UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AERONÁUTICA Y DEL ESPACIO

GRADO EN INGENIERÍA AEROESPACIAL

TRABAJO FIN DE GRADO

Uso de Graph Neural Networks para el desarrollo de modelos térmicos surrogados de sistemas espaciales

AUTOR: Ernesto ENFEDAQUE MEDINA

ESPECIALIDAD: Propulsión Aeroespacial

TUTOR DEL TRABAJO: David GONZÁLEZ BÁRCENA

Junio de 2025

Contents

List of Figures **i**

List of Tables **v**

Acronyms **vii**

1 Introduction **1**

2 State of the Art Review **4**

3 Theoretical Foundations of Thermal Control **9**
3.1 Thermal Control Subsystem 9
3.2 Heat Transfer 10
3.2.1 Conduction 10
3.2.2 Convection 11
3.2.3 Radiation 13
3.3 Space Thermal Environment in Earth Orbit 14
3.3.1 Direct Solar Radiation 15
3.3.2 Albedo 16
3.3.3 Planetary Radiation, OLR 16
3.4 Lumped Parameter Thermal Method (LPTM) and its Connection to Graph-Based Models 16
3.4.1 Principles of the Lumped Parameter Method 16
3.4.2 Historical Context and Mathematical Basis 17
3.4.3 Construction and Application in ESATAN-TMS 18
3.4.4 Relation to Graphs and Modern Machine Learning 18
3.4.5 Advantages and Use Cases 18

4 Theoretical Foundations of Deep Learning **20**
4.1 Fundamentals of ANNs 21
4.2 Characteristics and Training of ANNs 21
4.2.1 Architecture 21
4.2.2 Hidden Units and Activation Functions 22

4.2.3 Dataset . . . 24
4.2.4 Hyper-parameters . . . 24
4.2.5 Loss Functions . . . 25
4.2.6 Optimization functions . . . 26
4.3 Most common types of Neural Networks . . . 27
4.3.1 Multi-Layer Perceptron (MLP) . . . 27
4.3.2 Convolutional Neural Network (CNN) . . . 28
4.3.3 Recurrent Neural Network (RNN) . . . 30
4.3.4 Graph Neural Network (GNN) . . . 32
4.4 Challenges and Limitations of Artificial Neural Networks . . . 40
4.4.1 Delving into hyper-parameter sensitivity . . . 41

5 Objectives and Scope of the Thesis 43

6 Comparison of several GNN architectures solving a steady 2D PCB 45
6.1 Thermal Model PCB . . . 46
6.1.1 Model Simplifications . . . 47
6.1.2 Boundary Conditions . . . 48
6.1.3 Lumped Parameter Method . . . 50
6.1.4 Heat Flux Vector . . . 51
6.1.5 Conductivity Coupling Matrix . . . 51
6.1.6 Radiative Coupling Matrix: . . . 52
6.1.7 Equation Resolution . . . 53
6.2 Insights of some decisions taken . . . 53
6.3 Graph Convolutional Network . . . 56
6.3.1 Hyper-parameter optimization for GCN and results obtained . . . 56
6.4 Graph Attention Network (GAT) . . . 64
6.4.1 Hyper-parameter optimization for GAT and results obtained . . . 64
6.5 Graph SAmple and aggreGatE (Graph SAGE) . . . 70
6.5.1 Hyper-parameter optimization for SAGE and results obtained . . . 70
6.6 Neural Network Convolution (NNConv) . . . 74
6.6.1 Hyper-parameter optimization for NNConv and results obtained . . . 74
6.7 Comparison between the GNN architectures . . . 78

7 Description of the 3D ESATAN-TMS model to use in training 80
7.1 Generating syntethic data . . . 87
7.2 Results . . . 87
7.2.1 Graph Convolutional Network (GCN) . . . 88
7.2.2 Graph Attention Network (GAT) . . . 91
7.2.3 Graph SAmple and aggreGatE (Graph SAGE) . . . 96

7.2.4 Neural Network Convolution (NNConv) 102
7.3 Comparison between the four different architectures 106

8 Conclusions and Future Steps **110**
8.1 Conclusions 110
8.2 Future Work 111
References 113

List of Figures

List of Figures

3.1 Radiation in Earth orbit [1] 15

4.1 Diagram of a MLP [2]. 22
4.2 Graphical representation of common activation functions. 23
4.3 CNN architecture [3]. 29
4.4 RNN architecture [4]. 31
4.5 LSTM architecture [5]. 32
4.6 GNN architecture [6]. 33
4.7 GCN model representation. 36
4.8 GAT model representation. 37
4.9 SAGE model representation. 38
4.10 NNConv model representation. 39
4.11 Generation of edge matrixes on NNConv. 39

6.1 Temperature distribution of a PCB with random boundary conditions. 47
6.2 Location of the power dissipation in a regular 13×13 plate. 49
6.3 Example of interface temperatures located at the corners. 50
6.4 Evolution of the MSE of a GCN 2D model with the number of cases. 54
6.5 Evolution of the MSE of a GCN 2D model with the number of cases, truncated. . . . 54
6.6 Evolution of the MSE with the number of layers in GCN 2D architecture. 57
6.7 Evolution of the MSE with the number of layers in GCN 2D architecture, truncated. 57
6.8 Target and predicted temperatures by the GCN 2D. 59

6.9 Absolute error in Kelvin per node. GCN 2D only with MSE. 60
6.10 Evolution of the training and validation loss, all normalized. GCN 2D, only with MSE. 61
6.11 Absolute error in Kelvin per node. GCN 2D with the combined loss function. 63
6.12 Absolute error in Kelvin per node. GAT 2D with the combined loss function. 67
6.13 Absolute error in Kelvin per node. GAT 2D only with MSE. 69
6.14 Absolute error in Kelvin per node. SAGE 2D with the combined loss function. . . . 72
6.15 Absolute error in Kelvin per node. SAGE 2D only with MSE. 73
6.16 Absolute error in Kelvin per node. NNConv 2D with the combined loss function. . . 76
6.17 Absolute error in Kelvin per node. NNConv 2D only using the MSE. 77

7.1 Geometry of the satellite. 81
7.2 Geometry of the satellite, with the MLI and one wall hidden. 82
7.3 Elements that dissipate power in the satellite. 82
7.4 IR emissivity of the satellite. 83
7.5 Conductivity of the different materials used in the satellite. 83
7.6 Model representation as a graph, with external ID numeration. 85
7.7 Model representation as a graph, displaying nodes that dissipate power with external ID numeration. 86
7.8 Evolution of training loss and validation loss, all normalized. GCN 3D, only with MSE. 89
7.9 Absolute error in Kelvin per node. GCN 3D only with MSE. 90
7.10 Absolute error in Kelvin per node. GCN 3D with the combined loss function. 91
7.11 Evolution of training loss and validation loss, all normalized. GAT 3D asymmetrical, only with MSE. 92
7.12 Absolute error in Kelvin per node. GAT 3D asymmetrical, only with MSE. 93
7.13 Absolute error in Kelvin per node. GAT 3D symmetrical, only with MSE. 94
7.14 Absolute error in Kelvin per node. GAT 3D symmetrical, with the combined loss function. 96
7.15 Evolution of training loss and validation loss, all normalized. SAGE 3D asymmetrical, only with MSE. 97
7.16 Absolute error in Kelvin per node. SAGE 3D asymmetrical, with the MSE. 98
7.17 Evolution of training loss and validation loss, all normalized. SAGE 3D symmetrical, only with MSE. 99
7.18 Absolute error in Kelvin per node. SAGE 3D symmetrical, with the MSE. 100
7.19 Absolute error in Kelvin per node. SAGE 3D symmetrical, with the combined loss function. 101
7.20 Evolution of training loss and validation loss, all normalized. SAGE 3D symmetrical, with the combined loss function. 102
7.21 Absolute error in Kelvin per node. NNConv 3D asymmetrical, with the MSE. 104
7.22 Absolute error in Kelvin per node. NNConv 3D symmetrical, with the MSE. 105

