monitoring

While physics and solver oriented monitors safeguard against violations of known laws and numerical pathologies, they cannot address a more fundamental limitation of neural surrogates: the model may produce plausible yet incorrect outputs in regimes where it has little or no support from training data. To address this, monitoring

strategies turn to uncertainty quantification as a means of assessing the reliability of surrogate predictions at runtime. The central idea is that a surrogate should not only provide an output but also a measure of confidence, enabling the simulation workflow to detect when predictions are potentially unreliable.

2.2.1 Uncertainty in neural networks

Conventional supervised learning approaches seek a mapping $f : \mathbb{X} \to \mathbb{Y}$ from training data

$$\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n, \qquad x_i \in \mathbb{X},\ y_i \in \mathbb{Y}.$$

This paradigm results in trained models that produce only one value $\hat{y}$ as a prediction associated to each input x. However, a more complete and informative quantity that a model should output would be the predictive distribution:

$$p(y \mid x, \mathcal{D}_n),$$

whose spread reflects two fundamental sources of uncertainty: aleatoric and epistemic uncertainty.

Aleatoric uncertainty. This source of uncertainty represents the irreducible uncertainty inherent in the data. It refers to the notion of randomness and arises mainly due to either noise related to the process of capturing the data or the randomness of the phenomenon we are trying to predict (Malinin, 2019). It can also arise from an extra level of complexity not present in the data (e.g. missing features) (Hüllermeier and Waegeman, 2021).

In deep learning, aleatoric uncertainty can be directly integrated into the definition of the loss function. Consider a neural network with parameters $\theta \in \mathcal{W}$ that defines a predictive distribution for y given x. A common choice for regression is a Gaussian with input-dependent mean and variance,

$$y \mid x, \theta \sim \mathcal{N}\Big(\mu(x;\theta),\, \sigma^2(x;\theta)\Big), \tag{2.1}$$

where $\mu(x;\theta)$ models the signal and $\sigma^2(x;\theta)$ captures aleatoric uncertainty. Training proceeds by minimizing the negative log-likelihood (NLL):

$$L(\theta; \mathcal{D}_n) = \frac{1}{n} \sum_{i=1}^{n} \left[\frac{\Big(y_i - \mu(x_i;\theta)\Big)^2}{2\sigma^2(x_i;\theta)} + \frac{1}{2} \log \sigma^2(x_i;\theta) \right] + \text{const},$$

so the network learns both the mean and the variance from data. If the variance is assumed constant (i.e., $\sigma^2(x;\theta) \equiv \sigma^2$), the NLL reduces to the mean squared error (MSE):

$$L(\theta; \mathcal{D}_n) \propto \frac{1}{n} \sum_{i=1}^{n} \Big(\mu(x_i;\theta) - y_i\Big)^2.$$

The learning objective is then

$$\theta^* \in \arg\min_{\theta \in \mathcal{W}} L(\theta; \mathcal{D}_n).$$

Aleatoric uncertainty is qualified as irreducible because it is internal to the data itself and does not represent a lack of knowledge. On the contrary, it is an indivisible part of the information present in the data: adding additional data will only give us a more accurate estimation of this uncertainty but does not help reduce it.

Epistemic uncertainty. This source of uncertainty is related mainly to the model lack of understanding or knowledge of a specific input for which the model is making a prediction (Malinin, 2019). The first factor in epistemic uncertainty is related to the fact that the training dataset is just a sample of all the possible observable data and can be prone to sampling bias. Thus, adding/removing data or entirely resampling the distribution will give us different optimal parameters, ultimately giving rise to a distribution of values for the unknown set of parameters that describe the distribution accurately. The error that results from this is conventionally called **estimation error**. The second factor is that, the chosen class of models itself does not include the *true* function we seek to represent. The error that results from this is conventionally called **approximation error** (Shalev-Shwartz and Ben-David, 2014). These factors corroborate that uncertainty can be higher in regions of the input space where samples are sparse or limited in training data. The model might not have sufficient information to make confident predictions in these areas (Gal, 2016; Kendall and Gal, 2017). As a general rule of thumb, epistemic uncertainty in a particular region of space correlates inversely with the number of data samples present in that region. The principle of estimation error and approximation error in epistemic uncertainty is illustrated in Figure 2.1.

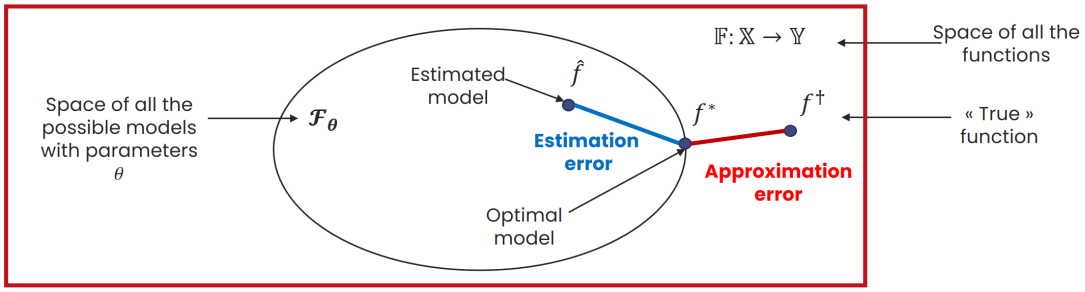


Figure 2.1: Sources of epistemic uncertainty.

Epistemic uncertainty is also called reducible uncertainty since it can be reduced by gathering more data or using a more sophisticated model. However, this reduction is limited by the available resources and the complexity of the actual function. Therefore methods for estimating and quantifying epistemic uncertainty are necessary.

This estimation is a challenging task in the context of deep learning and machine learning in general. The absence of a universally agreed-upon definition of uncertainty and a standardized way of quantifying it adds complexity to this challenge (Zidek and van Eeden, 2003). Methods based on probability theory, such as Bayesian inference (Arbel et al., 2024), are often used to represent and estimate uncertainty. Probability theory presents a natural way to quantify uncertainty and a systematic way to express our lack of complete knowledge. Probabilistic models are widely used in machine learning to estimate epistemic uncertainty.

In probabilistic modeling, epistemic uncertainty is captured by treating the model parameters as random variables. A prior $p(\theta)$ encodes assumptions about plausible parameter values before observing data, and a likelihood $p(\mathcal{D}_n \mid \theta) = \prod_{i=1}^{N} p(y_i \mid x_i, \theta)$ specifies how the data might be generated given θ. Bayes' rule then yields the posterior distribution

$$p(\theta \mid \mathcal{D}_n) \;\propto\; p(\mathcal{D}_n \mid \theta)\, p(\theta),$$

which aggregates prior assumptions with evidence from the data. This distribution is not specific to any data point but depends on the training data. We estimate epistemic uncertainty for a single input by measuring how much the variations in θ according to $p(\theta|\mathcal{D}_n)$ influence the output variations. A prediction influenced more by the variations of θ is less sure. In other words, the epistemic uncertainty of a model for an input can be represented by the variance of the model output over a sample of $p(\theta|\mathcal{D}_n)$. In the previous regression example (equation 2.1), the predictive mean and variance then decompose as

$$\mathbb{E}[y \mid x, \mathcal{D}_n] \;\approx\; \mathbb{E}_{\theta\sim p(\theta|\mathcal{D}_n)}\Big[\mu(x;\theta)\Big],$$

$$\mathrm{Var}(y \mid x, \mathcal{D}_n) \;\approx\; \underbrace{\mathbb{E}_{\theta\sim p(\theta|\mathcal{D}_n)}\Big[\sigma^2(x;\theta)\Big]}_{\text{aleatoric}} \;+\; \underbrace{\mathrm{Var}_{\theta\sim p(\theta|\mathcal{D}_n)}\Big[\mu(x;\theta)\Big]}_{\text{epistemic}}.$$

Here, the variability of $\mu(x;\theta)$ across plausible parameter values reflects epistemic uncertainty.

2.2.2 Epistemic uncertainty quantification and estimation

Direct estimation of $p(\theta|D)$ is not tractable in the case of neural network models due to the dimension of the parameters θ. Hence, Bayesian approximation methods such as Markov Chain Monte Carlo (MCMC) (Andrieu et al., 2003) and variational inference (VI) (Graves, 2011) are commonly used. Neural networks trained in this manner are called Bayesian neural networks (MacKay, 1992; Jospin et al., 2020; Arbel et al., 2024; Papamarkou et al., 2024). Influential uncertainty quantification methods such as MC-Dropout (Gal and Ghahramani, 2016) and Deep Ensembles (Lakshminarayanan et al., 2017) are approximations of Bayesian methods.

Other approaches have been developed in the literature that utilize the Bayesian framework. From this, we mention Deep Gaussian processes (Damianou and Lawrence, 2013) have recently generalized Gaussian processes (Rasmussen and Williams, 2006) and even recurrent processes (Mattos et al., 2016) to represent dynamic systems. In such methods, predictions correspond to a normal distribution. Thus, if the predicted variance is high, the confidence in the output is low. The main weakness of these methods lies in handling a large amount of data or high-dimensional data, as in our case.

Epistemic uncertainty quantification has been widely used in surrogate modeling as a way to assess the trustworthiness of predictions, flag out-of-distribution regimes, set conservative safety margins, and guide the collection of additional high-fidelity data or experiments where knowledge gaps are most significant (Abulawi et al., 2025; Olivier et al., 2021; Chae et al., 2025; Mondal and Sharma, 2025).

While these developments underscore the value of epistemic uncertainty quantification in enhancing the reliability and utility of surrogate models, important challenges and limitations remain that warrant closer examination. Epistemic uncertainty estimation and quantification is a dynamic and evolving area of research. Researchers are continuously working to develop more accurate, efficient, and comprehensive methods for understanding and quantifying epistemic uncertainty. Some recent reviews of uncertainty quantification methods are presented in Abdar et al. (2021); Gawlikowski et al. (2021). Nonetheless, estimating epistemic uncertainty remains an open problem (Postels et al., 2022).

2.2.3 Limitations of epistemic uncertainty quantification

Despite its value, epistemic uncertainty quantification in surrogate modeling faces several limitations. A key limitation is the compute burden, both in training and at inference. Fully Bayesian neural networks are expensive to train and deploy at scale, which pushes practitioners toward approximations (variational inference, MC-dropout, small ensembles) that can trade off calibration and robustness (Sharma and Jennings, 2021). At test time, epistemic uncertainty quantification often adds extra inference cost: MC-Dropout estimates uncertainty via multiple stochastic forward passes, Bayesian neural networks typically draw multiple posterior samples to approximate the predictive distribution and deep ensembles require evaluating multiple independently trained models per input. Some works have attempted to reduce the computation cost (training and inference) of ensembles by using specific model architectures (Laurent et al., 2023).

A further limitation is that epistemic uncertainty itself is often not well-calibrated: under distribution shift, common uncertainty methods can yield confidence that no longer tracks error. This is due to a problem known as **model misspecification** (Hüllermeier and Waegeman, 2021; Lahlou et al., 2021). Figure 2.2 illustrates a toy task for which the ground truth is available. In this task, the misspecified Gaussian Process (GP) gives an erroneous representation of uncertainty over unseen inputs.

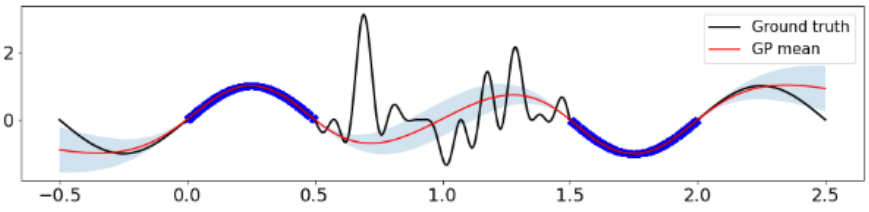


Figure 2.2: An example of model misspecification using a Gaussian Process (GP) fitted on the blue points. The blue points represent the result of a Ground truth function for some inputs in $[0, 0.5] \cup [1.5, 2]$. The fitted GP grossly underestimate the uncertainty over $[0.5, 1.5]$ (Lahlou et al., 2021).

As stated in Section 2.2.1, epistemic uncertainty has two significant factors: estimation and approximation error. Bayesian methods only account for estimation error. The other factor which describes the discrepancy between the chosen model and the actual function is not accounted for when using such methods, which leads to an

underestimation of epistemic uncertainty and possibly confidently wrong predictions. While one can recalibrate the reported uncertainty, that fix is inherently limited: recalibration mainly adjusts the variance-like part associated with estimation error. It does not remove the bias-like component induced by approximation error. The result is a well-tuned variance around a misspecified mean (Lahlou et al., 2021; Kato et al., 2022).

The question posed by epistemic uncertainty is: how confident can we be in the predictions or outcomes of a model, given the limitations in our knowledge? As we showed in this section, this question is challenging to answer. In our case, although relatively slow, calculating the actual function is possible using the baseline computation code. Therefore, we can calculate the proper answer if the model is uncertain. Hence, the decision-making process requires us to answer a binary question: for a specific input point, do we trust the output of the surrogate model or should we run the computation code instead? This question is conceptually easier to answer, since it does not require us to associate a number with the uncertainty of the model for an input. Consequently, this question opens the door to other methods that tackle the problem of epistemic uncertainty without necessarily quantifying it.

2.3 Out-of-Distribution (OOD) detection

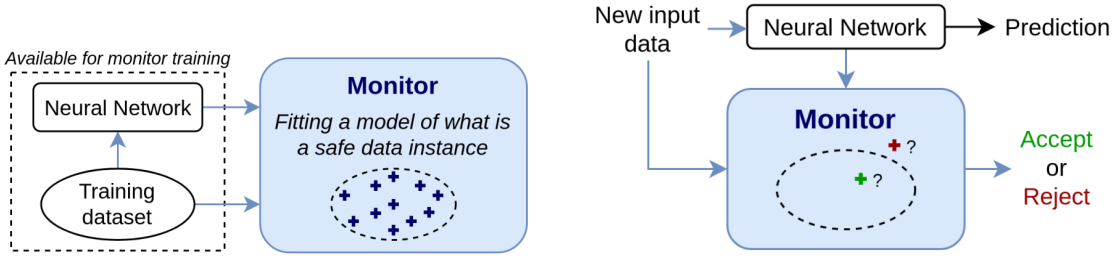
Out-of-distribution (OOD) detection, is a critical aspect of ensuring the reliability and safety of machine learning models. When deploying machine learning models in real-world applications, it is crucial to detect instances where the model encounters data that significantly deviates from the training distribution. Such scenarios may fall outside the scope of the model and can lead to unreliable predictions, potentially causing severe consequences in safety-critical applications like autonomous vehicles, medical diagnosis, or financial systems.

Detecting OOD samples is challenging because conventional machine learning models give predictions regardless if the data is beyond their training distribution or not. They generally provide erroneous predictions for inputs fundamentally different from the data seen during training.

Epistemic uncertainty estimation can serve as a natural detector for OOD samples. When a model encounters an input unlike anything seen during training, its prediction confidence should decrease due to the inherent lack of knowledge about such inputs. High epistemic uncertainty signals that the model recognizes the input as unfamiliar, a characteristic shared with OOD samples.

Other methods have been proposed, such as density-based (Lee et al., 2018), distance-based (Papernot and McDaniel, 2018), and reconstruction-based (Denouden et al., 2018) methods, to name a few. Recent works also exploit the phenomenon of "neural collapse" and the underlying geometrical properties to propose OOD detection methods (Ammar et al., 2023). A recent line of work proposes to combine statistical notions of data depth with distance measures in feature space. For example, Nguyen et al. (2024) introduce the *Lens Depth* (LD) combined with Fermat distance to quantify out-of-domain uncertainty in neural network predictions. Their method is appealing because it is non-parametric, requires no retraining, and can be applied post-hoc to any classifier. On standard benchmarks, LD-Fermat yields competitive

or superior uncertainty estimates compared to strong baselines, while remaining computationally lightweight. A survey of OOD detection methods and an OOD detection method benchmark are respectively presented in Yang et al. (2024, 2022).



(a) Monitoring at inference time (runtime). (b) Monitor training (design time).

Figure 2.3: Neural network runtime monitoring is a research area aiming to characterize safe data instances, in order to reject unsafe predictions during inference (Guérin et al., 2023)

This type of method for OOD detection can be integrated into a monitor to detect possible unsafe data. Figure 2.3 shows the typical functioning of a runtime monitor. It constantly evaluates the incoming data or observations to determine whether they fall within the expected distribution of the system training data. The monitor assesses the internal behaviour of the neural network when it makes predictions on incoming data points and compares it to the typical behaviour of the neural network on in-distribution data. When it encounters data that significantly deviates from the expected behaviour, it raises an alert or takes a predetermined action to mitigate potential issues. Runtime monitors for OOD detection are essential in identifying and addressing unexpected inputs or conditions, thus enhancing the robustness and reliability of neural networks in unanticipated situations.

2.4 Out-of-Model-Scope (OMS) detection

The definition of the in-distribution domain is ambiguous, as it is unclear what level of deviation from the training data is necessary to call an input OOD. Adversarial attacks add small perturbations specifically designed to mislead the neural network model to the input data, leading to a performance drop (Chaouai et al., 2022; Xu et al., 2020). These adversarial examples represent a challenge to OOD detection, as rejecting them will lead to rejecting inputs where a robust model should provide correct predictions but accepting them using a non-robust model would lead to erroneous predictions.

Guérin et al. (2023) make a case for Out-of-Model-Scope (OMS) detection as a better conceptual approach to monitoring compared to out-of-distribution detection. They define the model scope as the set of data points where the model output is correct. Figure 2.4 gives an illustration of how prioritizing OOD detection can lead away from the main goal of identifying potentially incorrect predictions.

So the task of OMS detection is to design a monitor that detects possible erroneous outputs even if the inputs belong to the training distribution. The OOD detection

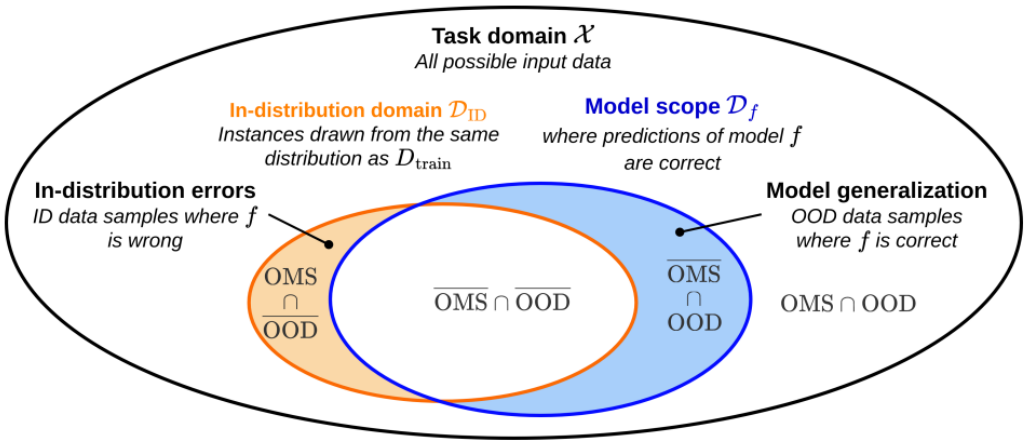


Figure 2.4: Differences between OOD and OMS detection (Guérin et al., 2023).

setting assumes the model will have high accuracy on points similar to training data. However, this assumption can be wrong, mainly when the training data contains outliers or the model performs poorly on some instances of training data. Furthermore, the OOD detection framework does not consider the generalisation capabilities of the model, which may lead to the rejection of some perfectly valid predictions.

Chapter 3

Developing and validating a runtime monitoring approach based on bounding boxes abstraction and activation patterns

As presented in the previous chapter, epistemic uncertainty refers to the uncertainty that arises due to a lack of knowledge or information. It is associated with our limited understanding of a system or a process and with the inherent limitations of model architecture and training data sampling. The main challenge discussed is the dependence of the model on the distribution of the training data. For example, a metamodel construction, such as a neural network, relies on the database used in the training phase. When the neural network is deployed in a production setting, one must ensure that the new evaluations of the metamodel are valid, especially for instances that stray from the training distribution. Hence, there is a need for a well-defined operational domain that can be used to identify potential prediction errors before they can propagate through the system. Pointedly, the task of OMS (Out-of-Model Scope) detection gives the conceptual framework that aligns best with this endeavour.

In this chapter, we propose to build a runtime monitor for OOD and OMS detection for several compelling reasons. An efficient runtime monitor offers real-time monitoring capabilities, enabling us to promptly detect and respond to deviations or anomalies during system execution, ensuring relevance and effectiveness as the system behaviour evolves. Furthermore, the runtime monitor provides comprehensive insights into the system activities and interactions, facilitating in-depth analysis and informed decision-making. This choice is motivated by the need for a responsive and versatile detection solution tailored to the demands of our specific application.

We introduce a novel runtime monitoring approach based on activation patterns (Raghu et al., 2017) and bounding boxes (Henzinger et al., 2020). While most existing methods treat the neural network as a black box, the proposed approach tries to detect novel input by observing the hidden features of the neural network. The proposed method leverages the internal activation patterns of the neural network and bounding boxes to ensure that the network behaves as expected during runtime.

The basic idea behind it is grouping training data into clusters according to how they activate the neurons in the neural network and then defining the acceptable input domain by defining lower and upper bounds on the hidden features of the neural network over each one of these clusters. Hence, the constructed monitor will reject an input if the hidden features resulting from this input forward pass through the model are not within the bounds of any cluster. Our goal in this chapter is to give a mathematical explanation and justification for this method by illustrating how constructing such a monitor is akin to defining a process that checks if a point belongs to an envelope encapsulating the training data. This approach ultimately aims to enhance the reliability of neural networks during their deployment and execution. A specific benefit of this approach is that it does not require a specific training process, it can be applied in a post-hoc manner on models that are already trained.

3.1 Basic terminology and notation

Feed-forward neural networks are a special type of parameterized functions that takes an input and produces an output:

$$\mathcal{N} : X \to Y.$$

Typically, we have

$$X = \mathbb{R}^{n_{\text{in}}} \text{ and } Y = \mathbb{R}^{n_{\text{out}}},$$

where n_{in} is the dimension of the input and n_{out} is the dimension of the output.

A k-layer neural network $\mathcal{N}_k$ is defined as a composition of functions $f^{(l)}$ associated to different layers l:

$$\begin{aligned} \mathcal{N}_k(x) &= g \circ f^{(k)} \circ ... \circ f^{(2)} \circ f^{(1)}(x), \\ f^{(l)}(z^{(l-1)}) &= \sigma(W^{(l)} z^{(l-1)} + b^{(l)}), \\ z^{(l)} &= f^{(l)}(z^{(l-1)}), \\ z^{(0)} &= x, \end{aligned}$$

where $b^{(l)}$ is a bias column vector in $\mathbb{R}^{n^{(l)}}$, $W^{(l)}$ is the $n^{(l)} \times n^{(l-1)}$ weight matrix, $z^{(l-1)}$ is the input of layer l, σ is the activation function of the intermediate layers (it is the same for all of the intermediate layers), and g is the output activation function, it can be different from the activation function of the intermediate layers. Each function $f^{(l)}$ maps its inputs from $\mathbb{R}^{n^{(l-1)}}$ into $\mathbb{R}^{n^{(l)}}$

$$f^{(l)} : \mathbb{R}^{n^{(l-1)}} \to \mathbb{R}^{n^{(l)}}.$$

In this section, we will look at the behaviour of the ReLU activation function. To simplify, we adopt the same notation for $\sigma : \mathbb{R} \to \mathbb{R}$ and $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ (element-wise function) such as for $x \in \mathbb{R}^n$,

$$\sigma(x) = (\sigma(x_i))_{i=1,...,n}.$$

Each layer $f^{(l)}$ is composed of $n^{(l)}$ neurons, each neuron can be represented as function $f_i^{(l)} : \mathbb{R}^{n^{(l-1)}} \to \mathbb{R}$ such as

$$f_i^{(l)}(z^{(l-1)}) = \sigma(W_i^{(l)} z^{(l-1)} + b_i^{(l)}),$$

where $W_i^{(l)}$ is the i^{th} line of $W^{(l)}$.

3.2 Bounding Box abstraction: foundations, advances, and impact

Henzinger et al. (2020) introduces a runtime monitoring approach for neural networks aimed at detecting novel inputs. The central idea is to observe a network hidden layer activation during operation and flag inputs that produce behaviours "outside the box". Henzinger et al. (2020) is strongly inspired by Abstract interpretation of software systems.

3.2.1 Interval abstraction: foundations and applications in deep learning

Interval abstraction is a static program analysis technique that approximates the range of possible values for program variables. Rather than tracking exact values, which can become computationally infeasible, interval abstraction represents each variable using an interval $[a, b]$, capturing all potential values the variable might assume during program execution. Analyzing these intervals makes it possible to detect potential runtime errors, such as division by zero or array out-of-bounds conditions, enabling proactive error identification.

Abstract interpretation, introduced by Cousot and Cousot (1976), provides the formal foundation for interval abstraction. Abstract interpretation supplies a systematic framework for creating a sound approximation of a program's concrete semantics, known as the abstract semantics. It accomplishes this by operating within an abstract domain and using two key mappings: an abstraction function and a concretisation function. The abstraction function maps a set of real values onto the smallest interval that encompasses them, while the concretisation function performs the reverse operation, mapping an interval back to the set of real values it represents.

A critical result from abstract interpretation is its soundness theorem, which guarantees that applying sequences of operations within the abstract domain, known as abstract transformers, results in intervals that safely over-approximate all possible outcomes of the corresponding concrete operations. This over-approximation ensures that if a correctness property is verified using the abstract intervals, it also holds for all concrete inputs.

However, interval abstraction can be overly conservative, sometimes significantly overestimating variable ranges. This issue, known as the dependency problem or wrapping effect in Krukowski et al. (2024), arises because interval abstraction typically treats variables independently, thereby losing precision through an accumulation of approximations. Consequently, if an interval-based analysis indicates that a property

does not hold, it does not necessarily imply that all concrete inputs violate the property.

To mitigate the wrapping effect, researchers have proposed refined abstract domains, such as zonotopes and affine arithmetic (De Figueiredo and Stolfi, 2004), that explicitly capture linear correlations among variables. While these refined methods increase computational complexity, they considerably reduce the independence-induced inaccuracies, resulting in tighter and more precise interval estimations.

In deep learning, interval abstraction typically involves propagating intervals through a neural network—a process commonly known as Interval Bound Propagation (IBP). IBP propagates input ranges through the network layers, resulting in an interval (or box) that encloses all possible outputs for a given range of inputs. This method is particularly valuable for uncertainty quantification, robustness analysis, and formal verification.

In uncertainty quantification, IBP explicitly represents uncertainties in inputs or model parameters as intervals. The network propagates these intervals to produce output intervals that function as confidence bands around predictions. For instance, Interval Neural Networks (INNs) serve as deterministic, lightweight wrappers that can be added to any trained neural network, enabling the generation of interpretable uncertainty estimates. Specifically, INNs operate by slightly perturbing the input into an interval range, then observing the resulting interval outputs as measures of uncertainty. Oala et al. (2021) illustrate this approach in image reconstruction, where INNs successfully identified regions within medical images likely to contain reconstruction errors, as evidenced by significantly widened output intervals. Similarly, Betancourt and Muhanna (2021) introduce Deep Interval Neural Networks (DINNs), a method explicitly designed to train neural networks to produce interval outputs given interval inputs. In DINNs, both the network's inputs and weights can be treated as intervals during training, and the loss function explicitly penalizes large output intervals, ensuring tight, meaningful uncertainty bounds. Their experimental results in safety-critical scenarios, such as air pollution forecasting using uncertain sensor data, demonstrate DINN's capability to generate accurate and reliably bounded predictions.

In robustness analysis and formal verification, interval abstraction plays a dual role: first, it serves as a means of generating formal certificates that verify the network exhibits desired behavior within specific input ranges; second, it provides a systematic training mechanism designed explicitly to achieve these certified behaviors. Specifically, by propagating intervals through the network, IBP ensures that all potential outputs within an input range remain within predetermined safety or performance bounds.

3.2.2 Bounding box abstraction

Bounding box abstraction (Henzinger et al., 2020) extends interval abstraction concepts to the context of novelty detection in machine learning. While interval abstraction traditionally uses sound approximations to verify program safety within defined boundaries, bounding box abstraction utilizes similar principles for monitoring purposes rather than verification. For verification, the central question is whether a

specific variable or group of variables satisfies a given property within a defined input interval. In contrast, bounding box abstraction focuses on dynamically monitoring whether variables remain within normative boundaries established from previously observed safe data points and flags inputs that cause variables to exceed these predetermined boundaries, thus serving primarily as a runtime check rather than a formal verification method. For instance, a simple implementation might establish intervals by taking input variables' minimum and maximum values over a set of safe datapoints (e.g. a subset of the training data). This results in a monitor that accepts all the points within a hyperrectangular region in the input space encompassing known safe datapoints, and rejects the rest.

A critical aspect of bounding box abstraction, inherited from interval abstraction, is the concept of overapproximation. While interval abstraction ensures the safety of all points within a verified interval, similarly bounding box abstraction guarantees acceptance of its defining dataset. However, it does not guarantee overapproximation for all non-novel data. To address this, Henzinger et al. (2020) introduce the concept of abstraction convergence: an abstraction converges when adding new points no longer expands it. In such cases, the monitor accepts all non-novel data; thus, if the monitor rejects an input, the input must be novel. A non-converged abstraction is, therefore, in the peculiar position of being simultaneously overapproximated yet overly restrictive. To manage this issue, Henzinger et al. (2020) propose grouping feature vectors into clusters using k-means and encapsulating each cluster with a bounding box. An input is accepted if it falls within any of these boxes. Employing multiple smaller boxes allows the monitor to more accurately capture non-convex support in the feature space, reducing overapproximation. Additionally, to mitigate the restrictive nature of the abstraction, the authors propose enlarging the bounding boxes by an enlargement factor. Consequently, tuning the monitor through clustering granularity or the enlargement factor provides a flexible trade-off between recall and precision in novelty detection.

Henzinger et al. (2020) introduce bounding box abstraction within a classification context for monitoring a pre-trained neural network. They base their approach on the assumption that layers near the output of a neural network tend to form class-specific patterns (Zeiler and Fergus (2014)). Following this assumption, Henzinger et al. (2020) utilize the outputs of neurons in a fully connected layer close to the network's output as monitoring variables. To create the bounding boxes, they first select subsets D_k from the training set, where each subset consists of data points correctly classified as belonging to class k. They then extract the corresponding neuron outputs at the selected layer, resulting in labeled vectors. In the next phase, they use a clustering algorithm to group the labeled vectors. The authors note that the applying this clustering significantly enhances precision. Specifically, their implementation employs k-means clustering, which requires specifying the number of clusters beforehand. To determine an optimal cluster count, they iteratively increase the number of clusters until the relative improvement of the sum-of-squares metric falls below a predefined threshold, which serves as an input parameter. Consequently, multiple sets of bounding boxes are constructed, with each set dedicated to a particular class k. When inputs are propagated through the neural network at runtime, the monitoring procedure observes the output vector $v^{(l)}$ from layer l. It then verifies whether this

output vector falls within any bounding box constructed for the predicted class $\hat{k}$. If the vector is contained within a bounding box, the monitor accepts the input, indicating that the network has processed the input in a usual manner compared its training behavior. If not, the monitor rejects the input, indicating that the network processed it unusually.

3.2.3 Literature review

Since Henzinger et al. (2020), several researchers have published important follow-up work, notably Wu et al. (2023)'s introduction of "resolution boxes". This work aims to systematically tune the clustering granularity of box abstractions. Wu et al. (2023) formalize the notion of a box's resolution: essentially, each box's size (or coverage) relative to the data. They define a clustering coverage metric that quantitatively measures how well an abstraction covers the known data in feature space. By analyzing the coverage as a function of the clustering parameters, they can identify ranges of the number of clusters (or distance thresholds) that are "sub-optimal", thus guiding the user to select a near-optimal abstraction without exhaustive trial-and-error. Importantly, Wu et al. (2023) also explore using correct and incorrect behaviors in the network to build monitors. In other words, they cluster the correctly classified training activations and examine how known failure cases (if available) or misclassified samples are distributed in the activation space to assess the separability of classes and the risk of overlap between class abstractions. Their method can flag situations where the abstractions for different classes are too close or overlapping by checking the size and intersection of the constructed boxes, potentially revealing ambiguous regions in the feature space. The result is a more systematic way to configure monitors, moving toward automated monitor design.

Wu et al. (2024) extend this line and apply the Outside-the-Box idea to object detectors, which output multiple proposals per image rather than a single classification. They manage this by constructing feature-space boxes for each predicted object class in a detection model (like Faster R-CNN). They show that a union of convex boxes could form a non-convex decision boundary in the feature space that effectively filters out anomalous detection proposals, outperforming prior OOD methods adapted from classification to detection.

Hashemi et al. (2024) look at combining abstraction-based monitors with other detection methods. They observe that Henzinger et al. (2020)'s monitor considers correlations between neurons by clustering whole activation vectors. The other approach is to monitor each neuron individually for out-of-range values (e.g., modeling each neuron's activation distribution as a Gaussian from training). The Gaussian-based method flags a neuron when its value lies far in the tail of that neuron's training distribution. Hashemi et al. (2024) propose a hybrid monitor that leverages both mechanisms: it checks each neuron against its typical range and checks the overall pattern against known clusters. By combining these, they aim to improve detection. Intuitively, the per-neuron Gaussian test might catch extreme outliers that still incidentally fall inside a coarse box. In contrast, the box clusters catch dependency anomalies that per-neuron checks would miss.

These approaches operate on representations from a single layer, typically the

pre-final layer. In the next section, we develop a rigorous geometrical interpretation of this type of method, clarifying the role of feature geometry in OOD separability. Building on this perspective, we generalize the approach to multiple layers, rather than relying solely on the pre-final embedding. This multi-layer extension both broadens the theoretical foundation of the method and is expected to yield more robust detection.

3.3 On the geometric structure of neural network bounding box

The bounding box abstraction defines an operational domain through the hidden features of the neural network. The idea is that the hidden features are all continuous functions from the input space to the set of real numbers. So, defining upper and lower bounds on this set of functions to construct a bounding box implies defining a domain in the input space representing the intersection of a series of sub-level and super-level sets.

When an instance propagates through a neural network, each neuron takes the previous layer outcome and linearly transforms it before passing it through an activation function. We can consider this series of linear transformations as a mapping that we can use to define a bounding box over some point set.

In this section, we use the notations described in section 3.1.

Definition 1. *(Neural network transformation) Let $\mathcal{N}$ be a neural network with fixed trainable parameters θ, the transformation $\phi_\theta : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{\#\text{neurons}}$ is defined as:*

$$\phi_\theta(x) = (\phi^{(l)}_{\theta,i}(x))_{i,l} = (W^{(l)}_i z^{(l-1)}(x) + b^{(l)}_i)_{i,l}. \tag{3.1}$$

Definition 2. *(Neural network bounding box) Let $\mathcal{N}$ be a neural network, a bounding box over a finite point set $A \subset \mathbb{R}^{n_{\text{in}}}$ and for a fixed vector of trainable parameters θ is*

$$\mathcal{B}(A, \theta) := \prod_{l,i} \left[\min_{x \in A}(\phi^{(l)}_{\theta,i}(x)),\ \max_{x \in A}(\phi^{(l)}_{\theta,i}(x)) \right].$$

The region of space delimited by $\mathcal{B}(A, \theta)$ is

$$\mathcal{R}_{\mathcal{B}}(A, \theta) := \{x \in \mathbb{R}^{n_{\text{in}}} \mid \phi_\theta(x) \ \in \mathcal{B}(A, \theta)\}.$$

Definition 3. *(Single layer bounding box) The bounding box over a layer l of $\mathcal{N}$ over a finite point set $A \subset \mathbb{R}^{n_{\text{in}}}$ and for a fixed vector of trainable parameters θ is*

$$\mathcal{B}^{(l)}(A, \theta) := \prod_{i} \left[\min_{x \in A}(\phi^{(l)}_{\theta,i}(x)),\ \max_{x \in A}(\phi^{(l)}_{\theta,i}(x)) \right].$$

The region of space delimited by $\mathcal{B}^{(l)}(A, \theta)$ is

$$\mathcal{R}^{(l)}_{\mathcal{B}}(A, \theta) := \{x \in \mathbb{R}^{n_{\text{in}}} \mid \phi^{(l)}_\theta(x) \ \in \mathcal{B}^{(l)}(A, \theta)\}.$$

The shape of $\mathcal{R}_{\mathcal{B}}(A, \theta)$ depends on the architecture of the neural network and on the parameters θ. In this section, we will analyse the resulting shape of bounding box in the input space. To illustrate the approach, we train a neural network on the two-moons classification task (Pedregosa et al., 2011) and show the applications of these concepts to this dataset. The neural network has four layers each with 16 neuron and a ReLU activation function. This example will serve as a running illustration throughout the rest of the section.