7.23 Absolute error in Kelvin per node. NNConv 3D symmetrical, with the combined loss function. 106
7.24 Evolution of the validation and training loss of a NNConv 3D model with the number of cases. 108
7.25 Absolute error in Kelvin per node. NNConv 3D symmetrical reduced Dataset, with the MSE. 109

List of Tables

List of Tables

3.1 Temperature requirements of subsystems [7] 10

6.1 Hyper-parameters used for GCN 2D training. 59
6.2 Evaluation metrics and loss components of the GCN 2D trained model using the MSE as the loss function. 61
6.3 Evaluation metrics and loss components of the GCN 2D trained model with the combined loss function. 64
6.4 Hyper-parameters used for GAT 2D training 66
6.5 Evaluation metrics and loss components of the GAT trained model with the combined loss function. 68
6.6 Evaluation metrics and loss components of the GAT trained model using the MSE as the loss function. 69
6.7 Hyper-parameters used for SAGE 2D training. 70
6.8 Evaluation metrics and loss components of the SAGE trained model with the combined loss function. 71
6.9 Evaluation metrics and loss components of the SAGE trained model using the MSE as the loss function. 73
6.10 Hyper-parameters used for NNConv 2D training 75
6.11 Evaluation metrics and loss components of the NNConv trained model with the combined loss function. 75
6.12 Evaluation metrics and loss components of the NNConv trained model using the MSE as the loss function. 77

6.13 Comparison of hyper-parameters and performance across GNN architectures 78

7.1 Node and colour assignment for each instrument. 83
7.2 Component classification and associated ID ranges. 84
7.3 Hyper-parameters used for GCN 3D training. 88
7.4 Evaluation metrics and loss components of the GCN 3D trained model only using the MSE. 89
7.5 Evaluation metrics and loss components of the GCN 3D trained model with the combined loss function. 90
7.6 Hyper-parameters used for GAT 3D training. 91
7.7 Evaluation metrics and loss components of the trained GAT 3D model only with the MSE and the asymmetrical Dataset. 92
7.8 Evaluation metrics and loss components of the GAT 3D trained model only with the MSE and the symmetrical Dataset. 94
7.9 Evaluation metrics and loss components of the GAT 3D trained model with the combined loss function and the symmetrical Dataset. 95
7.10 Hyper-parameters used for SAGE 3D training. 96
7.11 Evaluation metrics and loss components of the SAGE 3D trained model only with the MSE and the asymmetrical Dataset. 97
7.12 Evaluation metrics and loss components of the SAGE 3D trained model only with the MSE and the symmetrical Dataset. 98
7.13 Evaluation metrics and loss components of the SAGE 3D trained model with the combined loss function and the symmetrical Dataset. 100
7.14 Hyper-parameters used for NNConv 3D training. 103
7.15 Evaluation metrics and loss components of the NNConv 3D trained model only using the MSE and the asymmetrical Dataset. 103
7.16 Evaluation metrics and loss components of the NNConv 3D trained model only with the MSE and the symmetrical Dataset. 104
7.17 Evaluation metrics and loss components of the NNConv 3D trained model with the combined loss function and the symmetrical Dataset 105
7.18 Comparison of hyper-parameters and performance across GNN architectures, tested on the 3D ESATAN-TMS model. 107
7.19 Evaluation metrics and loss components of the NNConv 3D trained model only using the MSE and the reduced symmetrical 464 cases Dataset. 108
7.20 Evaluation metrics and loss components of the NNConv 3D trained model only using the MSE and the reduced symmetrical 100 cases Dataset. 109

Acronyms

Adam Adaptive Moment Estimation 27, 28

AI Artificial Intelligence 20

ANN Artificial Neural Network , 1, 2, 4–8, 21, 24–27, 40, 41, 56, 57, 70

AU Astronomical Unit 15

CFD Computational Fluid Dynamics 2, 4, 43, 80

CNN Convolutional Neural Network i, , 5–7, 22, 28–30, 43, 45, 46, 54, 110

FC Fully Connected 22

FEM Finite Element Method 2, 4, 5, 7, 8, 43, 80

GAT Graph Attention Network , 34, 36–38, 64, 65, 71, 75, 78, 91, 96–98, 107, 109, 111

GCN Graph Convolutional Network i, , 18, 34–36, 38, 53, 56–62, 64–67, 70, 71, 74, 75, 78, 88, 92, 110

GD Gradient Descent 27

GNN Graph Neural Network i, , 2, 5–9, 18, 32–35, 43, 45, 46, 56, 58, 78, 80, 87, 110, 111

GNS Graph Network Simulator 6, 7, 34, 35

GPU Graphics Processing Unit 40, 46

Graph SAGE Graph SAmple and aggreGatE , 34, 37, 38, 70, 74, 75, 78, 96, 101, 103, 109–111

GRU Gated Recurrent Units 32

IR Infrared 10, 13

Leaky ReLU Leaky Rectified Linear Unit 23

LPM Lumped Parameter Model 7

LPTM Lumped Parameter Thermal Method , 2, 5, 9, 16–18, 43, 74, 80, 84, 106, 110

LPTN Lumped Parameter Thermal Network 4, 6

LSTM Long Short-Term Memory i, 6, 31, 32, 37

MAE Mean Absolute Error 25, 61, 64, 68, 69, 71, 73, 75, 77, 89, 90, 92, 94, 95, 97, 98, 100, 103–105, 108, 109

ML Machine Learning 20, 24, 26, 41

MLI Multi-Layer Insulation 80, 81

MLP Multi-Layer Perceptron i, , 5–7, 22, 27, 28, 30, 38, 46, 87, 102

MPNN Message Passing Neural Network 18, 33, 34, 38, 56

MSE Mean Squared Error i, 7, 25, 28, 53, 54, 56–59, 61–64, 68, 69, 71–73, 75–78, 88–90, 92, 94, 95, 97, 98, 100, 101, 103–110

NNConv Neural Network Convolution , 38–40, 74, 78, 87, 102, 106–108, 111

ODE Ordinary Differential Equations 50

OLR Outgoing Long-wave Radiation , 14, 16

PCB Printed Circuit Board , 1, 2, 4, 7, 8, 10, 36, 43, 45–50, 56, 68, 70, 71, 74, 75, 77, 87–89, 92, 102, 106, 107, 110, 111

PDE Partial Differential Equations 4, 5

PINN Physics-Informed Neural Network 45

ReLU Rectified Linear Unit 7, 23, 28–30, 38, 40, 55, 102

RMSE Root Mean Square Error 45, 61, 63, 64, 66, 68, 69, 71, 73, 75–78, 89, 90, 92, 94, 95, 97, 98, 100, 101, 103–106, 108–111

RMSProp Root Mean Square Propagation 27

RNN Recurrent Neural Network i, , 6, 28, 30, 31

SGD Stochastic Gradient Descent 27, 28

SiLU Sigmoid Linear Unit 23, 35, 55, 87, 102

SVD Singular Value Decomposition 45

TanH Hyperbolic Tangent 23, 28, 30

TNN Thermal Neural Network 5, 6

TPU Tensor Processing Unit 40

UV Ultraviolet 13

1

Introduction

One of the most critical conditions when carrying out missions in the space environment is temperature. Satellites are exposed to highly variable thermal loads and must be designed to withstand extreme temperatures. Depending on their orbit, they may experience rapid transitions from extremely high temperatures to cryogenic conditions. Therefore, the thermal control system must be carefully designed to maintain operating temperatures within an acceptable range for onboard instruments. An efficient thermal control system is vital for the operability and longevity of satellites.

To address these challenges, various thermal control techniques have been developed, designed to maintain component temperatures within acceptable ranges. Since convection does not occur in space, thermal control becomes significantly more complex compared to applications on Earth. The absence of convection forces reliance on other methods, such as radiation and conduction, to manage heat.

One of the most critical components in every satellite are the Printed Circuit Board (PCB)s. These components form the heart of the electrical systems due to their compact nature and diverse functionality. Given the limited weight and space available for components in space missions, PCBs are indispensable in satellite design.

However, due to the complexity of PCBs, manufacturers are often unable to determine exactly where and how much heat is dissipated within them. While most manufacturers provide the total heat dissipation of the PCB, accurately predicting the temperature distribution remains a challenge. This issue is still under investigation in the field.

The importance of ANNs in modern applications cannot be overstated. From speech recognition and computer vision to fraud detection and personalized medicine, their applications span a wide range of fields. In aerospace, particularly for thermal management, the ability to predict the thermal

conditions of PCBs accurately can significantly improve design, reduce costs, and enhance system reliability and lifespan.

Traditional methods like FEM, CFD, or LPTM are well-established and accurate but often require high computational resources and long runtimes, particularly for high-resolution 3D models or transient simulations. These methods typically involve solving large systems of equations iteratively, making them impractical for real-time applications or design iterations where rapid feedback is essential.

In contrast, ANN-based approaches, once trained, can offer near-instantaneous predictions. This makes them particularly attractive for real-time thermal monitoring, fault detection, or onboard thermal management in satellites. GNNs, in particular, are well suited for thermal modelling due to their ability to directly operate on the graph-like structure of thermal networks. Unlike conventional ANNs, GNNs can capture local and global dependencies between nodes, preserve spatial relationships, and naturally incorporate the topology of the thermal model. These characteristics make them highly effective for approximating temperature distributions across a PCB or a full satellite, offering a promising alternative to computationally intensive simulations.

In this thesis, the capabilities of GNNs in predicting steady-state temperatures within a PCB will be explored, in comparison to traditional numerical solvers based on the first law of thermodynamics. To accomplish this objective, a detailed comparison between different GNN architectures will be presented, highlighting their advantages and limitations. First, these architectures will be tested against a 2D PCB steady state temperature prediction problem. For this problem, a synthetic Dataset will be used as ground truth. The solver that generates the solutions has been programmed on python.

This thesis aims to predict the temperature distribution within a PCB, leveraging the capabilities of Artificial Neural Network (ANN)s, particularly Graph Neural Network (GNN)s. The power of GNNs in this domain lies in their ability to model complex systems such as the heat dissipation in a PCB, which can be represented as a graph of interconnected nodes with corresponding temperatures.

The motivation for using GNNs arises from the fact that the PCB solver discretizes the PCB into a finite number of nodes, each with a temperature and connections to other nodes, which is analogous to the structure of a graph.

Once the effectiveness of GNNs has been demonstrated for the 2D PCB problem, this analysis will be extended to more complex 3D scenarios. In these cases, a simplified model of a complete satellite will be used, with the goal of developing a fast prediction tool. This approach aims to reduce the large computational costs associated with traditional solvers, such as Finite Element Method (FEM), Computational Fluid Dynamics (CFD), or Lumped Parameter Thermal Method (LPTM).

The goal of these comparisons and analysis is to determine the suitability of GNNs for solving real thermal models, within an accuracy threshold pre-defined and with a fewer computational cost,

almost for instant predictions.

2

State of the Art Review

The implementation of neural networks in thermal modelling is a rapidly developing field, with works ranging from predicting the temperature of a house and its interior rooms, taking into account interactions with other houses, the environment, and heat flows between rooms [8], to real-time temperature prediction on a PCB.

To accurately solve thermal models, one can use CFD, the FEM based on PDE, or LPTN. The latter is the most commonly used so far due to its relatively low number of variables compared to the former, while still maintaining high reliability. However, the drawback of this physical model is that it requires advanced knowledge of the physical system, as well as a deep understanding of the geometry and physical characteristics of the entire model [9].

The motivation behind the use of ANNs to solve thermal problems stems from their ability to make rapid predictions, even when facing previously unseen scenarios. Once appropriately trained, these models can generalize from known data to accurately estimate solutions for new conditions, offering a significant speed advantage over traditional numerical solvers. This predictive efficiency is particularly valuable in applications where multiple cases need to be evaluated quickly, for example, in design optimization, uncertainty quantification, or real-time simulations. In contrast, conventional thermal analysis tools, such as lumped parameter thermal solvers (e.g., ESATAN-TMS), often require considerable computational resources and time just to solve a single case.

Moreover, ANNs contribute to the democratization of thermal modelling. Traditional methods often require deep expertise in heat transfer physics, mesh generation, boundary condition selection, and solver configuration. In contrast, once a neural network model is trained, it can provide accurate results with minimal user intervention or domain-specific knowledge. This opens the door for non-experts to explore and interact with thermal models in a more intuitive way.

The field of neural networks applied to thermal analysis is rapidly evolving, with ongoing research

focused on improving their performance in real-world, complex cases. Key areas of development include reducing the amount of training data required, increasing generalization capacity, and enhancing prediction accuracy. These improvements aim to make ANNs not only faster but also more reliable for high-fidelity simulations. Ultimately, the integration of ANNs into thermal modelling workflows paves the way for advanced applications such as digital twins, which require real-time feedback, scalability, and adaptability to changing conditions.

Neural Networks for Steady-State Thermal Modelling

Regarding their architecture, the ANNs that have been primarily used in these analyses are of the Multi-Layer Perceptron (MLP) type [10, 11]. While they can learn various patterns, MLPs are inefficient in terms of computational cost and learning speed due to their full connectivity between neurons, which leads to a significantly longer training time. Their usefulness relies on global node connections, adding global relationships that can, and often do, have a significant impact on the predicted outcome, but in reality many of these relations are casual and not something that should be learned [12].

To overcome these spatial limitations and better align neural architectures with the underlying physics and geometry of thermal systems, alternative models have been proposed. While MLPs cannot learn spatial dependencies effectively due to their lack of localized processing, Convolutional Neural Network (CNN)s were developed to address this issue. CNNs are particularly effective in recognizing spatial patterns in structured data, such as images or 2D thermal maps, through convolutional layers that capture local interactions.

However, CNNs are inherently limited to regular grid-like geometries and struggle to generalize when applied to unstructured or three-dimensional domains. This becomes a significant issue in realistic thermal problems, such as those involving complex 3D geometries or irregular node distributions, where CNNs cannot properly represent spatial relationships.

In contrast, Graph Neural Network (GNN)s provide a more flexible and powerful framework. GNNs operate directly on graph-structured data, which aligns naturally with the discretization used in thermal solvers such as FEM or LPTM. They represent the system as a set of nodes and edges, allowing them to learn how thermal information propagates through arbitrary topologies. This makes GNNs especially suited for 3D thermal modelling, where spatial relationships are complex and irregular. Thus, while MLPs and CNNs may suffice for simplified or 2D problems with regular geometries, GNNs are significantly more capable for solving real-world 3D thermal problems.

To address these spatial challenges while leveraging the development of ANNs, a new model was proposed by Wilhelm Kirchgässner [9], known as TNN, which combines physics with trainable neural networks for solving PDEs. This approach enables the learned parameters to have physical meaning, unlike a purely ANN-based model. This also allows for greater accuracy in scenarios out-

side the training dataset, without requiring an exhaustive understanding of the material properties, geometry, or physical model, since the TNN is capable of learning all of this. The TNN combines the physical model of LPTNs with a type of ANN similar to RNNs [9].