3.3.1 Bounding box over the first layer

The shape of the region corresponding to the bounding box over the first layer can be explicitly defined.

Lemma 1. *For $\mathcal{N}$ be a neural network and $A \subset \mathbb{R}^{n_{\text{in}}}$ a finite point set, we have:*

$$\mathcal{R}_{\mathcal{B}}^{(1)}(A, \theta) \text{ is a polytope} \iff \operatorname{rank}(W^{(1)}) = n_{\text{in}}.$$

We note that the stated condition of full-rank is almost always true since the number of neurons of the first layer is often significantly higher than the input dimension.

Proof. By definition

$$\mathcal{R}_{\mathcal{B}}^{(1)}(A, \theta) = \{x \in \mathbb{R}^{n_{\text{in}}} \mid \min_{x \in A}(\phi_{\theta,i}^{(1)}(x)) - b_i^{(1)} \leq W_i^{(1)} x \leq \max_{x \in A}(\phi_{\theta,i}^{(1)}(x)) - b_i^{(1)}\}. \quad (3.2)$$

The right–hand side is an intersection of $2n^{(1)}$ closed half-spaces, hence always a convex polyhedron. It is a *polytope* precisely when it is bounded. We demonstrate below that this region is indeed bounded if and only if $\operatorname{rank}(W^{(1)}) = n_{\text{in}}$.

$\Longrightarrow$) Assume that $\operatorname{rank}(W^{(1)}) < n_{\text{in}}$. Then there is a non-zero vector $v \in \ker W^{(1)}$. Pick any $x_0 \in \mathcal{R}_{\mathcal{B}}^{(1)}(A, \theta)$ and note that for every $t \in \mathbb{R}$,

$$\phi_{\theta}^{(1)}(x_0 + tv) = W^{(1)}(x_0 + tv) + b^{(1)} = W^{(1)}x_0 + b^{(1)} \in \mathcal{B}.$$

Thus $x_0 + tv \in \mathcal{R}_{\mathcal{B}}^{(1)}(A, \theta)$ for all $t \in \mathbb{R}$, so the region contains an entire affine line and is unbounded. Consequently, if $\mathcal{R}_{\mathcal{B}}^{(1)}(A, \theta)$ is a polytope, we must have $\operatorname{rank}(W^{(1)}) = n_{\text{in}}$.

$\Longleftarrow$) Conversely, suppose $\operatorname{rank}(W^{(1)}) = n_{\text{in}}$, so $W^{(1)}$ is injective. Let $\sigma_{\min} > 0$ be its smallest singular value. Then:

$$\|W^{(1)}x\|_2 \;\geq\; \sigma_{\min}\|x\|_2 \qquad \forall\, x \in \mathbb{R}^{n_{\text{in}}}.$$

Let:

$$M := \max_{i=1,\ldots,n^{(1)}} \max\Big\{ \,|\min_{x \in A}(\phi_{\theta,i}^{(1)}(x)) - b_i^{(1)}|,\; |\max_{x \in A}(\phi_{\theta,i}^{(1)}(x)) - b_i^{(1)}| \,\Big\},$$

so that for every x satisfying (3.2), $\|W^{(1)}x\|_2 \leq \sqrt{n^{(1)}}\, M$. Note that M is correctly defined because A is a finite set. Combining with the inequality above gives

$$\|x\|_2 \;\leq\; \frac{\sqrt{n^{(1)}}\, M}{\sigma_{\min}},$$

a uniform bound for all $x \in \mathcal{R}_{\mathcal{B}}^{(1)}(A, \theta)$. Hence the set is bounded, and being an intersection of finitely many half-spaces, it is indeed a polytope. □

The level sets we show in Figures 3.1 represent lines with the same value over the grid for a neuron as the lower or upper bound over the training data of each neuron in the first layer of the neural network trained on the two-moons example specified earlier. We note that the envelope formed by the bounding box of this first layer is a convex polygon.

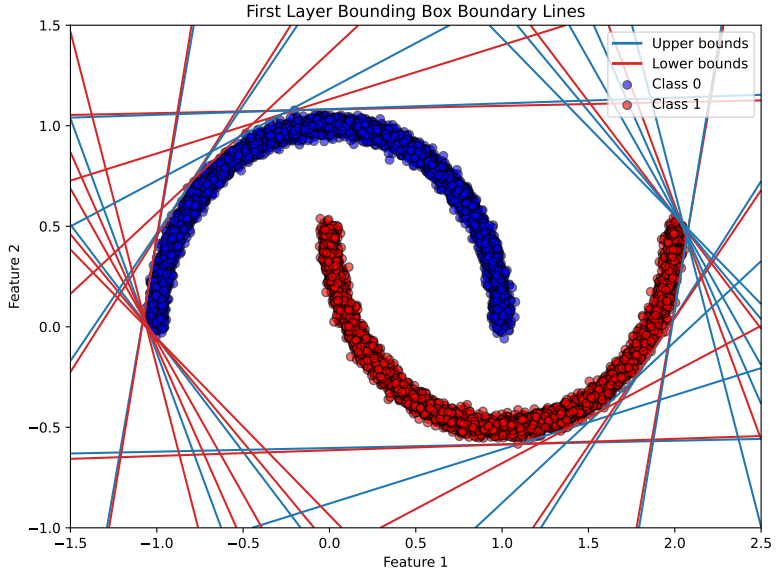


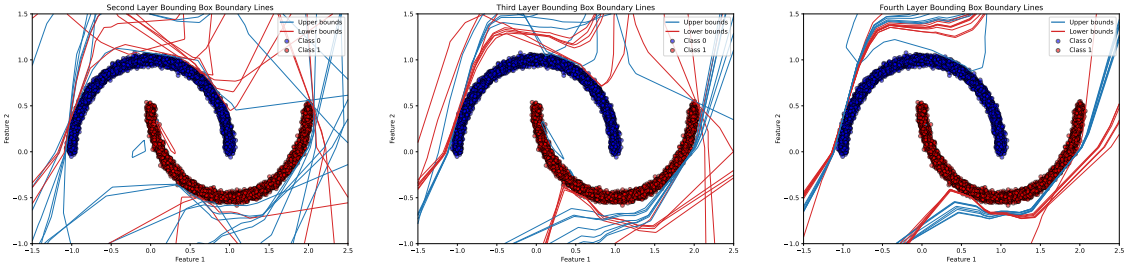
Figure 3.1: Bounding box of the first layer, level sets in blue represent the upper bounds and in red lower bounds over the whole dataset.

Figure 3.2 shows the levels sets for the next layers. The shape of the bounding boxes for the other layers does not exhibit a specific structure. In the next section, we will discuss the geometrical origine/interpretaion of these shapes.

3.3.2 Activation patterns

A neural network composed entirely of linear activation functions behaves purely as a linear transformation, mapping inputs directly to outputs through linear combinations. Consequently, when the activation functions are piecewise linear, such as ReLU, the neural network becomes piecewise linear. Since, in every region of the input space where all activations are linear simultaneously, the network operates as a linear function locally.

In such piecewise linear networks, the input space is partitioned or "shattered" into distinct regions, each characterized by a unique linear relationship between



(a) Second layer (b) Third layer (c) Fourth layer

Figure 3.2: Neural Network bounding boxes of the other layers, level sets in violet represent the lower bounds and in red the upper bounds over the whole dataset.

inputs and outputs. For networks employing ReLU activations, boundaries between these regions occur where at least one ReLU unit transitions from inactive (output zero for negative input) to active (output equal to input for positive input). Each unique combination of ReLU activations, active and inactive neurons, defines an *activation pattern*. Correspondingly, each activation pattern specifies a distinct *activation region*, which comprises all input points that produce identical patterns of neuron activations for a given neural network configuration with fixed parameters.

Definition 4. *(Activation Patterns/Regions) Let $\mathcal{N}$ be a neural network with* ReLU *activation and an identity output activation. An activation pattern for $\mathcal{N}$ is the assignment of a bit to each neuron:*

$$a := (a_i^{(l)}) \in \mathcal{A} := \{0,1\}^{\#\text{neurons}}.$$

For fixed trainable parameters θ, there is a mapping $\Phi_\theta : \mathbb{R}^{n_{\text{in}}} \to \mathcal{A}$, such as

$$\Phi_\theta(x) := a \quad \textit{with} \quad a_i^{(l)} = \begin{cases} 0 & \textit{if} \quad z_i^{(l-1)} \leq 0 \\ 1 & \textit{if} \quad z_i^{(l-1)} > 0. \end{cases}$$

The activation region corresponding to a and θ is

$$\mathcal{R}(a,\theta) := \Phi_\theta^{-1}(a) = \{x \in \mathbb{R}^{n_{\text{in}}} | \Phi_\theta(x) = a\}.$$

Understanding the activation regions within a neural network is essential for understanding the geometric structure and overall behavior of the function it represents. Specifically, these activation regions determine the shape and segmentation of the neural network's functional domain across its multiple layers.

The activation region is a convex polyhedron resulting from the intersection of half-spaces defined by the linear inequalities representing the neurons' activation states.

Lemma 2. *(Convexity of activation regions) Let $\mathcal{N}$ be a* ReLU *neural network. Then every activation region $\mathcal{R}(a,\ \theta)$, with an activation pattern a and a vector of trainable parameters θ, is a convex polyhedron.*

The proof follows directly from the definition of a convex polyhedron.

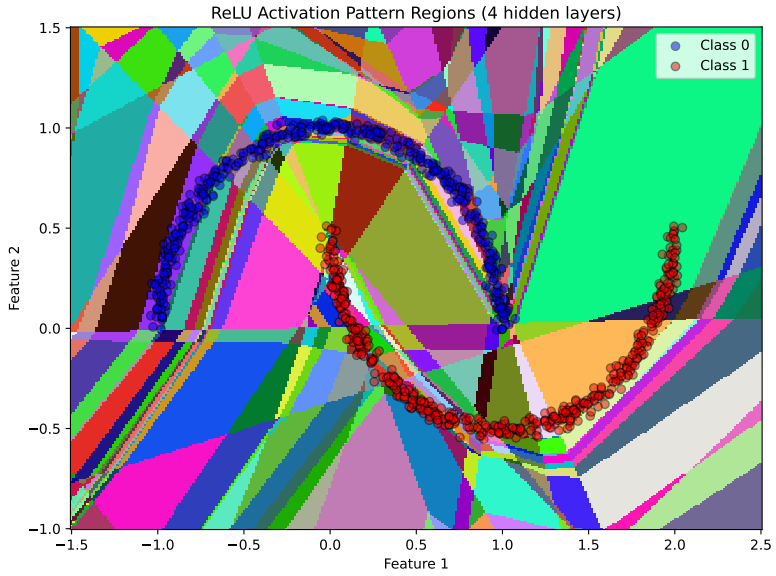


Figure 3.3: Activation regions of the neural network, on each partition of space the activation pattern is constant.

Returning to our two-moon example, Figure 3.3 illustrates the activation regions of the neural network, each activation region is represented in a different colour. We note that they form convex polyhedra.

We can extend lemma 1 to all layers for points inside the same activation region using the fact that the neural network is linear on said activation region:

Lemma 3. *For $\mathcal{N}$ be a neural network and $A \subset \mathbb{R}^{n_{\text{in}}}$ a finite point set such as there is an activation pattern a with non-empty activation region such as $A \subset \mathcal{R}(a, \theta)$, $\mathcal{R}_{\mathcal{B}}(A, \theta)$ is convex and*

$$\mathcal{R}_{\mathcal{B}}(A, \theta) \text{ is a polytope} \iff \operatorname{rank}(W^{(1)}) = n_{\text{in}}.$$

Consequently, a region enclosed by a bounding box, defined over a finite set of points and a neural network where the weight matrix of the first layer has full rank relative to the input space, forms a union of bounded polytopes. These polytopes correspond to activation regions determined by the possible activation patterns of points within the bounding box. For example, if the set of points exhibits only two adjacent activation patterns a, meaning they differ at most in one bit, the region enclosed by the bounding box will consist of exactly two bounded polytopes. Conversely, if there are three adjacent activation patterns, meaning they differ at most in two bits, the region will comprise at most four bounded polytopes, depending on whether the fourth possible activation pattern has a nonempty activation region.

Formally. Let $d \in \mathbb{N}^*$ the number of neurons in a neural network and let a bounding box

$$\mathcal{B}(A, \theta) \;=\; \Big\{ x \in \mathbb{R}^d \;\Big|\; \ell_i \leq x_i \leq u_i \text{ for all } i = 1, \ldots, d \Big\},$$

be given, with lower and upper bounds $\ell = (\ell_1, \ldots, \ell_d)$ and $u = (u_1, \ldots, u_d)$ satisfying $\ell_i \le u_i$. Define

$$P \;:=\; \Big\{ i \in \{1, \ldots, d\} \,\Big|\, \ell_i < 0 < u_i \Big\}, \qquad n_b \;:=\; |P|,$$

so n_b counts the coordinates in which the box spans both negative and positive values (i.e. "crosses the origin").

For every subset $S \subseteq P$ let

$$B_S \;:=\; \Big\{ x \in \mathcal{B}(A, \theta) \,\Big|\, x_i \ge 0 \text{ if } i \in S,\;\; x_i \le 0 \text{ if } i \in P \setminus S \Big\}.$$

Then each B_S lies entirely inside a single orthant (all coordinates have a fixed sign pattern), and

$$\mathcal{B}(A, \theta) \;=\; \bigcup_{S \subseteq P} B_S, \qquad \text{with at most } 2^{n_b} \text{ (non-empty) boxes } B_S.$$

Consequently:

$$\left[\mathcal{R}_{\mathcal{B}}(A, \theta) \;=\; \bigcup_{S \subseteq P} \mathcal{R}_{\mathcal{B}_S} \quad \text{with at most } 2^{n_b} \text{ (non-empty) polytopes } \mathcal{R}_{\mathcal{B}_S} \right] \iff \left[\operatorname{rank}\!\left(W^{(1)}\right) = n_{\text{in}} \right].$$

Here, n_b denotes the number of possible activation patterns, not the number of patterns actually grouped into the cluster. Under this interpretation, bounding boxes drawn around (approximately) spherical clusters in activation pattern space achieve near-minimal coverage. Consequently, clustering methods that favor spherical structure, or equivalently, that minimize the maximum pairwise distance (cluster diameter), are appropriate. Consequently, we can cluster points according to their activation patterns using algorithms optimized for spherical clusters. We explore this concept further in the subsequent section.

3.4 Constructing a bounding box monitor using activation patterns

Previous studies typically focus on monitoring a single layer in neural networks. However, as demonstrated earlier, incorporating all layers for monitoring provides significant advantages, primarily by enhancing the precision of the domain bounded by the bounding boxes. Henzinger et al. (2020) discuss two strategies for leveraging multiple layers in monitoring: one approach involves building separate monitors for each layer and only accepting an input if all the monitors accept it. The other approach concatenates outputs from all layers and employs a single monitor. Although Henzinger et al. (2020) acknowledge the latter approach as theoretically more sensible, they ultimately adopt the first option for two primary reasons. First, they assume inter-layer dependencies to be less important. Second, they highlight a practical issue, the curse of dimensionality, which significantly affects the clustering

step due to diminished contrast between nearest and farthest point distances in high-dimensional spaces. To address the dimensionality problem while retaining the benefits of clustering, we propose utilizing activation patterns, distinct from the monitoring variables, as alternative clustering features.

Furthermore, the original method proposed by Henzinger et al. (2020) does not provide an anomaly score for the bounding box abstraction. It only indicates whether inputs fall inside or outside the defined bounds. Such an approach is conservative and overly restrictive, as even minor deviations result in outright rejection. Introducing a meaningful anomaly score would overcome this limitation and enable a more nuanced evaluation, enabling direct comparisons with other Out-of-Distribution (OOD) detection approaches.

In this section, we first elaborate on the rationale behind adopting activation patterns as clustering features. We then introduce our proposed anomaly score and conclude by presenting a comprehensive overview of the monitoring procedure.

3.4.1 Activation patterns as clustering features

Clustering in high-dimensional spaces faces substantial challenges due to the concentration of distances, where pairwise distances become increasingly uniform and lose discriminative power. This phenomenon, known as the curse of dimensionality, severely limits the efficacy of direct feature-based clustering, particularly constraining how many neurons a single monitor can effectively handle. An alternative approach is clustering based on activation patterns using the Hamming distance:

Definition 5. *(Hamming distance) Let x and y be two binary vectors in $\{0,1\}^n$. The Hamming distance $d(x,y)$ between x and y is the number of elements (bits) that differ between both vectors:*

$$d(x,y) = \text{Card}(\{i \in [\![1,\ldots,n]\!] \mid x_i \neq y_i\}).$$

This method aligns naturally with the geometric insights discussed previously, as the combinatorial nature of activation patterns within a bounding box is directly related to its coverage. In a vacuum, the Hamming distance on the activation pattern space also remains subject to the curse of dimensionality, though less severely than in continuous feature spaces. The combinatorial structure of the pattern space allows for more structured reasoning about similarity and neighborhoods than in continuous high-dimensional metric spaces.

Activation patterns significantly reflect the capacity and expressivity of neural networks. In particular, the number of linear regions induced by networks has become an established metric for assessing expressivity, as networks with more regions can approximate increasingly complex functions. Although counting activation regions exactly is NP-hard in general (Wang (2022)), various theoretical methods provide bounds or estimations (Montúfar et al. (2014); Serra et al. (2017)). These analyses typically use combinatorial arguments involving hyperplane arrangements defined by ReLU activations, where each neuron, through its ReLU activation boundary, partitions the input space, with deeper networks exponentially multiplying these partitions via hierarchical layer stacking, effectively creating repeated subdivisions.

Telgarsky (2015) construction of a 2^L-sawtooth function using a ReLU network with $O(L)$ neurons exemplifies this exponential growth.

Nevertheless, maximum theoretical capacity differs significantly from practical network usage. Raghu et al. (2017) confirm this by measuring network expressivity by tracking the trajectory complexity, defined as the number of activation region boundary crossings along random input lines, during training. Despite initial increases, they found that the number of region crossings tends to plateau or grow sub-exponentially as training proceeds. In other words, the network does create more linear pieces to fit the data, though not in the combinatorially explosive way one might expect if it were using all available neurons independently. Furthermore, Hanin and Rolnick (2019a,b) demonstrate that typical neural networks, both at random initialization and following conventional training, do not attain worst-case activation complexity. Instead, their observed activation patterns are substantially simpler than those worst-case constructions. Specifically, they observe that the number of linear regions grows approximately linearly with the total number of neurons rather than exponentially, which explains, in part, why over-parameterized networks can generalize well. Also, they show that although the network function space is vast, stochastic gradient descent may only explore a small, structured subset biased toward "simpler" functions (in terms of activation regions). For instance, slight Gaussian perturbations to parameters in Telgarsky (2015) sawtooth network drastically simplify its activation partitions (many teeth disappear). Thus, gradient descent implicitly favors simpler, broader partitions covering data groups rather than complex, arbitrary tilings.

These results show that neural networks possess immense combinatorial expressivity through activation patterns. However, training regimes and implicit regularization significantly restrict this capacity in practice as neural networks tend toward simplified, structured activation partitions rather than utilizing their full combinatorial complexity. Consequently, clustering samples using activation patterns will significantly alleviate dimensionality-related difficulties.

Besides the reduced number of activation regions and mitigating the curse of dimensionality, another crucial advantage of clustering via activation patterns lies in their tendency to align with the intrinsic structure of data. Novak et al. (2018), analyzing neural networks on image classification tasks, introduced sensitivity measures related to the geometry of activation regions. Their findings indicate that networks exhibiting better generalization capabilities possess larger, smoother activation regions around data points, enhancing robustness against input perturbations within the data manifold. Conversely, networks trained on randomly labeled data, which enforce memorization, produce highly irregular decision boundaries with numerous small activation regions around individual data points. This results in significant sensitivity, where small perturbations easily shift the input across region boundaries and alter outputs.

Furthermore, employing Hamming distance for clustering facilitates the recovery of task-relevant invariances encoded by neural networks, grouping inputs eliciting similar activation behaviors. ReLU networks encode invariances through a mechanism known as space folding, in which spatially distant inputs map to similar activation patterns. Lewandowski et al. (2025) characterize this phenomenon by analyzing

linear input paths and corresponding activation trajectories represented within a Hamming cube. Ideally, a linear path in input space would correspond to a convex path in activation space. However, the folding phenomenon disrupts this linearity, yielding non-convex trajectories. Notably, Lewandowski et al. (2025) observe cases where the Hamming distance to the initial input point decreased despite increases in Euclidean distance in input space. They define a measure based on range metrics reflecting deviations in Hamming distance from linear interpolation to quantify space folding. Their analysis of synthetic benchmarks (e.g., CantorNet) and real datasets (e.g., MNIST) demonstrate that folding increases with network depth and is more pronounced in networks with better generalization. Thus, networks with higher generalization performance exhibit greater degrees of space folding, indicating that learned invariances manifest structurally as folds. Lewandowski et al. (2025) conclude that network generalization capabilities are closely tied to their capacity for space folding and invariance learning.

We add that the Hamming distance also corresponds to the difference in linear transformation applied to the concerned activation regions. Notably, a small Hamming distance between the activation patterns of two inputs implies subtle, low-dimensional differences in the learned representation, even if the original inputs are not close in the input space. Accordingly, clustering the training data using the Hamming distance between activation patterns effectively uses the network-learned representation of the input space.

Especially, we can state the following lemma:

Lemma 4. *A Hamming distance of one between two activation patterns implies a rank-one difference in the linear transformation matrix that corresponds to these activation patterns.*

Proof. Let us fix an activation pattern α, see Definition 4. Then the network's mapping from input x to output y can be written as a single affine linear transformation:

$$y = Ax + b$$

where A and b depend on the pattern α of which neurons are active. For a network with L layers, each layer ℓ transforms its input $h^{(\ell-1)}$ into a pre-activation

$$z^{(\ell)} = W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}.$$

Once an activation pattern α is fixed, the ReLU operation for each layer reduces to a linear selection. This can be represented by a diagonal matrix $D^{(\ell)}$ whose diagonal entries are either 1 (active neuron) or 0 (inactive neuron). Thus, on a given region:

$$h^{(\ell)} := \mathrm{ReLU}(z^{(\ell)}) = D^{(\ell)} \left(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)} \right),$$

where $D^{(\ell)}$ is determined by the sign pattern of $z^{(\ell)}$.

Chaining these together from the input to the output, the linear transformation matrix is:

$$A = \prod_{k=0}^{L-1} D^{(L-k)} W^{(L-k)}.$$

Now, consider that we move to a region of the input space where the activation pattern differs at *exactly one neuron*. That means for some layer i and some neuron j in that layer, the corresponding diagonal entry in $D^{(i)}$ flips from 1 to 0, or from 0 to 1, and we denote by $D'^{(i)}$ the corresponding diagonal matrix. The linear transformation matrix A becomes:

$$A' = A_1 D'^{(i)} W^{(i)} A_2,$$

where:

$$A_1 = \prod_{k=0}^{L-i+1} D^{(L-k)} W^{(L-k)}, \text{ and } A_2 = \prod_{k=L-i-1}^{L-1} D^{(L-k)} W^{(L-k)}.$$

Let us say that originally $D^{(i)}$ had a 0 at position j, and now $D'^{(i)}$ has a 1 there. This means previously the j-th neuron's output of layer i was set to zero, and now it is allowed through. Algebraically, $D'^{(i)} - D^{(i)}$ is the diagonal matrix with a one in the (j, j) position and 0 elsewhere. Substituting back into the full product yields:

$$A' - A = A_1 \left(D'^{(i)} - D^{(i)} \right) W^{(i)} A_2.$$

Since $D'^{(i)} - D^{(i)}$ is zero everywhere except one diagonal entry, this difference has a rank-1 effect on the overall matrix A. In other words, adding or removing one active neuron at layer i corresponds to adding or removing a single direction of linear influence. □

In summary, in this subsection, we provide rationales for applying clustering on activation patterns before defining the bounding boxes. In the two moons example, we employ the *agglomerative clustering* method with Hamming distance (from `scikit-learn`, Pedregosa et al., 2011), the resulting bounding boxes are shown in Figure 3.4.

In order to build the clusters, we first extract the training data point that belong to the same class k of the classification task and we compute the clustering features. It is also possible to exclude data points that are misclassified by the network or classified with a low confidence probability to eliminate noisy instances that could degrade OOD detection performance. We obtain therefore a set of clusters $\mathcal{C}_k$ associated to each class k. Once the training data are clustered, we compute the monitoring variables represented by the intermediate features associated to every training data point. Then, we define the bounding boxes for each cluster by computing the bounds of the range of the monitoring variables associated to each training data point in the cluster. Formally, let us consider a cluster $\mathbf{c} \in \mathcal{C}_k$ associated to a class k, the function $\phi_{\theta,i}^{(\ell)}$ that associates an input x to the corresponding monitoring variable for the neuron or channel i of the layer ℓ. The lower bound $L_{\mathbf{c},i}^{(\ell)}$ and upper bound $U_{\mathbf{c},i}^{(\ell)}$ are defined by:

$$L_{\mathbf{c},i}^{(\ell)} = \min_{x \in \mathbf{c}} \phi_{\theta,i}^{(\ell)}(x) \quad \text{and} \quad U_{\mathbf{c},i}^{(\ell)} = \max_{x \in \mathbf{c}} \phi_{\theta,i}^{(\ell)}(x). \tag{3.3}$$

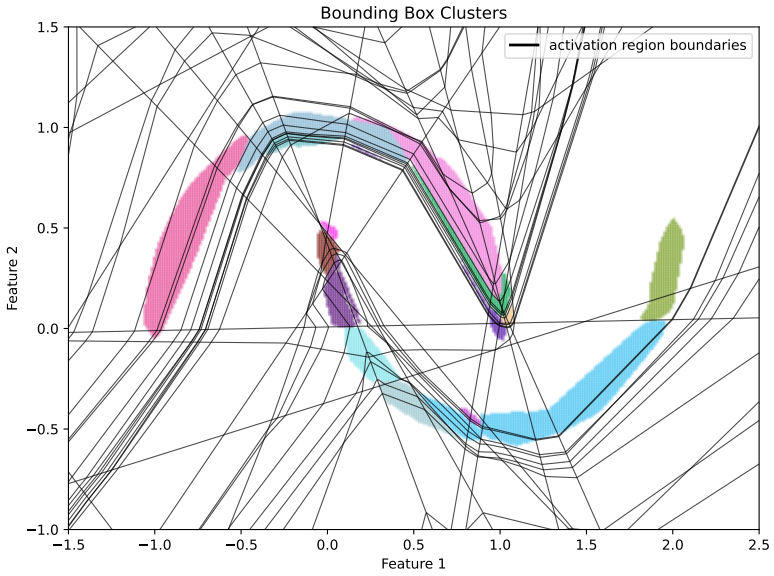


Figure 3.4: Bounding boxes defined over points according to their activation patterns. Activation boundaries shown in black.

3.4.2 Bounding Box Anomaly Score

Henzinger et al. (2020) do not propose an explicit anomaly score for bounding box abstractions. Instead, their approach verifies whether an input lies inside or outside regions defined by an ensemble of bounding boxes. Such an approach is inherently conservative, as any violation of these bounding boxes leads to rejection. To partially alleviate this issue, Henzinger et al. (2020) suggests enlarging bounding boxes by a calibration factor. Nevertheless, this still results in a binary decision process, unable to differentiate slight deviations from substantial ones. Moreover, defining a progressive anomaly score would facilitate direct comparisons with alternative OOD detection techniques. Subsequent studies have considered employing distance-based metrics as anomaly scores but their effectiveness in high-dimensionality contexts remains to be examined. Alternative heuristics are also worth considering, such as using the calibration idea as an anomaly score by taking the minimum factor necessary to include a test sample.

We propose a novel anomaly scoring approach, termed *Bounding Box Anomaly Scoring* (BBAS). BBAS quantifies anomaly by counting how many neuron states exceed their predefined cluster-specific bounds, assigning the anomaly score as the minimal count across all clusters. Thus, the anomaly score of inputs varies from 0, indicating an input fully contained within at least one cluster's bounding box, to a maximum equal to the total number of monitored variables, indicating a complete violation of bounds across all clusters. The BBAS thus introduces flexibility, accommodating inputs that marginally violate certain cluster bounds but may still be relevant. Formally, the Anomaly Score for a specific input x^* that we want to test is defined as:

$$\mathrm{AS}(x^*) = \min_{\text{cluster } c} \sum_{l=1}^{N_{\text{layers}}} \sum_{i=1}^{N_l} \mathbb{1}(z_i^{(l)}(x^*) \notin [L_{c,i}^{(l)}, U_{c,i}^{(l)}]), \tag{3.4}$$

where $z_i^{(l)}(x^*)$ is the pre-activation of neuron i of hidden layer l for input x, $n^{(l)}$ is the number of neurons in layer l, $L_{c,i}^{(l)}$ and $U_{c,i}^{(l)}$ are respectively the pre-computed lower and upper bounds for neuron i of hidden layer l for cluster c.

For a classification problem we modify BBAS to be the number of times neuron states exceed predefined bounds within each cluster and selecting the minimum count across all clusters associated with the predicted class $\hat{k}$, denoted as $\mathcal{C}_{\hat{k}}$. Formally, the Anomaly Score AS for a specific input x^* associated to a predicted class $\hat{k}$ is defined as:

$$\mathrm{AS}(x^*, \hat{k}) = \min_{\mathbf{c} \in \mathcal{C}_{\hat{k}}} \sum_{\ell=1}^{N_{\text{layers}}} \sum_{i=1}^{N_\ell} \mathbb{1}(z_i^{(l)}(x^*) \notin [L_{\mathbf{c},i}^{(\ell)}, U_{\mathbf{c},i}^{(\ell)}]), \tag{3.5}$$

where N_ℓ is the number of monitoring variables in layer ℓ.

The Anomaly Score (3.5) can be directly used as a scoring function to identify OOD. However, it focuses only on the predicted class associated to the tested input point and ignores the other classes. This predicted class is representative when its associated probability by the neural network is high. When the entropy of the predicted probabilities is high, meaning that the model *hesitates* between several classes, the predicted class is not representative anymore and the other classes must be considered. Therefore, we propose an Aggregated Anomaly Score in the form of the average of the individual Anomaly Scores over all the classes k weighted by the predicted probabilities $f_\theta(x^*)_k$ associated to an input x^*:

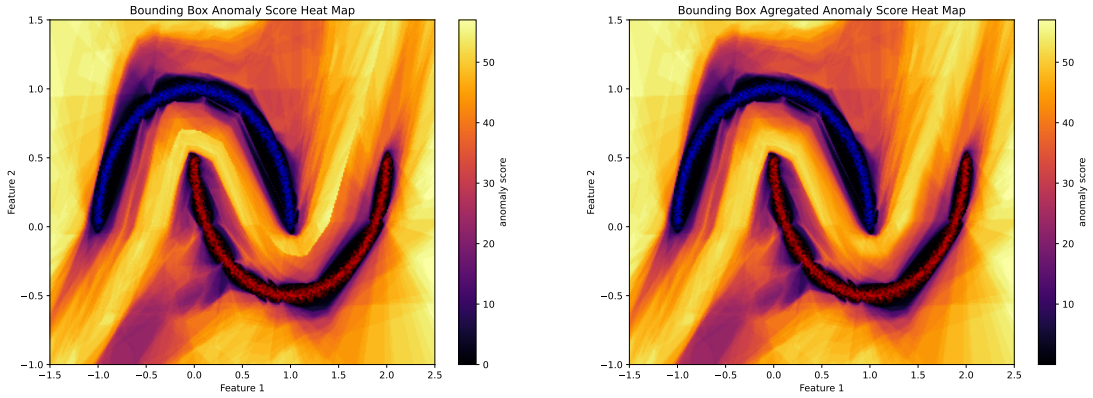
$$\mathrm{AggAS}(x^*) = \sum_{k=1}^{K} f_\theta(x^*)_k \times \mathrm{AS}(x^*, k). \tag{3.6}$$

The complexity of the Aggregated Anomaly Score (3.6) is linear with the number of classes, which is more expensive to compute than the single Anomaly Score. Still, the computation is simple enough and easily parallelizable.

Figure 3.5 show both anomaly scores on the two moons dataset. Training samples are overlaid as points. The background heatmap encodes anomaly: cooler tones indicate low score, warmer tones indicate high score. There is no discernible difference between the panels. The absence of differences follows from the setting where the two classes are fully separated, aggregation does not change the result relative to the standard AS.

3.4.3 Constructing a bounding box monitor

The proposed approach is illustrated in Figure 3.6. It consists of two primary algorithms. Algorithm "Constructing the monitor" performs an initial abstraction construction by first clustering in-distribution training samples. Subsequently, for each identified cluster, a bounding box abstraction is generated by calculating bounds for neuron activations or other selected monitoring features. Algorithm "Runtime monitoring" utilizes the constructed bounding box abstraction to compute an anomaly



(a) Anomaly Score (b) Aggregated Anomaly Score

Figure 3.5: Comparison between the aggregated anomaly map (left) and the standard anomaly score (AS) map (right) on the two moons dataset.

score for test samples. Specifically, it evaluates the degree to which neuron activation states deviate from the predefined bounds, thereby quantifying the anomaly level of each input as an Anomaly Score.

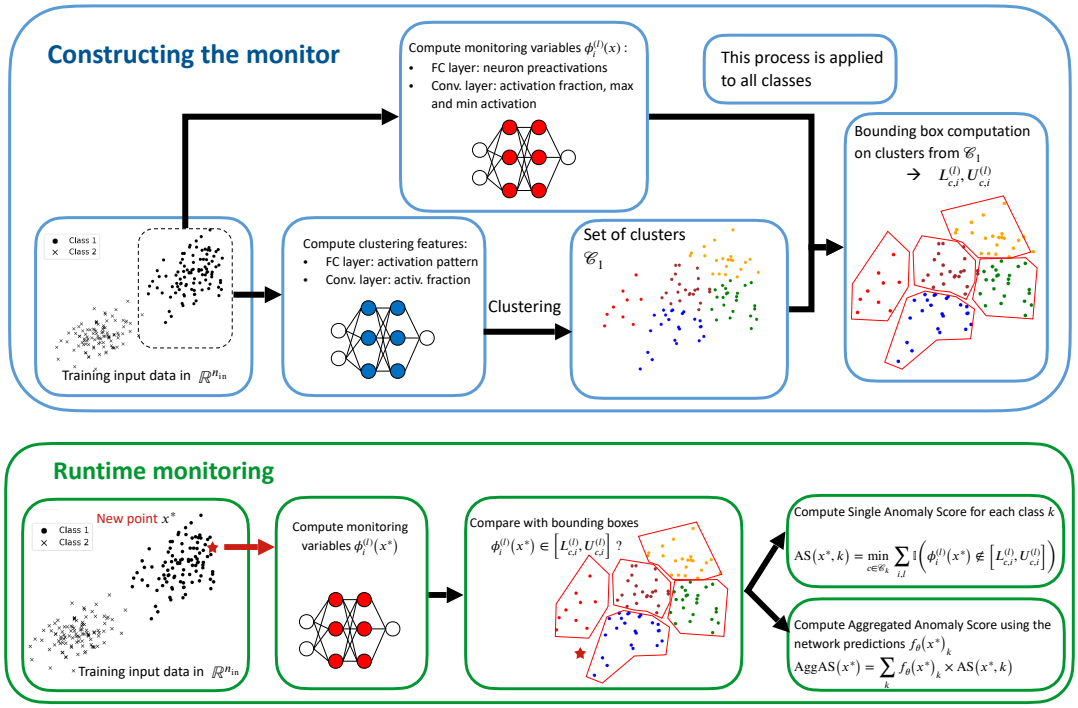


Figure 3.6: Complete BBAS pipeline processing: from constructing the monitor to runtime monitoring.

The framework described by these two algorithms is intentionally modular and broadly applicable. It can readily accommodate alternative clustering algorithms (e.g., K-means, hierarchical clustering, or DBSCAN), diverse monitoring features, including activations from different layers or neuron subsets, and various anomaly-

scoring methodologies beyond BBAS.

3.4.4 Using the monitor in OMS detection

Relating to Section 2.4, using the whole training dataset to construct the monitor may be the best choice only if the neural network performs well over all instances of this data set. The objective of a monitor in the context of surrogate models is also to be able to detect potential inaccurate prediction, even if the input belongs to the training distribution. Therefore, to build the bounding boxes, excluding the points where the neural network has less than acceptable answers is a better approach. Formally, it consists in selecting elements $x \in A^{\text{inMS}} \subset$ Training set to build the bounding box, where A^{inMS} corresponds to examples whose neural network predictions are correct according to a dedicated metric.

Furthermore, the monitor could update the bounding boxes at runtime in case there are instances for which the output is admissible and nonetheless found to be outside the OMS boxes. More precisely, if an example is detected as possible OMS, we can run the baseline computation code to guarantee safety. Then, if we realize that the prediction of the neural network is acceptable compared to the computation code answer, we can state that this example is admissible and add this point in the A^{inMS} set to update the bounding boxes accordingly. Nevertheless, updating the bounding boxes at runtime could be risky and appropriate safeguards should be implemented.

3.5 Experiments and comparison using OpenOOD benchmark

In this section, we apply the proposed approach on the OpenOOD benchmark Yang et al. (2022) that is designed for a fair comparison of the OOD detection methods, using standardized dataset and models. This benchmark is designed on Computer Vision problems, image classification, that involves Convolutional Neural Networks (CNNs). Therefore, we propose an adaptation of our method to CNNs, by defining specific monitoring variables and clustering features. Then, we conduct an ablation study to assess the importance of the selection of the monitoring variables and the impact of the clustering algorithm. Finally, we compare the performance of our approach to state-of-the-art *post-hoc* methods, meaning that it is applied only at inference time with no specific training procedure of the model.

For a first insight of the benchmark results, we provide in Figure 3.7 the performance of BBAS compared to state-of-the-art methods on CIFAR datasets, showing the relevance of our apporach for the OOD detection problem.

3.5.1 Extension to Convolutional Neural Networks

While Henzinger et al. (2020) focused on fully connected neural networks, numerous applications in computer vision use mainly convolutional neural networks (CNNs). In CNNs, a convolutional layer is equivalent to the repeated application

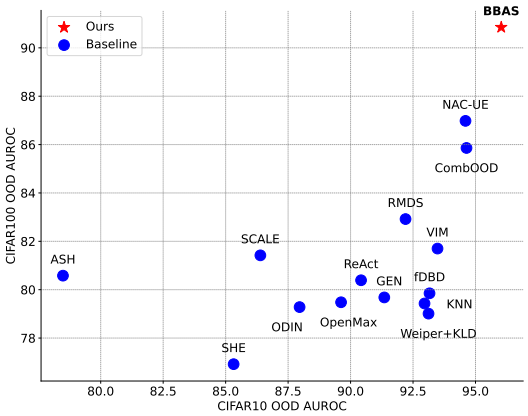


Figure 3.7: AUROC scores (%) for OOD identification performance on CIFAR-10 and CIFAR-100 averaged over four OOD datasets, where BBAS demonstrate very strong performance. See Section A.1 and A.2 for details.

of a fully connected layer across spatially overlapping input patches. The resulting feature map encodes the distribution of channel preactivation responses over an ensemble of local receptive fields within the input space. In the spirit of the bounding box abstraction, one can consider using the upper and lower bounds of the channel feature as a transformation. However, unlike the single neuron preactivation output, such approach does not account for the activation status of the entire channel. We propose below to construct monitoring variables that are adapted to convolutional layers.

Let $x \in \mathbb{R}^{n_{\text{in}}}$ be an input to a CNN composed of L layers, and let $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_L$ be the corresponding preactivation tensors for all layers. For each convolutional layer $\ell \in \{1, 2, \ldots, L\}$, the preactivation tensor can be decomposed in each feature channel:

$$\mathbf{A}^{(\ell)} = [\mathbf{A}_1^{(\ell)}, \mathbf{A}_2^{(\ell)}, \ldots, \mathbf{A}_{C_\ell}^{(\ell)}]^\top \in \mathbb{R}^{C_\ell \times H_\ell \times W_\ell},$$

where C_ℓ is the number of feature channels in layer ℓ and H_ℓ, W_ℓ are the spatial dimensions of layer ℓ feature map.

Based on these notations, we build three monitoring variables as vectors in $\mathbb{R}^{C_\ell}$ for each convolutional layer ℓ: the *activation fraction vector* $\boldsymbol{f}^{(\ell)}$, and the *minimum* and *maximum activation vectors* $\boldsymbol{m}^{(\ell)}$ and $\boldsymbol{M}^{(\ell)}$, with entries for channel $c \in \{1, \ldots, C_\ell\}$ defined by

$$f_c^{(\ell)} = \frac{1}{H_\ell W_\ell} \sum_{h=1}^{H_\ell} \sum_{w=1}^{W_\ell} \mathbb{1}(A_c^{(\ell)}[h, w] > 0), \tag{3.7}$$

$$m_c^{(\ell)} = \min_{1 \le h \le H_\ell,\, 1 \le w \le W_\ell} A_c^{(\ell)}[h, w], \tag{3.8}$$

$$M_c^{(\ell)} = \max_{1 \le h \le H_\ell,\, 1 \le w \le W_\ell} A_c^{(\ell)}[h, w]. \tag{3.9}$$

A last monitoring variable, denoted $\boldsymbol{z}^{(L)}$, is obtained by aggregating and flattening the output of the last convolutional layer. Finally, the transformation ϕ_θ that produces the monitoring variables is defined as:

$$\phi_\theta(x) = [\boldsymbol{f}^{(1)}, \boldsymbol{m}^{(1)}, \boldsymbol{M}^{(1)}, \ldots, \boldsymbol{f}^{(L)}, \boldsymbol{m}^{(L)}, \boldsymbol{M}^{(L)}, \boldsymbol{z}^{(L)}]^\top.$$

Our method is highly flexible: alternative monitoring variables can be employed, fewer monitoring variables can be used, or they can be applied to only a subset of the layers.

In the case of CNNs with ReLU activation functions, the geometric interpretation of activation patterns is similar to fully connected neural networks. However, the translational symmetry of CNNs leads us to consider other descriptive monitoring variables than the direct value of the neurons. Similarly, we do not use the direct activation of the neurons as clustering features. Instead, for convolutional layers, we use the activation fractions, defined in Equation (3.7) and we apply a hierarchical clustering based on the Manhattan distance.

3.5.2 Experimental setup

For the experimental setting, we use OpenOOD[1] (Zhang et al., 2023b), a standardized benchmark for OOD detection. OpenOOD is based on image classification datasets: CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), ImageNet-200 and ImageNet-1K (Deng et al., 2009). For each dataset, it uses a set of OOD data originated from other image datasets. MNIST (Deng, 2012), SVHN (Netzer et al., 2011), Textures (Cimpoi et al., 2014), Places365 (Zhou et al., 2017) are used as FarOOD for CIFAR datasets; iNaturalist (Horn et al., 2017), Textures (Cimpoi et al., 2014), OpenImage-O (Kuznetsova et al., 2018) are used as FarOOD for ImageNet datasets. Tiny ImageNet (Le and Yang, 2015) and CIFAR-100 are used as NearOOD for the CIFAR-10 dataset; Tiny ImageNet and CIFAR-10 are used as NearOOD for the CIFAR-100 dataset. SSB-hard (Vaze et al., 2022) and NINCO (Bitterwolf et al., 2023) are used as NearOOD for both ImageNet datasets. OpenOOD summarizes the performances of different OOD detection methods on each dataset based on the AUROC metric, allowing for a fair comparison between the methods.

Moreover, a set of pre-trained models is provided for applying post-hoc methods without re-training, and thereby simplifying the benchmark process and leveling the playing field.

3.5.3 Ablation study

We conduct the ablation study on FarOOD datasets derived from the CIFAR-10 and CIFAR-100 benchmarks. MNIST, SVHN, Textures, and Places365 are used as OOD datasets. We employ pre-trained ResNet-18 (He et al., 2016) models for these experiments. In our approach, we remove all misclassified training samples. This procedure aims to eliminate potentially noisy instances, which can degrade both in-distribution accuracy and OOD detection performance. We employ the *agglomerative clustering* method with Manhattan distance from `scikit-learn` (Pedregosa et al., 2011).

Our baseline configuration uses the *complete* linkage method from the *agglomerative clustering*, and sets the number of clusters for each class to the integer part of the square root of the number of considered training points in the class. Furthermore, features are extracted from layers at the end of each residual block group as well as

[1] https://zjysteven.github.io/OpenOOD/

the aggregated result fed to the dense layer at the end. The ablation experiments starts with this baseline and studies the impact of each choice on the AUROC scores. The results of this ablation study are summarized in Table 3.1.

Table 3.1: Ablation study on layer, channel statistics and anomaly score choices. We report AUROC (%) and FPR@95 (%), averaged over FarOOD datasets for both CIFAR-10 and CIFAR-100. **Bold** values represent the best metric for each ablation study.

	CIFAR-10		CIFAR-100	
	AUROC ↑	**FPR95 ↓**	**AUROC ↑**	**FPR95 ↓**
a) Layer Choice				
Final layers	96.02 ± 0.13	16.16 ± 0.48	$\mathbf{88.79 \pm 1.65}$	$\mathbf{37.64 \pm 6.23}$
All layers	$\mathbf{97.69 \pm 0.09}$	$\mathbf{10.30 \pm 0.22}$	87.59 ± 1.42	40.48 ± 5.32
b) Channel Stat				
Activ fraction (3.7)	95.76 ± 0.21	16.95 ± 0.50	88.59 ± 1.91	39.31 ± 6.41
Min activation (3.8)	95.53 ± 0.15	17.61 ± 0.48	86.84 ± 1.29	42.49 ± 4.16
Max activation (3.9)	94.62 ± 0.15	20.13 ± 0.03	88.43 ± 1.20	39.53 ± 4.34
All combined	$\mathbf{96.02 \pm 0.13}$	$\mathbf{17.24 \pm 0.53}$	$\mathbf{88.79 \pm 1.65}$	$\mathbf{37.64 \pm 6.23}$
c) Anomaly Score				
Single AS (3.5)	96.02 ± 0.13	$\mathbf{16.16 \pm 0.48}$	88.79 ± 1.65	37.64 ± 6.238
Aggregated AS (3.6)	$\mathbf{96.03 \pm 0.14}$	16.42 ± 0.34	$\mathbf{90.78 \pm 0.91}$	$\mathbf{36.10 \pm 3.61}$

a) Layer choice. We compare two feature extraction strategies: (1) using the complete set of network layers and (2) extracting features only from the final layer of each group of residual blocks. We observe that while employing all layers leads to performance improvements in CIFAR-10, it slightly decreases them in CIFAR-100. This behavior suggests that no single approach consistently dominates, and the choice between using all layers or just a subset may be informed by practical considerations such as computational cost or memory constraints rather than a strict performance requirement.

b) Channel statistics. We investigate the impact of the choice of the channel-wise statistics as monitoring variables for the convolutional layers. We study the choice of each individual quantities and their combined use. Single-statistic variants are the *activation fraction* (3.7), *min activation* (3.8), and *max activation* (3.9). They provide all decent performance, but we note that combining the three yields a notable improvement. Hence, in our final framework, we choose to use the three quantities as monitoring variables.

c) Anomaly score. Finally, we analyse the impact of the choice of the Anomaly Score, using either the single Anomaly Score (3.5) or the Aggregated Anomaly Score (3.6). The two scores show very close performance on CIFAR-10, while the Aggregated Anomaly Score is significantly better on CIFAR-100. Such as the layer choice, the use of the Aggregated Anomaly Score improves performances across the board and this improvement is more important for classification tasks that involve a large number of classes.

3.5.4 Clustering parameters

We focus here more specifically on the impact of the clustering parameters. To understand how these affect OOD detection, we vary both the linkage method, *complete*, *average*, and *single*, and the number of clusters. The range of number of clusters has been adapted to the two dataset sizes. The results are shown on Figure 3.8.

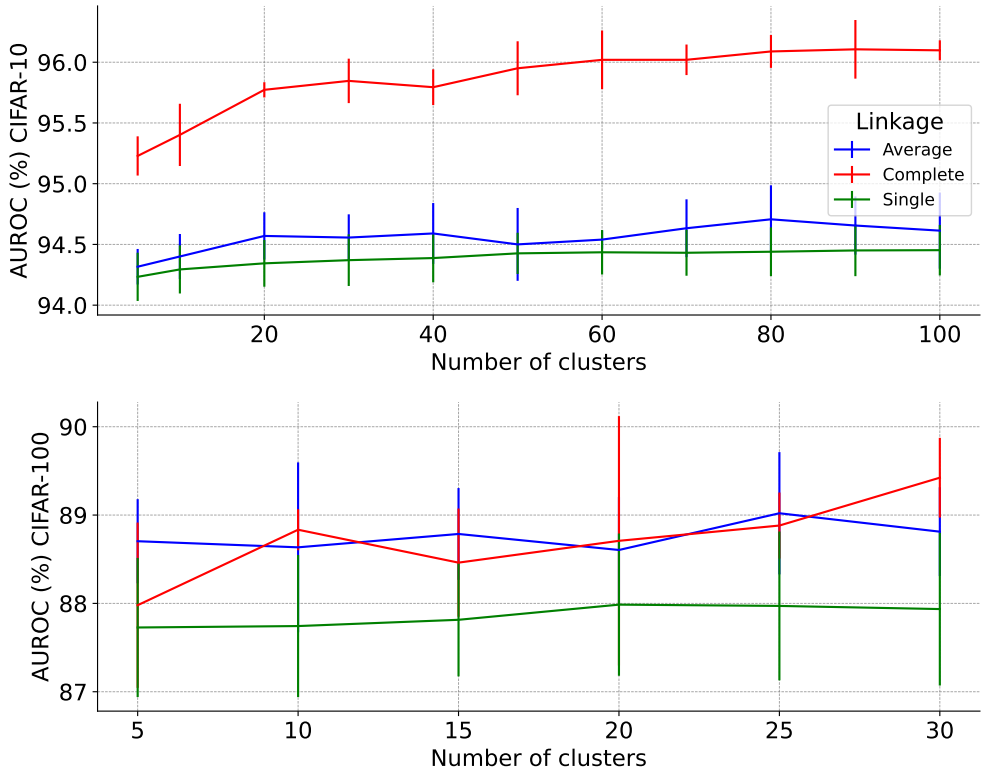


Figure 3.8: AUROC scores on CIFAR-10 (top) and CIFAR-100 (bottom), averaged over all FarOOD datasets, depending on the number of clusters and linkage method.

On the CIFAR-10 test, *complete* linkage achieves the best performance, although the improvement over the *single* and *average* linkages is limited. On the CIFAR-100 test, the linkage methods do not show significantly different results, regardless of the chosen number of clusters. Across all tested configurations, the method demonstrates consistently strong performance, indicating its robustness to changes in clustering parameters. Although slight variations arise under different linkages and cluster counts, these differences remain relatively small (less than 2% in AUROC score), underscoring that our approach is not overly sensitive to specific clustering setups.

3.5.5 Comparison with state-of-the-art methods

FarOOD setting. Table 3.2 shows the performance of the leading post-hoc methods across several FarOOD benchmarks and the results for our BBAS method. For these comparisons, we apply the same configuration as the baseline described in Section 3.5.3

associated with the Aggregated Anomaly Score. We use a set of pre-trained ResNet models (He et al., 2016) as detailed in OpenOOD.

The detailed results of these methods as well as several other state-of-the-art baselines from OpenOOD are given in Appendix A for CIFAR-10, CIFAR-100, ImageNet-200, and ImageNet-1k.

The results show that our BBAS significantly outperforms all of the approaches on both CIFAR-10 and CIFAR-100 benchmarks. On the ImageNet-200 benchmark, SCALE and ASH slightly outperform our method, however the difference remains within the margin of error as shown in the detailed Tables in Appendix A. Finally in ImageNet-1k FarOOD benchmark, our method shows satisfying results placing at the fifth rank in the leading post-hoc methods.

It is noticeable that BBAS achieves consistent good performance on all the listed datasets. Indeed, the diversity of the OpenOOD benchmark makes achieving good or state-of-the-art performance on all datasets extremely difficult. Overall, our method shows excellent performance on the CIFAR benchmarks while achieving a satisfying result on the ImageNet datasets.

Table 3.2: Average performances on FarOOD benchmarks using ResNet architectures. Best performance is in **bold**, second best is underlined. We indicate a – when the result is not provided in the original paper of the method nor in the OpenOOD leader board.

Method	CIFAR-10		CIFAR-100		ImageNet-200		ImageNet-1K	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
General approach methods								
NAC-UE Liu et al. (2024)	18.31	94.60	40.14	86.98	–	91.02	–	95.29
VIM Wang et al. (2022)	25.05	93.48	50.74	81.70	27.20	91.26	24.67	92.68
ASH Djurisic et al. (2023)	79.03	78.49	59.20	80.58	27.29	93.90	19.49	95.74
SCALE Xu et al. (2024)	–	86.39	–	81.42	–	**93.98**	**16.53**	**96.53**
Distance-based methods								
MDS Lee et al. (2018)	32.22	89.72	72.26	69.39	61.66	74.72	62.92	74.25
MDSEns Lee et al. (2018)	61.47	73.90	66.74	66.00	80.96	69.27	82.77	74.25
RMDS Ren et al. (2021a)	25.35	92.20	52.81	82.92	32.45	88.06	40.91	86.38
KNN Sun et al. (2022)	24.27	92.96	53.65	82.40	27.27	93.16	34.13	90.18
BBAS (Ours)	**16.42**	**96.03**	**35.04**	**90.85**	**23.99**	93.67	25.31	93.70

NearOOD setting. NearOOD problem is the other case study included in the OpenOOD benchmark. This case uses a comparison dataset whose distribution is similar, but represent a semantic shift from the original dataset. Table 3.3 shows the performance of the top two methods (AUROC) as well as the other methods mentioned in the FarODD setting. We reuse the exact models and configuration from the FarOOD comparison.

The results show that, except for the CIFAR-10 benchmark, BBAS is not always among the top performers in the NearOOD problem for the ResNet architectures. The full set of results is presented in Appendix A.

We note that to date, no single method has consistently outperformed its competitors across every dataset and model architecture (Tajwar et al., 2021). However, the BBAS method has room for improvement thanks to its flexibility.

Table 3.3: Average performances on NearOOD benchmarks using ResNet architectures. Best performance is in **bold**, second best is underlined. We indicate a – when the result is not provided in the original paper of the method nor in the OpenOOD leader board.

Method	CIFAR-10		CIFAR-100		ImageNet-200		ImageNet-1K	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
General approach methods								
NAC-UE Liu et al. (2024)	**30.80**	**90.98**	–	75.90	–	78.52	–	74.43
VIM Wang et al. (2022)	44.84	88.68	62.63	74.98	59.19	78.68	71.35	72.08
ASH Djurisic et al. (2023)	86.78	75.27	65.71	78.20	64.89	82.38	63.32	78.17
SCALE Xu et al. (2024)	–	84.84	–	**80.99**	–	**84.84**	**59.76**	**81.36**
Distance-based methods								
MDS Lee et al. (2018)	49.90	84.20	83.53	58.69	79.11	61.93	85.45	55.44
MDSEns Lee et al. (2018)	92.96	60.43	95.88	46.31	91.75	54.32	93.52	49.67
RMDS Ren et al. (2021a)	38.89	89.80	**55.46**	80.15	**54.02**	82.57	65.04	76.99
KNN Sun et al. (2022)	34.01	90.64	61.22	80.18	60.18	81.57	70.87	71.10
BBAS (Ours)	34.59	90.81	65.36	76.02	62.47	81.00	68.41	76.72

3.5.6 Extension to transformer architectures

Although the development of BBAS was mainly centered on dense and convolutional architectures, the method itself is not restricted to these types of architectures. To apply BBAS to any architecture, we need only to decide on the clustering variables and the monitoring variables. To show this, we evaluated BBAS on transformer architectures (ViT-B/16 (Dosovitskiy et al., 2020) and Swin-T (Liu et al., 2021)) using the same previously specified hyper-parameters and choosing the output of their last encoder block as both monitoring and clustering variables. We follow the OpenOOD evaluation protocol exactly, using the official torchvision checkpoints (Marcel and Rodriguez, 2010). The results of some of the methods used on the ResNet architecture benchmark are not available for transformer architectures, especially because these methods are not directly transposable to this type of network structure. We have retained other methods with good performances for comparison: VIM (Wang et al., 2022), MDS (Lee et al., 2018), RMDS (Ren et al., 2021a), SHE (Zhang et al., 2023a), GEN (Liu et al., 2023). Extended results are provided in Appendix A.

Table 3.4: Results for Transformer tests on ImageNet-1k (AUROC). Best performance is in **bold**, second best is underlined.

Method	FarOOD		NearOOD	
	ViT-B/16	Swin-T	ViT-B/16	Swin-T
VIM (Wang et al., 2022)	**92.84**	**93.20**	77.03	75.24
MDS (Lee et al., 2018)	92.60	91.49	79.04	75.18
RMDS (Ren et al., 2021a)	92.60	92.34	80.09	78.34
SHE (Zhang et al., 2023a)	92.42	89.84	76.11	76.59
GEN (Liu et al., 2023)	91.35	90.98	76.30	78.97
WeiPer+KLD (Granz et al., 2024)	90.32	–	75.00	–
ASH (Djurisic et al., 2023)	51.56	44.64	53.21	46.47
BBAS (Ours)	91.76	92.17	**80.97**	**80.51**

On the contrary to the ResNet architecture, it is noticeable that BBAS is the top-performing method for NearOOD detection on transformers. Our method also demonstrates strong performance on these architectures, placing it among the best methods for the FarOOD tests.

3.5.7 Discussion on the choice of parameters

Our method introduces a limited number of tunable parameters. As demonstrated in the ablation study in Section 3.5.3, these parameters have a minimal impact on the overall performance. Even suboptimal choices still yield relevant results in the experiments. The stability of BBAS with respect to its parameters allowed us to forgo the validation data provided by the OpenOOD framework for tuning purposes. The performances are close enough that there is no objective way to decide the best choice of parameters. The motivation behind the choice of the number of clusters as the square root of the number of examples and the complete linkage in clustering is to obtain clusters that are the most spherical, without having a large number of clusters. This is particularly relevant for CIFAR10 as shown on Figure 3.8.

Nevertheless, BBAS is built upon several key design choices, primarily the selection of monitoring variables, clustering features, the clustering method, and the OOD score. Other choices for the aforementioned design decisions still yield relevant performance, suggesting that the method is robust to variations in its configuration. Nevertheless, exploring alternative strategies, such as adaptive weighting schemes for monitoring variables or more sophisticated clustering techniques, may further enhance detection capabilities and improve the overall stability of the approach.

Finally, while this Section and associated experiments focused on classification tasks, our method can be applied to other deep learning problems, such as detecting OOD samples in regression tasks. In the case of regression, the main adaptation consists in considering only one set of data and not extracting each class individually, and it forces the use of the single Anomaly Score. We apply our BBAS approach to a surrogate model for OpenCalphad in Chapter 4.

3.6 Conclusion

This chapter introduces BBAS, a novel deep learning framework for out-of-distribution detection that leverages monitoring variables derived from hidden features of the neural network. By integrating a clustering approach, BBAS refines bounding box coverage to enhance OOD detection. Additionally, we propose heuristics for computing a score that quantifies the risk of a new input being out-of-distribution.

BBAS has been successfully applied to multiple convolutional neural network and transformer architectures trained for image classification. We conducted experiments on various OOD detection benchmarks and demonstrated that our method successfully distinguishes OOD samples from in-distribution data, particularly in scenarios involving covariate shift. The performances are competitive against the leading state-of-the-art post-hoc methods, achieving the best performances on CIFAR datasets, and consistent satisfying results on ImageNet datasets.

Future work could explore refining the method through another choice of monitoring variables, clustering features, and clustering algorithms. There are also other variations that can be explored in the Anomaly Score definition, such as adding a weighting scheme to each bounding box interval that takes into account the importance of the channel to the prediction. Furthermore, integrating this approach into unsupervised learning frameworks, such as active learning (Ren et al., 2021b), could further increase its applicability.

Chapter 4

Metamodeling OpenCalphad

Before addressing our leading problem, emulating CARACAS, we start with a simpler one to test the BBAS approach of Chapter 3 on a physical application. The objective in this chapter is the emulation of a component in a thermo-diffusion simulation loop. This component is called OpenCalphad (Sundman et al., 2015), a robust computational framework for thermodynamic modelling and simulation in materials science and engineering. It is based on the CALPHAD (Calculation of Phase Diagrams) approach, used to predict multicomponent materials' thermodynamic properties and phase equilibria. It involves the development of thermodynamic databases that contain the necessary information about the interactions between different elements in a material system. These databases combine experimental data, such as phase equilibria, heat capacity, and enthalpy measurements, with first-principles calculations and empirical models. OpenCalphad, as the name suggests, is an open-source implementation of the CALPHAD methodology. It provides a suite of software tools and libraries that enable researchers to perform various thermodynamic calculations, such as phase diagram calculations, equilibrium calculations, and property optimization.

Simulating the thermochemical behavior of nuclear fuel under operational transients is a complex multiphysics problem that often requires the coupling of thermal, chemical, and diffusion processes. In particular, the redistribution of elements such as oxygen, uranium, and plutonium in oxide fuel (U-Pu-O) under a steep temperature gradient (e.g., during power ramps) is a critical scenario. During such transients, local regions of the fuel may approach the incipient melting threshold, especially if the oxygen distribution is non-uniform (Introïni et al., 2021). Predicting this behavior requires resolving solid-state diffusion driven by chemical potential gradients, which is more accurate than using simple concentration gradients. In the conventional approach, this is achieved by performing a CALPHAD equilibrium calculation at each time step and at every spatial mesh node to obtain local properties such as chemical potentials and phase-dependent mobilities. In our case, a single 1D simulation with 40 spatial nodes and 1000 time steps would entail 40,000 OpenCalphad equilibrium computations. Realistic multi-dimensional simulations could easily push this into the millions of calls.

Broadly speaking, the coupling between a Gibbs Energy Minimiser, such as OpenCalphad, with a diffusive transport model can be time consuming, depending

on the complexity of the thermodynamic description (not addressed in this work), the number of chemical elements involved, and the stiffness of the diffusion problem, which dictates a small time-step size. Indeed, during the minimization, the set of stable phases can change because the algorithm updates the compositions of all stable and metastable phases to find the equilibrium. Also, as for all iterative methods, the minimization algorithm leads more or less rapidly to a local equilibrium depending on the initial estimate of the set of stable phases. The presence of a greater number of chemical elements results in a more expansive thermodynamic landscape and a more intricate global minimisation process. For instance, the calculations performed with PLEIADES Fuel Performance codes involve chemical systems with a minimum of fifteen elements, including U, Pu, O and several fission products. The use of a substitution model is deemed appropriate in order to reduce the overcost of the thermodynamic calculations on the overall simulation.

In this chapter, focus is made on the analysis of a problem that has been simplified in order to establish a general, reproducible approach for more complex studies, including multiphase multicomponent system (Introïni et al., 2021, 2020). We develop a neural network-based surrogate model for OpenCalphad to accelerate simulations of species diffusion in a single-phase U-Pu-O solid solution under a temperature gradient. By using a surrogate model, we aim to retain the predictive accuracy of full thermodynamic calculations while cutting down the computational cost. This is particularly valuable for simulating the onset of fuel melting (incipient melting) and oxygen redistribution during power transients, where time steps must be small and spatial resolution fine to capture steep gradients, leading to an enormous number of thermodynamic evaluations. Finally, by using the surrogate model together with BBAS, which routes flagged cases to OpenCalphad, we aim to retain the predictive accuracy of full thermodynamic calculations while cutting down the computational cost across different deployment scenarios.

4.1 Coupled thermodiffusion model

4.1.1 Description of the physical problem

We consider a cylindrical nuclear fuel pellet composed of a ternary oxide solid solution $U_{1-y}Pu_yO_{2-x}$, with y denoting Pu fraction and x reflecting oxygen non-stoichiometry. The pellet is subjected to an imposed radial temperature gradient (higher temperature at the center, cooler at the rim), for example, due to a power pulse or ramp. Under these conditions, oxygen tends to redistribute, migrating from hot regions to cooler regions, because temperature affects the local chemical potential of oxygen. This redistribution can lead to local changes in stoichiometry (O/M ratio) and even the onset of melting in the hot central region if the oxygen potential rises sufficiently. In the system of partial differential equations associated with this problem, the diffusion fluxes depend on the mobilities and chemical potentials of the elements. These contributions are calculated at each time step and at each node of the mesh using the thermodynamic solver, OpenCalphad.

We focus on a simplified scenario with a single solid phase present: no distinct oxide phases or molten phase initially, although melting initiation may be approached.

The system is modeled in one dimension (radial line from pellet center to periphery) taking advantage of assumed symmetry, with a spatial mesh of nodes spanning the radius, as represented in Figure 4.1.

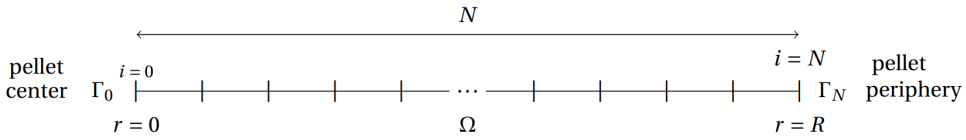


Figure 4.1: One-dimensional radial discretization of the fuel pellet, illustrating the mesh of spatial nodes used to model diffusion from the center to the periphery (Introïni et al., 2021).

4.1.2 Partitioned two-stage simulation loop

The thermo-diffusion simulation of interest involves two components, one for thermodynamic equilibrium calculations and the other for the diffusion transport calculations in a uniphase ternary system. The simulation first defines an initial state: spatial profiles of the molar quantities $N_e^0(.)$ for each element $e \in \{\mathrm{O}, \mathrm{U}, \mathrm{Pu}\}$, together with the imposed temperature profile $T(.)$. At every time step, a partitioned scheme is used.

Stage 1 (thermodynamics): at each spatial node, OpenCalphad is queried with the current values of $\{N_e(t, r_i),\, T(t, r_i)\}$ to obtain the chemical potentials $\mu_e(t, r_i)$ and mobilities $M_e(t, r_i)$. These thermodynamic quantities are the driving forces and kinetic coefficients needed for the diffusion model.

Stage 2 (transport): these quantities drive the multicomponent diffusion update, yielding new molar fractions $\{x_e(t + \delta t, r)\}$ and consequently new molar quantities $\{N_e(t + \delta t, r)\}$ for use in the thermodynamic equilibrium calculations in the first stage of the next time step, closing the loop. A set of coupled partial differential equations (PDEs) governs this transport due to molar fractions gradients. In this uniphase ternary system, the inter-diffusion equation describes the time evolution of the molar fractions profiles within the material:

$$\begin{aligned}
\frac{\partial x_O}{\partial t} &= \nabla \cdot [M_{O,U} \nabla(\mu_O - \mu_U) + M_{O,Pu} \nabla(\mu_O - \mu_{Pu})], \\
\frac{\partial x_U}{\partial t} &= \nabla \cdot [M_{O,U} \nabla(\mu_U - \mu_O) + M_{U,Pu} \nabla(\mu_U - \mu_{Pu})], \\
\frac{\partial x_{Pu}}{\partial t} &= \nabla \cdot [M_{O,Pu} \nabla(\mu_{Pu} - \mu_O) + M_{U,Pu} \nabla(\mu_{Pu} - \mu_U)],
\end{aligned} \tag{4.1}$$

where:

$M_{e,e'}$ is the interdiffusion coefficient between the elements e and e' ,

Q_o is the kinetic contribution of the heat transport of oxygen.

The interdiffusion coefficient is calculated using the diffusion coefficient and the molar fraction of each element:

$$M_{e,e'} = (M_e x_{e'} + M_{e'} x_e) x_e x_{e'}. \tag{4.2}$$

This partitioned approach effectively captures thermal diffusion, where composition changes are driven not simply by concentration gradients but by gradients in chemical potential that include temperature-dependent contributions. A comprehensive description of the general problem is detailed in Introïni et al. (2021).

Figure 4.2 provides a summary of the simulation loop. The OpenCalphad component (in red) represents the stage 1 and the Thermodiffusion simulation block (in dark blue) represents the stage 2. Therefore, the objective of this work is to develop a reliable and fast surrogate model of the OpenCalphad block to accelerate the simulation loop.

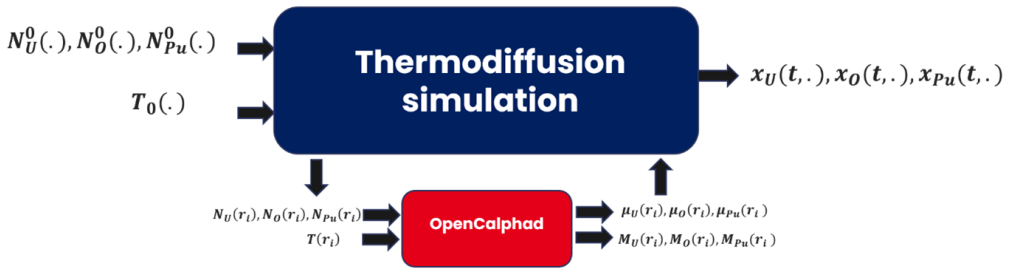


Figure 4.2: Schematic overview of the simulation loop.

4.2 Surrogate model development

4.2.1 Data generation

We adopted a strategy of generating training data via simulations of the single-phase U-Pu-O thermodiffusion problem using the actual OpenCalphad coupling. By randomly varying initial conditions and simulation parameters, we obtain a broad sampling of thermodynamic states that the model might encounter. Specifically, we run 1000 coupled diffusion simulations with different initial compositions and temperature profiles.

- All simulations start with a spatially uniform mixture of elements. The initial composition is randomly sampled: $N_O^0 \sim U(1.975, 1.999)$, $N_U^0 \sim U(0.6, 0.9)$, $N_{Pu}^0 = 1 - N_U^0$. Corresponding to 60-90% U/(U+Pu) and a total Actinides elements molar quantity of 1. All reflecting a slight hypostoichiometry to nearly stoichiometric O/M ratio in the oxide, since 2.0 would be exact stoichiometry.

- A parabolic temperature profile, as represented on Figure 4.3, is imposed, characterized by a minimum system temperature $T_{\min}$ and held constant in time for each simulation. The minimum system temperature is randomly sampled: $T_{\min} \sim U(800, 1100)$.

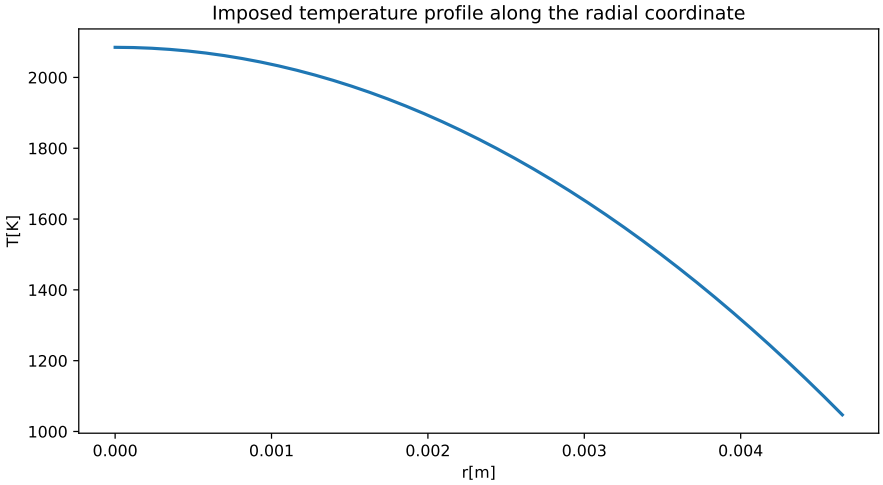


Figure 4.3: Example of a temperature profile used for a simulation: temperature ($T[K]$) in terms of the pellet radius ($r[m]$).

All simulations are 1D with 40 spatial points evenly spaced along the pellet radius and are run for 1000 time steps of $\delta t = 0.5$ s, totaling 500 s of simulated time. A reflective symmetry is applied at the pellet center to mimic a central axis.

Dataset for the surrogate model. During each such simulation, OpenCalphad is called repeatedly at all nodes and time steps, producing a dataset of 40000 input-output pairs per simulation for the surrogate. Each data sample consists of the four inputs (N_U, N_{Pu}, N_O, T) and the corresponding six outputs $(\mu_U, \mu_{Pu}, \mu_O, M_U, M_{Pu}, M_O)$ computed by OpenCalphad. At the outset, we randomly sampled 25% of the data from each simulation. We then partition the collected data by simulation: 70% of the 1000 simulations are allocated for training, with the remaining 30% divided equally between validation and testing sets.

4.2.2 Preprocessing

Prior to training, all input and output variables are scaled to improve numerical stability and convergence of the neural network. Two distinct scalers are adopted depending on the nature of the quantity. For the chemical potentials μ_e, molar quantities N_e, and temperature T, we employ a standard scaling procedure. Each raw variable X is transformed according to:

$$S(X) = \frac{X - \mu}{\sigma}, \tag{4.3}$$

where μ and σ respectively denote the mean and standard deviation of X across the training dataset. This transformation ensures that the resulting scaled variables have approximately zero mean and unit variance.

Mobilities M_e typically span several orders of magnitude, which makes direct standard scaling less relevant. To address this, we apply a logarithmic scaling with exponent normalization. The transformation is defined as:

$$S(X) = \frac{\log_2\left(X \times 2^{-n}\right) - \mu}{\sigma}, \tag{4.4}$$

where

- n is the mean exponent of the mobility values,
- μ is the mean of $\log_2(X \times 2^{-n})$ over the training dataset,
- σ is the corresponding standard deviation.