Neural Networks for Transient Thermal Modelling

All of the architectures above were used for steady state thermal problems, but they are not adequate for solving transient problems. While CNNs and MLPs have demonstrated strong performance in image recognition and static regression tasks, their application to transient thermal problems presents several important limitations. These problems are inherently time-dependent, governed by the evolution of temperature distributions over time in response to dynamic boundary conditions, internal heat sources, and material properties. Classical feed-forward architectures lack the mechanisms required to capture such temporal dynamics.

To begin with, they lack temporal memory, CNNs and MLPs process inputs independently, without memory or recurrence. As such, they cannot inherently model how thermal states evolve over time, which is essential in transient heat transfer. In addition, these networks are designed for static input-output mappings, such as the relationship between geometry and steady-state temperature fields. However, transient thermal problems involve a sequence of state transitions, which requires memory-aware modelling approaches.

Moreover, heat transfer follows time-dependent partial differential equations (e.g., the transient heat conduction equation). Without encoding temporal information explicitly into the input, CNNs and MLPs fail to capture the cumulative and causal nature of thermal propagation. Incorporating time as an input variable into feed-forward models significantly increases the dimensionality of the input space, which often demands large datasets across many time steps and may still result in poor generalization across unseen temporal intervals.

These limitations make CNNs, MLPs, and even TNNs suboptimal for directly solving transient thermal problems. Instead, architectures explicitly designed to model temporal sequences, such as RNNs and LSTMs, are better suited. These models incorporate memory mechanisms that allow them to learn how the system evolves over time, capturing both short-term and long-term dependencies within thermal dynamics.

Conceptually, RNNs can be seen as the temporal counterpart of MLPs, LSTMs as the equivalent of CNNs for time sequences, and graph-based architectures such as GNS are the natural extension of GNNs to transient [13] [14], sequential graph data. These recurrent or memory-enhanced models provide the necessary mechanisms to capture time-evolving behaviour in thermal systems.

For transient problems, RNNs and LSTMs architectures have been used to solve problems such as weather prediction, mechanical system interactions, biomedical signals, and even stock and demand forecasts. Nevertheless, transient thermal modelling networks are being developed to give fast

solutions where traditional solvers are slow, as it happens in predicting the thermal behaviour of components on a PCB, internal temperatures of batteries in real-time, and satellite thermal control, given that temperatures vary noticeably in an orbit.

Motivation and Application in this Work

The latter problem is the one that motivates this Bachelor's thesis. In this work, a GNN-based ANN will be investigated and developed for the thermal modelling of a PCB, and its results will be compared between different architectures. Once the accuracy and effectiveness of those architectures are evaluated, the next milestone will be the development of the same GNNs so that a complete 3D model with the necessary boundary conditions can be introduced and solved within a 3 K threshold to label it as effective.

Using GNNs for solving thermal 3D models lies in the similarity between how Lumped Parameter Model (LPM)s are discretized, with nodes and connections between them, as it happens also in FEM. Graph Neural Network (GNN)s are structured in nodes and edges, which act as connectors between nodes and as pathways through which the ANN passes information. While it is true that other architectures are simpler and sufficient for solving 2D problems, such as CNNs, as previously mentioned, when a third dimension is added and the geometry is no longer discretized with regular patterns, GNNs provide the best performance and capability to solve complex problems with virtually any geometry. This architecture is also capable of learning how to solve transient problems, although in the present work only the steady-state problem will be addressed.

The implementation of GNNs in thermal modelling is an emerging field, with constant research and improvements. Notable milestones include the precise modelling of temperature variations in a building [8]. In this paper, the authors proposed a new model with physical constraints, enabling real-time prediction of normalized temperatures with significantly improved accuracy. They reported that the Mean Squared Error (MSE) was consistently lower than with traditional machine learning architectures. The hyper-parameters used included 2 layers with 16 nodes each and a learning rate of 0.001, resulting in a normalized open-loop MSE of 0.0012.

To achieve the best results, GNNs can be constrained to obey physical laws. One such example is a study by Quercus Hernández [14], in which the principles of energy conservation and entropy inequality were enforced using the GENERIC formalism [15].

Another relevant study was conducted by Helios Sanchis-Alepuz, where a dynamic dissipation model was developed to solve a 3D PCB thermal problem by combining GNNs with thermodynamic principles [16]. In this case, a GNS architecture was used, comprising an encoder and a decoder. The encoder consisted of two hidden layers of an MLP with 128 neurons per layer, each node and edge having 128 attributes. A SELU activation function and LayerNorm were used between layers to improve stability. The decoder mirrored the encoder structure but employed ReLU activations.

The model was trained using a batch size of 10 and the Adam optimizer. Each epoch required 14 minutes, and the training lasted between 700 and 1000 epochs. The study concluded that, despite a maximum error of 1.4% (mostly localized around heat sources), the proposed method was more efficient than solving the same problem using FEM [16].

In the present thesis, a GNN will be developed and tested for solving a steady 2D PCB. Once the most efficient GNN architecture is identified, it will be applied to solve 3D ESATAN-TMS steady-state models, with the aim of crafting an ANN capable of solving complete 3D models and significantly reducing computational cost once trained.

3

Theoretical Foundations of Thermal Control

This chapter introduces the key concepts required to understand the thermal behaviour of systems operating in space environments. It aims to establish a solid theoretical basis for the methods and models used throughout this thesis, including both classical approaches and their integration with modern data-driven techniques.

The discussion begins with an overview of thermal control in spacecraft, highlighting the challenges posed by the space environment and the specific temperature requirements of various subsystems. Then, the chapter reviews the fundamental mechanisms of heat transfer—conduction, convection, and radiation—with a focus on their relevance and limitations in orbit.

Finally, the chapter explores the Lumped Parameter Thermal Method (LPTM), a widely adopted modelling technique in the aerospace industry, and its strong connection to graph-based representations. This serves as a bridge toward the neural network architectures used later in this work, particularly Graph Neural Network (GNN), which naturally align with the structure and logic of LPTM formulations.

3.1 Thermal Control Subsystem

Thermal control is the discipline, implemented as a subsystem in satellites, responsible for maintaining the temperature and thermal gradients of an object or system within predefined limits [17]. This discipline includes both active thermal control, which uses energy to dissipate or transfer heat to a specific component or system, and passive control, which achieves its goal through surface coatings, special paints and mechanical joints.

The importance of thermal control in space lies in the operational and survival needs of different systems and components in this environment. Depending on whether it is an IR detector, a PCB, or the satellite's own structure, temperature requirements will vary, as shown in Table 3.1 [7].

Table 3.1: Temperature requirements of subsystems [7]

Temperature	$\mathbf{T}_{min}[^\circ\mathbf{C}]$	$\mathbf{T}_{max}[^\circ\mathbf{C}]$
Electronics (housing)	-10	50
Batteries	0	20
Solar panels	-100	120
Antenna	-65	95
Hydrazine tank	10	50
IR detectors	-223	-173
Inactive structure	-100	100

Temperature Gradients	
Optoelectronic equipment	$\Delta T < 5^\circ$C
High-resolution cameras	$\Delta T < 0.1^\circ$C
Detectors (CCD)	$\Delta T < 0.01^\circ$C

Temperature Stability	
Electronics (housing)	$\frac{dT}{dt} < 5^\circ$C/hour
Detectors (CCD), during observation periods	$dT < 0.1^\circ$C

Heat transfer occurs when two sources are at different temperatures, with heat flowing from the hotter to the colder one. The three main heat transfer mechanisms are: conduction, radiation, and convection. However, in the space environment, convection does not significantly contribute to heat transfer.

3.2 Heat Transfer

3.2.1 Conduction

Conduction refers to the transfer of thermal energy within a material driven by a temperature gradient. The underlying mechanism involves the interaction of microscopic particles such as atoms, molecules, or electrons. This mechanism differs depending on whether the material is a solid or a fluid.

In fluids, conduction occurs primarily through collisions between molecules and, to a lesser extent, free electrons. In solids, the conduction mechanism depends on the material type: in metallic solids,

it is dominated by the movement of free electrons, while in non-metallic solids, it primarily occurs through lattice vibrations (phonons) [7, 18].