This preprocessing scheme first rescales the mobility values by a power of 2 chosen to center the data around unity (via the factor 2^{-n}), then applies a logarithmic transformation to compress the range, followed by standard scaling. The result is a well-conditioned distribution suitable for neural network training.

4.2.3 Methodology

We design a deep feed-forward neural network to serve as the surrogate model. We choose a fully-connected (dense) network architecture with 16 hidden layers, each containing 512 neurons with Tanh activation function and a skip connection that links each layer input forward to the next layer. The network expects 4 input variables corresponding to (N_U, N_{Pu}, N_O, T). The output layer has 6 neurons which produce the surrogate predictions for (μ_U, μ_{Pu}, μ_O) and (M_U, M_{Pu}, M_O) simultaneously.

We train the network by minimizing a custom loss function $\mathcal{L}$ incorporating the classical Mean Squared Error (MSE) loss $\mathcal{L}_\alpha$ and an additional term $\mathcal{L}_\beta$ considering the MSE loss of Equation (4.1) parameters:

$$\mathcal{L}(y, \hat{y}) = \mathcal{L}_\alpha(y, \hat{y}) + \mathcal{L}_\beta(y, \hat{y}), \tag{4.5}$$

with:

$$\mathcal{L}_\alpha(y, \hat{y}) = \sum_{e \in \{O,Pu,U\}} ((\mu_e - \hat{\mu}_e)^2 + (M_e - \hat{M}_e)^2), \tag{4.6}$$

and

$$\mathcal{L}_\beta(y, \hat{y}) = \sum_{\{e,e'\} \subset \{O,Pu,U\}} (((\mu_e - \mu_{e'}) - (\hat{\mu}_e - \hat{\mu}_{e'}))^2 + (M_{e,e'} - \hat{M}_{e,e'})^2). \tag{4.7}$$

Indeed, the transport equation does not consider only the value of the chemical potential and the mobilities themselves, but the differences of the chemical potentials and interdiffusion coefficients among the several elements. Network training is performed using the training dataset ($\approx 6.9 \times 10^6$ input-output pairs), with the AdamW optimizer (Loshchilov and Hutter, 2017), a `ReduceOnPlateau` scheduler, and early stopping procedure based on the validation loss (20 epochs without improvement of the loss). The implementation was carried out in the `PyTorch` framework (Paszke et al., 2019).

4.2.4 Results

Point-wise accuracy. After training, the neural network surrogate is evaluated on the held-out test set (15% of the simulation data). Table 4.1 provides the performances of the neural network for the chemical potentials diffusion coefficients, diffusion coefficients, derived chemical potential difference and interdiffusion coefficients. The predictions generated by the neural network exhibit high accuracy, with minimal errors observed: the surrogate achieved a coefficient of determination R^2 greater than 99.9% for all predicted quantities.

-	Mean absolute error	Maximum absolute error	R^2
μ_O	11.77	528.10	0.9999
μ_U	28.23	1281.66	0.9999
μ_{Pu}	27.35	1253.92	0.9999
M_O	7.56×10^{-15}	3.68×10^{-13}	0.9999
M_U	1.11×10^{-17}	4.70×10^{-15}	0.9998
M_{Pu}	6.33×10^{-24}	7.30×10^{-22}	0.9999
$\mu_O - \mu_U$	36.75	1809.76	0.9999
$\mu_O - \mu_{Pu}$	35.61	1782.03	0.9999
$\mu_U - \mu_{Pu}$	5.52	214.84	0.9999
$M_{O,U}$	3.26×10^{-16}	2.14×10^{-14}	0.9999
$M_{O,Pu}$	3.93×10^{-17}	1.60×10^{-15}	0.9999
$M_{U,Pu}$	3.72×10^{-20}	1.68×10^{-17}	0.9998

Table 4.1: Accuracy metrics for neural-network predicted chemical potentials, diffusion coefficients, derived chemical potential difference and interdiffusion coefficients.

Application in a simulation. High accuracy on static test data is necessary but not sufficient. To assess the practical applicability of our approach, we test the neural network as a surrogate model inside a simulation by replacing OpenCalphad with it. We take a representative test-case scenario: 500 s of thermodiffusion, similar conditions as the training simulations ($N_O^0 = 1.985$, $N_U^0 = 0.88$, $N_{Pu}^0 = 0.12$, $T_{\min} = 1047$). We run it in two ways: once using the true OpenCalphad coupling at each step, and once using the Neural Network to provide μ_e and M_e at each step. The evolution of the composition profiles in both cases can then be compared. Simulations are executed in serial mode on a standard workstation with an Intel CPU (48 cores, 2.4 GHz) and 64 GiB of RAM using a single-threaded implementation without parallelization.

Figures 4.4, 4.5 and 4.6 respectively show the evolutions of the molar quantity, the chemical potential and the mobility of oxygen in terms of pellet radius. The full OpenCalphad-coupled simulation (solid lines) is compared to the neural-network surrogate simulation (dashed lines) at 1, 10, 100, 200, and 500 seconds. The results obtained using the surrogate model show a remarkable level of accuracy and agreement with the simulation outcomes using OpenCalphad. For brevity, only the oxygen-related quantities are shown here; the corresponding results for uranium and plutonium exhibit the same level of agreement between the surrogate

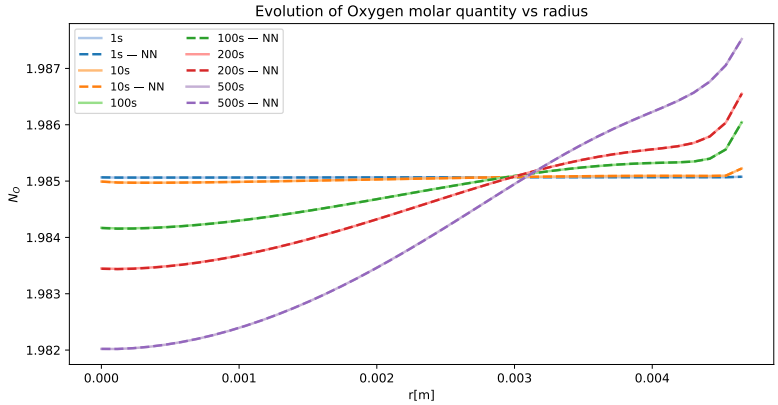


Figure 4.4: Evolution of the oxygen molar quantity (N_O) vs pellet radius ($r[m]$).

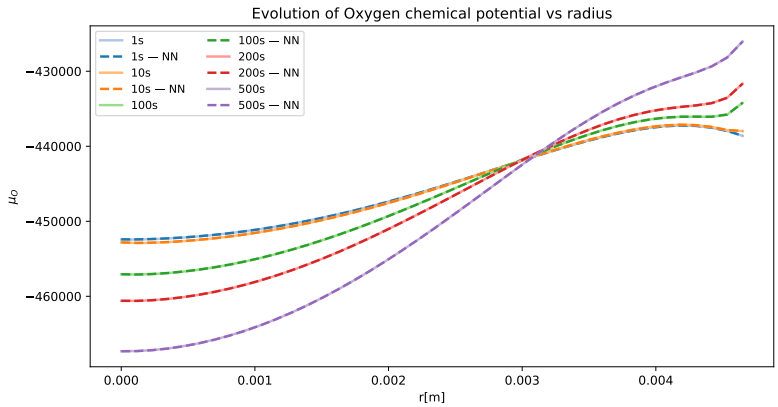


Figure 4.5: Evolution of oxygen chemical potential (μ_O) vs pellet radius ($r[m]$).

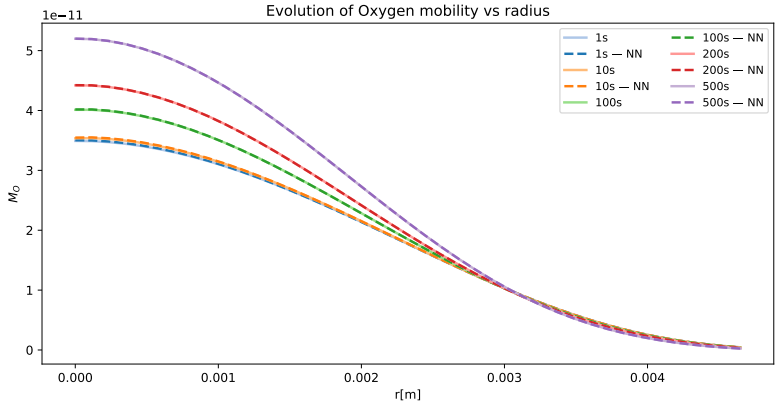


Figure 4.6: Evolution of oxygen mobility (M_O) vs pellet radius ($r[m]$).

and OpenCalphad, with no discernible deviation over the entire simulation time (Results are provided in Appendix B). More precisely, the deviation at the end of the simulation is, in average, less than 0.001%, when employing the neural network model instead of OpenCalphad.

The simulation using the neural network model closely matches the results from the original simulation while being significantly faster. With the original code, the average simulation time is approximately 71.2 seconds, of which around 61.6 seconds is spent on computing OpenCalphad. By employing the neural network, the total simulation time is reduced to about 16.2 seconds, with only 5.5 seconds devoted to the neural network representing a more than 10-fold decrease in computational time. We note that an important advantage of the neural network surrogate is that its computational efficiency improves relative to OpenCalphad as the problem scales, making the neural-network approach increasingly beneficial for large-scale or multi-dimensional simulations.

4.3 Application of bounding box abstraction for Out-of-Model Scope (OMS) detection

4.3.1 Operational domain of the neural network

A key consideration when deploying a surrogate model in lieu of a physics-based solver is ensuring its predictions remain valid during the simulation. As noted, the neural network is only guaranteed to be accurate within the domain it was trained on, which in our case corresponds to the ranges of composition and temperature sampled by the training simulations. If, during a new simulation, the system strays into a significantly different state (for instance, an unusual combination of U/Pu ratio and O/M ratio that was not represented, or a temperature below 800 K), the network prediction may become unreliable. Unlike the neural network, which has no inherent mechanism to handle conditions outside its training data and may produce unphysical outputs, the CALPHAD model is built on a thermodynamic database with a well-defined operational range.

The operational domain of the neural network is the result of the generation process of inputs, i.e., the domain where the surrogate is trained and expected to operate reliably. For simulation parameters $\theta = \left(N_O^0,\, N_U^0,\, N_{Pu}^0,\, T_{\min}\right)$ running the simulation for time horizon $t_{\max}$ and on a mesh $\mathcal{M}$ yields a trajectory of model inputs

$$\mathcal{T}(\theta, t_{\max}, \mathcal{M}) \;=\; \{\, x(m,t;\theta) \in \mathbb{R}^4 \mid m \in \mathcal{M},\; t \in [0, t_{\max}] \,\}, \tag{4.8}$$

with

$$x(m,t;\theta) = \Big(N_U(m,t;\theta),\, N_{Pu}(m,t;\theta),\, N_O(m,t;\theta),\, T(m,t;\theta)\Big), \tag{4.9}$$

are the simulation output on mesh m at time t with initial parameters θ.

Let Θ be the random distribution used to draw the simulation parameters described in section 4.2.1 and let X neural network training distribution.

We have X is supported on:

$$\operatorname{supp}(X) \;=\; \bigcup_{\theta \sim \Theta} \mathcal{T}(\theta, t_{\max}, \mathcal{M}). \tag{4.10}$$

By construction, the surrogate model is expected to operate reliably on average on this domain as it has been verified empirically against held-out test simulations. Since each $\mathcal{T}(\theta, t_{\max}, \mathcal{M})$ is generated by the nonlinear thermodiffusion solver, this distribution is analytically intractable. This raises the question of the reliability of the neural network under other simulations that are similar in setup. This justifies the need for a mechanism that verifies whether a given input lies inside the operational domain of the neural network, independently of the generation process.

While the operational domain of the neural network can be described structurally as the set of inputs generated by simulation trajectories, this characterization is not directly verifiable in practice: given a new input, we cannot test whether it belongs to $\mathrm{supp}(X)$. For this reason, it is more meaningful to define the model operational domain in terms of its predictive performance. Specifically, we consider the surrogate to operate reliably only on those inputs where its error, measured by the same loss function $\ell(\cdot)$ used during training, remains below a prescribed tolerance ε. Formally, the operational domain is expressed as:

$$\mathcal{D}_{\mathrm{op}}(\varepsilon) = \{\, x \mid \ell(x) \leq \varepsilon \,\}. \tag{4.11}$$

This performance-based characterization is empirically verifiable, since $\ell(x)$ can be computed by comparing the surrogate predictions to OpenCalphad results on any sampled input. Moreover, it provides a principled ground-truth reference to evaluate our out-of-model-scope detection method: an effective detector should flag precisely those inputs that fall outside $\mathcal{D}_{\mathrm{op}}(\varepsilon)$. We note that the choice of $\ell(\cdot)$ as the training loss is a natural and consistent option, but ultimately it should be regarded as a suggestion: other error metrics may be more appropriate depending on the application context.

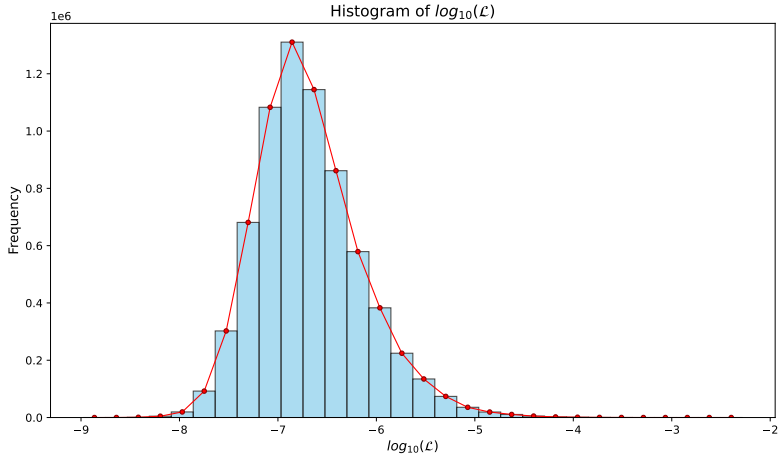


Figure 4.7: Histogram of $\log_{10}$ of the loss values over the training dataset.

A practical question in defining the operational domain is the choice of the tolerance ε. Since the surrogate is trained to minimize the loss $\ell(x)$, a natural starting point is the empirical distribution of $\ell(x)$ on the training set. The histogram of $\log_{10} \ell(x)$ shown in Figure 4.7 provides a direct view of the error profile across all inputs of the simulations used during training. A reasonable strategy is therefore to

define ε in terms of the quantiles of this distribution, for instance, at the 95th or 99th percentile, to exclude outliers. At the same time, domain knowledge can be incorporated by choosing the threshold depending on the sensitivity of the diffusion simulation to errors.

To define the model scope, we set an upper threshold on the loss of the neural network at $\varepsilon = 10^{-4}$. We chose this value empirically: simulations with average errors below 10^{-4} exhibit end states that are visually and quantitatively indistinguishable from those of the reference OpenCalphad-based simulations. Although we did not conduct a formal sensitivity analysis, this threshold comes from observations on a representative set of simulation cases. As a side effect, the chosen value also excludes the right tail of the loss distribution (1455 points out of more than 6.9×10^6 samples).

We define the binary classification task of distinguishing between in-scope and out-of-model-scope (OMS) inputs as:

$$\text{Class}(x) = \begin{cases} 0, & x \in \mathcal{D}_{\text{op}} \quad \text{(in-scope)}, \\ 1, & x \notin \mathcal{D}_{\text{op}} \quad \text{(OMS)}. \end{cases} \tag{4.12}$$

4.3.2 Data generation for OMS detection evaluation

To evaluate the out-of-model-scope (OMS) detection during deployment, we broaden the distribution used to generate test simulations. Hence, it contains both in-scope (i.e., within the training domain of the surrogate) and OMS conditions. We run 1000 coupled diffusion simulations with the OpenCalphad thermodynamic code, and from each we retain a randomly sampled 25% of the data for analysis. For these simulations, we employ the configuration described in section 4.1. The random sampling of these parameters is modified as follows (modified values highlighted in **bold face**):

- Initial composition: $N_O^0 \sim U(\mathbf{1.95}, 1.999)$, $N_U^0 \sim U(\mathbf{0.4}, \mathbf{0.95})$, $N_{Pu}^0 = 1 - N_U^0$.
- Minimum system temperature: $T_{\min} \sim U(\mathbf{750}, 1100)$.

Figure 4.8 shows the distribution of the loss values over this new dataset and Table 4.2 summarizes the performances of the model. As expected, the performance of the neural network drops significantly when evaluated on the newly broadened distribution, with approximately 54% of the sampled points fell outside the scope of the surrogate model. The model is applied to compositions and temperatures outside the training domain, which require extrapolation. As a result, the model deviates significantly on both predicted and derived variables.

4.3.3 Implementation and evaluation of BBAS

To apply the bounding box abstraction method of Chapter 3, we first extract the intermediate features from the surrogate neural network. Specifically, we record the activations of all hidden layers for every training sample. Before constructing the bounding box, we exclude training points that already violate the predefined OMS condition (Equation 4.11). For this initial study, we construct a single bounding

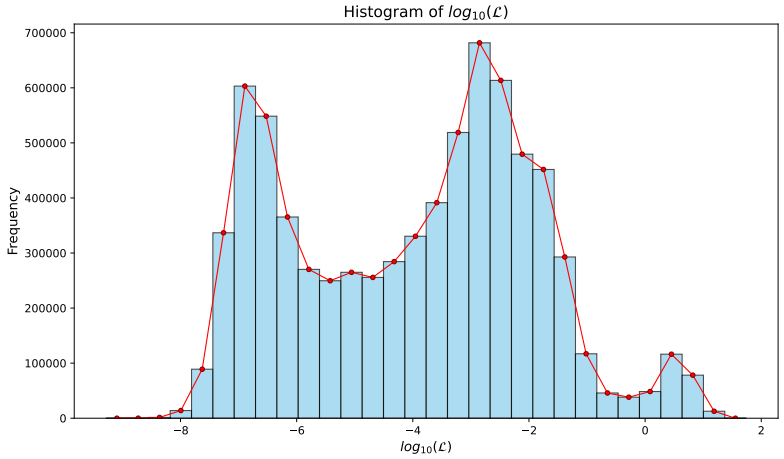


Figure 4.8: Histogram of $\log_{10}$ of the loss values over the extended test dataset.

-	Mean absolute error	Maximum absolute error	R^2
μ_O	3370.02	212389.00	0.8942
μ_U	7560.49	568012.34	0.9349
μ_{Pu}	5958.24	521982.99	0.9526
M_O	1.89×10^{-12}	1.99×10^{-9}	0.9744
M_U	6.45×10^{-13}	2.51×10^{-9}	0.0375
M_{Pu}	4.50×10^{-20}	1.08×10^{-15}	0.1652
$\mu_O - \mu_U$	10890.38	780401.38	0.9150
$\mu_O - \mu_{Pu}$	9284.06	734371.97	0.9238
$\mu_U - \mu_{Pu}$	2066.42	88395.76	0.9340
$M_{O,U}$	9.76×10^{-14}	1.50×10^{-10}	0.8834
$M_{O,Pu}$	2.93×10^{-14}	5.19×10^{-11}	0.9650
$M_{U,Pu}$	3.37×10^{-15}	1.40×10^{-11}	0.0328

Table 4.2: Accuracy metrics for neural-network predicted chemical potentials, diffusion coefficients, derived chemical potential difference and interdiffusion coefficients for the extended test dataset.

box encompassing all training data. While it is possible to partition the data into multiple clusters, each with its own bounding box, our tests show that using additional bounding boxes does not improve the OMS identification results in any significant way. Thus, the single bounding box abstraction provides a sufficiently accurate characterization of the model scope for this application.

We apply the BBAS scoring procedure to both the held-out test data and to the extended test dataset. To show the relationship between anomaly detection and surrogate accuracy, we plot in Figures 4.9 and 4.10 the two-dimensional histograms of anomaly scores versus the loss of the model for both held-out and extended test data. For each dataset, we construct the 2D histogram with

$$x = \log_{10}(\text{anomaly score} + 1), \qquad y = \log_{10}(\text{loss})\,.$$

The histogram for the held-out test data shows a dense cluster at anomaly score $= 0$ and low $\log_{10}(\text{loss})$. Quantitatively, only about 2% of the held-out test samples exhibit a nonzero anomaly score. This indicates that BBAS correctly characterizes the training distribution.

The histogram for the extended test data shows a positive correlation between the anomaly score and the surrogate prediction loss. 28% of the samples are assigned an anomaly score of zero. Only a negligible fraction of these (152 points out of more than 2×10^6 samples) were outside the model scope but nevertheless received an anomaly score of zero.

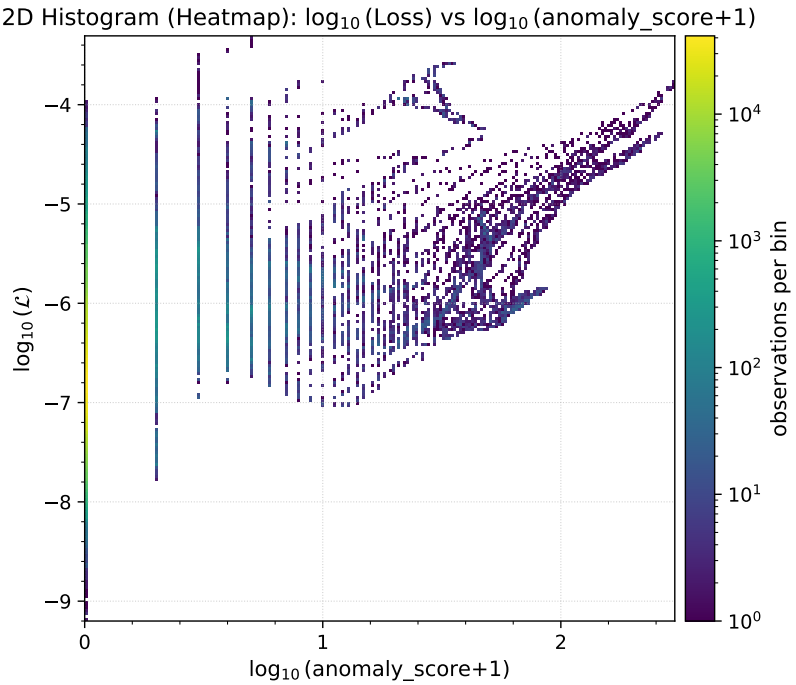


Figure 4.9: Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\text{anomaly score} + 1)$ on held-out test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$).

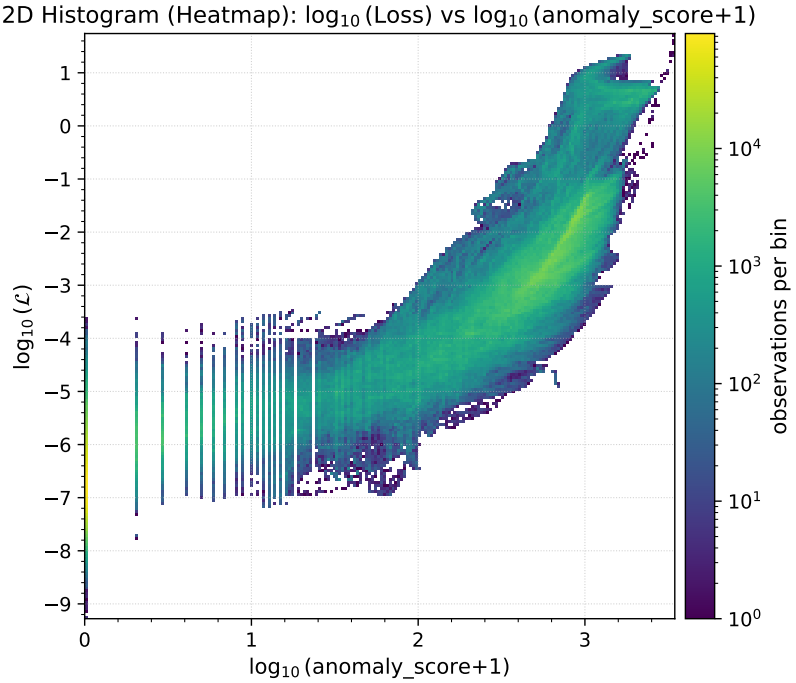


Figure 4.10: Two-dimensional histograms of $\log_{10}$(loss) vs $\log_{10}$(anomaly score + 1) on extended test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$).

4.3.4 Comparison with deep-ensemble

Deep ensembles are a widely used technique for estimating epistemic uncertainty in neural networks (Lakshminarayanan et al., 2017). The central idea is to train M independent models $\{f_{\theta_m}(x)\}_{m=1}^{M}$ with different random initializations and/or optionally different data orderings. At inference, the ensemble prediction is given by the average

$$\bar{f}(x) = \frac{1}{M} \sum_{m=1}^{M} f_{\theta_m}(x), \tag{4.13}$$

while the predictive variance

$$\sigma^2(x) = \frac{1}{M} \sum_{m=1}^{M} \left\| f_{\theta_m}(x) - \bar{f}(x) \right\|^2, \tag{4.14}$$

serves as an estimator of epistemic uncertainty. High predictive variance indicates that the ensemble members disagree, signaling low model confidence and possible extrapolation.

Deep ensembles being one of the main techniques for uncertainty estimation in Deep Learning approaches, we use it as a baseline to compare the performances of BBAS, both in terms of OMS identification and execution speed. For our application, we generate an ensemble of $M = 8$ models, each with the same architecture and training procedure as the surrogate described in Section 4.2.3, differing only in their random seeds. The ensemble achieves lower predictive errors on the held-out test dataset: on average 3.60×10^{-7} compared to 6.15×10^{-7} for a single model. The number of points in the extended test dataset that are considered out-of-scope, when

applying the same criterion as for the single model, remains approximately the same (about 54%). To compare directly with the bounding-box abstraction, we generate in Figure 4.11 and 4.12 the two-dimensional histograms of $\log_{10}(\sigma^2)$ versus $\log_{10}(\text{loss})$ for both the held-out test set and the extended test set, respectively.

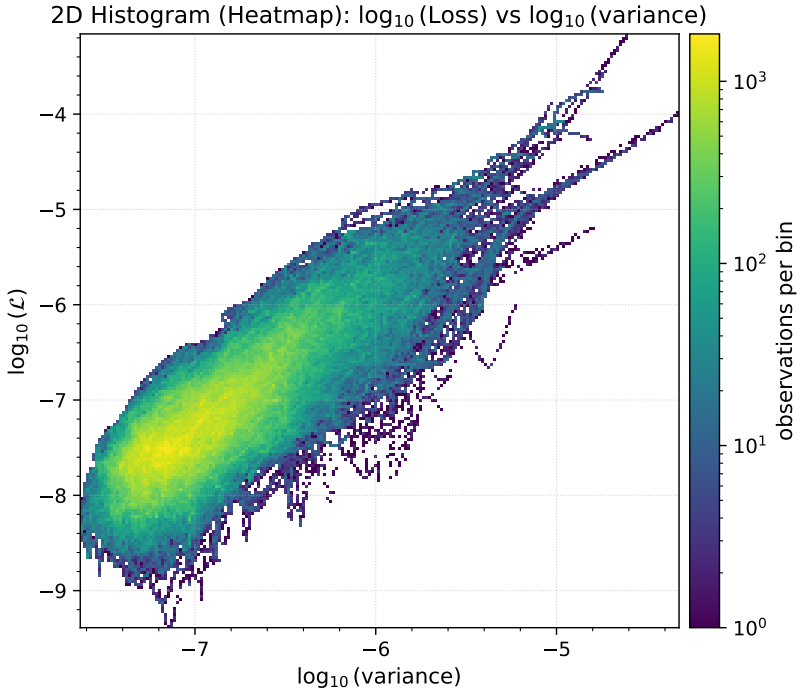


Figure 4.11: Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\sigma^2)$ on held-out test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$).

As we have observed in Figure 4.9, for the held-out test data, the bounding-box anomaly score shows essentially no correlation with the surrogate prediction error: the vast majority of samples have an anomaly score of zero. In contrast, the deep ensemble variance maintains a meaningful spread even on the test set, and visually exhibits a consistent relationship between uncertainty and error. For the extended test data, both methods show a positive correlation between uncertainty and loss. The correlation is stronger for the deep ensembles (Pearson $r \approx 0.97$) compared to the bounding-box anomaly score (Pearson $r \approx 0.93$).

As discussed in Section 4.3.1, the task of OMS detection can be formulated as a binary classification problem, where the goal is to separate inputs that lie within the surrogate's training distribution from those that fall outside. We compare our proposed bounding-box with anomaly score (BB-AS) method against the deep ensemble baseline. The results are reported in Table 4.3 using four common metrics: the false positive rate at 95% true positive rate (FPR@95), the area under the ROC curve (AUROC), the area under the precision–recall curve (AUPR), and the relative computation time.

The table highlights the trade-off between predictive performance and efficiency. Deep ensembles achieve the best overall detection performance across the accuracy metrics (lowest FPR@95, highest AUROC and AUPR), consistent with their role as a state-of-the-art approach for epistemic uncertainty quantification. However,

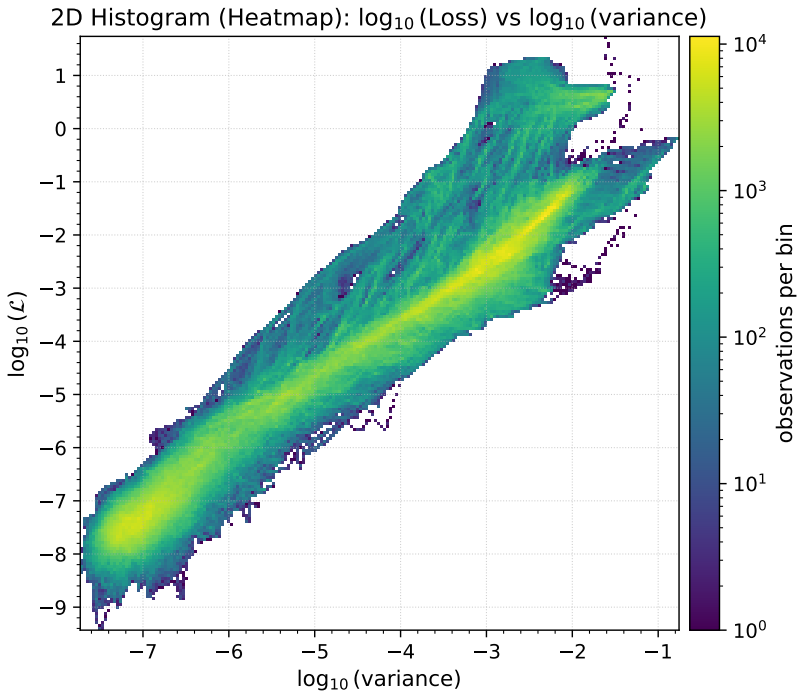


Figure 4.12: Two-dimensional histograms of $\log_{10}(\text{loss})$ vs $\log_{10}(\sigma^2)$ on extended test data (threshold for OMS data: $\log_{10}(\mathcal{L}) = -4$).

Method	FPR@95 ↓	AUROC ↑	AUPR ↑	Comp Time ↓
BB-AS (ours)	0.053	0.991	0.992	$\approx \times\mathbf{1}$
Deep Ensemble (Lakshminarayanan et al., 2017)	**0.025**	**0.996**	**0.996**	$\times 8$

Table 4.3: Performance comparison among the two methods for out-of-model scope data detection (bold for best performance). Computation time is given with reference to the execution of a single neural network model.

this improved accuracy comes at a computational cost: an ensemble of eight models requires eight times the inference time of a single network. Consequently, it almost cancels out the gain of computing time by using the surrogate model in this particular case (factor of 10, as described in Section 4.2.4).

In contrast, BBAS achieves competitive results (AUROC of 0.991 and AUPR of 0.992), with only a modest increase in FPR@95. Its primary advantage is efficiency: because it adds only inexpensive range checks to a single network evaluation, its runtime is effectively identical to that of a single model.

In summary, deep ensembles remain a highly effective method in tasks where epistemic uncertainty estimation is important. However, in the present case of monitoring for OMS detection, the marginal gain in accuracy offered by deep ensembles does not outweigh their computational overhead. The BBAS method, while slightly less accurate, delivers near-equivalent detection performance at essentially no additional cost, making it a more practical solution for monitoring surrogate models in large-scale simulation workflows.

4.3.5 Determining the threshold for the anomaly score

In practice, we must define a threshold τ to build a binary detector associated to the anomaly score $s(x)$

$$\hat{y}_\tau(x) = \mathbb{1}\{s(x) > \tau\},$$

which classifies x as OMS when $s(x) > \tau$. This rule induces the usual trade-off between true positive rate (TPR) and false positive rate (FPR):

$$\mathrm{TPR}(\tau) = \Pr(\hat{y}_\tau = 1 \mid y = 1), \qquad \mathrm{FPR}(\tau) = \Pr(\hat{y}_\tau = 1 \mid y = 0).$$

Conformal prediction offers a statistically grounded method (Lei et al., 2018). Within the split-conformal framework, let $\mathcal{C} = \{x_i\}_{i=1}^n$ be a calibration set drawn under conditions representative of deployment so that future in-scope queries are exchangeable with $\mathcal{C}$. Denote the anomaly scores by $s_i = s(x_i)$ and their order statistics by $s_{(1)} \leq \cdots \leq s_{(n)}$. The one-sided level-$\alpha$ threshold is then the order statistic

$$\tau \;=\; s_{(k)}, \qquad k = \Big\lceil (n+1)(1-\alpha) \Big\rceil.$$

Under exchangeability, this rank-based construction yields a finite-sample, distribution-free guarantee on the in-scope false positive rate:

$$\mathbb{P}\Big\{ s(\tilde{x}) > \tau \Big\} \leq \alpha \quad \text{for any future in-scope } \tilde{x}.$$

Equivalently, τ is the empirical $(1-\alpha)$-quantile of the calibration-score distribution with the standard $(n{+}1)$ correction and the guarantee concerns only in-scope data.

The split-conformal guarantee rests on a simple but necessary premise: the calibration scores must be exchangeable with those of future in-scope queries. In other words, the calibration set should contain only in-scope inputs drawn under conditions representative of deployment.

We assessed the effect of the calibration source on split-conformal thresholding at $\alpha = 0.05$ under two regimes that differ in their plausibility of exchangeability with deployment:

Regime A) Held-out test set. The calibration set consists of only in-scope points taken from the held-out test set. However, these points are not exchangeable with the evaluation setting (the remainder of the extended test suite), so the finite-sample false-positive control guaranteed by split-conformal does not apply. Results for Regime A are therefore relevant for diagnostic only.

Regime B) 10% of extended test set. The calibration set consists of only in-scope points selected from 100 simulations within the extended test set (10% of all simulations). For the in-scope portion of the remaining extended test set (90% of all simulations) that lie in the same regime as the calibration set, exchangeability with this calibration source is plausibly satisfied, so the split-conformal in-scope FPR guarantee is expected to hold.

Calibration regime	Exchangeable?	α	Threshold τ	In-scope FPR ↓	OMS TPR ↑
A) Held-out test (in-scope points)	No	0.05	0	0.3784	0.9999
B) Extended test (100 sims, in-scope points)	Yes	0.05	198	0.0389	0.9433

Table 4.4: Conformal thresholding at level $\alpha = 0.05$ with two calibration sources. Metrics are computed on the remaining extended test set: in-scope FPR on its in-scope subset, OMS TPR on its OMS subset.

Table 4.4 shows the performance of BBAS for both regimes. We observe that, at level $\alpha = 0.05$, calibrating with exchangeable OMS simulations (Regime B), for which the in-scope portion of the extended test set is exchangeable with the calibration data, yields a nontrivial threshold ($\tau = 198$) that controls the in-scope FPR (0.0389) while maintaining high OMS TPR (0.9433). In contrast, calibrating on a non-exchangeable in-scope data from the held-out test set (Regime A) violates the exchangeability assumption with respect to the OMS test, leading to a degenerate threshold ($\tau = 0$) and an inflated in-scope FPR (0.3784) and as a result a near-perfect OMS TPR (0.9999) which is overly conservative.

Starting a new simulation family, we can adopt as an initial operating point the conservative threshold estimated on the test set: the decision rule admits as in-scope only inputs with very low anomaly scores and thus allows only a tiny fraction of genuinely out-of-scope points to pass. This favours safety (few false negatives on OMS) at the cost of rejecting more borderline in-scope inputs. During deployment, the threshold can then be recalibrated using feedback from the simulation itself: whenever an input is rejected, the reference evaluation (e.g. OpenCalphad) provides a ground-truth loss; rejected inputs that nevertheless satisfy a predefined tolerance are appended to a rolling calibration buffer. Periodic re-computation of the split-conformal threshold on this buffer progressively adapts the operating point to the encountered workload while restoring exchangeability for the in-scope subset. In this way, we start from a deliberately strict test-set threshold to minimise OMS admissions, and then relax it only where we can empirically confirm that rejections were unwarranted.