In all cases, energy is transferred from higher-energy (hotter) particles to lower-energy (colder) ones, making conduction the dominant heat transfer mechanism within solid bodies.

The rate of heat conduction is governed by Fourier's law of heat conduction [19]:

$$\boldsymbol{q} = -k\nabla T \ [\mathrm{W/m^2}] \tag{3.1}$$

where:

- $\boldsymbol{q}$ is the heat flux vector [$\mathrm{W/m^2}$],
- k is the thermal conductivity of the material [$\mathrm{W/(m \cdot K)}$],
- ∇T is the temperature gradient [$\mathrm{K/m}$].

The negative sign indicates that heat flows from higher to lower temperatures.

Because of the lower density and weaker intermolecular interactions, fluids generally have much lower thermal conductivity than solids. For example, air and water have thermal conductivities that are several orders of magnitude lower than metals like copper or aluminium.

However, in fluids, another mechanism, convection, often dominates heat transfer. Unlike conduction, convection involves the bulk movement of fluid particles and can lead to much higher effective heat transfer rates.

Understanding these distinctions is crucial for modelling thermal behaviour in engineering applications, especially when analysing electronic systems, heat exchangers, or environmental conditions where both solid and fluid domains interact.

3.2.2 Convection

Convection can be defined as conduction in the presence of a moving fluid, with conduction being the limiting case of convection when the fluid is at rest. Convection requires a moving fluid in contact with another fluid or solid, transferring heat via molecular collisions and free electron flow. Convection is generally classified into two main types:

- **Natural (or free) convection**: Driven by buoyancy forces that arise due to temperature-induced density differences in the fluid.
- **Forced convection**: Driven by an external source such as a pump, fan, or flow over a surface, which actively induces fluid motion.

In both cases, heat transfer occurs more efficiently than by conduction alone, as moving fluid carries energy from one region to another.

Governing Equation:
The rate of convective heat transfer between a solid surface and a fluid is typically described by Newton's law of cooling:

$$q = h \cdot A \cdot (T_s - T_\infty)$$

where:

- q is the heat transfer rate [W],
- h is the convective heat transfer coefficient [$W/(m^2 \cdot K)$],
- A is the surface area through which heat is transferred [m^2],
- T_s is the surface temperature [K],
- T_∞ is the temperature of the surrounding fluid [K].

The convective heat transfer coefficient h depends on various factors, including:

- Fluid properties (viscosity, thermal conductivity, specific heat, density),
- Flow conditions (velocity, turbulence),
- Geometry and orientation of the surface,
- Temperature gradients.

Dimensionless Numbers:
Convection phenomena are often analysed using dimensionless numbers that characterize the nature and intensity of the flow:

- **Reynolds number** (Re): Measures the ratio of inertial to viscous forces; distinguishes between laminar and turbulent flow.
- **Prandtl number** (Pr): Relates momentum diffusivity to thermal diffusivity.
- **Grashof number** (Gr): Analogous to Reynolds number in natural convection; relates buoyancy to viscous forces.
- **Nusselt number** (Nu): Measures the enhancement of heat transfer due to convection relative to pure conduction.

For many engineering applications, empirical or semi-empirical correlations are used to estimate the Nusselt number and, thus, the convective heat transfer coefficient h, based on flow geometry and regime.

Relevance in Thermal Modelling
Convection is often the dominant mode of heat transfer in systems involving fluids in motion, such as cooling of electronic components by air, heat exchangers, or fluid transport in microchannels. In thermal simulations of real-world systems, correctly modelling convective boundary conditions is essential for accurately predicting temperature distributions.

However, as this paper is focused on space environment, convection will not be taken into account in the models.

3.2.3 Radiation

Thermal radiation is the transfer of heat in the form of electromagnetic waves emitted by matter due to its temperature. The frequency spectrum of thermal radiation ranges from Infrared (IR) to Ultraviolet (UV) and its associated with molecular, atomic, and electronic transitions. Unlike conduction and convection, which require a medium (solid or fluid) for heat transfer, radiation can occur through a vacuum and does not involve the motion or interaction of particles, so it is the fastest form of energy transfer since it travels at the speed of light and is not attenuated in vacuum [20].

All bodies with a temperature above absolute zero emit thermal radiation. The amount and characteristics of this radiation depend on the surface temperature and properties such as emissivity.

An important concept is the blackbody: an idealized object that perfectly emits and absorbs thermal radiation, absorbing *all* incident radiation regardless of wavelength.

Stefan–Boltzmann Law
The total thermal radiation emitted per unit surface area by a blackbody is governed by the Stefan–Boltzmann law:

$$q = \sigma T^4$$

where:

- q is the radiative heat flux [$\mathrm{W/m^2}$],
- $\sigma = 5.670 \times 10^{-8}$ $\mathrm{W/(m^2 \cdot K^4)}$ is the Stefan–Boltzmann constant,
- T is the absolute temperature of the emitting surface [K].

For real surfaces, the equation becomes:

$$q = \varepsilon \sigma T^4$$

where $\varepsilon \in [0, 1]$ is the emissivity of the surface, a material-dependent property that quantifies how efficiently a surface emits radiation compared to a blackbody.

Radiation Exchange Between Surfaces:
When two surfaces exchange thermal radiation, the radiative heat transfer depends on the view factor and the fourth powers of their absolute temperatures. The amount of heat per second that is emitted from 1 and reaches 2 is defined as:

$$\dot{Q}_{1\to 2} = A_1 F_{1\to 2} \varepsilon_1 \sigma T_1^4$$

Where F is the view factor, which is defined between two surfaces 1 and 2 as the fraction of the radiation leaving surface 1 that reaches surface 2 [7]. More accurate modelling considers not only view factors (or configuration factors), but also surface geometry, and reflectivity.

Relevance in Engineering Applications
Radiation plays a crucial role in thermal systems where conduction and convection are limited or absent. Examples include:

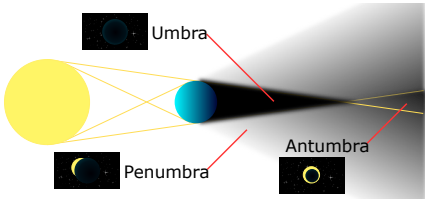
- Spacecraft thermal control (e.g., radiation to deep space),
- High-temperature furnaces and reactors,
- Electronics operating in vacuum conditions,
- Fire and combustion modelling.

In simulation and data-driven modelling, radiation is often challenging to capture due to its global dependence and non-linear nature. Recent thermal neural networks and GNN-based models address this by incorporating radiative boundary conditions or by learning effective radiative behaviour through training data.

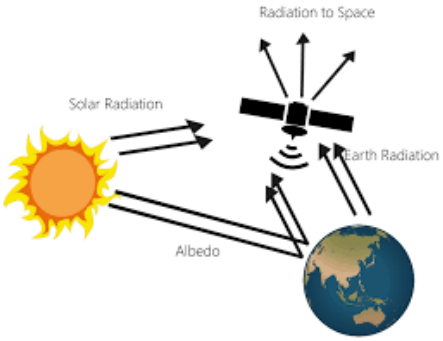
3.3 Space Thermal Environment in Earth Orbit

Objects in space must withstand extreme and harsh conditions as it is shown in Figure 3.1b. On one hand, there is direct solar radiation and the portion reflected by the Earth's surface called Albedo. Additionally, the Earth itself emits radiation, known as Outgoing Long-wave Radiation (OLR), which also affects the object. Lastly, deep space can be modelled as a blackbody with an equivalent temperature of 2.7 K.

These factors make thermal control more complex, as objects can rapidly move from sunlit regions to shadow (umbra and penumbra), creating strong temperature gradients.



(a) Umbra and penumbra regions [21].



(b) Radiative interactions [22].

Figure 3.1: Radiation in Earth orbit [1]

To better understand thermal radiation in space, some key concepts must be defined:

3.3.1 Direct Solar Radiation

Bodies orbiting the Earth (i.e., located at 1 AU) receive a solar irradiance of $G_s = 1366.1$ [W/m^2]. Solar irradiance is defined as the rate at which solar energy strikes a surface perpendicular to the Sun's rays at Earth's orbit. This value is an average, but in reality, it varies with solar distance and solar cycles, making it important to account for these variations in thermal modelling. In the present thesis it is assumed that solar irradiance is constant, yielding the absorbed heat by the orbiting body as:

$$\dot{Q}_{Sol} = \alpha G_s A_{sat} \cos\theta \quad [W] \tag{3.2}$$

Where:

- α is the **absorptance**, defined as the fraction of the incident solar radiation that is absorbed by the surface (dimensionless, $0 \leq \alpha \leq 1$).
- A_{sat} is the total area of one surface of a plate.
- θ is the angle between the normal vector to the surface and the direction of the incoming solar radiation.

3.3.2 Albedo

This phenomenon arises from the reflection of solar radiation by the Earth's surface, which then strikes the orbiting object. It represents indirect solar radiation incidence [19]. The albedo coefficient a is defined as the fraction of solar radiation reflected by a planet. In this case, the view factor $F_{E\to sat}$ represents the fraction of radiation leaving Earth surface that reaches another surface [7]. An interesting property of the view factor is the following, which is used to obtain $F_{sat\to E}$:

$$F_{1\to 2}A_1 = F_{2\to 1}A_2 \tag{3.3}$$

The incident heat is given by:

$$\dot{Q}_{Alb} = aG_sA_{sat}F_{sat\to E}\cos\theta \quad [W] \tag{3.4}$$

3.3.3 Planetary Radiation, OLR

This is the radiation emitted by the entire planet due to its temperature, which is derived from the Stefan-Boltzman law. It includes radiation from the surface, oceans, and atmospheric gases [23]. Although these elements have different emissivities, a weighted average is generally assumed (ε_E). The incident heat in an orbiting body from this source is calculated as:

$$\dot{Q}_{IR} = A_{sat}F_{sat\to E}\varepsilon_E\sigma T^4 \quad [W] \tag{3.5}$$

3.4 Lumped Parameter Thermal Method (LPTM) and its Connection to Graph-Based Models

The Lumped Parameter Thermal Method, also known as the *Network Analogy* method, is a foundational technique for modelling heat transfer in complex systems, particularly in the aerospace industry. It is implemented in tools such as ESATAN-TMS and Thermica, and enables efficient simulation of both steady-state and transient thermal behaviours [24].

3.4.1 Principles of the Lumped Parameter Method

LPTM approximates a continuous thermal domain as a discrete network of nodes (thermal masses) interconnected by conductors (heat transfer paths). This abstraction originates from the electrical-thermal analogy, where temperature is analogous to voltage, heat flow to electric current, thermal capacitance to electrical capacitance, and conductance to resistance.

Each node represents a region with uniform temperature and thermal capacity C_i, and it is connected to other nodes via:

- **Linear conductors** K_{ij}: Modelling conduction, convection, or other linear mechanisms.
- **Radiative conductors** R_{ij}: Modelling radiative heat exchange using view factors and emissivities.
- **Heat sources** Q_i: Representing external or internal power inputs.

The general transient heat balance at each node i is expressed as:

$$C_i \frac{dT_i}{dt} = \sum_{j \neq i} K_{ij}(T_j - T_i) + \sum_{j \neq i} R_{ij}(T_j^4 - T_i^4) + Q_i$$

In steady-state conditions, the time derivative vanishes, resulting in an algebraic non-linear equation system:

$$\sum_{j \neq i} K_{ij}(T_j - T_i) + \sum_{j \neq i} R_{ij}(T_j^4 - T_i^4) + Q_i = 0$$

3.4.2 Historical Context and Mathematical Basis

Historically, this method evolved from analogue computing: in the early 20^{th} century, thermal problems were solved using physical electrical networks of resistors and capacitors [24]. This analogue foundation later evolved into digital simulation techniques, and the LPTM was formally established as a numerical method by Southwell [25], Emmons [26], and Dusinberre [27].

From a mathematical standpoint, the LPTM corresponds to a first-order finite-difference approximation of the heat conduction equation:

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \quad \text{where} \quad \alpha = \frac{\kappa}{\rho c}$$

Where:

- κ is the thermal conductivity of the material [W/(m·K)], which quantifies its ability to conduct heat.
- ρ is the mass density of the material [kg/m^3].
- c is the specific heat capacity [J/(kg·K)], representing the amount of energy required to raise the temperature of one kilogram of the material by one kelvin.

By discretizing the spatial domain into a mesh of thermal nodes and applying energy conservation, the LPTM provides a system of ordinary differential equations that can be solved using time integration methods such as Euler or Crank-Nicolson.

3.4.3 Construction and Application in ESATAN-TMS

In ESATAN-TMS, the user defines a network of nodes and their physical properties, including:

- **Capacitance** $C_i = \rho_i c_i V_i$.
- **Conductance** values between connected nodes.
- **Boundary conditions**: Prescribed temperatures or heat fluxes.

For example, a metal plate may be divided into square cells, each represented by a node, and conductors are defined based on the material's thermal conductivity κ, dimensions, and boundary conditions. For regular grids, thermal conductance between two adjacent nodes is:

$$K_{ij} = \kappa \frac{A}{L}$$

where A is the cross-sectional area and L is the distance between nodes.

3.4.4 Relation to Graphs and Modern Machine Learning

The LPTM can be naturally interpreted as a graph:

- **Nodes** in the graph correspond to physical components (e.g., surfaces, subsystems).
- **Edges** represent thermal couplings, both conductive and radiative.
- **Node attributes** include temperature, heat capacity, internal or external power.
- **Edge attributes** include conductance values K_{ij}, radiation factors R_{ij}, or geometric coefficients.

This structure closely aligns with modern Graph Neural Network (GNN) models, particularly those designed for thermal surrogate modelling. For instance, Graph Convolutional Network (GCN) and Message Passing Neural Network (MPNN) operate over the same graph topology defined by the LPTM, allowing the integration of physics-based and data-driven approaches.

3.4.5 Advantages and Use Cases

The LPTM is ideal for space applications due to:

- Its intuitive physical structure.
- Flexibility to model complex geometries and mixed coupling types.

- Compatibility with experimental and analytical data.
- Suitability for both steady-state and transient analyses.

It allows fast yet accurate modelling of thermal environments in satellites, instruments, and components, and serves as the backbone of tools like ESATAN-TMS for pre-flight thermal validation.

4

Theoretical Foundations of Deep Learning

In the present chapter, we explore the fundamental concepts of machine learning and deep learning, with a particular focus on their application to thermal analysis problems. We will introduce the specific types of models used, explain their structure and training process, and discuss how they can be adapted to graph-based representations of physical systems.

To begin, it is important to define what ML truly entails. Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that mimics the way humans learn by utilizing data and algorithms. Through exposure to large datasets and the application of statistical techniques, machine learning models can be trained to make predictions, such as assigning labels to images, generating the next word in a sentence, or recommending movies based on previous user preferences.

Within the field of machine learning, there exists a variety of model types, including decision trees, Bayesian networks, and neural networks. The latter refers to a family of algorithms that learn by example and are particularly effective in solving complex, non-linear problems. When these neural networks comprise multiple hidden layers, the approach is commonly referred to as deep learning.

Deep learning involves neural networks with three or more layers that attempt to simulate the behaviour of the human brain in processing data. These layers are organized in a hierarchical structure reminiscent of biological neurons, enabling the system to achieve high accuracy and handle vast and complex datasets. As a result, deep learning models excel in tasks such as image recognition and natural language processing [28].