4.4 Conclusion and perspectives

In conclusion, the proposed methodology enables a hybrid workflow in which the high-fidelity numerical component is invoked whenever the runtime monitor rejects a surrogate prediction. This selective fallback preserves most of the computational gains of the surrogate while maintaining trustworthiness, thereby increasing confidence in the overall simulation accuracy. By combining a fast predictor with a principled rejection mechanism and a guaranteed-accurate backup, the coupled system delivers reliable and accurate results across a wide range of operating conditions.

This chapter is a demonstration study for the relevance of the BBAS monitoring method of Chapter 3 of a surrogate model in a physics simulation context. The classical speed-up approach consists of precomputing thermodynamic/kinetic quantities (e.g., Gibbs energy, chemical potentials, mobilities) on a grid of composition/temperature and use fast interpolation at runtime. This tabulation strategy avoids repeated equilibrium solves and is robust provided the simulation remains inside the tabulated domain. However, as additional chemical elements, phases, and state variables are introduced, the effective state space grows rapidly and we inevitably encounter the curse of dimensionality: any naive attempt at space-filling or uniformly random coverage becomes prohibitively expensive and still leaves large regions poorly represented.

In most practical fuel problems, the initial state distribution is at least partly informed by manufacturing and material accounting data and by irradiation conditions, these priors can be encoded to produce realistic initial conditions. By contrast, the evolution of the system in a strongly coupled multiphysics setting is not known a priori: trajectories depend nonlinearly on chemical potentials, mobilities, and temperature gradients. Consequently, when the dimensionality increases and closed-form trajectory priors are unavailable, one is compelled to adopt the simulation-driven approach used here: generate input-output pairs along ensembles of representative simulations. In short, the present policy is best viewed as a pragmatic scaffold for high-dimensional settings: start from simulation generated data reflecting credible scenarios and rely on the OMS monitor for new simulation settings.

Beyond the present demonstration, this framework lays the groundwork for a general and reproducible approach applicable to more complex studies, most notably, integrated simulations of nuclear-fuel behavior spanning normal operation through severe-accident conditions. In such regimes, the monitor-guarded surrogate can focus computational effort where it matters most, while the original solver ensures correctness beyond the model domain, offering a scalable path to high-fidelity fuel performance analysis.

Chapter 5

Surrogate modeling of fission gas behavior

Accurate modelling of fission gas behaviour remains one of the central challenges in nuclear fuel performance simulation. In uranium dioxide UO_2 fuel operated under pressurized water reactor (PWR) conditions, xenon and krypton atoms are continuously produced during fission and undergo a complex sequence of transport, trapping, and release processes. Their accumulation in bubbles within the fuel matrix and at grain boundaries leads to swelling and degradation of thermal conductivity. At the same time, their eventual release into the free volume of the rod raises internal pressure and can strongly affect cladding integrity. For these reasons, every modern fuel performance code incorporates a fission gas model that resolves the temporal and spatial evolution of these processes and their impact on global rod behaviour.

CARACAS (CAlcul RApide du Comportement des gAz de fiSsion) (Jomard et al., 2014), developed by CEA, is an advanced simulation module for fission gas behavior embedded in the ALCYONE (Introïni et al., 2024) fuel performance code for PWRs. It is part of the ALCYONE fuel performance code, which is tasked with predicting fuel rod behavior under normal and transient operation. CARACAS was introduced to provide a more mechanistic and detailed description of fission gas processes than earlier empirical models, while still using robust, fast algorithms suitable for routine industrial simulations. Within ALCYONE, CARACAS predicts the swelling and release quantities needed for thermo-mechanical calculations, and its predictions have been benchmarked against extensive experimental databases (Introïni et al., 2024). Despite these strengths, mechanistic models like CARACAS remain relatively costly to evaluate. The need to integrate stiff coupled equations over long irradiation histories requires implicit solvers and adaptive time stepping. This computational bottleneck motivates the search for surrogate models that can reproduce the predictions of CARACAS at far lower cost.

The objective of this chapter is to design, train, and validate a neural network surrogate for CARACAS. The surrogate is intended to emulate the evolution of fission gas populations and the resulting macroscopic outputs such as swelling strain and fission gas release fraction. By replacing the integration of coupled rate equations with a forward evaluation of a trained neural network, the computational cost per call can be reduced by orders of magnitude.

Due to project constraints, the codes CARACAS and ALCYONE are not directly accessible for this work. Instead, authorized team members run them to generate the simulations used as training and testing data for the surrogate model. The main limitation of this setup is the reduced control over the produced data and it prevents leveraging deeper knowledge of the codes' computation processes that could otherwise be integrated into model training and development to improve efficiency.

5.1 CARACAS inner workings and integration within ALCYONE

5.1.1 CARACAS physical model

Grain microstructure. CARACAS works at the scale of a representative spherical grain and its interfaces. In the unrestructured state, the interior is a crystal matrix bounded by interfaces that include occasional fabrication porosity. At high local burnup typically near the pellet rim, the microstructure partially restructures into high-burnup structure (HBS) with finer subgrains and a dense population of nanometric pores.

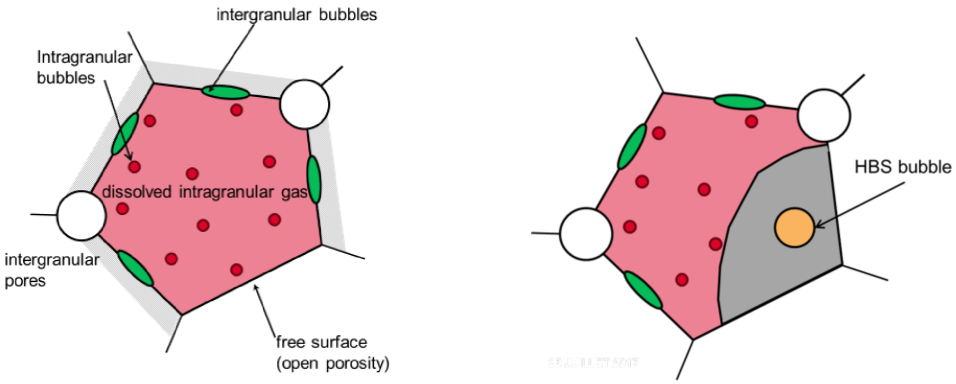


Figure 5.1: Left: Representative grain schematic with fission-gas populations. Right: Representative grain schematic with high burnup structure. Source: CEA.

Fission gas populations and evolution. From the start of irradiation, fission gas atoms are generated uniformly throughout the grain material in proportion to the local fission rate. Due to the low solubility of noble gases in UO_2, fission gas atoms are captured by defects and intragranular bubbles, with a fraction diffusing to the grain boundary over time. At boundaries, gas accumulates in intergranular bubbles, is released into open porosity once bubbles interlink, or escapes through a free surface when a connected path exists. Figure 5.1 (Left) depicts an illustration of a representative grain where the grain interior contains a dissolved intragranular gas population (light red) that is continually drained into intragranular bubbles (red dots), the grain faces host intergranular bubbles (green lenses) and intergranular

pores (white circles). As burnup increases, the microstructure partially transforms into HBS wich modifies release and retention properties. Figure 5.1 (Right) illustrates a grey HBS zone replaces part of the grain periphery and contains characteristic HBS bubbles (yellow), while the unrestructured interior still shows dissolved atoms and intragranular bubbles as in Figure 5.1 (Left).

CARACAS Model structure. CARACAS tracks summary metrics of three distinct populations: gas dissolved in the lattice, gas stored in intragranular bubbles, and gas accumulated at interfaces (grain-boundary bubbles or, in restructured fuel, pores). The model uses a set of coupled equations to model and solve them for the summary metrics of each population. Several material-specific parameters underpin the CARACAS equations, such as diffusion coefficients, fission gas yield, grain size, etc. Many of these parameters are calibrated against separate-effect experiments and post-irradiation examinations to ensure the model reproduces observed fission gas release fractions and bubble sizes under typical PWR operating conditions.

CARACAS is computationally efficient. An important contributor to CARACAS efficiency is its lifecycle-conditioned model reduction. The set of active operators is restricted to the dominant processes at each irradiation stage. For example, in the initial phase, updates are limited to intragranular capture of fission gas, and boundary/release operators are inactive. In a separate example, the HBS is taken into acount only once restructuring becomes significant. When restructuring becomes complete in the affected region, CARACAS only tracks HBS-specific metrics in that zone (i.e., the model stops advancing the other gas populations there and follows only HBS gas populations).

5.1.2 Integration within ALCYONE

ALCYONE simulates a PWR fuel rod with two complementary coupling levels. At the global-rod level, the axially stacked slices of the rod exchange uniform state variables (e.g., rod internal pressure, total fission-gas release) within a fixed-point loop and adaptive time stepping. At the local-slice level, each axial slice is solved on a pellet-gap-cladding mesh (in 1D/2D/3D variants) using partitioned fixed-point iterations that resolve, among other processes, heat conduction, thermo-mechanical contact, and gas behavior until convergence. Figure 5.2 is a scheme that summarizes the computational architecture of ALCYONE: a global rod-level loop with adaptive time stepping, iterating over axial slices and exchanging uniform rod variables (left), a local slice-level partitioned multiphysics loop that iterates heat conduction, contact/mechanics, and gas behavior to convergence (centre), and a panel that illustrates the pellet-gap-clad discretizations used per slice for the 1D axisymetric case, 2D sector with symmetry/crack planes, and a 3D 1/32-pellet fragment (right).

CARACAS is integrated at the slice (local) level as a pointwise fission-gas model. It is called independently at each fuel mesh node within the local fixed-point loop. It takes as input the relevant thermo-mechanical and irradiation fields (e.g., temperature history, fission rate, burnup), and returns updated gas state variables and release terms. ALCYONE aggregates these nodal results to update slice values and propagates them to the coupled slice and rod scale models. In

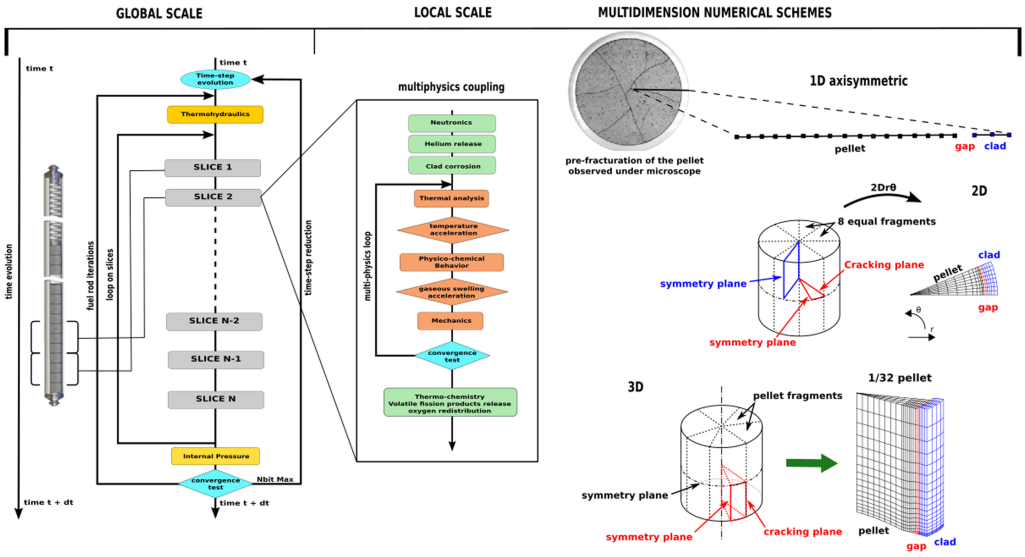


Figure 5.2: Schematic representation of the ALCYONE multiphysics computational scheme with the different levels of description (Introïni et al., 2024).

particular, CARACAS is strongly coupled with the thermal and mechanical solvers of ALCYONE.

Thermal coupling. ALCYONE thermal module provides the temperature field in the fuel, radially and axially, at each time step. CARACAS uses these local temperatures to update diffusion rates and to evaluate any temperature-dependent release criteria. For example, higher temperature increases the diffusion coefficient in CARACAS, accelerating gas migration to grain boundaries. In the opposite direction, the presence of fission gas bubbles degrades the thermal conductivity of the fuel, causing the local temperature to rise further. Thus, thermal and gas models are implicitly coupled: temperature drives gas diffusion, and gas-induced porosity in turn influences temperature.

Mechanical coupling. ALCYONE mechanical solver computes fuel and cladding deformation, stresses, and cracks. It supplies CARACAS with mechanical state parameters that influence gas behavior. For example, the mechanical model produces a plastic porosity variable that represents grain-boundary cracking. ALCYONE passes this directly to CARACAS, which uses it as a criterion for gas release. Conversely, CARACAS returns fission-gas swelling strains that are used as volumetric strains in the mechanical equilibrium, altering local contact pressures and the stress state, which in turn can trigger more cracking and release. The net effect is a feedback between gas-driven swelling and cracking-driven release at every time step.

5.1.3 CARACAS numerical model

In the CARACAS numerical model, each mesh point carries an internal state split into two groups of variables:

- $Y_n \in \mathbb{R}^{d_y}$: the subset of variables at step n governed by a system of coupled ODEs that the time integrator advances, with $d_y = 22$. These variables are visible to ALCYONE. They can be divided into three main groups of variables:

$$Y_n \;=\; \Big[\, y_{\text{gas_pop}} \,\|\, y_{\text{hbs_frac}} \,\|\, y_{\text{hbs_pop}} \,\Big],$$

  where:

  - $y_{\text{gas_pop}}$: gas population metrics (a vector of dimension $d_{\text{gas_pop}} = 14$).
  - $y_{\text{hbs_frac}}$: the local fraction of HBS (a scalar between 0 and 1).
  - $y_{\text{hbs_pop}}$: HBS gas population metrics (a vector of dimension $d_{\text{hbs_pop}} = 7$).

- $A_n \in \mathbb{R}^{d_a}$: auxiliary variables (flags, triggers, state descriptors, etc.) at step n used to define which system of equations to solve, with $d_a = 20$.

At the start of a simulation, ALCYONE initializes all internal variables, providing (Y_0, A_0). At a generic step t_n, CARACAS receives from ALCYONE a set of driving variables, $U_{n+1} \in \mathbb{R}^{d_u}$ with $d_u = 14$, and a time step δt_{n+1}, with $t_{n+1} = t_n + \delta t_{n+1}$.

Continuous-time model. Given the current auxiliary state A_n and the driving variables U_{n+1}, fixed over the step, the ODE subsystem reads:

$$\dot{Y}(t) = F\Big(Y(t), A_n, U_{n+1}\Big), \qquad t \in (t_n, t_{n+1}], \tag{5.1}$$

with $Y(t_n) = Y_n$. The role of A_n is to select and parameterize the form of F, e.g. via flags and regime switches.

Time discretization. Given $(A_n, U_{n+1}, \delta t_{n+1})$, the ODE solver advances Y across the macro-interval $[t_n, t_{n+1}]$. The solver may internally subdivide or adapt the step. We only record the net one-step map at time t_{n+1}, such that $Y_{n+1} = Y(t_{n+1})$. We write:

$$Y^*_{n+1} = \Phi\Big(Y_n, A_n, U_{n+1}, \delta t_{n+1}\Big), \tag{5.2}$$

where Φ denotes the aggregate propagator over the time step.

Auxiliary update. After computing Y^*_{n+1}, CARACAS applies some operations to this variable to enforce physical constraints and obtain a corrected value of Y_{n+1}. Then, it updates the auxiliary variables using the new ODE state and the step driving variables. We denote both operations as $\mathcal{G}$ that we do not explicitate in this work:

$$Y_{n+1}, A_{n+1} = \mathcal{G}\Big(A_n, Y^*_{n+1}, U_{n+1}\Big). \tag{5.3}$$

This update may flip flags, trigger regime changes, or refresh other local descriptors that will affect the next-step assembly of Equation 5.1. In particular, two flags are of interest. The first flag is activated when the fraction of the HBS structure exceeds a certain threshold; in this case, the set of variables describing HBS gas populations is activated along with their corresponding ODEs. The second flag deactivates the normal gas population metrics when the fraction of the HBS structure exceeds another threshold.

5.2 Data generation and preprocessing

5.2.1 Data generation

We construct the training corpus from physics-based simulations performed with ALCYONE for UO_2 fuel rods operating under normal PWR conditions. The starting point is a representative set of $N_0 = 57$ experimental rods previously used to validate ALCYONE. For each rod r, we take its measured irradiation (time-dependent power) history and its associated manufacturing parameters, run a baseline ALCYONE simulation over the rod's full life from fresh fuel to discharge, and record the time-evolving CARACAS relevant variables for each mesh point. Denote ALCYONE inputs by the pair $\big(p_r(t), \mathbf{m}_r\big)$, where $p_r(t)$ is the power history and $\mathbf{m}_r$ collects manufacturing parameters. To broaden coverage around the validated operating envelope, we augment each baseline case by applying a power multiplier $\alpha(t)$ to the entire power history while holding $\mathbf{m}_r$ fixed, i.e.,

$$\tilde{p}_{r,\alpha}(t) = \alpha(t)\, p_r(t). \tag{5.4}$$

For every rod, we generate 8 such augmented inputs, with α chosen for $\tilde{p}_{r,\alpha}$ to stay within normal PWR bounds, and rerun ALCYONE. This yields 9 simulations per rod, thus a total of $N = 513$ simulations.

For each simulation, we keep a separate trajectory for all mesh points, across all slices. Each trajectory consists of CARACAS checkpoints at every time step $\{(Y_k, A_k)\}_{k=0}^{n}$, and exogenous inputs for each transition $\{(U_k, \delta t_k)\}_{k=1}^{n}$. These trajectories are of variable length n ranging from roughly 40 to 2000 steps depending on the case. The corresponding physical time span covered by a trajectory varies from a few months to multiple years. Time steps are highly irregular as well, with δt_k ranging from a few seconds to several weeks.

5.2.2 Preprocessing

For preprocessing, we split the simulations at the level of the original reference simulation. We grouped all instances derived from a given reference run, the baseline and all re-runs following change in power (Equation 5.4), into a single subset in a 0.70/0.15/0.15 train/validation/test split. This grouping prevents leakage across subsets and ensures that re-runs appear only within the same split as their source reference simulation. Within each split, every trajectory is segmented into fixed-length windows of n_f time steps; windows shorter than n_f are zero-padded, and we store a binary mask $m_i \in \{0,1\}^{n_f}$ to mark valid positions. The resulting dataset is:

$$\mathcal{D} = \Big\{ \mathcal{T}_i \Big\}_{i=1}^{N}, \qquad \mathcal{T}_i := \Big((Y_{i,0}, A_{i,0}), \{(Y_{i,k}, A_{i,k})\}_{k=1}^{n_f}, \{(U_{i,k}, \delta t_{i,k})\}_{k=1}^{n_f}, m_i \Big). \tag{5.5}$$

We perform normalization feature-wise and time-invariant. Statistics are estimated on the training set, with padded entries excluded, and applied unchanged to validation and test sets.

1. **Outputs Y.** Non-fraction features Y^j being positive, they are min-max scaled with training minima and maxima:

$$\tilde{Y}^{(j)} = \frac{Y^{(j)} - y^{(j)}_{\min}}{y^{(j)}_{\max} - y^{(j)}_{\min}}, \tag{5.6}$$

   while fraction features are left unchanged.

2. **Auxiliary variables A.** Non-flag, non-fraction features A^j can be negative or positive, they are then standardized with training means and standard deviations:

$$\tilde{A}^{(j)} = \frac{A^{(j)} - \mu^{(j)}_A}{\sigma^{(j)}_A}, \tag{5.7}$$

   whereas binary flags and fraction features are left unchanged.

3. **ALCYONE variables U.** All features U^j, being possibly negative or positive, are also standardized using their means and standard deviations:

$$\tilde{U}^{(j)} = \frac{U^{(j)} - \mu^{(j)}_U}{\sigma^{(j)}_U}. \tag{5.8}$$

4. **Time-step δt.** Time-step values are scaled to $[0, 1]$ by division by the maximum time step observed in the training split:

$$\tilde{\delta t} = \frac{\delta t}{\delta t_{\max}}. \tag{5.9}$$

To simplify notation in the following sections, we drop the tildes and write normalized variables without them, unless specified.

5.3 Methodology

5.3.1 Direct approach: next-step prediction

Over a step $[t_n, t_{n+1}]$ with piecewise-constant drivings U_{n+1} and step sizes δt_{n+1}, The solver in CARACAS returns only the net one-step map Y_{n+1}, while CARACAS internally update its auxiliary variables A_{n+1}. Given this interface, a natural direct approach is to learn a next-step surrogate that imitates this map directly.

Concretely, we train a neural network to predict only the next step using as input all the variables of the current step (intern, auxiliary and driving variables):

$$\hat{Y}_{n+1} = f_\theta\Big(Y_n, A_n, U_{n+1}, \delta t_n\Big), \qquad \theta \in \Theta, \tag{5.10}$$

by minimizing a single-step supervised loss:

$$\ell\Big(f_\theta(Y_n, A_n, U_{n+1}, \delta t_{n+1}),\, Y_{n+1}\Big) = \Big\| f_\theta(Y_n, A_n, U_{n+1}, \delta t_{n+1}) - Y_{n+1} \Big\|_2^2. \tag{5.11}$$

-	Mean absolute error	Maximum absolute error	R^2
$Y^{(1)}$	2.7751×10^{23}	3.2278×10^{25}	0.99996
$Y^{(2)}$	2.4490×10^{23}	3.1006×10^{25}	0.99996
$Y^{(3)}$	2.5534×10^{23}	2.6900×10^{25}	0.99995
$Y^{(4)}$	2.9320×10^{23}	1.7831×10^{25}	0.99993
$Y^{(5)}$	1.8356×10^{23}	1.6980×10^{25}	0.99995
$Y^{(6)}$	2.8421×10^{23}	2.7335×10^{25}	0.99995
$Y^{(7)}$	3.9650×10^{22}	2.5584×10^{24}	0.99999
$Y^{(8)}$	4.0000×10^{22}	2.5606×10^{24}	0.99993
$Y^{(9)}$	4.8000×10^{-5}	2.1940×10^{-3}	0.99991
$Y^{(10)}$	8.7180×10^{-10}	7.5894×10^{-8}	0.99984
$Y^{(11)}$	3.1000×10^{-5}	1.7850×10^{-3}	0.99986
$Y^{(12)}$	4.3280×10^{21}	1.6426×10^{23}	0.99937
$Y^{(13)}$	2.2000×10^{-5}	1.3130×10^{-3}	0.99992
$Y^{(14)}$	9.8600×10^{-4}	3.9393×10^{-2}	0.99986

Table 5.1: Accuracy metrics for neural-network predicted normal gas population metrics computed when the values are active.

-	Mean absolute error	Maximum absolute error	R^2
$Y^{(15)}$	1.2200×10^{-4}	3.5110×10^{-2}	0.99999
$Y^{(16)}$	1.5800×10^{23}	4.2754×10^{24}	0.99999
$Y^{(17)}$	1.4344×10^{23}	4.4170×10^{24}	0.99999
$Y^{(18)}$	1.1983×10^{23}	4.3731×10^{24}	0.99999
$Y^{(19)}$	1.0052×10^{23}	4.0676×10^{24}	0.99999
$Y^{(20)}$	8.6064×10^{22}	3.4328×10^{24}	0.99997
$Y^{(21)}$	3.8141×10^{23}	1.5111×10^{25}	0.99999
$Y^{(22)}$	6.6×10^{-5}	2.1170×10^{-3}	0.99999

Table 5.2: Accuracy metrics for the neural-network predicted local fraction of HBS ($Y^{(15)}$) and HBS gas population metrics computed when the values are active.

where f_θ is a fully connected feed-forward neural network with 16 layers and 512 neurons per layer, using ReLU activation.

Table 5.1 and Table 5.2 summarize the performance prediction of the model on the test set. This model shows good one-step accuracy on held-out pairs with regression coefficient very close to 1.

However, incorporating f_θ inside a simulation requires rollouts that repeatedly feed predictions back into the model:

$$\hat{Y}_{n+1} = f_\theta\big(\hat{Y}_n, \hat{A}_n, U_{n+1}, \delta t_{n+1}\big), \qquad \hat{Y}_{n+2} = f_\theta\big(\hat{Y}_{n+1}, \hat{A}_{n+1}, U_{n+2}, \delta t_{n+2}\big), \quad \text{etc.}$$

In this closed-loop use, small one-step errors accumulate and can amplify. Figure 5.3 compares the evolution of some of the variables across time computed with the neural network and the reference simulation where the driving variables are assumed unchanged (no coupling with ALCYONE). It tests only the interaction with the auxiliary update. This result illustrates the problem of error accumulation by producing undesirable oscillations. The single-step objective fits f_θ under the data distribution of true states Y_n, but during simulation the model is driven by its own predictions $\hat{Y}_n$. Crucially, the auxiliary variable A is itself computed from Y and as a result, any error in $\hat{Y}_{n+1}$ immediately contaminates $\hat{A}_{n+1}$. This contaminated auxiliary state then feeds into the next prediction which then pushes the predictor further away from the true trajectory. The resulting drift is stronger than simple error accumulation because it modifies the effective dynamics themselves, more precisely, the system of equations to solve. This compounded distributional shift means that low one-step test error does not guarantee stable multi-step behavior.

5.3.2 Proposed approach: trajectory prediction

In order to address the previous problem, instead of fitting only a single-step map, we propose training the surrogate to reproduce the dynamics consistently over an entire trajectory $\mathcal{T}$.

Given an initial condition (Y_0, A_0) and the subsequent ALCYONE variables:

$$\{(U_k, \delta t_k)\}_{k=1}^{n_h}, \qquad n_h \;=\; \max\{\, k \le n_f : m_k = 1 \,\},$$

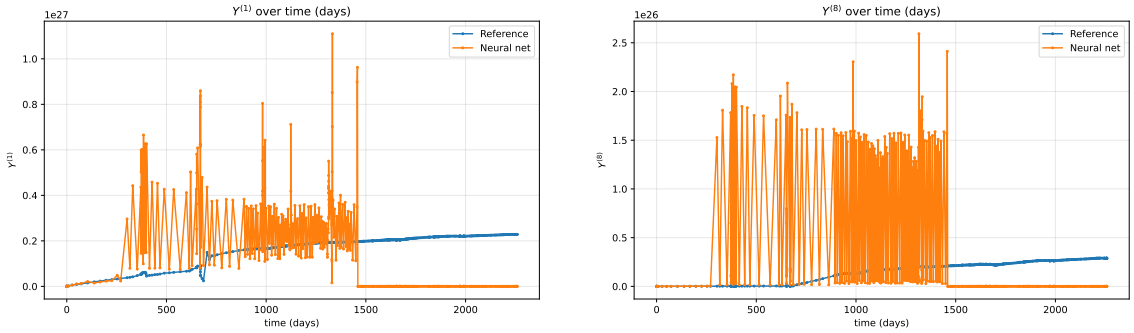
the model rollout is defined recursively as:

$$\hat{Y}_k^* \;=\; f_\theta\big(\hat{Y}_{k-1}, \hat{A}_{k-1}, U_k, \delta t_k\big), \qquad \hat{Y}_k, \hat{A}_k \;=\; \mathcal{G}\big(\hat{A}_{k-1}, \hat{Y}_k^*, U_k\big).$$

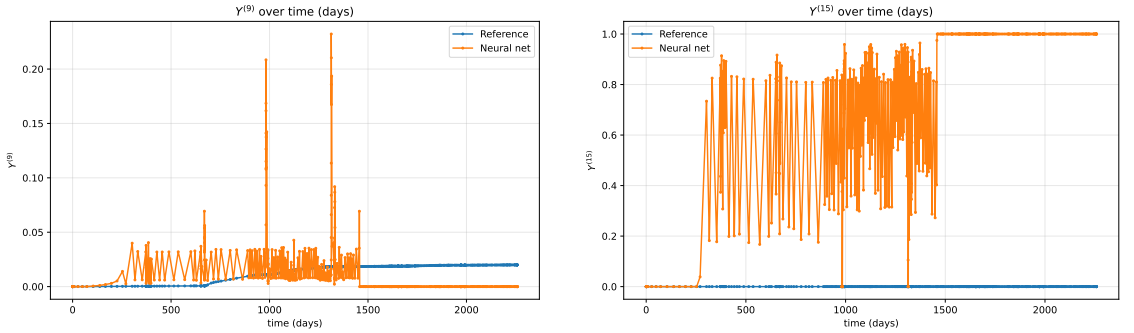
We can then minimize a trajectory loss that penalizes the discrepancy between predicted and the true trajectorie:

$$\mathcal{L}_\theta(\mathcal{T}) \;=\; \sum_{k=1}^{n_h} \Big(\ell\big(\hat{Y}_k,\, Y_k\big) + \ell\big(\hat{A}_k,\, A_k\big)\Big).$$

In our case, however, encoding the auxiliary update $\mathcal{G}$ is not within the scope of the project. Indeed, $\mathcal{G}$ represents almost the entirety of CARACAS code base, and attempting to reproduce it would be equivalent to reimplementing CARACAS.



(a) $Y^{(1)}$ (b) $Y^{(8)}$



(c) $Y^{(9)}$ (d) $Y^{(15)}$

Figure 5.3: Comparison of reference data and the neural net next-step predictor model across time (days) in simulation for example variables.

With this in mind, we set out to emulate the observed trajectory of Y over a time horizon, conditioned only on an initialization (Y_0, A_0) and the subsequent ALCYONE variables $(U_k, \delta t_k)$ using a neural-network based surrogate model. The internal state of CARACAS (Y_k, A_k) can be viewed as a carrier of two kinds of context: traces of the past (history of $(U, \delta t)$ and the initial condition) and cues about which dynamical regime or model is appropriate. To keep prediction from entangling with these modeling choices, we separate the model from the record it relies on by introducing an internal latent representation Z that encodes the relevant variables recoverable from the observables. It is then expected that Z encodes the evolution of the auxiliary variables A without feeding them explicitly to the network. This leaves the predictive pathway insulated from solver-specific internals while preserving the information needed to advance Y. In this case, the surrogate does not implement the internal auxiliary update $\mathcal{G}$ of CARACAS and never consumes or returns A_k for $k \geq 1$ during an uninterrupted run. Formally, we seek a mapping

$$\boxed{(\hat{Y}_1, \ldots, \hat{Y}_n) \;=\; \mathcal{S}_\theta\Big((Y_0, A_0),\, \{(U_k, \delta t_k)\}_{k=1}^{n}\Big).} \tag{5.12}$$

The formulation admits restarts from any checkpoint (Y_k, A_k), enabling starts at arbitrary fuel-life moments. During operation, CARACAS can, in parallel, update the auxiliary variables A using the surrogate outputs so that A remains synchronized for potential hand-offs. The construction of Z will be detailed in the next section.

5.3.3 Model components

Data-driven latent dynamics models have emerged as a powerful approach for simulating stiff deterministic systems. These models follow an encode-evolve-decode strategy (Vijayarangan et al., 2024; Wu et al., 2022; Regazzoni et al., 2024; Watter et al., 2015). The main idea behind these models is to feed the model with trajectory data, so that the model encodes the variables into a latent space, where the evolution along the trajectory is decoupled from the auxiliary update mechanism, and finally the advanced latent variable is then decoded back to the initial space to get next step prediction.

Formally, let E_ϕ, F_θ, and D_ψ be three fully connected neural networks with ReLU activations, with parameters ϕ, θ, ψ, respectively:

$$E_\phi : \mathbb{R}^{d_y+d_a} \to \mathbb{R}^{d_z}, \quad F_\theta : \mathbb{R}^{d_z+d_u} \to \mathbb{R}^{d_z}, \quad D_\psi : \mathbb{R}^{d_z} \to \mathbb{R}^{d_y}.$$

Encoder. E_ϕ is an encoder that maps observations to a latent state. In a general case, we denote the latent state by $Z_i \in \mathbb{R}^{d_z}$ and a specific latent state provided by the encoder as $\hat{Z}_i^{\text{enc}}$:

$$\hat{Z}_i^{\text{enc}} \;=\; E_\phi(Y_i, A_i) \quad \text{for } i = 0, \ldots, n_f. \tag{5.13}$$

Latent dynamics (Euler update). F_θ is a a latent dynamics module that produces a velocity in latent space given the current latent and exogenous input. We evolve the latent state with a residual step scaled by δt:

$$Z_{i+1} \;=\; Z_i \;+\; \delta t_{i+1}\, F_\theta\Big(Z_i, U_{i+1}\Big), \qquad i = 1, \ldots, n_f. \tag{5.14}$$

Here F_θ outputs a latent velocity in $\mathbb{R}^{d_z}$. This scheme is the forward-Euler discretization of a continuous-time latent ODE $\dot{Z} = F_\theta(Z, U)$ that is expected to be learned by the neural network using the data.

Decoder. D_ψ is the decoder that maps a latent state back to the observation space:

$$\hat{Y}_i \;=\; D_\psi(Z_i), \qquad \hat{Y}_i^{\text{ae}} \;=\; D_\psi\Big(E_\phi(Y_i, A_i)\Big),$$

with outputs in $\mathbb{R}^{d_y}$.

Architectures. In our specific application, these three components have the following architecture:

- **Encoder** E_ϕ: 5 layers and 499 neurons per layer, mapping inputs to a latent space of dimension $d_z = 26$.
- **Latent dynamics model** F_θ: 6 layers and 285 neurons per layer, operating on the 26-dimensional latent states and rolled out via Equation (5.14).
- **Decoder** D_ψ: 5 layers and 499 neurons per layer, mapping latent states back to observables.

All hidden layers use ReLU activation. Decoder outputs are applied element-wise as: `softplus` for all outputs except a designated subset of fraction outputs $\mathcal{F}$, which use `sigmoid` (PyTorch implementations).

5.3.4 Training losses

Training uses three complementary objectives:

$$\mathcal{L} \;=\; \mathcal{L}_{\text{rec}} \;+\; \mathcal{L}_{\text{lat}} \;+\; \mathcal{L}_{\text{pred}}. \tag{5.15}$$

Autoencoding (encoder–decoder) reconstruction.

$$\mathcal{L}_{\text{rec}} \;=\; \frac{1}{n_f+1}\sum_{i=0}^{n_f}\Big\|D_\psi\Big(E_\phi(Y_i,A_i)\Big)-Y_i\Big\|_2^2. \tag{5.16}$$

This term enforces that the encoder–decoder pair is approximately identity on the data manifold.

Latent consistency.

$$\mathcal{L}_{\text{lat}} \;=\; \frac{1}{n_f}\sum_{i=1}^{n_f}\Big\|Z_i-E_\phi(Y_i,A_i)\Big\|_2^2. \tag{5.17}$$

This aligns the rolled-out latent Z_i with the encoder's instantaneous estimate $\hat{Z}_i^{\text{enc}}$, tying the dynamics to the representation. The latent dynamics module F_θ is implicitly invoked in the evolution of Z_i, through the update rule described in Equation (5.14).

Prediction loss.

$$\mathcal{L}_{\text{pred}} \;=\; \frac{1}{n_f}\sum_{i=1}^{n_f}\Big\|D_\psi(Z_i)-Y_i\Big\|_2^2. \tag{5.18}$$

We optimize all models using end-to-end backpropagation on the composite loss above with AdamW and decoupled weight decay. To prevent exploding updates, gradients are clipped by global norm each step. After every epoch, we monitor validation loss and apply a `ReduceLROnPlateau` scheduler. Early stopping is triggered if no improvement is observed for 20 epochs, after which we restore the checkpoint with the minimum validation loss.

5.4 Results

After training, the neural network surrogate model is evaluated on the held-out test set (Section 5.2.1) across three settings: point-wise prediction, full trajectory rollout, and integration into simulation.

5.4.1 Single-step evaluation

The neural network is assessed in a point-wise manner by comparing its predictions against the ground-truth simulation outputs at individual time steps. We indeed want to verify that the trajectory model is still reliable for a one-step prediction and we do not lose accuracy compared to the direct approach developed in Section 5.3.1. In this setup, the model is fed with the true values (Y_n, A_n) and we compare its

prediction only for the next step $\hat{Y}_{n+1}$ with the expected value Y_{n+1}. Table 5.3 and Table 5.4 summarize the predictive performance and show that the surrogate achieved high accuracy, with low error across all state variables. These results indicate that the model is able to approximate one-step dynamics reliably.