4.1 Fundamentals of ANNs

An Artificial Neural Network is a computational framework inspired by the neural architecture of animal brains [29]. It is composed of interconnected units called *neurons*, which are arranged in layers and linked through weighted connections.

Each neuron takes an input, usually a real-valued vector, applies a linear transformation using weights and biases, and then passes the result through a non-linear activation function. The outcome is sent to neurons in the subsequent layers.

Formally, an ANN approximates a function through a sequence of transformations, expressed as:

$$\boldsymbol{y} = f(\boldsymbol{x}) = f^{(L)}(f^{(L-1)}(\dots f^{(1)}(\boldsymbol{x})))$$

where each $f^{(l)}$ denotes a layer of neurons, and $\boldsymbol{x}$, $\boldsymbol{y}$ represent the input and output vectors, respectively.

The connectivity between neurons can vary based on the type of network:

- **Feed-forward networks**: Data flows only from input to output.
- **Recurrent networks**: Include feedback loops to handle temporal sequences.
- **Convolutional networks**: Use local connectivity to capture spatial dependencies.

In the next sections, we examine how these architectures are built, trained, and utilized to address complex challenges, particularly in the thermal analysis of spacecraft systems.

4.2 Characteristics and Training of ANNs

4.2.1 Architecture

Artificial Neural Networks are usually designed with layers of neurons, as shown in Figure 4.1. These layers progressively transform the input data into higher-level features from input to output.

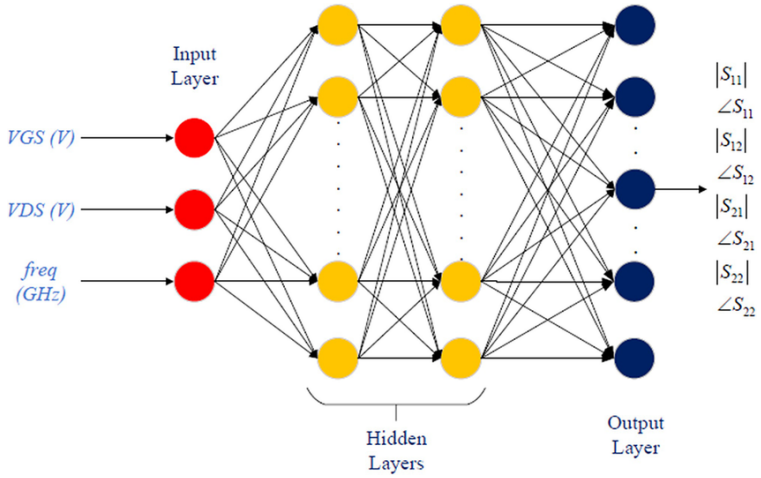


Figure 4.1: Diagram of a MLP [2].

The key types of layers include:

- **Input layer**: Takes the raw data and forwards it unchanged. Its size matches the dimensionality of the input vector.
- **Hidden layers**: Apply linear operations followed by non-linear activations. Increasing the number of these layers enhances the model's capacity to learn complex patterns but also raises the risk of over-fitting and adds computational complexity.
- **Output layer**: Delivers the network's final result. In classification tasks, it often uses softmax or sigmoid activations. In regression tasks, such as those addressed here, it typically includes one neuron per target variable, like node temperatures.

Neurons can be connected in various configurations:

- **Fully Connected (FC) layers**: Every neuron connects to all neurons in the next layer.
- **Pooling layers**: Reduce dimensionality and sensitivity by summarizing local outputs. These are common in CNNs.

The choice of architecture layer depth, width, and connectivity must be tailored to the problem being solved.

4.2.2 Hidden Units and Activation Functions

Each hidden unit in a neural network performs a non-linear operation on its input, composed of a linear transformation followed by an activation function:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{a} = g(\boldsymbol{z})$$

Here, $\boldsymbol{x}$ is the input vector, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weight matrix and bias vector, and g is the activation function applied element-wise.

The five most common activation functions, both for autoregression and classification problems, are illustrated in Figure 4.2.

- **Rectified Linear Unit (ReLU)**: $g(z) = \max(0, z)$
- **Leaky Rectified Linear Unit (Leaky ReLU)**: $g(z) = \begin{cases} z, & \text{si } z \geq 0 \\ \alpha z, & \text{si } z < 0 \end{cases}$
- **Sigmoid**: $g(z) = \frac{1}{1+e^{-z}}$
- **Sigmoid Linear Unit (SiLU)**: $g(z) = z \cdot \sigma(z)$
- **Hyperbolic Tangent (tanh)**: $g(z) = \tanh(z)$

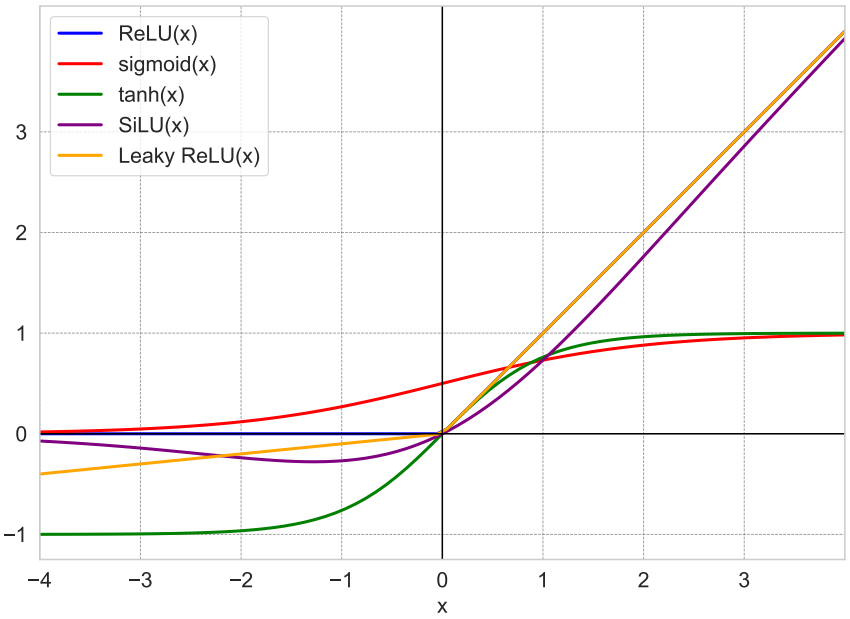


Figure 4.2: Graphical representation of common activation functions.

While sigmoid and TanH are smooth and bounded, they can lead to vanishing gradients in deep networks. ReLU and its variants are preferred for their computational efficiency and better gradient flow.

Normalizing input features is crucial, especially when they vary in magnitude (e.g., $T \sim \mathcal{O}(10^2)$, $\dot{Q} \sim \mathcal{O}(10^0)$). Without normalization, training can be hindered by gradient saturation.

4.2.3 Dataset

It is evident that the network requires access to data, not only the inputs it will process, but also the corresponding outputs it is expected to predict. This collection of input–output pairs forms what is known as a dataset, which serves as the foundation of any deep learning application. Datasets are essential for enabling the model to learn the desired mappings between inputs and outputs. However, this structure specifically applies to supervised learning, where labelled data is available to guide the training process.

In most applications the sequence is not a problem. Nevertheless in problems where time plays a role, preserving the sequence of data is essential to maintain dynamic behaviour.

Datasets are typically categorized as:

- **Experimental**: Gathered from real-world sensors or measurements.
- **Synthetic**: Created from simulations or numerical models.

For effective training and evaluation, datasets are divided into:

- **Training set**: Used to update the model parameters (usually 70–80% of the data).
- **Validation set**: Monitors performance to guide hyper-parameter tuning.
- **Test set**: Reserved for the final evaluation of the trained model. The network has not seen any data from this set.