-	Mean absolute error	Maximum absolute error	R^2
$Y^{(1)}$	6.6341×10^{23}	8.5611×10^{25}	0.99956
$Y^{(2)}$	5.8842×10^{23}	8.4218×10^{25}	0.99957
$Y^{(3)}$	5.9644×10^{23}	7.9136×10^{25}	0.99954
$Y^{(4)}$	5.9604×10^{23}	6.2550×10^{25}	0.99948
$Y^{(5)}$	5.6325×10^{23}	3.8308×10^{25}	0.99952
$Y^{(6)}$	6.3772×10^{23}	7.0565×10^{25}	0.99966
$Y^{(7)}$	1.5499×10^{23}	1.1894×10^{25}	0.99987
$Y^{(8)}$	6.5066×10^{22}	3.0542×10^{24}	0.99989
$Y^{(9)}$	5.6000×10^{-5}	1.2770×10^{-3}	0.99991
$Y^{(10)}$	1.1069×10^{-9}	1.2922×10^{-7}	0.99978
$Y^{(11)}$	6.6×10^{-5}	1.2900×10^{-3}	0.99981
$Y^{(12)}$	1.7730×10^{21}	5.7733×10^{23}	0.99991
$Y^{(13)}$	4.7000×10^{-5}	2.7220×10^{-3}	0.99943
$Y^{(14)}$	1.3240×10^{-3}	5.4288×10^{-2}	0.99976

Table 5.3: Individual time steps accuracy metrics for neural-network predicted normal gas population metrics computed when the values are active.

-	Mean absolute error	Maximum absolute error	R^2
$Y^{(15)}$	7.4500×10^{-4}	3.5332×10^{-2}	0.99999
$Y^{(16)}$	1.9684×10^{23}	6.2043×10^{24}	0.99999
$Y^{(17)}$	2.0170×10^{23}	6.9602×10^{24}	0.99999
$Y^{(18)}$	1.7187×10^{23}	7.3519×10^{24}	0.99999
$Y^{(19)}$	1.3497×10^{23}	7.5825×10^{24}	0.99998
$Y^{(20)}$	1.0597×10^{23}	4.7902×10^{24}	0.99994
$Y^{(21)}$	5.0437×10^{23}	1.5221×10^{25}	0.99998
$Y^{(22)}$	6.4000×10^{-5}	4.8940×10^{-3}	0.99998

Table 5.4: Individual time steps accuracy metrics for neural-network predicted local fraction of HBS ($Y^{(15)}$) and HBS gas population metrics computed when the values are active.

5.4.2 Trajectory evaluation

To assess long-horizon predictive performance, the surrogate is rolled out over entire simulation trajectories by recursively feeding its own latent variable predictions. More precisely, for these tests, we feed the neural network with the initial state (Y_0, A_0) and we compute recursively all the subsequent values $\hat{Y}_{n+1}$ by evolving

(through F_θ) and decoding (through D_ψ) the latent variable $\hat{Z}_n$. Importantly, these tests are performed on simulations that extend well beyond the trajectory lengths used during training, providing a measure of the model ability to generalize outside its training horizon. At each time step, point-wise errors are computed and then aggregated across the entire test data set. Table 5.5 and Table 5.6 summarize the performance prediction of the model on the test set over entire simulation trajectories. As expected, the aggregated accuracy is slightly lower than the isolated single-step results previously presented in Section 5.4.1, reflecting the effect of error accumulation over long horizons.

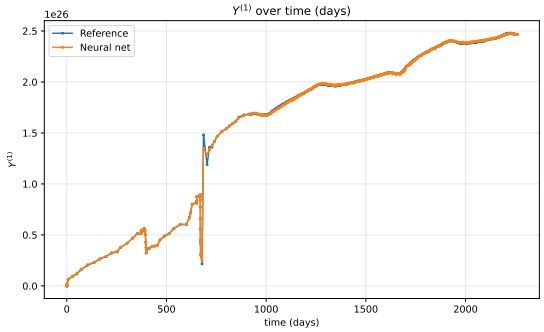
-	Mean absolute error	Maximum absolute error	R^2
$Y^{(1)}$	9.1278×10^{23}	1.3191×10^{26}	0.99865
$Y^{(2)}$	8.6090×10^{23}	1.2938×10^{26}	0.99862
$Y^{(3)}$	8.3811×10^{23}	1.2239×10^{26}	0.99860
$Y^{(4)}$	7.9450×10^{23}	1.0823×10^{26}	0.99864
$Y^{(5)}$	6.8932×10^{23}	7.9508×10^{25}	0.99890
$Y^{(6)}$	9.1130×10^{23}	1.0922×10^{26}	0.99891
$Y^{(7)}$	2.5577×10^{23}	1.4213×10^{25}	0.99955
$Y^{(8)}$	1.2949×10^{23}	7.1560×10^{24}	0.99954
$Y^{(9)}$	9.9×10^{-5}	1.5820×10^{-3}	0.99969
$Y^{(10)}$	2.0841×10^{-9}	1.2922×10^{-7}	0.99916
$Y^{(11)}$	$1. \times 10^{-4}$	3.5530×10^{-3}	0.99950
$Y^{(12)}$	2.3494×10^{21}	1.1619×10^{24}	0.99856
$Y^{(13)}$	5.7000×10^{-5}	8.0150×10^{-3}	0.99875
$Y^{(14)}$	2.4030×10^{-3}	5.5494×10^{-2}	0.99925

Table 5.5: Trajectory accuracy metrics for neural-network predicted normal gas population metrics computed when the values are active.

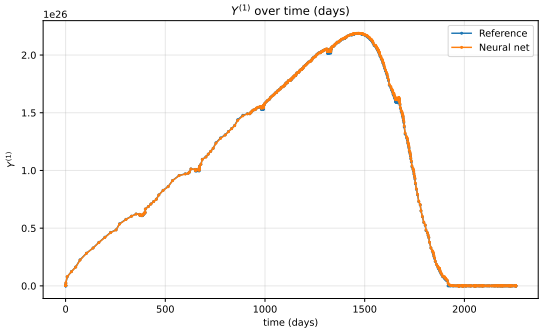
-	Mean absolute error	Maximum absolute error	R^2
$Y^{(15)}$	2.8100×10^{-4}	9.7970×10^{-2}	0.99952
$Y^{(16)}$	6.1079×10^{23}	1.3728×10^{25}	0.99980
$Y^{(17)}$	5.8635×10^{23}	1.3784×10^{25}	0.99978
$Y^{(18)}$	5.5850×10^{23}	1.3770×10^{25}	0.99974
$Y^{(19)}$	5.1065×10^{23}	1.3331×10^{25}	0.99962
$Y^{(20)}$	3.6224×10^{23}	1.0002×10^{25}	0.99909
$Y^{(21)}$	1.8153×10^{24}	3.8940×10^{25}	0.99974
$Y^{(22)}$	2.5300×10^{-4}	5.0860×10^{-3}	0.99961

Table 5.6: Trajectory accuracy metrics for the neural-network predicted local fraction of HBS ($Y^{(15)}$) and HBS gas population metrics computed when the values are active.

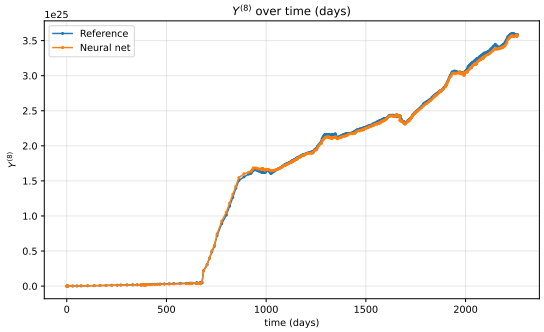
However, the surrogate maintains stability and preserves the qualitative behavior of the underlying system. It cannot be directly compared to the point-wise model developed in Section 5.3.1 since it is not integrated in the simulation (it is only a trajectory prediction without coupling with ALCYONE).



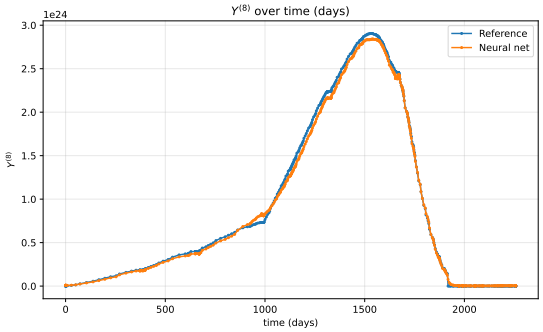
(a) $Y^{(1)}$ at pellet center



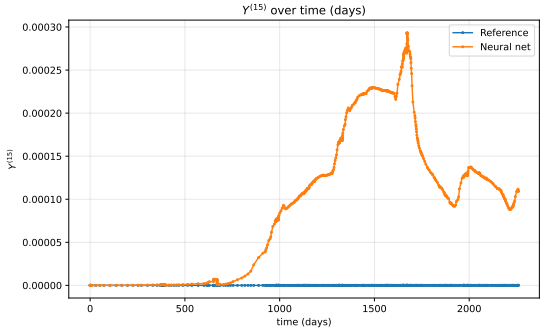
(b) $Y^{(1)}$ at pellet periphery



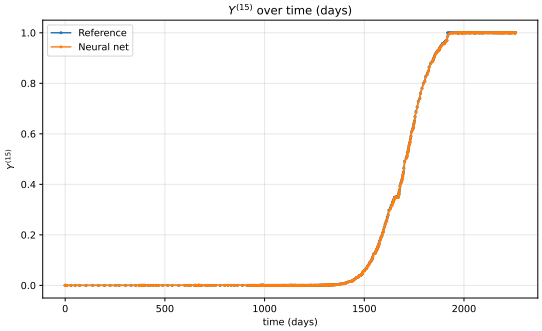
(c) $Y^{(8)}$ at pellet center



(d) $Y^{(8)}$ at pellet periphery

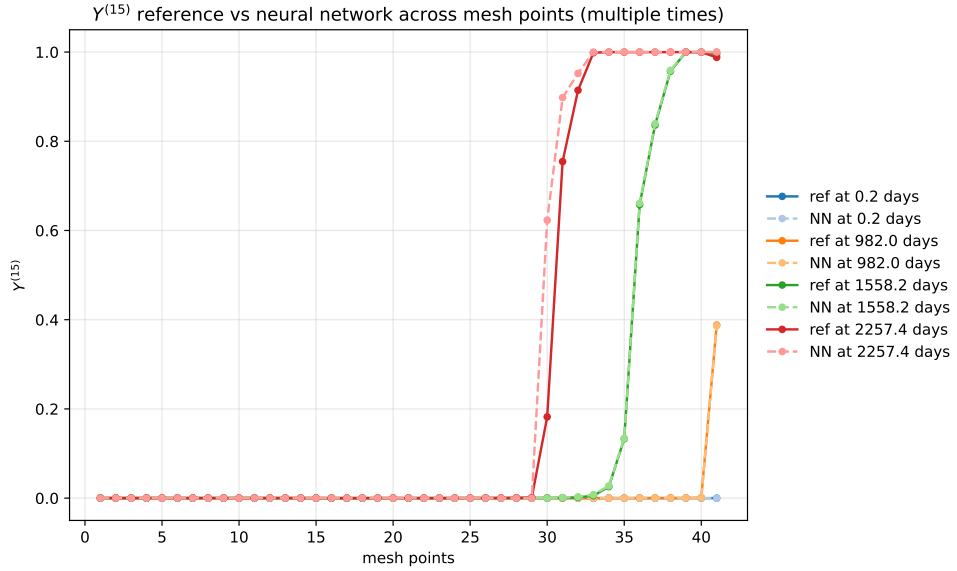


(e) $Y^{(15)}$ at pellet center

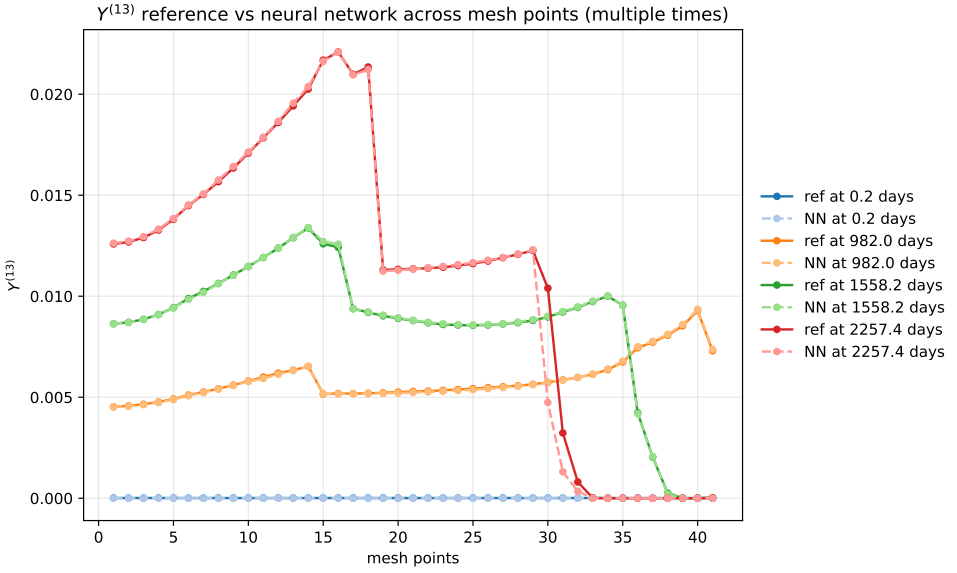


(f) $Y^{(15)}$ at pellet periphery

Figure 5.4: Neural network predictions vs. reference simulation for multiple variables over a full trajectory at two spatial locations close to the pellet center (left column) and close to the pellet periphery (right column).



(a) $Y^{(15)}$



(b) $Y^{(13)}$

Figure 5.5: Neural network predictions vs. reference simulation for $Y^{(15)}$ and $Y^{(13)}$ over mesh points at different times.

Figure 5.4 shows a comparison between the surrogate and the numerical model over a selected set of variables at the pellet centre and periphery. The deviations from the reference simulations grow gradually with time but remain bounded, showing that the model can function beyond its training scope. However, the model fails to capture episodes of strong oscillations, tending to underestimate their amplitude. Figure 5.4a shows an example of this for variable $Y^{(1)}$ at a centre of pellet trajectory where the reference exhibits a large-amplitude oscillation and the model underestimates the amplitude near peaks. In addition, we observe that small oscillatory artifacts appear in the surrogate outputs, seeping in from oscillations present in the driving variables. Figure 5.4e illustrates the small oscillatory ripples in the surrogate output that are absent in the reference value. Also, relative errors are proportionally larger when the underlying state values are small, while large-magnitude values are approximated more accurately. For instance, Figure 5.4d shows a periphery trajectory about an order of magnitude smaller than Figure 5.4c, with deviations more pronounced in the periphery. Finally, the model successfully replicates the behavior of the numerical model across some phase transitions, while failing to do so for others.

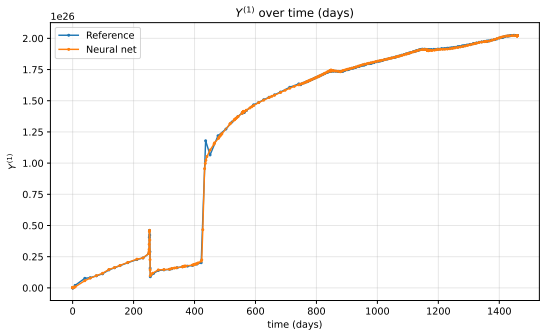
As a case in point, Figure 5.5 presents the evolution of two variables across a rod cross-section at multiple points in the fuel lifetime. The first variable $Y^{(15)}$ undergoes a single phase transition that is well captured until a certain stage, after which the discrepancy grows and is most pronounced near end of life. The second variable $Y^{(13)}$ exhibits two phase transitions: one is accurately reproduced throughout the lifetime, while the other deviates similarly to $Y^{(15)}$.

5.4.3 In simulation evaluation

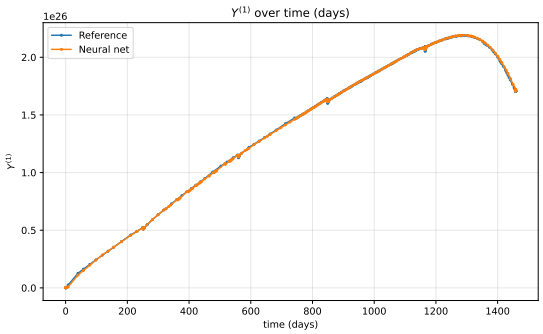
To assess the surrogate performance in a full simulation environment, the neural network model has been integrated into ALCYONE and tested in three simulation runs. Of these, only one run completed to the prescribed final time. The remaining two runs terminated prematurely due to non-convergence of the mechanical submodel.

Figures 5.6 and 5.7 show the time evolution comparisons between the reference and the neural network model of the same selection of variables as in Figure 5.4, corresponding to the completed coupled simulation and a prematurely terminated coupled simulation, respectively. Notably, these results indicate that the proposed approach mitigates the problem observed with the point-wise model proposed in Section 5.3.1. Across the coupled runs, most variables exhibite no significant deviations relative to the corresponding independent (uncoupled) NN run from Section 5.4.2. For these, differences are subtle and not readily noticeable. As an illustration, Figures 5.6a, 5.6b, 5.6c, 5.6e, and 5.6f are qualitatively similar to their counterparts in Figure 5.4. Sometimes the surrogate shows small but noticeable shifts, these deviations do not alter the overall qualitative behavior of the system, but they do manifest as slight offsets in amplitude or phase compared to the uncoupled run. For instance, Figure 5.6c illustrates this behavior, showing modest offsets relative to the independent reference while preserving the overall trajectory shape.

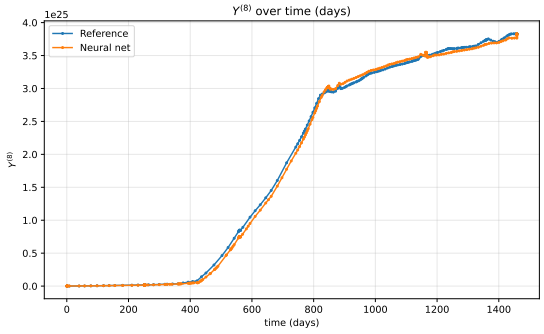
Finally, for the variable $Y^{(9)}$, relatively large shifts are observed from the outset of the simulation, particularly at small values. Figure 5.8 exemplifies this effect. The top panel reports the variable behavior in the independent (uncoupled) run,



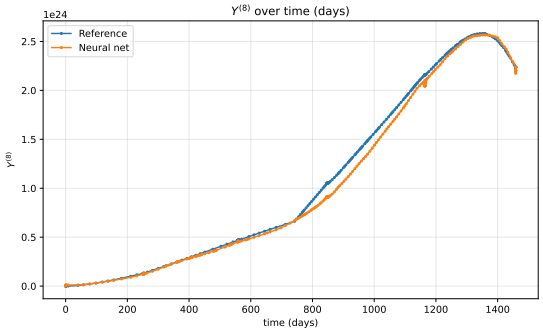
(a) $Y^{(1)}$ at pellet center



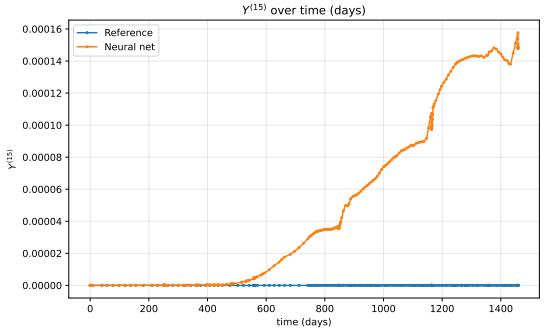
(b) $Y^{(1)}$ at pellet periphery



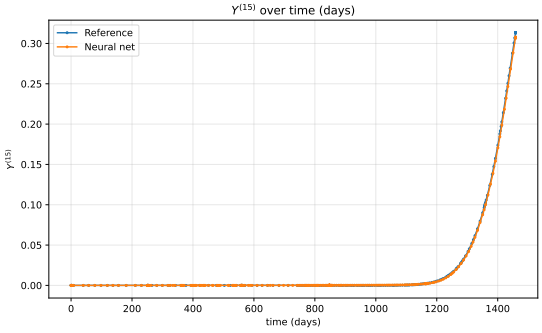
(c) $Y^{(8)}$ at pellet center



(d) $Y^{(8)}$ at pellet periphery

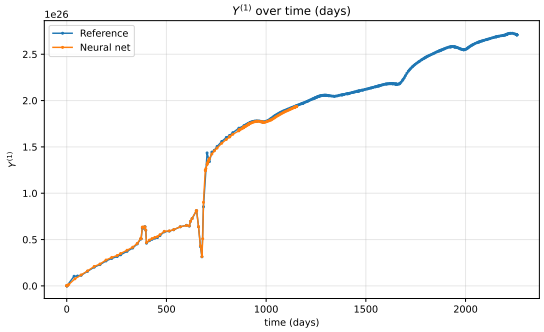


(e) $Y^{(15)}$ at pellet center

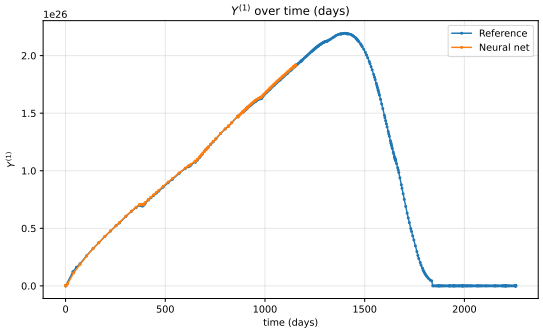


(f) $Y^{(15)}$ at pellet periphery

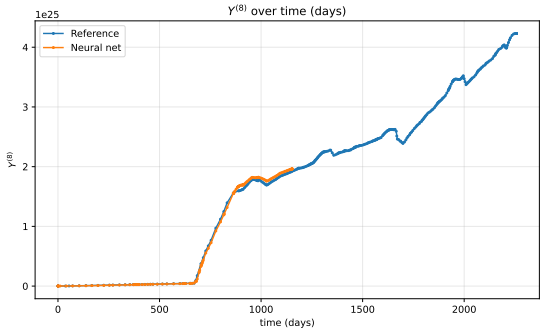
Figure 5.6: Neural network predictions vs. reference simulation for multiple variables over a completed simulation integration case at two spatial locations close to the pellet center (left column) and close to the pellet periphery (right column).



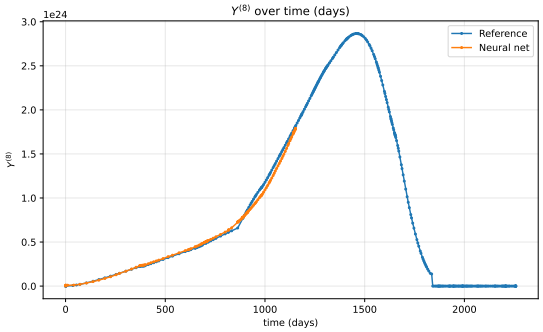
(a) $Y^{(1)}$ at pellet center



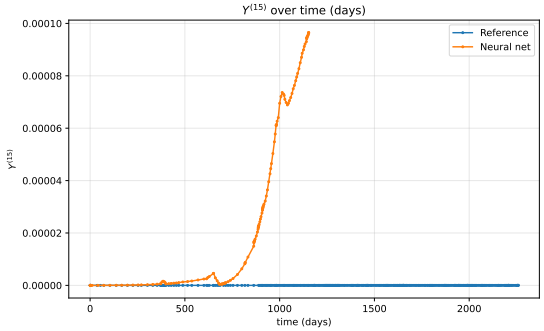
(b) $Y^{(1)}$ at pellet periphery



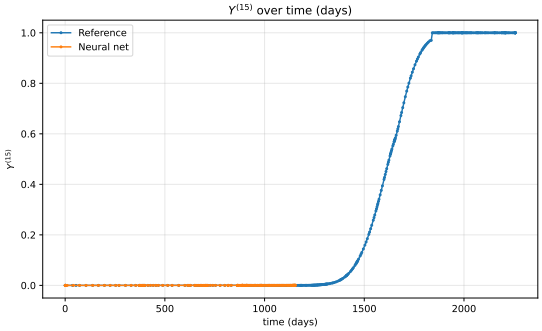
(c) $Y^{(8)}$ at pellet center



(d) $Y^{(8)}$ at pellet periphery



(e) $Y^{(15)}$ at pellet center



(f) $Y^{(15)}$ at pellet periphery

Figure 5.7: Neural network predictions vs. reference simulation for multiple variables over an incomplete simulation integration case at two spatial locations close to the pellet center (left column) and close to the pellet periphery (right column).

where the neural network closely follows the reference trajectory across the full time horizon. The middle and bottom panels report the same variable under coupling: in these cases, the neural network response departs from the reference shortly after initialization, with the largest relative discrepancies occurring at low amplitudes.

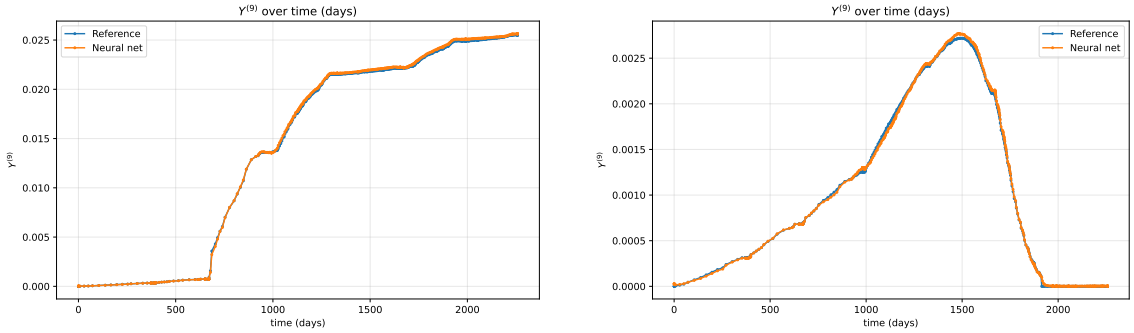
Here, two mechanisms plausibly explain the departures observed in Figure 5.8. First, the numerical model applies correction terms to the neural network outputs to enforce consistency constraints, as explained in the description of Equation (5.3). While these corrections improve physical plausibility, they are not constant but depend on the auxiliary variables. Minor errors made by the neural network at early stages lead to auxiliary variable values that deviate from those in the reference simulation. These deviations trigger corrections that are not consistent with the reference trajectory. As a result, the imposed corrections amplify rather than mitigate the neural network error. Second, once this imbalance exceeds the convergence threshold of the mechanical solver, the iteration process fails to stabilize, and the simulation terminates.

A meaningful runtime comparison between the surrogate-coupled simulation and the reference simulation has not been feasible during the time of this project. ALCYONE architecture does not support input vectorization, so the surrogate is called independently for each spatial point rather than in parallel batches. This constraint introduces substantial overhead in the coupled setting, preventing the neural network, designed for vectorized evaluation, from achieving its natural efficiency. As a result, the measured runtimes primarily reflect limitations of the solver implementation rather than the surrogate intrinsic cost. Enabling batched evaluation (e.g., via an interface layer that aggregates queries across all spatial points before calling the surrogate) would allow vectorized surrogate calls and make the overall simulation significantly faster. To achieve this objective, a substantion modification of ALCYONE architecture is then required.

5.4.4 Comment on the application of BBAS monitoring for OMS Detection

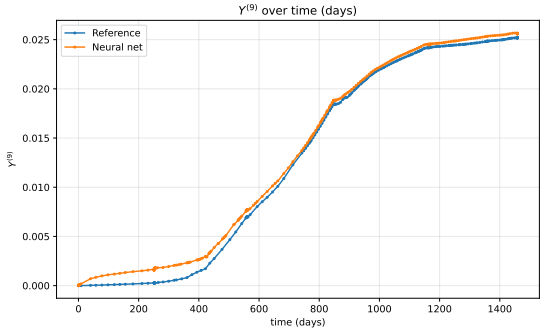
Given the project timeline, BBAS deployment on CARACAS was deferred to future work. Within the available timeframe, we prioritized establishing a surrogate that integrates robustly with the coupled simulation and completing the associated verification, which we judged prerequisite to meaningful monitoring. In addition, a credible OMS evaluation requires a curated set of out-of-scope data, assembling such data to an adequate standard could not be completed within the thesis window due to the time-consuming simulations.

While the BBAS method is applicable to CARACAS surrogate, its deployment would require several targeted adjustments considering the specific architecture of the model (composed of three components) and the quantity of training data. First, the algorithm should be applied separately to each of the three submodels, with submodel-specific thresholds. Second, threshold calibration should be performed per submodel: each submodel is associated to an independant anomaly score. Then, some aggregation rule (e.g., max or veto) should be applied to these anomaly scores to form a single alarm. Finally, another practical constraint for deploying BBAS on

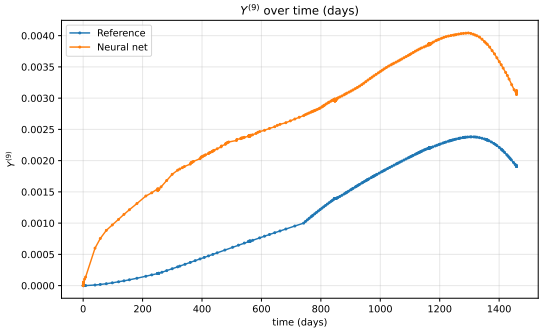


(a) $Y^{(9)}$ at pellet center

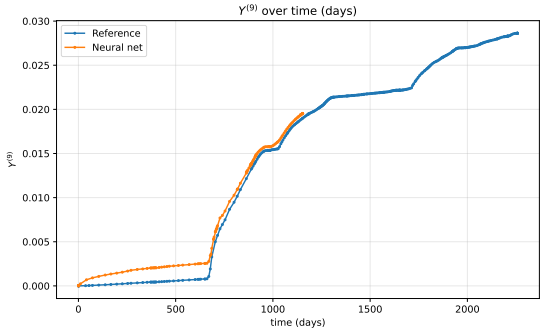
(b) $Y^{(9)}$ at pellet periphery



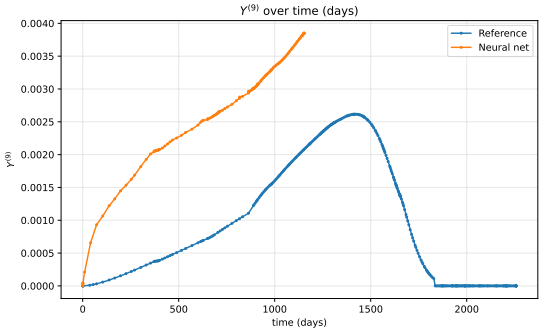
(c) $Y^{(9)}$ at pellet center



(d) $Y^{(9)}$ at pellet periphery



(e) $Y^{(9)}$ at pellet center



(f) $Y^{(9)}$ at pellet periphery

Figure 5.8: Neural network predictions compared with reference simulations for $Y^{(9)}$ across three runs: (top) the trajectory evolution example from Section 5.4.2, (middle) the completed simulation integration case, and (bottom) the incomplete integration case.

CARACAS is the volume of evaluation points, which makes agglomerative clustering computationally infeasible. A scalable remedy is to learn the activation-pattern clusters on a subsample and then process the remaining points in a single pass, assigning each to its nearest centroid and expanding the corresponding box bounds when a point falls outside its enclosure. These steps are straightforward but were out of reach within the present schedule. They define a clear path for BBAS integration on CARACAS surrogate model in subsequent work.

5.5 Conclusion and perspectives

This chapter has presented the development and integration of a neural network surrogate for fission gas behaviour into ALCYONE. It evaluates its performance both as a standalone predictor and when embedded in the solver. The results show clear potential and also expose current limitations that arise at the interface between the surrogate and ALCYONE. Together, the findings outline a clear path toward improving the surrogate model.

Modeling choices in the original solver and their impact. The original numerical solver is designed to focus on the most relevant physical phenomena for each local regime. This design yields excellent computational performance, but it also introduces behavioral and functional discontinuities: the governing update relations can switch abruptly when regime indicators change. As a consequence, spatially adjacent points can follow different update rules and therefore exhibit markedly different dynamics despite being close in space and and subject to similar comparable driving variables. For a data-driven surrogate that assumes smoothness in its input-output map, these regime switches manifest as sharp transitions that are intrinsically hard to approximate with a single smooth function. The surrogate was able to learn a smoothed representation of these transitions, capturing the dominant trends. This smoothing comes at a cost, it forces the network to expend capacity on implicitly encoding change-points and phase shifts in its latent space, such representations are intrinsically approximate and therefore imprecise.

The auxiliary update mechanism. In the present study, the auxiliary update mechanism was treated as black-box components, and the surrogate was conditioned on its observable effects rather than on its internal logic. This arrangement is consistent with the project scope and accessibility constraints to CARACAS and ALCYONE. Nevertheless, any extension of this work would benefit from explicitly encoding the auxiliary update function and using it as input to the latent dynamics so that the model can anticipate regime changes. The encoding would also provide additional training objectives for the model outputs. The encoding would likely yield immediate gains in accuracy and efficiency. Moreover, the loss can include physical constraints.

Embedding physical constraints in the loss. To promote consistency among outputs and align the surrogate with governing physics, the training objective

can include physical constraint penalty terms. In addition, downstream quantities computed from the surrogate outputs and consumed by ALCYONE can be included in the loss, aligning training with what the solvers actually use.

Solver integration and development workflow. Progress would be accelerated by adding a dedicated testing interface for surrogates within ALCYONE that supplies batched inputs to the model, runs it against standardized test scenarios, and returns clear diagnostics. This interface would shorten development cycles, enable reproducible benchmarking across model variants, simplify the evaluation of surrogate design choices, and provide a path to future performance improvements.

Chapter 6

Conclusion and perspectives

This thesis examined the reliability of neural-network surrogate models. These surrogates can emulate physical codes even with high-dimensional inputs and richly parameterized experiments, but reliability must be handled differently from numerical solvers: numerical models have a well-defined scope, whereas deep networks are tied to their training distribution and offer only average-error guarantees, leaving them vulnerable, even "within distribution", to rare cases where the learned mapping no longer applies. We argued that model misspecification constrains uncertainty derived from uncertainty on the model itself: such estimates remain meaningful in-distribution, but require recalibration outside the model support. This motivated reframing reliability as out-of-model-scope (OMS) detection and embedding standard OOD techniques within a runtime-verification framework. Among candidate methods, a bounding-box abstraction based method proved interesting: our geometric interpretation shows how it can approximate arbitrary operational domains, and how aggregating information across multiple layers, rather than relying only on the pre-final layer, captures complementary geometric cues. Evaluation on a widely used benchmark indicates the approach is promising and merits further study and domain-specific integration.

This thesis concentrated on an important surrogate target: local constitutive models embedded in multiphysics simulations, which are invoked at every mesh point and time step and therefore can dominate runtime and scalability. We focused on two such models in PWR fuel analysis: OpenCalphad for thermochemical evaluations and CARACAS for fission-gas behavior. For OpenCalphad, we applied the methodology end-to-end and used the monitoring framework to enforce reliability at runtime, triggering fallback to the high-fidelity model whenever out-of-scope inputs were detected so that simulation integrity was preserved. For CARACAS, we analyzed the limits of a data-only surrogate trained without explicit physical constraints: despite high predictive accuracy, the model integrated poorly with the simulation workflow, underscoring the need to incorporate physics-based terms to the loss to achieve stable, simulation-compatible behavior.

Perspectives

This section outlines how the thesis findings translate into a concrete roadmap for reliable neural-network surrogates in multiphysics workflows. The perspectives for the specific applications on OpenCalphad and CARACAS are given in Section 4.4 and Section 5.5, respectively. Here, we propose the following axes for future studies on the proposed monitoring approach to provide reliability guarantees for surrogate models in physics simulations.

Method efficiency

Box budget. While the bounding-box abstraction can, in principle, approximate any compact domain to arbitrary precision, this asymptotic claim is of limited practical value: exhaustive partitioning by activation patterns is not possible with limited data. A more compelling justification should connect the method to the generalization mechanisms of neural networks if possible. A practical open question is the optimal range for the number of boxes: too few boxes underfit the operational domain and too many may unnecessarily raise computational cost.

Feature weighting. The current monitor treats all feature dimensions equally, which is unlikely to be optimal. Introducing a principled weighting of features may therefore be beneficial, potentially improving sensitivity to genuinely informative directions while reducing the influence of noisy or redundant components.

Scalable scoring. Bounding-box monitors scale favorably in memory. However, naïve scoring can be limiting when the number of boxes increases. Future work can adapt algorithmic ideas from approximate nearest neighbors and spatial indexing to accelerate queries (Johnson et al., 2019). It would help conserve the benefits of using a surrogate model approach to accelerate simulations.

Monitor exploitation

Incremental and principled updates. A key advantage of the bounding-box abstraction is cheap precision: adding a box costs roughly as much memory as storing two points, and updating existing bounds requires no additional cost. However, indiscriminate expansion risks codifying accidental accuracy on isolated observations. If a point is rejected by the monitor but the model happens to be accurate on that single evaluation, it must not automatically trigger a box update. Future work can therefore formalize update rules.

One simple policy is an admission buffer: if an input is deemed in-scope but has only a small anomaly score, the corresponding box may be updated. Another is to relax bounds that generate repeated false positives for a specific box in a controlled manner. A complementary strategy is to leverage spatiotemporal coherence: treat a flagged in-scope point as incidental when neighboring evaluations in space or time are correctly rejected, and as persistent when false positives recur locally, using these patterns to decide whether and how to adjust box limits.

Active learning. The monitoring approach can also serve as a compass for data acquisition. During deployment, inputs that are genuinely flagged as out-of-model-scope can be passed to a secondary novelty monitor dedicated to active learning, whose role would be to suppress redundant logging from the same region of state/feature space. Only novel OMS cases will be recorded, yielding a compact, high-value dataset that can be fed back into subsequent training cycles to refine the surrogate in deficient regions.

Appendix A

Detailed experiments results for Chapter 3

This section presents the detailed results of the comparison with the state-of-the-art methods using the OpenOOD benchmark. Sections A.1, A.2, A.2.1 and A.3 present the results respectively for CIFAR-10, CIFAR-100, ImageNet-200 and ImageNet-1k. The reported results come from the associated papers and from the OpenOOD leader board. The performance is not always indicated in details in the associated paper. In this case, we indicate a symbol "−" in the tables.

We provide here the list of the compared state-of-the-art methods and their associated paper: OpenMax (Bendale and Boult, 2015), MSP (Hendrycks and Gimpel, 2017), TempScale (Guo et al., 2017), ODIN (Liang et al., 2018), MDS and MDSEns (Lee et al., 2018), RMDS (Ren et al., 2021a), Gram (Sastry and Oore, 2020), EBO (Liu et al., 2020), OpenGAN (Kong and Ramanan, 2021), GradNorm (Huang et al., 2021), ReAct (Sun et al., 2021), MLS and KLM (Hendrycks et al., 2022), VIM (Wang et al., 2022), KNN (Sun et al., 2022), DICE (Sun and Li, 2022), RankFeat (Song et al., 2022), ASH (Djurisic et al., 2023), SHE (Zhang et al., 2023a), GEN (Liu et al., 2023), Weiper+KLD (Granz et al., 2024), NAC-UE (Liu et al., 2024), CombOOD (Rajasekaran et al., 2024), SCALE (Xu et al., 2024), fDBD (Liu and Qin, 2024).

A.1 Full CIFAR-10 results

Table A.1: CIFAR-10 FarOOD Benchmark results on the ResNet18 architecture.

Method	MNIST		SVHN		Textures		Places365		Average	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
OpenMax	23.33 ± 4.67	90.50 ± 0.44	25.40 ± 1.47	89.77 ± 0.45	31.50 ± 4.05	89.58 ± 0.60	38.52 ± 2.27	88.63 ± 0.28	29.69 ± 1.21	89.62 ± 0.19
MSP	23.64 ± 5.81	92.63 ± 1.57	25.82 ± 1.64	91.46 ± 0.40	34.96 ± 4.64	89.89 ± 0.71	42.47 ± 3.81	88.92 ± 0.47	31.72 ± 1.84	90.73 ± 0.43
TempScale	23.53 ± 7.05	93.11 ± 1.77	26.97 ± 2.65	91.66 ± 0.52	38.16 ± 5.89	90.01 ± 0.74	45.27 ± 4.50	89.11 ± 0.52	33.48 ± 2.39	90.97 ± 0.52
ODIN	23.83 ± 12.34	95.24 ± 1.96	68.61 ± 0.52	84.58 ± 0.77	67.70 ± 11.06	86.94 ± 2.26	70.36 ± 6.96	85.07 ± 1.24	57.62 ± 4.24	87.96 ± 0.61
MDS	27.30 ± 3.55	90.10 ± 2.41	25.96 ± 2.52	91.18 ± 0.47	27.94 ± 4.20	92.69 ± 1.06	47.67 ± 4.54	84.90 ± 2.54	32.22 ± 3.40	89.72 ± 1.36
MDSEns	**1.30** ± 0.51	**99.17** ± 0.41	74.34 ± 1.04	66.56 ± 0.58	76.07 ± 0.17	77.40 ± 0.28	94.16 ± 0.33	52.47 ± 0.15	61.47 ± 0.48	73.90 ± 0.27
RMDS	21.49 ± 2.32	93.22 ± 0.80	23.46 ± 1.48	91.84 ± 0.26	25.25 ± 0.53	92.23 ± 0.23	31.20 ± 0.28	91.51 ± 0.11	25.35 ± 0.73	92.20 ± 0.21
Gram	70.30 ± 8.96	72.64 ± 2.34	33.91 ± 17.35	91.52 ± 4.45	94.64 ± 2.71	62.34 ± 8.27	90.49 ± 1.93	60.44 ± 3.41	72.34 ± 6.73	71.73 ± 3.20
EBO	24.99 ± 12.93	94.32 ± 2.53	35.12 ± 6.11	91.79 ± 0.98	51.82 ± 6.11	89.47 ± 0.70	54.85 ± 6.52	89.25 ± 0.78	41.69 ± 5.32	91.21 ± 0.92
OpenGAN	79.54 ± 19.71	56.14 ± 24.08	75.27 ± 26.93	52.81 ± 27.60	83.95 ± 14.89	56.14 ± 18.26	95.32 ± 4.45	53.34 ± 5.79	83.52 ± 11.63	54.61 ± 15.51
GradNorm	85.41 ± 4.85	63.72 ± 7.37	91.65 ± 2.42	53.91 ± 6.36	98.09 ± 0.49	52.07 ± 4.09	92.46 ± 2.28	60.50 ± 5.33	91.90 ± 2.23	57.55 ± 3.22
ReAct	33.77 ± 18.00	92.81 ± 3.03	50.23 ± 15.98	89.12 ± 3.19	51.42 ± 11.42	89.38 ± 1.49	44.20 ± 3.35	90.35 ± 0.78	44.90 ± 8.37	90.42 ± 1.41
MLS	25.06 ± 12.87	94.15 ± 2.48	35.09 ± 6.09	91.69 ± 0.94	51.73 ± 6.13	89.41 ± 0.71	54.84 ± 6.51	89.14 ± 0.76	41.68 ± 5.27	91.10 ± 0.89
KLM	76.22 ± 12.09	85.00 ± 2.04	59.47 ± 7.06	84.99 ± 1.18	81.95 ± 9.95	82.35 ± 0.33	95.58 ± 2.12	78.37 ± 0.33	78.31 ± 4.84	82.68 ± 0.21
VIM	18.36 ± 1.42	94.76 ± 0.38	19.29 ± 0.41	94.50 ± 0.48	21.14 ± 1.83	95.15 ± 0.34	41.43 ± 2.17	89.49 ± 0.39	25.05 ± 0.52	93.48 ± 0.24
KNN	20.05 ± 1.36	94.26 ± 0.38	22.60 ± 1.26	92.67 ± 0.30	24.06 ± 0.55	93.16 ± 0.24	30.38 ± 0.63	91.77 ± 0.23	24.27 ± 0.40	92.96 ± 0.14
DICE	30.83 ± 10.54	90.37 ± 5.97	36.61 ± 4.74	90.02 ± 1.77	62.42 ± 4.79	81.86 ± 2.35	77.19 ± 12.60	74.67 ± 4.98	51.76 ± 4.42	84.23 ± 1.89
RankFeat	61.86 ± 12.78	75.87 ± 5.22	64.49 ± 7.38	68.15 ± 7.44	59.71 ± 9.79	73.46 ± 6.49	43.70 ± 7.39	85.99 ± 3.04	57.44 ± 7.99	75.87 ± 5.06
ASH	70.00 ± 10.56	83.16 ± 4.66	83.64 ± 6.48	73.46 ± 6.41	84.59 ± 1.74	77.45 ± 2.39	77.89 ± 7.28	79.89 ± 3.69	79.03 ± 4.22	78.49 ± 2.58
SHE	42.22 ± 20.59	90.43 ± 4.76	62.74 ± 4.01	86.38 ± 1.32	84.60 ± 5.30	81.57 ± 1.21	76.36 ± 5.32	82.89 ± 1.22	66.48 ± 5.98	85.32 ± 1.43
GEN	23.00 ± 7.75	93.83 ± 2.14	28.14 ± 2.59	91.97 ± 0.66	40.74 ± 6.61	90.14 ± 0.76	47.03 ± 3.22	89.46 ± 0.65	34.73 ± 1.58	91.35 ± 0.69
Weiper+KLD	19.98 ± 4.08	94.40 ± 1.47	19.48 ± 4.08	94.30 ± 0.41	19.48 ± 0.18	93.20 ± 0.46	31.88 ± 1.20	90.60 ± 0.24	23.72 ± 0.79	93.12 ± 0.34
NAC-UE	15.14 ± 2.60	94.86 ± 1.36	14.33 ± 1.24	96.05 ± 0.47	17.03 ± 0.59	95.64 ± 0.44	**26.73** ± 0.80	91.85 ± 0.28	18.31 ± 0.92	94.60 ± 0.50
CombOOD	–	91.28 ± –	–	**99.06** ± –	–	95.98 ± –	–	92.27 ± –	–	94.64 ± –
SCALE	–	–	**5.80** ± –	98.72 ± –	23.42 ± –	94.97 ± –	38.69 ± –	91.74 ± –	–	86.39 ± –
fDBD	–	–	–	–	–	–	–	–	–	93.16 ± –
BBAS	14.71 ± 1.69	96.26 ± 0.67	7.93 ± 0.43	98.66 ± 0.11	**15.83** ± 0.36	**96.60** ± 0.11	27.21 ± 0.82	**92.61** ± 0.32	**16.42** ± 0.34	**96.03** ± 0.14

Table A.2: CIFAR-10 NearOOD Benchmark results on the ResNet18 architecture.

Method	CIFAR-100		TIN		Average	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
OpenMax	48.06 ± 3.25	86.91 ± 0.31	39.18 ± 1.44	88.32 ± 0.28	43.62 ± 2.27	87.62 ± 0.29
MSP	53.08 ± 4.86	87.19 ± 0.33	43.27 ± 3.00	88.87 ± 0.19	48.17 ± 3.92	88.03 ± 0.25
TempScale	55.81 ± 5.07	87.17 ± 0.40	46.11 ± 3.63	89.00 ± 0.23	50.96 ± 4.32	88.09 ± 0.31
ODIN	77.00 ± 5.74	82.18 ± 1.87	75.38 ± 6.42	83.55 ± 1.84	76.19 ± 6.08	82.87 ± 1.85
MDS	52.81 ± 3.62	83.59 ± 2.27	46.99 ± 4.36	84.81 ± 2.53	49.90 ± 3.98	84.20 ± 2.40
MDSEns	91.87 ± 0.10	61.29 ± 0.23	92.66 ± 0.42	59.57 ± 0.53	92.26 ± 0.20	60.43 ± 0.26
RMDS	43.86 ± 3.49	88.83 ± 0.35	33.91 ± 1.39	90.76 ± 0.27	38.89 ± 2.39	89.80 ± 0.28
Gram	91.68 ± 2.24	58.33 ± 4.49	90.06 ± 1.59	58.98 ± 5.19	90.87 ± 1.91	58.66 ± 4.83
EBO	66.60 ± 4.46	86.36 ± 0.58	56.08 ± 4.83	88.80 ± 0.36	61.34 ± 4.63	87.58 ± 0.46
OpenGAN	94.84 ± 3.83	52.81 ± 7.69	94.11 ± 4.21	54.62 ± 7.68	94.48 ± 4.01	53.71 ± 7.68
GradNorm	94.54 ± 1.11	54.43 ± 1.59	94.89 ± 0.60	55.37 ± 0.41	94.72 ± 0.82	54.90 ± 0.98
ReAct	67.40 ± 7.34	85.93 ± 0.83	59.71 ± 7.31	88.29 ± 0.44	63.56 ± 7.33	87.11 ± 0.61
MLS	66.59 ± 4.44	86.31 ± 0.59	56.06 ± 4.82	88.72 ± 0.36	61.32 ± 4.62	87.52 ± 0.47
KLM	90.55 ± 5.83	77.89 ± 0.75	85.18 ± 7.60	80.49 ± 0.85	87.86 ± 6.37	79.19 ± 0.80
VIM	49.19 ± 3.15	87.75 ± 0.28	40.49 ± 1.55	89.62 ± 0.33	44.84 ± 2.31	88.68 ± 0.28
KNN	37.64 ± 0.31	89.73 ± 0.14	30.37 ± 0.65	91.56 ± 0.26	34.01 ± 0.38	90.64 ± 0.20
DICE	73.71 ± 7.67	77.01 ± 0.88	66.37 ± 7.68	79.67 ± 0.87	70.04 ± 7.64	78.34 ± 0.79
RankFeat	65.32 ± 3.48	77.98 ± 2.24	56.44 ± 5.76	80.94 ± 2.80	60.88 ± 4.60	79.46 ± 2.52
ASH	87.31 ± 2.06	74.11 ± 1.55	86.25 ± 1.58	76.44 ± 0.61	86.78 ± 1.82	75.27 ± 1.04
SHE	81.00 ± 3.42	80.31 ± 0.69	78.30 ± 3.52	82.76 ± 0.43	79.65 ± 3.47	81.54 ± 0.51
GEN	58.75 ± 3.97	87.21 ± 0.36	48.59 ± 2.34	89.20 ± 0.25	53.67 ± 3.14	88.20 ± 0.30
Weiper+KLD	37.42 ± 0.91	89.70 ± 0.27	30.70 ± 0.43	91.38 ± 0.43	34.06 ± 0.49	90.54 ± 0.29
NAC-UE	**35.06** ± 0.30	89.78 ± 0.31	**26.53** ± 0.21	91.98 ± 0.24	**30.80** ± 0.13	90.98 ± 0.25
CombOOD	–	**90.08** ± –	–	**92.18** ± –	–	**91.13** ± –
SCALE	–	–	–	–	–	84.84 ± –
fDBD	–	–	–	–	–	90.42 ± –
BBAS	38.22 ± 2.11	89.91 ± 0.29	30.96 ± 1.71	91.70 ± 0.41	34.59 ± 1.90	90.81 ± 0.35

A.2 Full CIFAR-100 results

Table A.3: CIFAR-100 FarOOD Benchmark results on the ResNet18 architecture.

Method	MNIST		SVHN		Textures		Places365		Average	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
OpenMax	53.82 ± 4.74	76.01 ± 1.39	53.20 ± 1.78	82.07 ± 1.53	56.12 ± 1.91	80.56 ± 0.09	54.85 ± 1.42	79.29 ± 0.40	54.50 ± 0.68	79.48 ± 0.41
MSP	57.23 ± 4.68	76.08 ± 1.86	59.07 ± 2.53	78.42 ± 0.89	61.88 ± 1.28	77.32 ± 0.71	56.62 ± 0.87	79.22 ± 0.29	58.70 ± 1.06	77.76 ± 0.44
TempScale	56.05 ± 4.61	77.27 ± 1.85	57.71 ± 2.68	79.79 ± 1.05	61.56 ± 1.43	78.11 ± 0.72	56.46 ± 0.94	79.80 ± 0.25	57.94 ± 1.14	78.74 ± 0.51
ODIN	45.94 ± 3.29	83.79 ± 1.31	67.41 ± 3.88	74.54 ± 0.76	62.37 ± 2.96	79.33 ± 1.08	59.71 ± 0.92	79.45 ± 0.26	58.86 ± 0.79	79.28 ± 0.21
MDS	71.72 ± 2.94	67.47 ± 0.81	67.21 ± 6.09	70.68 ± 6.40	70.49 ± 2.48	76.26 ± 0.69	79.61 ± 0.34	63.15 ± 0.49	72.26 ± 1.56	69.39 ± 1.39
MDSEns	**2.83** ± 0.86	**98.21** ± 0.78	82.57 ± 2.58	53.76 ± 1.63	84.94 ± 0.83	69.75 ± 1.14	96.61 ± 0.17	42.27 ± 0.73	66.74 ± 1.04	66.00 ± 0.69
RMDS	52.05 ± 6.28	79.74 ± 2.49	51.65 ± 3.68	84.89 ± 1.10	53.99 ± 1.06	83.65 ± 0.51	**53.57** ± 0.43	**83.40** ± 0.46	52.81 ± 0.63	82.92 ± 0.42
Gram	53.53 ± 7.45	80.71 ± 4.15	20.06 ± 1.96	95.55 ± 0.60	89.51 ± 2.54	70.79 ± 1.32	94.67 ± 0.60	46.38 ± 1.21	64.44 ± 2.37	73.36 ± 1.08
EBO	52.62 ± 3.83	79.18 ± 1.37	53.62 ± 3.14	82.03 ± 1.74	62.35 ± 2.06	78.35 ± 0.83	57.75 ± 0.86	79.52 ± 0.23	56.59 ± 1.38	79.77 ± 0.61
OpenGAN	63.09 ± 23.25	68.14 ± 18.78	70.35 ± 2.06	68.40 ± 2.15	74.77 ± 1.78	65.84 ± 3.43	73.75 ± 8.32	69.13 ± 7.08	70.49 ± 7.38	67.88 ± 7.16
GradNorm	86.97 ± 1.44	65.35 ± 1.12	69.90 ± 7.94	76.95 ± 4.73	92.51 ± 0.61	64.58 ± 0.13	85.32 ± 0.44	69.69 ± 0.17	83.68 ± 1.92	69.14 ± 1.05
ReAct	56.04 ± 5.66	78.37 ± 1.59	50.41 ± 2.02	83.01 ± 0.97	55.04 ± 0.82	80.15 ± 0.46	55.30 ± 0.41	80.03 ± 0.11	54.20 ± 1.56	80.39 ± 0.49
MLS	52.95 ± 3.82	78.91 ± 1.47	53.90 ± 3.04	81.65 ± 1.49	62.39 ± 2.13	78.39 ± 0.84	57.68 ± 0.91	79.75 ± 0.24	56.73 ± 1.33	79.67 ± 0.57
KLM	73.09 ± 6.67	74.15 ± 2.59	50.30 ± 7.04	79.34 ± 0.44	81.80 ± 5.80	75.77 ± 0.45	81.40 ± 1.58	75.70 ± 0.24	71.65 ± 2.01	76.24 ± 0.52
VIM	48.32 ± 1.07	81.89 ± 1.02	46.22 ± 5.46	83.14 ± 3.71	46.86 ± 2.29	85.91 ± 0.78	61.57 ± 0.77	75.85 ± 0.37	50.74 ± 1.00	81.70 ± 0.62
KNN	48.58 ± 4.67	82.36 ± 1.52	51.75 ± 3.12	84.15 ± 1.09	53.56 ± 2.32	83.66 ± 0.83	60.70 ± 1.03	79.43 ± 0.47	53.65 ± 0.28	82.40 ± 0.17
DICE	51.79 ± 3.67	79.86 ± 1.89	49.58 ± 3.32	84.22 ± 2.00	64.23 ± 1.65	77.63 ± 0.34	59.39 ± 1.25	78.33 ± 0.66	56.25 ± 0.60	80.01 ± 0.18
RankFeat	75.01 ± 5.83	63.03 ± 3.86	58.49 ± 2.30	72.14 ± 1.39	66.87 ± 3.80	69.40 ± 3.08	77.42 ± 1.96	63.82 ± 1.83	69.45 ± 1.01	67.10 ± 1.42
ASH	66.58 ± 3.88	77.23 ± 0.46	46.00 ± 2.67	85.60 ± 1.40	61.27 ± 2.74	80.72 ± 0.70	62.95 ± 0.99	78.76 ± 0.16	59.20 ± 2.46	80.58 ± 0.66
SHE	58.78 ± 2.70	76.76 ± 1.07	59.15 ± 7.61	80.97 ± 3.98	73.29 ± 3.22	73.64 ± 1.28	65.24 ± 0.98	76.30 ± 0.51	64.12 ± 2.70	76.92 ± 1.16
GEN	53.92 ± 5.71	78.29 ± 2.05	55.45 ± 2.76	81.41 ± 1.50	61.23 ± 1.40	78.74 ± 0.81	56.25 ± 1.01	80.28 ± 0.27	56.71 ± 1.59	79.68 ± 0.75
Weiper+KLD	55.51 ± 5.58	77.93 ± 2.09	59.80 ± 2.70	79.55 ± 0.97	59.63 ± 1.55	78.56 ± 0.79	56.90 ± 0.85	80.00 ± 0.24	57.96 ± 0.98	79.01 ± 0.54
NAC-UE	21.97 ± 6.62	93.15 ± 1.63	24.39 ± 4.66	92.40 ± 1.26	40.65 ± 1.94	89.32 ± 0.55	73.57 ± 1.86	73.05 ± 0.37	40.14 ± 1.86	86.98 ± 0.37
CombOOD	–	77.74 ± –	–	**97.69** ± –	–	90.19 ± –	–	77.85 ± –	–	85.86 ± –
SCALE	–	–	22.05 ± –	96.29 ± –	34.20 ± –	92.34 ± –	85.04 ± –	72.66 ± –	–	81.42 ± –
fDBD	–	–	–	–	–	–	–	–	–	79.85 ± –
BBAS	26.52 ± 18.51	94.44 ± 3.60	**17.56** ± 6.00	96.99 ± 0.99	**27.39** ± 1.47	**95.14** ± 0.34	68.70 ± 2.87	76.84 ± 0.31	**35.04** ± 6.27	**90.85** ± 1.23

Table A.4: CIFAR-100 NearOOD Benchmark results on the ResNet18 architecture.

Method	CIFAR-10		TIN		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	60.17 ± 0.97	74.38 ± 0.37	52.99 ± 0.51	78.44 ± 0.14	56.58 ± 0.73	76.41 ± 0.25
MSP	58.91 ± 0.93	78.47 ± 0.07	50.70 ± 0.34	82.07 ± 0.17	54.80 ± 0.33	80.27 ± 0.11
TempScale	58.72 ± 0.81	79.02 ± 0.06	50.26 ± 0.16	82.79 ± 0.09	54.49 ± 0.48	80.90 ± 0.07
ODIN	60.64 ± 0.56	78.18 ± 0.14	55.19 ± 0.57	81.63 ± 0.08	57.91 ± 0.51	79.90 ± 0.11
MDS	88.00 ± 0.49	55.87 ± 0.22	79.05 ± 1.22	61.50 ± 0.28	83.53 ± 0.60	58.69 ± 0.09
MDSEns	95.94 ± 0.16	43.85 ± 0.31	95.82 ± 0.12	48.78 ± 0.19	95.88 ± 0.04	46.31 ± 0.24
RMDS	61.37 ± 0.24	77.75 ± 0.19	<u>49.56</u> ± 0.90	82.55 ± 0.02	55.46 ± 0.41	80.15 ± 0.11
Gram	92.71 ± 0.64	49.41 ± 0.58	91.85 ± 0.86	53.91 ± 1.58	92.28 ± 0.29	51.66 ± 0.77
EBO	59.21 ± 0.75	79.05 ± 0.11	52.03 ± 0.50	82.76 ± 0.08	55.62 ± 0.61	80.91 ± 0.08
OpenGAN	78.83 ± 3.94	63.23 ± 2.44	74.21 ± 1.25	68.74 ± 2.29	76.52 ± 2.59	65.98 ± 1.26
GradNorm	84.30 ± 0.36	70.32 ± 0.20	86.85 ± 0.62	69.95 ± 0.79	85.58 ± 0.46	70.13 ± 0.47
ReAct	61.30 ± 0.43	78.65 ± 0.05	51.47 ± 0.47	82.88 ± 0.08	56.39 ± 0.34	80.77 ± 0.05
MLS	59.11 ± 0.64	79.21 ± 0.10	51.83 ± 0.70	82.90 ± 0.05	55.47 ± 0.66	81.05 ± 0.07
KLM	84.77 ± 2.95	73.91 ± 0.25	71.07 ± 0.59	79.22 ± 0.28	77.92 ± 1.31	76.56 ± 0.25
VIM	70.59 ± 0.43	72.21 ± 0.41	54.66 ± 0.42	77.76 ± 0.16	62.63 ± 0.27	74.98 ± 0.13
KNN	72.80 ± 0.44	77.02 ± 0.25	49.65 ± 0.37	<u>83.34</u> ± 0.16	61.22 ± 0.14	80.18 ± 0.15
DICE	60.98 ± 1.10	78.04 ± 0.32	54.93 ± 0.53	80.72 ± 0.30	57.95 ± 0.53	79.38 ± 0.23
RankFeat	82.78 ± 1.56	58.04 ± 2.36	78.40 ± 0.95	65.72 ± 0.22	80.59 ± 1.10	61.88 ± 1.28
ASH	68.06 ± 0.44	76.48 ± 0.30	63.35 ± 0.90	79.92 ± 0.20	65.71 ± 0.24	78.20 ± 0.15
SHE	60.41 ± 0.51	78.15 ± 0.03	57.74 ± 0.73	79.74 ± 0.36	59.07 ± 0.25	78.95 ± 0.18
GEN	**58.87** ± 0.69	**79.38** ± 0.04	49.98 ± 0.05	83.25 ± 0.13	<u>54.42</u> ± 0.33	<u>81.31</u> ± 0.08
Weiper+KLD	<u>59.90</u> ± 0.69	<u>79.20</u> ± 0.10	**48.78** ± 0.24	**83.54** ± 0.07	**54.34** ± 0.24	**81.37** ± 0.02
NAC-UE	–	–	–	–	–	75.90 ± –
CombOOD	–	75.39 ± –	–	82.14 ± –	–	78.76 ± –
SCALE	–	–	–	–	–	80.99 ± –
fDBD	–	–	–	–	–	81.26 ± –
BBAS	70.88 ± 1.79	72.72 ± 0.78	59.85 ± 0.33	79.32 ± 0.11	65.36 ± 0.95	76.02 ± 0.44

A.2.1 Full ImageNet-200 results:

Table A.5: ImageNet-200 FarOOD Benchmark results on the ResNet18 architecture.

Method	iNaturalist		Textures		OpenImage-O		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	24.53 ± 0.96	92.32 ± 0.32	36.80 ± 0.55	90.21 ± 0.07	38.03 ± 0.49	88.07 ± 0.14	33.12 ± 0.66	90.20 ± 0.17
MSP	26.48 ± 0.73	92.80 ± 0.25	44.58 ± 0.68	88.36 ± 0.13	35.23 ± 0.18	89.24 ± 0.02	35.43 ± 0.38	90.13 ± 0.09
TempScale	24.39 ± 0.79	93.39 ± 0.25	43.57 ± 0.77	89.24 ± 0.11	34.04 ± 0.31	89.84 ± 0.02	34.00 ± 0.37	90.82 ± 0.09
ODIN	22.39 ± 1.87	94.37 ± 0.41	42.99 ± 1.56	90.65 ± 0.20	37.30 ± 0.59	90.11 ± 0.15	34.23 ± 1.05	91.71 ± 0.19
MDS	58.53 ± 0.75	75.03 ± 0.76	58.16 ± 0.84	79.25 ± 0.33	68.29 ± 0.28	69.87 ± 0.14	61.66 ± 0.27	74.72 ± 0.26
MDSEns	83.37 ± 0.70	62.16 ± 0.73	72.27 ± 0.48	80.70 ± 0.48	87.26 ± 0.10	64.96 ± 0.51	80.96 ± 0.38	69.27 ± 0.57
RMDS	24.70 ± 0.90	90.64 ± 0.46	37.80 ± 1.32	86.77 ± 0.38	34.85 ± 0.31	86.77 ± 0.22	32.45 ± 0.79	88.06 ± 0.34
Gram	85.54 ± 0.40	65.30 ± 0.20	80.87 ± 1.20	80.53 ± 0.37	86.66 ± 1.27	67.72 ± 0.58	84.36 ± 0.78	71.19 ± 0.24
EBO	26.41 ± 2.29	92.55 ± 0.50	41.43 ± 1.85	90.79 ± 0.16	36.74 ± 1.14	89.23 ± 0.26	34.86 ± 1.30	90.86 ± 0.21
OpenGAN	60.13 ± 9.79	75.32 ± 3.32	66.00 ± 9.97	70.58 ± 4.66	66.34 ± 8.44	73.54 ± 4.48	64.16 ± 9.33	73.15 ± 4.07
GradNorm	61.31 ± 2.86	86.06 ± 1.90	66.88 ± 3.59	86.07 ± 0.36	71.16 ± 0.23	80.66 ± 1.09	66.45 ± 0.22	84.26 ± 0.87
ReAct	22.97 ± 2.25	93.65 ± 0.88	29.67 ± 1.35	92.86 ± 0.47	32.86 ± 0.74	90.40 ± 0.35	28.50 ± 0.95	92.31 ± 0.56
MLS	25.09 ± 2.04	93.12 ± 0.45	41.25 ± 1.86	90.60 ± 0.16	35.76 ± 0.74	89.62 ± 0.21	34.03 ± 1.21	91.11 ± 0.19
KLM	26.66 ± 1.61	91.80 ± 0.21	50.24 ± 1.26	86.13 ± 0.12	45.81 ± 0.59	87.66 ± 0.17	40.90 ± 1.08	88.53 ± 0.11
VIM	27.34 ± 0.38	90.96 ± 0.36	20.39 ± 0.17	94.61 ± 0.12	33.86 ± 0.63	88.20 ± 0.18	27.20 ± 0.30	91.26 ± 0.19
KNN	24.46 ± 1.06	93.99 ± 0.36	24.45 ± 0.29	95.29 ± 0.02	32.90 ± 1.12	90.19 ± 0.32	27.27 ± 0.75	93.16 ± 0.22
DICE	29.66 ± 2.62	91.81 ± 0.79	40.96 ± 1.87	91.53 ± 0.21	38.91 ± 1.16	89.06 ± 0.34	36.51 ± 1.18	90.80 ± 0.31
RankFeat	98.00 ± 0.80	33.08 ± 4.68	99.40 ± 0.68	29.10 ± 2.57	95.77 ± 0.85	52.48 ± 4.44	97.72 ± 0.75	38.22 ± 3.85
ASH	22.49 ± 2.24	**95.10** ± 0.47	25.65 ± 0.80	94.77 ± 0.19	33.72 ± 0.97	**91.82** ± 0.25	27.29 ± 1.12	93.90 ± 0.27
SHE	34.38 ± 3.48	91.43 ± 1.28	45.58 ± 2.42	90.51 ± 0.19	46.54 ± 1.34	87.49 ± 0.70	42.17 ± 1.24	89.81 ± 0.61
GEN	**22.03** ± 0.98	93.70 ± 0.18	42.01 ± 0.92	90.25 ± 0.10	**32.25** ± 0.31	90.13 ± 0.06	32.10 ± 0.59	91.36 ± 0.10
Weiper+KLD	–	–	–	–	–	–	–	–
NAC-UE	–	–	–	–	–	–	–	91.02 ± –
CombOOD	–	–	–	–	–	–	–	92.57 ± –
SCALE	–	–	–	–	–	–	–	**93.98** ± –
fDBD	–	–	–	–	–	–	–	93.45 ± –
BBAS	25.19 ± 1.02	93.38 ± 0.22	**8.90** ± 0.22	**98.22** ± 0.07	37.87 ± 1.52	89.42 ± 0.45	**23.99** ± 0.76	93.67 ± 0.20

Table A.6: ImageNet-200 NearOOD Benchmark results on the ResNet18 architecture.

Method	SSB-hard		NINCO		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	72.37 ± 0.11	77.53 ± 0.08	54.59 ± 0.54	83.01 ± 0.17	63.48 ± 0.25	80.27 ± 0.10
MSP	**66.00** ± 0.10	80.38 ± 0.03	43.65 ± 0.75	86.29 ± 0.11	54.82 ± 0.35	83.34 ± 0.06
TempScale	66.43 ± 0.26	**80.71** ± 0.02	43.21 ± 0.70	86.67 ± 0.08	54.82 ± 0.23	83.69 ± 0.04
ODIN	73.51 ± 0.38	77.19 ± 0.06	60.00 ± 0.80	83.34 ± 0.12	66.76 ± 0.26	80.27 ± 0.08
MDS	83.65 ± 0.47	58.38 ± 0.58	74.57 ± 0.15	65.48 ± 0.46	79.11 ± 0.31	61.93 ± 0.51
MDSEns	92.13 ± 0.05	50.46 ± 0.36	91.36 ± 0.16	58.18 ± 0.42	91.75 ± 0.10	54.32 ± 0.24
RMDS	65.91 ± 0.27	80.20 ± 0.23	**42.13** ± 1.04	84.94 ± 0.28	**54.02** ± 0.58	82.57 ± 0.25
Gram	85.68 ± 0.85	65.95 ± 1.08	87.13 ± 1.89	69.40 ± 1.07	86.40 ± 1.21	67.67 ± 1.07
EBO	69.77 ± 0.32	79.83 ± 0.02	50.70 ± 0.89	85.17 ± 0.11	60.24 ± 0.57	82.50 ± 0.05
OpenGAN	88.07 ± 2.23	55.08 ± 1.84	80.23 ± 5.71	64.49 ± 4.98	84.15 ± 3.85	59.79 ± 3.39
GradNorm	82.17 ± 0.62	72.12 ± 0.43	83.17 ± 0.21	73.39 ± 0.63	82.67 ± 0.30	72.75 ± 0.48
ReAct	71.51 ± 1.92	78.97 ± 1.33	53.47 ± 2.46	84.76 ± 0.64	62.49 ± 2.19	81.87 ± 0.98
MLS	69.64 ± 0.37	80.15 ± 0.01	49.87 ± 0.94	85.65 ± 0.09	59.76 ± 0.59	82.90 ± 0.04
KLM	78.19 ± 2.30	77.56 ± 0.18	62.33 ± 2.66	83.96 ± 0.12	70.26 ± 0.64	80.76 ± 0.08
VIM	71.28 ± 0.49	74.04 ± 0.31	47.10 ± 1.10	83.32 ± 0.19	59.19 ± 0.71	78.68 ± 0.24
KNN	73.71 ± 0.31	77.03 ± 0.23	46.64 ± 0.73	86.10 ± 0.12	60.18 ± 0.52	81.57 ± 0.17
DICE	70.84 ± 0.30	79.06 ± 0.05	52.91 ± 1.20	84.49 ± 0.24	61.88 ± 0.67	81.78 ± 0.14
RankFeat	90.79 ± 0.37	58.74 ± 0.94	93.32 ± 0.11	55.10 ± 2.52	92.06 ± 0.23	56.92 ± 1.59
ASH	72.14 ± 0.97	79.52 ± 0.37	57.63 ± 0.98	85.24 ± 0.08	64.89 ± 0.90	82.38 ± 0.19
SHE	72.64 ± 0.30	78.30 ± 0.20	60.96 ± 1.33	82.07 ± 0.33	66.80 ± 0.74	80.18 ± 0.25
GEN	66.79 ± 0.26	80.75 ± 0.03	43.61 ± 0.61	**86.60** ± 0.08	55.20 ± 0.20	83.68 ± 0.06
Weiper+KLD	–	–	–	–	–	–
NAC-UE	–	–	–	–	–	78.52 ± –
CombOOD	–	–	–	–	–	**95.74** ± –
SCALE	–	–	–	–	–	84.84 ± –
fDBD	–	–	–	–	–	84.27 ± –
BBAS	72.49 ± 0.89	77.21 ± 0.38	52.44 ± 2.08	84.79 ± 0.36	62.47 ± 1.44	81.00 ± 0.35

A.3 Full ImageNet-1k results

Table A.7: ImageNet-1k FarOOD Benchmark results on the ResNet50 architecture.

Method	iNaturalist		Textures		OpenImage-O		Average	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
OpenMax	25.29	92.05	40.26	88.10	37.39	87.62	34.31	89.26
MSP	43.34	88.41	60.87	82.43	50.13	84.86	51.45	85.23
TempScale	37.63	90.50	56.90	84.95	45.40	87.22	46.64	87.56
ODIN	35.98	91.17	49.24	89.00	46.67	88.23	43.96	89.47
MDS	73.81	63.67	42.79	89.80	72.15	69.27	62.92	74.25
MDSEns	84.23	61.82	73.31	79.94	90.77	60.80	82.77	67.52
RMDS	33.67	87.24	48.80	86.08	40.27	85.84	40.91	86.38
Gram	67.89	76.67	58.80	88.02	75.39	74.43	67.36	79.71
EBO	31.30	90.63	45.77	88.70	38.09	89.06	38.39	89.47
OpenGAN	–	–	–	–	–	–	–	–
GradNorm	32.03	93.89	43.27	92.05	68.46	84.82	47.92	90.25
ReAct	16.72	96.34	29.64	92.79	32.58	91.87	26.31	93.67
MLS	30.61	91.17	46.17	88.39	37.88	89.17	38.22	89.57
KLM	38.52	90.78	52.40	84.72	48.89	87.30	46.60	87.60
VIM	30.68	89.56	10.51	**97.97**	32.82	90.5	24.67	92.68
KNN	40.80	86.41	17.31	97.09	44.27	87.04	34.13	90.18
DICE	33.37	92.54	44.28	92.04	47.83	88.26	41.83	90.95
RankFeat	94.40	40.06	76.84	70.90	90.26	50.83	87.17	53.93
ASH	14.04	97.07	15.26	96.90	29.15	93.26	19.49	95.74
SHE	34.06	92.65	35.27	93.6	55.02	86.52	41.45	90.92
GEN	26.10	92.44	46.22	87.59	34.50	89.26	35.61	89.76
Weiper+KLD	13.59	97.49	22.17	96.18	30.49	92.94	22.08	95.54
NAC-UE	–	96.52	–	97.90	–	91.45	–	95.29
CombOOD	–	87.13	–	–	–	86.60	–	90.24
SCALE	**9.51**	**98.02**	11.90	97.63	**28.18**	**93.95**	**16.53**	**96.53**
fDBD	–	–	–	–	–	–	–	92.77
BBAS	24.17	94.18	**10.39**	97.76	41.38	89.17	25.31	93.70

Table A.8: ImageNet-1k NearOOD Benchmark results on the ResNet50 architecture.

Method	SSB-hard		NINCO		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	77.33	71.37	60.81	78.17	69.07	74.77
MSP	74.49	72.09	56.88	79.95	65.68	76.02
TempScale	73.9	72.87	55.1	81.41	64.5	77.14
ODIN	76.83	71.74	68.16	77.77	72.5	74.75
MDS	92.1	48.5	78.8	62.38	85.45	55.44
MDSEns	95.19	43.92	91.86	55.41	93.52	49.67
RMDS	77.88	71.77	52.2	82.22	65.04	76.99
Gram	89.39	57.39	83.87	66.01	86.63	61.7
EBO	76.54	72.08	60.58	79.7	68.56	75.89
OpenGAN	–	–	–	–	–	–
GradNorm	78.24	71.9	79.54	74.02	78.89	72.96
ReAct	77.55	73.03	55.82	81.73	66.69	77.38
MLS	76.2	72.51	59.44	80.41	67.82	76.46
KLM	84.71	71.38	60.36	81.9	72.54	76.64
VIM	80.41	65.54	62.29	78.63	71.35	72.08
KNN	83.36	62.57	58.39	79.64	70.87	71.1
DICE	77.96	70.13	66.9	76.01	72.43	73.07
RankFeat	89.63	55.89	94.03	46.08	91.83	50.99
ASH	**73.66**	72.89	52.97	83.45	63.32	78.17
SHE	76.3	71.08	69.72	76.49	73.01	73.78
GEN	75.73	72.01	54.9	81.7	65.32	76.85
Weiper+KLD	74.12	**74.73**	**48.67**	**85.37**	61.39	80.05
NAC-UE	–	–	–	–	–	74.43
CombOOD	–	–	–	–	–	**95.22**
SCALE	–	–	–	–	**59.76**	81.36
fDBD	–	–	–	–	–	76.63
BBAS	79.23	71.51	57.6	81.93	68.41	76.72

Table A.9: ImageNet-1k FarOOD benchmark results on the ViT-B/16 architecture.

Method	iNaturalist		Textures		OpenImage-O		Average	
	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓	AUROC ↑
OpenMax	19.62	94.93	73.07	85.52	73.82	87.36	55.5	89.27
MSP	42.4	88.19	56.46	85.06	56.19	84.86	51.69	86.04
TempScale	43.09	88.54	58.16	85.39	59.98	85.04	53.74	86.32
ODIN	–	–	–	–	–	–	–	–
MDS	20.64	96.01	38.91	89.41	30.35	**92.38**	29.97	**92.6**
MDSEns	–	–	–	–	–	–	–	–
RMDS	19.47	96.1	37.22	89.38	29.57	92.32	28.76	**92.6**
Gram	–	–	–	–	–	–	–	–
EBO	83.56	79.3	83.66	81.17	88.82	76.48	85.35	78.98
OpenGAN	–	–	–	–	–	–	–	–
GradNorm	91.16	42.42	92.25	44.99	94.53	37.82	92.65	41.75
ReAct	48.25	86.11	55.88	86.66	57.67	84.29	53.93	85.69
MLS	72.94	85.29	78.94	83.74	85.82	81.6	79.23	83.54
KLM	43.48	89.59	50.12	86.49	51.75	87.03	48.45	87.7
VIM	**17.59**	95.72	40.35	90.61	29.61	92.18	29.18	92.84
KNN	27.75	91.46	33.23	91.12	34.82	89.86	31.93	90.81
DICE	47.9	82.5	54.83	82.21	52.57	82.22	51.77	82.31
RankFeat	–	–	–	–	–	–	–	–
ASH	97.02	50.62	98.5	48.53	94.79	55.51	96.77	51.56
SHE	22.16	93.57	**25.63**	**92.65**	33.57	91.04	**27.12**	92.42
GEN	22.92	93.54	38.3	90.23	35.47	90.27	32.23	91.35
Weiper+KLD	29.32	92.09	46.10	89.36	39.05	89.51	38.16	90.32
NAC-UE	–	–	–	–	–	–	–	–
CombOOD	–	–	–	–	–	–	–	–
SCALE	–	–	–	–	–	–	–	–
fDBD	–	–	–	–	–	–	–	–
BBAS	17.99	**96.44**	42.06	86.87	**29.12**	91.95	29.72	91.76

Table A.10: ImageNet-1k NearOOD benchmark results on the ViT-B/16 architecture.

Method	SSB-hard		NINCO		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	89.19	68.6	88.33	78.68	88.76	73.64
MSP	86.41	68.94	77.28	78.11	81.85	73.52
TempScale	87.35	68.55	81.88	77.8	84.62	73.18
ODIN	–	–	–	–	–	–
MDS	83.47	71.57	<u>48.77</u>	86.52	<u>66.12</u>	79.04
MDSEns	–	–	–	–	–	–
RMDS	84.52	**72.87**	**46.2**	**87.31**	**65.36**	<u>80.09</u>
Gram	–	–	–	–	–	–
EBO	92.24	58.8	94.14	66.02	93.19	62.41
OpenGAN	–	–	–	–	–	–
GradNorm	93.62	42.96	95.81	35.6	94.71	39.28
ReAct	90.46	63.1	78.51	75.43	84.49	69.26
MLS	91.52	64.2	92.97	72.4	92.25	68.3
KLM	88.35	68.14	66.14	80.68	77.25	74.41
VIM	90.04	69.42	57.41	84.64	73.73	77.03
KNN	86.22	65.98	54.73	82.25	70.47	74.11
DICE	89.77	59.05	81.1	71.67	85.44	65.36
RankFeat	–	–	–	–	–	–
ASH	93.5	53.9	95.37	52.51	94.43	53.21
SHE	85.73	68.04	56.02	84.18	70.88	76.11
GEN	<u>82.23</u>	70.09	59.33	82.51	70.78	76.3
Weiper+KLD	85.60	68.26	60.45	81.73	73.02	75.00
NAC-UE	–	–	–	–	–	–
CombOOD	–	–	–	–	–	–
SCALE	–	–	–	–	–	–
fDBD	–	–	–	–	–	–
BBAS	**80.92**	**75.34**	51.94	<u>86.60</u>	66.43	**80.97**

Table A.11: ImageNet-1k FarOOD benchmark results on the Swin-T architecture.

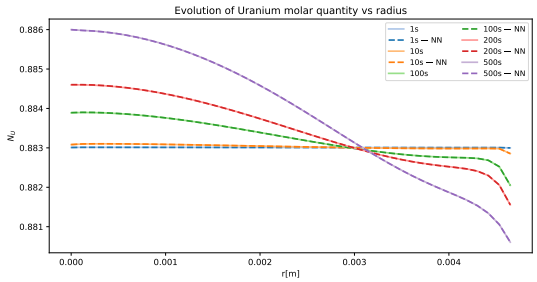
Method	iNaturalist		Textures		OpenImage-O		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	19.28	95.08	77.31	82.75	60.76	87.84	52.45	88.56
MSP	37.31	89.86	60.87	83.27	49.3	85.77	49.16	86.3
TempScale	36.66	90.37	62.55	83.68	50.93	86.14	50.05	86.73
ODIN	79.35	78.1	85.46	72.86	89.82	68.93	84.88	73.29
MDS	21.56	93.59	37.49	89.81	30.56	91.05	29.87	91.49
MDSEns	–	–	–	–	–	–	–	–
RMDS	18.44	95.58	39.18	89.26	28.91	92.17	28.84	<u>92.34</u>
Gram	–	–	–	–	–	–	–	–
EBO	61.52	85.13	84.45	78.96	80.81	79.86	75.59	81.32
OpenGAN	–	–	–	–	–	–	–	–
GradNorm	95.12	38.23	97.73	34.71	96.99	33.47	96.61	35.47
ReAct	31.55	90.03	54.66	86.99	42.81	87.66	43.01	88.23
MLS	49.19	88.99	79.8	81.68	74.45	83.72	67.81	84.8
KLM	34.96	90.92	49.12	86.04	45.91	87.99	43.33	88.32
VIM	<u>17.98</u>	<u>94.61</u>	**29.78**	**92.67**	**26.55**	<u>92.32</u>	**24.77**	**93.2**
KNN	30.94	88.93	<u>35.71</u>	<u>90.54</u>	35.71	88.63	34.12	89.37
DICE	98.23	46.82	88.3	77.03	96.73	57.47	94.42	60.44
RankFeat	–	–	–	–	–	–	–	–
ASH	94.25	46.62	96.16	41.66	93.74	45.64	94.71	44.64
SHE	33.42	92.72	50.98	88.86	53.89	87.93	46.1	89.84
GEN	20.51	94.23	44.57	88.15	32.69	90.56	32.59	90.98
Weiper+KLD	–	–	–	–	–	–	–	–
NAC-UE	–	–	–	–	–	–	–	–
CombOOD	–	–	–	–	–	–	–	–
SCALE	–	–	–	–	–	–	–	–
fDBD	–	–	–	–	–	–	–	–
BBAS	**13.81**	**96.77**	42.55	87.36	<u>27.92</u>	**92.36**	<u>28.10</u>	92.17

Table A.12: ImageNet-1k NearOOD benchmark results on the Swin-T architecture.

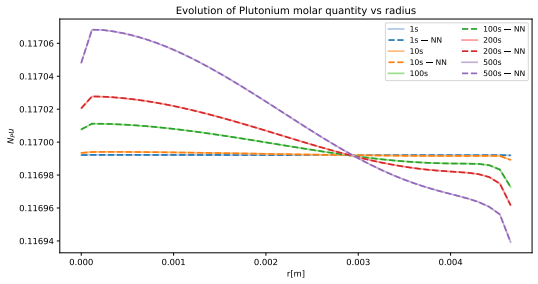
Method	SSB-hard		NINCO		Average	
	FPR95 ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑	**FPR95** ↓	**AUROC** ↑
OpenMax	85.5	71.56	72.58	81.66	79.04	76.61
MSP	80.9	71.78	61.21	81.72	71.06	76.75
TempScale	82.25	71.86	63.24	82.1	72.75	76.98
ODIN	86.45	66.94	86.05	69.35	86.25	68.15
MDS	83.6	68.61	53.71	81.75	68.65	75.18
MDSEns	–	–	–	–	–	–
RMDS	82.77	71.77	50.13	84.92	66.45	78.34
Gram	–	–	–	–	–	–
EBO	87.45	68.21	79.16	78.25	83.31	73.23
OpenGAN	–	–	–	–	–	–
GradNorm	93.57	50.36	94.06	44.81	93.82	47.58
ReAct	85.02	69.3	60.29	81.97	72.66	75.64
MLS	86.52	70.44	75	80.89	80.76	75.66
KLM	85.32	70.24	65.6	81.98	75.46	76.11
VIM	88.57	68.79	61.16	81.69	74.86	75.24
KNN	85	64.08	58.51	79.15	71.76	71.62
DICE	96.35	47.99	97.44	48.97	96.89	48.48
RankFeat	–	–	–	–	–	–
ASH	95.7	45.87	94.38	47.07	95.04	46.47
SHE	85.37	70.54	68.12	82.64	76.75	76.59
GEN	<u>79.19</u>	<u>72.75</u>	<u>48.05</u>	<u>85.18</u>	<u>63.62</u>	<u>78.97</u>
Weiper+KLD	–	–	–	–	–	–
NAC-UE	–	–	–	–	–	–
CombOOD	–	–	–	–	–	–
SCALE	–	–	–	–	–	–
fDBD	–	–	–	–	–	–
BBAS	**78.19**	**75.70**	**47.44**	**85.33**	**62.81**	**80.51**

Appendix B

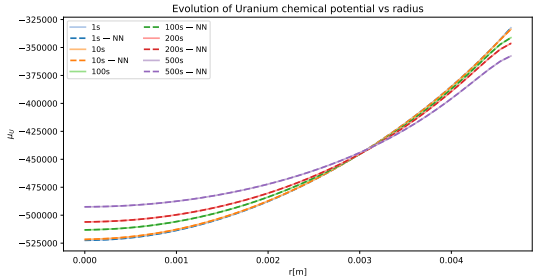
Extra results for Chapter 4



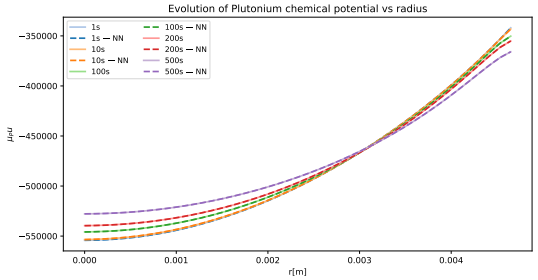
(a) Uranium molar quantity (N_U)



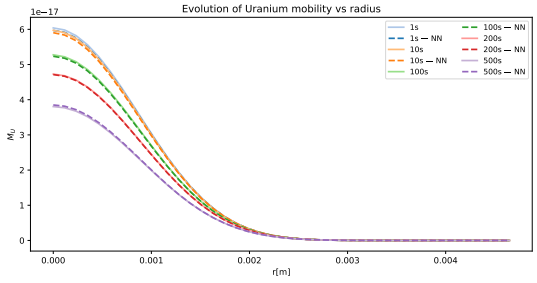
(b) Plutonium molar quantity (N_{Pu})



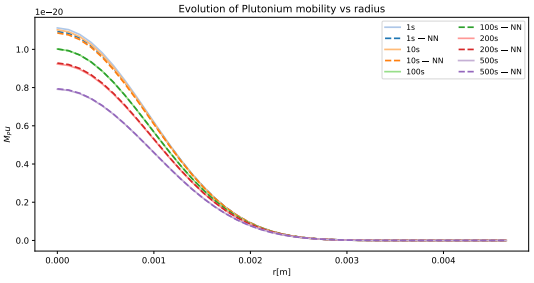
(c) Uranium chemical potential (μ_U)



(d) Plutonium chemical potential (μ_{Pu})



(e) Uranium mobility (M_U)



(f) Plutonium mobility (M_{Pu})

Figure B.1: Neural network and reference simulation for multiple variables evolution vs pellet radius ($r[m]$).

Bibliography

Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., Makarenkov, V., and Nahavandi, S. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297.

Abulawi, Z., Hu, R., Balaprakash, P., and Liu, Y. (2025). Bayesian optimized deep ensemble for uncertainty quantification of deep neural networks: a system safety case study on sodium fast reactor thermal stratification modeling. *Reliability Engineering & System Safety*, page 111353.

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. (2018). Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*.

Ammar, M. B., Belkhir, N., Popescu, S., Manzanera, A., and Franchi, G. (2023). Neco: Neural collapse based out-of-distribution detection. *arXiv preprint arXiv:2310.06823*.

Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, 50(1–2):5–43.

Antonante, P., Nilsen, H. G., and Carlone, L. (2023). Monitoring of perception systems: Deterministic, probabilistic, and learning-based fault detection and identification. *Artificial Intelligence*, 325:103998.

Antonello, F., Buongiorno, J., and Zio, E. (2023). Physics informed neural networks for surrogate modeling of accidental scenarios in nuclear power plants. *Nuclear Engineering and Technology*, 55(9):3409–3416.

Arbel, J., Pitas, K., Vladimirova, M., and Fortuin, V. (2024). A primer on Bayesian neural networks: review and debates. *Statistical Science*.

Attig, N., Gibbon, P., and Lippert, T. (2011). Trends in supercomputing: The european path to exascale. *Computer physics communications*, 182(9):2041–2046.

Bendale, A. and Boult, T. E. (2015). Towards open set deep networks. *CoRR*, abs/1511.06233.

Betancourt, D. and Muhanna, R. (2021). Interval deep learning for uncertainty quantification in safety applications. *CoRR*, abs/2105.06438.

Bitterwolf, J., Müller, M., and Hein, M. (2023). In or out? fixing imagenet out-of-distribution detection evaluation. In *Proceedings of the 40th International Conference on Machine Learning*, pages 2471–2506.

Chae, Y. H., Koo, S. R., Choi, J., and Kim, J. (2025). Enhanced learning for nuclear power plant condition diagnoses using information metric based on bayesian neural networks and umap. *Nuclear Engineering and Technology*, page 103886.

Chaouai, Z., Daniel, G., Martinez, J.-M., Limousin, O., and Benoit-Lévy, A. (2022). Application of adversarial learning for identification of radionuclides in gamma-ray spectra. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 1033:166670.

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , and Vedaldi, A. (2014). Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Cousot, P. and Cousot, R. (1976). Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*.

Damianou, A. and Lawrence, N. (2013). Deep Gaussian processes. In *AISTATS*.

De Figueiredo, L. H. and Stolfi, J. (2004). Affine arithmetic: concepts and applications. *Numerical algorithms*, 37(1):147–158.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*.

Denouden, T., Salay, R., Czarnecki, K., Abdelzad, V., Phan, B., and Vernekar, S. (2018). Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *arXiv preprint arXiv:1812.02765*.

Djurisic, A., Bozanic, N., Ashok, A., and Liu, R. (2023). Extremely simple activation shaping for out-of-distribution detection. In *The Eleventh International Conference on Learning Representations*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*.

Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A. M., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R., and Zhu, X. X. (2021). A survey of uncertainty in deep neural networks. *CoRR*, abs/2107.03342.

Granz, M., Heurich, M., and Landgraf, T. (2024). Weiper: Ood detection using weight perturbations of class projections. In *Advances in Neural Information Processing Systems*.

Graves, A. (2011). Practical variational inference for neural networks. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Gruver, N., Finzi, M., Stanton, S., and Wilson, A. G. (2022). Deconstructing the inductive biases of hamiltonian neural networks. *arXiv preprint arXiv:2202.04836*.

Guérin, J., Delmas, K., Ferreira, R., and Guiochet, J. (2023). Out-of-distribution detection is not all you need. 37(12):14829–14837.

Guillet, J.-L. and Guérin, Y., editors (2009). *Nuclear Fuels*. DEN Monographs — A Nuclear Energy Division Monograph. CEA Saclay and Groupe Moniteur (Éditions du Moniteur), Paris.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International Conference on Machine Learning*.

Hanin, B. and Rolnick, D. (2019a). Complexity of linear regions in deep networks. In *Proceedings of the 36th International Conference on Machine Learning*.

Hanin, B. and Rolnick, D. (2019b). Deep relu networks have surprisingly few activation patterns. In *Advances in Neural Information Processing Systems*.

Hashemi, V., Křetínskỳ, J., Rieder, S., Schön, T., and Vorhoff, J. (2024). Gaussian-based and outside-the-box runtime monitoring join forces. In *International Conference on Runtime Verification*, pages 218–228. Springer.

Hassija, V., Chamola, V., Mahapatra, A., Singal, A., Goel, D., Huang, K., Scardapane, S., Spinelli, I., Mahmud, M., and Hussain, A. (2024). Interpreting black-box models: a review on explainable artificial intelligence. *Cognitive Computation*, 16(1):45–74.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Hendrycks, D., Basart, S., Mazeika, M., Zou, A., Kwon, J., Mostajabi, M., Steinhardt, J., and Song, D. (2022). Scaling out-of-distribution detection for real-world settings. In *International Conference on Machine Learning*.

Hendrycks, D. and Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*. OpenReview.net.

Henzinger, T. A., Lukina, A., and Schilling, C. (2020). Outside the box: Abstraction-based monitoring of neural networks. *ECAI*.

Henzinger, T. A., Mazzocchi, N., and Saraç, N. E. (2023). Quantitative safety and liveness. In *Foundations of Software Science and Computation Structures*.

Hoemmen, M. (2010). *Communication-avoiding Krylov subspace methods*. University of California, Berkeley.

Horn, G. V., Aodha, O. M., Song, Y., Shepard, A., Adam, H., Perona, P., and Belongie, S. J. (2017). The inaturalist challenge 2017 dataset. *CoRR*, abs/1707.06642.

Huang, R., Geng, A., and Li, Y. (2021). On the importance of gradients for detecting distributional shifts in the wild. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 677–689. Curran Associates, Inc.

Hüllermeier, E. and Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110(3):457–506.

Introïni, C., Ramière, I., Sercombe, J., Michel, B., Helfer, T., and Fauque, J. (2024). Alcyone: the fuel performance code of the pleiades platform dedicated to pwr fuel rods behavior. *Annals of Nuclear Energy*, 207:110711.

Introïni, C., Sercombe, J., Ramière, I., and Le Tellier, R. (2021). Phase-field modeling with the taf-id of incipient melting and oxygen transport in nuclear fuel during power transients. *Journal of Nuclear Materials*, 556:153173.

Introïni, C., Sercombe, J., and Sundman, B. (2020). Development of a robust, accurate and efficient coupling between pleiades/alcyone 2.1 fuel performance code and the opencalphad thermo-chemical solver. *Nuclear Engineering and Design*, 369:110818.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Jomard, G., Struzik, C., Boulore, A., Mailhe, P., Auret, V., and Largenton, R. (2014). Caracas. an industrial model for the description of fission gas behavior in $lwr - uo_2$ fuel. *Proceedings of 2014 water reactor fuel performance meeting/ top fuel / LWR fuel performance meeting (WRFPM 2014)*.

Jospin, L. V., Buntine, W. L., Boussaïd, F., Laga, H., and Bennamoun, M. (2020). Hands-on bayesian neural networks - a tutorial for deep learning users. *CoRR*, abs/2007.06823.

Kadeethum, T., O'malley, D., Ballarin, F., Ang, I., Fuhg, J. N., Bouklas, N., Silva, V. L., Salinas, P., Heaney, C. E., Pain, C. C., et al. (2022). Enhancing high-fidelity nonlinear solver with reduced order model. *Scientific reports*, 12(1):20229.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440.

Kato, Y., Tax, D. M., and Loog, M. (2022). A view on model misspecification in uncertainty quantification. In *Benelux Conference on Artificial Intelligence*, pages 65–77. Springer.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in Bayesian deep learning for computer vision? *Advances in Neural Information Processing Systems*, 30.

Keyes, D. E., McInnes, L. C., Woodward, C., Gropp, W., Myra, E., Pernice, M., Bell, J., Brown, J., Clo, A., Connors, J., et al. (2013). Multiphysics simulations: Challenges and opportunities. *The International Journal of High Performance Computing Applications*, 27(1):4–83.

Kong, S. and Ramanan, D. (2021). Opengan: Open-set recognition via open data generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 813–822.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560.

Krizhevsky, A., Nair, V., and Hinton, G. (2009). CIFAR-10 and CIFAR-100 datasets. 6(1):1.

Krukowski, P., Wilczak, D., Tabor, J., Bielawska, A., and Spurek, P. (2024). Make interval bound propagation great again.

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J. R. R., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T., and Ferrari, V. (2018). The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *CoRR*, abs/1811.00982.

Lahlou, S., Jain, M., Nekoei, H., Butoi, V. I., Bertin, P., Rector-Brooks, J., Korablyov, M., and Bengio, Y. (2021). Deup: Direct epistemic uncertainty prediction. *Transactions on Machine Learning Research*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Laurent, O., Lafage, A., Tartaglione, E., Daniel, G., marc Martinez, J., Bursuc, A., and Franchi, G. (2023). Packed ensembles for efficient uncertainty estimation. In *The Eleventh International Conference on Learning Representations*.

Le, Y. and Yang, X. (2015). Tiny imagenet visual recognition challenge. *Vision Stanford*.

Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in Neural Information Processing Systems*.

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R. J., and Wasserman, L. (2018). Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111.

Lewandowski, M., Eghbalzadeh, H., Heinzl, B., Pisoni, R., and Moser, B. A. (2025). On space folds of relu neural networks.

Liang, S., Li, Y., and Srikant, R. (2018). Enhancing the reliability of out-of-distribution image detection in neural networks. *International Conference on Learning Representation*.

Liu, L. and Qin, Y. (2024). Fast decision boundary based out-of-distribution detector. In *International Conference on Machine Learning*.

Liu, W., Wang, X., Owens, J., and Li, Y. (2020). Energy-based out-of-distribution detection. *Advances in Neural Information Processing Systems*.

Liu, X., Lochman, Y., and Christopher, Z. (2023). Gen: Pushing the limits of softmax-based out-of-distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Liu, Y., TIAN, C. X., Li, H., Ma, L., and Wang, S. (2024). Neuron activation coverage: Rethinking out-of-distribution detection and generalization. In *The Twelfth International Conference on Learning Representations*.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

MacKay, D. J. C. (1992). A practical bayesian framework for backpropagation networks. *Neural Comput.*, 4(3):448–472.

Malinin, A. (2019). *Uncertainty Estimation in Deep Learning with application to Spoken Language Assessment*. PhD thesis, Christ's College.

Marcel, S. and Rodriguez, Y. (2010). Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488. Association for Computing Machinery.

Marrel, A., Marie, N., and De Lozzo, M. (2015). Advanced surrogate model and sensitivity analysis methods for sodium fast reactor accident assessment. *Reliability Engineering & System Safety*, 138:232–241.

Marzouk, Y. M. and Najm, H. N. (2009). Dimensionality reduction and polynomial chaos acceleration of bayesian inference in inverse problems. *Journal of Computational Physics*, 228(6):1862–1902.

Mattos, C. L. C., Dai, Z., Damianou, A., Forth, J., Barreto, G. A., and Lawrence, N. D. (2016). Recurrent gaussian processes.

Michel, B., Ramière, I., Viallard, I., Introini, C., Lainet, M., Chauvin, N., Marelle, V., Boulore, A., Helfer, T., Masson, R., et al. (2021). Two fuel performance codes of the pleiades platform: Alcyone and germinal. In *Nuclear Power Plant Design and Analysis Codes*, pages 207–233. Elsevier.

Mondal, A. and Sharma, S. L. (2025). Bayesian neural network for critical heat flux prediction: a unified approach with uncertainty quantification. *Applied Thermal Engineering*, page 127418.

Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Nguyen, H. V., Gamboa, F., Chhaibi, R., Zhang, S., Gratton, S., and Giaccone, T. (2024). Combining statistical depth and fermat distance for uncertainty quantification. *Advances in Neural Information Processing Systems*, 37:30207–30229.

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: an empirical study.

Oala, L., Heiß, C., Macdonald, J., März, M., Kutyniok, G., and Samek, W. (2021). Detecting failure modes in image reconstructions with interval neural network uncertainty. *Int. J. Comput. Assist. Radiol. Surg.*, 16(12):2089–2097.

Olivier, A., Shields, M. D., and Graham-Brady, L. (2021). Bayesian neural networks for uncertainty quantification in data-driven materials modeling. *Computer methods in applied mechanics and engineering*, 386:114079.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in Neural Information Processing Systems*, 32.

Papamarkou, T., Skoularidou, M., Palla, K., Aitchison, L., Arbel, J., Dunson, D., Filippone, M., Fortuin, V., Hennig, P., Hernández-Lobato, J. M., Hubin, A., Immer, A., Karaletsos, T., Khan, M. E., Kristiadi, A., Li, Y., Mandt, S., Nemeth, C., Osborne, M. A., Rudner, T. G. J., Rügamer, D., Teh, Y. W., Welling, M., Wilson, A. G., and Zhang, R. (2024). Position: Bayesian deep learning is needed in the age of large-scale AI. *International Conference on Machine Learning*.

Papernot, N. and McDaniel, P. (2018). Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning.

Parisot, J.-F., editor (2015). *Neutronics*. DEN Monographs — A Nuclear Energy Division Monograph. CEA Paris-Saclay and Groupe Moniteur (Éditions du Moniteur), Paris. ISSN 1950-2672.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

Paul, M. C., Mamun Molla, M., and Roditi, G. (2009). Large–eddy simulation of pulsatile blood flow. *Medical Engineering & Physics*, 31(1):153–159.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Postels, J., Segu, M., Sun, T., Sieber, L., Gool, L. V., Yu, F., and Tombari, F. (2022). On the practicality of deterministic epistemic uncertainty.

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2022). *Dataset shift in machine learning*. Mit Press.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. (2017). On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*.

Rahman, Q. M., Corke, P., and Dayoub, F. (2021). Run-time monitoring of machine learning for robotic perception: A survey of emerging trends. *IEEE Access*, 9:20067–20075.

Rajasekaran, M., Sajol, M. S. I., Berglind, F., Mukhopadhyay, S., and Das, K. (2024). *COMBOOD: A Semiparametric Approach for Detecting Out-of-distribution Data for Image Classification*.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA.

Ravaji, B. and Joshi, S. P. (2021). A crystal plasticity investigation of grain size-texture interaction in magnesium alloys. *Acta Materialia*, 208:116743.

Regazzoni, F., Pagani, S., Salvador, M., Dede', L., and Quarteroni, A. (2024). Learning the intrinsic dynamics of spatio-temporal processes through latent dynamics networks. *Nature Communications*, 15(1):1834.

Ren, J., Fort, S., Liu, J. Z., Roy, A. G., Padhy, S., and Lakshminarayanan, B. (2021a). A simple fix to mahalanobis distance for improving near-ood detection. *CoRR*, abs/2106.09022.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Gupta, B. B., Chen, X., and Wang, X. (2021b). A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40.

Sastry, C. S. and Oore, S. (2020). Detecting out-of-distribution examples with Gram matrices. In *International Conference on Machine Learning*.

Schumann, J., Moosbrugger, P., and Rozier, K. Y. (2015). R2u2: monitoring and diagnosis of security threats for unmanned aerial systems. In *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*.

Semenova, E. (2025). Case for a unified surrogate modelling framework in the age of ai. *arXiv e-prints*, pages arXiv–2502.

Sercombe, J., Masson, R., and Helfer, T. (2013). Stress concentration during pellet cladding interaction: Comparison of closed-form solutions with 2d(r,θ) finite element simulations. *Nuclear Engineering and Design*, 260:175–187.

Serra, T., Tjandraatmadja, C., and Ramalingam, S. (2017). Bounding and counting linear regions of deep neural networks. *CoRR*, abs/1711.02114.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press.

Sharma, H. and Jennings, E. (2021). Bayesian neural networks at scale: a performance analysis and pruning study. *The Journal of Supercomputing*, 77(4):3811–3839.

Sobes, V., Hiscox, B., Popov, E., Archibald, R., Hauck, C., Betzler, B., and Terrani, K. (2021). Ai-based design of a nuclear reactor core. *Scientific Reports*, 11:19646.

Song, Y., Sebe, N., and Wang, W. (2022). Rankfeat: Rank-1 feature removal for out-of-distribution detection. In *Advances in Neural Information Processing Systems*.

Sturm, P. O. and Wexler, A. S. (2022). Conservation laws in a neural network architecture: enforcing the atom balance of a julia-based photochemical model (v0. 2.0). *Geoscientific Model Development*, 15(8):3417–3431.

Suarez, E., Bockelmann, H., Eicker, N., Eitzinger, J., El Sayed, S., Fieseler, T., Frank, M., Frech, P., Giesselmann, P., Hackenberg, D., et al. (2025). Energy-aware operation of hpc systems in germany. *Frontiers in High Performance Computing*, 3:1520207.

Sun, Y., Guo, C., and Li, Y. (2021). React: Out-of-distribution detection with rectified activations. In *Advances in Neural Information Processing Systems*.

Sun, Y. and Li, Y. (2022). Dice: Leveraging sparsification for out-of-distribution detection. In *European Conference on Computer Vision*.

Sun, Y., Ming, Y., Zhu, X., and Li, Y. (2022). Out-of-distribution detection with deep nearest neighbors. In *International Conference on Machine Learning*.

Sundman, B., Kattner, U. R., Palumbo, M., and Fries, S. G. (2015). OpenCalphad - a free thermodynamic software. *Integr. Mater. Manuf. Innov.*, 4(1):1–15.

Tajwar, F., Kumar, A., Xie, S. M., and Liang, P. (2021). No true state-of-the-art? ood detection methods are inconsistent across datasets. *ArXiv*.

Telgarsky, M. (2015). Representation benefits of deep feedforward networks.

Vargas-Fajardo, J. A., Manvelyan-Stroot, D., Czech, C., Botazzoli, P., and Duddeck, F. (2025). Parametric model order reduction by box clustering with applications in mechatronic systems. *arXiv preprint arXiv:2505.11255*.

Vaze, S., Han, K., Vedaldi, A., and Zisserman, A. (2022). Open-set recognition: A good closed-set classifier is all you need. In *International Conference on Learning Representations*.

Vijayarangan, V., Uranakara, H. A., Barwey, S., Galassi, R. M., Malik, M. R., Valorani, M., Raman, V., and Im, H. G. (2024). A data-driven reduced-order model for stiff chemical kinetics using dynamics-informed training. *Energy and AI*, 15:100325.

von Rueden, L., Mayer, S., Sifa, R., Bauckhage, C., and Garcke, J. (2020). Combining machine learning and simulation to a hybrid modelling approach: Current and future directions. In Berthold, M. R., Feelders, A., and Krempl, G., editors, *Advances in Intelligent Data Analysis XVIII*.

Wang, H., Li, Z., Feng, L., and Zhang, W. (2022). Vim: Out-of-distribution with virtual-logit matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4921–4930.

Wang, Y. (2022). Estimation and comparison of linear regions for relu networks. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. International Joint Conferences on Artificial Intelligence Organization.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in Neural Information Processing Systems*, 28.

Wu, C., Falcone, Y., and Bensalem, S. (2023). Customizable reference runtime monitoring of neural networks using resolution boxes. In *International Conference on Runtime Verification*, pages 23–41. Springer.

Wu, C., He, W., Cheng, C.-H., Huang, X., and Bensalem, S. (2024). BAM: box abstraction monitors for real-time OoD detection in object detection. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2632–2638. IEEE.

Wu, T., Maruyama, T., and Leskovec, J. (2022). Learning to accelerate partial differential equations via latent global evolution. *Advances in Neural Information Processing Systems*, 35:2240–2253.

Xu, H., Ma, Y., Liu, H.-C., Deb, D., Liu, H., Tang, J.-L., and Jain, A. K. (2020). Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178.

Xu, J. and Zikatanov, L. (2017). Algebraic multigrid methods. *Acta Numerica*, 26:591–721.

Xu, K., Chen, R., Franchi, G., and Yao, A. (2024). Scaling for training time and post-hoc out-of-distribution detection enhancement. In *International Conference on Learning Representations*.

Yahiaoui, M., Daniel, G., Giraldi, L., Bruyelle, J., and Arbel, J. (2025a). Bounding box anomaly scoring for simple and efficient out-of-distribution detection.

Yahiaoui, M., Giraldi, L., Daniel, G., Introïni, C., and Arbel, J. (2025b). Reliable neural network model for accelerating coupled thermodiffusion simulation. In *International Congress on Advances in Nuclear Power Plants (ICAPP)*.

Yang, J., Wang, P., Zou, D., Zhou, Z., Ding, K., Peng, W., Wang, H., Chen, G., Li, B., Sun, Y., Du, X., Zhou, K., Zhang, W., Hendrycks, D., Li, Y., and Liu, Z. (2022). OpenOOD: Benchmarking generalized out-of-distribution detection. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Yang, J., Zhou, K., Li, Y., and Liu, Z. (2024). Generalized out-of-distribution detection: A survey. *International Journal of Computer Vision*, 132(12):5635–5662.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*.

Zhang, J., Fu, Q., Chen, X., Du, L., Li, Z., Wang, G., xiaoguang Liu, Han, S., and Zhang, D. (2023a). Out-of-distribution detection based on in-distribution data patterns memorization with modern hopfield energy. In *The Eleventh International Conference on Learning Representations*.

Zhang, J., Yang, J., Wang, P., Wang, H., Lin, Y., Zhang, H., Sun, Y., Du, X., Zhou, K., Zhang, W., Li, Y., Liu, Z., Chen, Y., and Hai, L. (2023b). OpenOOD v1.5: Enhanced Benchmark for Out-of-Distribution Detection. *arXiv preprint arXiv:2306.09301*.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2017). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhou, Y., Lu, Z., Hu, J., and Hu, Y. (2020). Surrogate modeling of high-dimensional problems via data-driven polynomial chaos expansions and sparse partial least square. *Computer Methods in Applied Mechanics and Engineering*, 364:112906.

Zidek, J. V. and van Eeden, C. (2003). Uncertainty, entropy, variance and the effect of partial information. *Lecture Notes-Monograph Series*, page 155–167.