Even though providing such amount of data is fundamental, proper splitting of the dataset is also essential to avoid over-fitting and ensure robust generalization. Over-fitting is a phenomenon in ML where a model learns not only the underlying patterns in the training data but also the noise and random fluctuations. As a result, the model performs very well on the training set but fails to generalize to new, unseen data, leading to poor performance on validation or test datasets.

4.2.4 Hyper-parameters

When designing an ANN we need to specify certain characteristics that the network will have. These parameters are defined by the user, and is what the network does not have to infer from the given data.

Main hyper-parameters are:

- **Epochs**: Number of complete passes through the entire training dataset by the network during training.

- **Layers**: Number of levels (input, hidden, and output) in the network, where each layer transforms the data before passing it on.
- **Hidden dimension**: Number of neurons (units) in a hidden layer. It determines the capacity of the model to learn complex patterns.
- **Batch size**: Number of training samples processed before the model's internal parameters are updated. A smaller batch size means more updates per epoch.
- **Learning rate**: Controls how much the model's weights are adjusted during training. A higher learning rate means faster but potentially less stable learning.
- **Activation functions**: Mathematical functions applied to a neuron's output to introduce nonlinearity into the network, allowing it to learn complex patterns and relationships in the data.

4.2.5 Loss Functions

As every other aspect in life, we have to measure how well a network is performing. With this aim, we introduce loss functions such as Mean Squared Error (MSE), Mean Absolute Error (MAE) and Cross-Entropy Loss. All of these functions measure how close the ANN is from the ground-truth, but, they all differ a bit.

- **Mean Squared Error (MSE)**: Is a widely used metric in machine learning for evaluating the accuracy of regression models. It is defined as the average of the squares of the differences between the actual and predicted values. By squaring the errors before averaging, MSE penalizes larger deviations more heavily than smaller ones, making it sensitive to significant prediction errors. A lower MSE value indicates better model performance, as it implies that the predicted outputs are closer to the true values. MSE is particularly important in regression tasks and serves as a common loss function for training models to minimize prediction errors. [30].

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

  where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of samples.
- **Mean Absolute Error (MAE)**: Is a straightforward and effective metric for assessing the performance of regression models. It calculates the average of the absolute differences between the predicted values and the actual targets. Unlike metrics that square the errors, MAE treats all deviations equally, regardless of whether they result from overestimations or underestimations. This characteristic makes MAE especially valuable when the goal is to quantify the

overall magnitude of prediction errors without emphasizing their direction [31].

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Cross-Entropy Loss**: Also referred to as log loss, is a metric commonly used in machine learning to assess the performance of classification models. Its value lies between 0 and 1, with lower values indicating better model accuracy. An ideal model would achieve a cross-entropy loss of 0. During training, the optimizer's objective is to minimize this loss as much as possible, bringing it closer to zero [32].

  For binary classification, the Cross-Entropy Loss is defined as:

$$\text{CE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

  where $y_i \in \{0, 1\}$ is the true label and $\hat{y}_i \in (0, 1)$ is the predicted probability.

  For multi-class classification with one-hot encoding:

$$\text{CE} = -\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

  where C is the number of classes, $y_{i,c}$ is the true label (1 if class c is the correct class for sample i, else 0), and $\hat{y}_{i,c}$ is the predicted probability for class c.

4.2.6 Optimization functions

The goal of a ML algorithm is to reduce the generalization error associated with parameter prediction. If the true distribution of the parameters were known, the problem could be addressed using an optimization algorithm. However, since the real distribution is unknown, the task becomes a Machine Learning problem. To address this, the expected loss on the training set is minimized by substituting the true distribution with the empirical distribution observed by the ANN. This approach, however, presents challenges such as over-fitting, which must be properly managed.

These functions are essential for any kind of ANN training, and they have been developed in order to give the best performance possible with few setbacks. A key mechanism that enables these optimization techniques is **backpropagation**.

Backpropagation is an algorithm used to efficiently compute the gradients of the loss function with respect to the model's parameters. It applies the chain rule of calculus to propagate the error backward through the network, layer by layer. This allows the network to update its weights in a way that minimizes the overall error during training.

- **Gradient Descent, GD**: It is a first-order optimization algorithm used to minimize a loss function by iteratively updating the model parameters in the direction of the steepest descent, defined by the negative gradient. It calculates the gradient using the entire training dataset at each iteration, which ensures stable convergence but can be computationally expensive for large datasets.
- **Stochastic Gradient Descent, SGD**: It is a variant of gradient descent that updates model parameters using only a single training example (or a small mini-batch) at each iteration. While this introduces more noise into the updates, it significantly reduces computation time and often helps the model escape local minima, making it suitable for large-scale learning.
- **Momentum**: The Momentum optimization technique improves the convergence speed of SGD by incorporating information from previous parameter updates. It accumulates a velocity vector in the direction of consistent gradients and helps smooth out oscillations, particularly in ravine-shaped loss surfaces. This results in faster and more stable convergence.
- **Root Mean Square Propagation, RMSProp**: It is an adaptive learning rate optimization algorithm that adjusts the learning rate for each parameter individually by maintaining an exponentially decaying average of squared gradients. This helps prevent the learning rate from diminishing too quickly and allows for more efficient training, especially in non-stationary settings.
- **Adaptive Moment Estimation, Adam**: It is a widely used optimization algorithm that combines the benefits of Momentum and RMSProp. It computes adaptive learning rates for each parameter using estimates of both the first moment (mean) and the second moment (uncentered variance) of the gradients. Adam is known for its fast convergence and robustness across a wide range of deep learning applications.

4.3 Most common types of Neural Networks

4.3.1 Multi-Layer Perceptron (MLP)

A MLP is a type of feed-forward Artificial Neural Network (ANN)) that forms the foundation of many modern deep learning architectures. It consists of multiple layers of computational units, known as *neurons*, which are arranged into three primary categories: an **input layer**, one or more **hidden layers**, and an **output layer**.

Each neuron in an MLP applies a linear transformation followed by a non-linear activation function. Mathematically, this transformation is expressed as:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{a} = g(\boldsymbol{z})$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the input vector, $\boldsymbol{W}$ is the weight matrix, $\boldsymbol{b}$ is the bias vector, $\boldsymbol{z}$ is the pre-activation output, $g(\cdot)$ is the activation function, and $\boldsymbol{a}$ is the final output of the neuron.

The **input layer** receives raw data, with each feature mapped to a neuron. The **hidden layers** perform a series of transformations to extract meaningful representations and capture complex, non-linear relationships. The **output layer** provides the final prediction or classification result, depending on the specific learning task (e.g., regression or classification).

A defining characteristic of MLPs is the use of non-linear activation functions such as ReLU, sigmoid, or TanH. These functions introduce non-linearity into the model, allowing it to learn and represent complex mappings between inputs and outputs. Without such activation functions, an MLP would reduce to a simple linear model, regardless of its depth.

MLPs are trained using supervised learning techniques. Model parameters (weights and biases) are adjusted to minimize a predefined loss function, such as MSE or Cross-Entropy, using optimization algorithms like SGD or Adam. The training process involves *back propagation*, which is an efficient method for computing gradients across layers using the chain rule.

Although MLPs are theoretically capable of approximating any continuous function given sufficient capacity (as stated by the Universal Approximation Theorem [33]), they are less effective in capturing spatial or sequential structures present in data like images or time series. For these tasks, architectures such as CCNN or RNN are often preferred.

Nevertheless, MLPs remain a core component in neural network theory and are widely used for tasks involving tabular data, basic regression or classification problems, and as baseline models for more complex systems [34] [29].

4.3.2 Convolutional Neural Network (CNN)

A CNN is a specialized type of deep neural network designed to process data with a grid-like structure, such as images (2D arrays of pixels) or time series (1D sequences). CNNs are particularly effective for tasks involving spatial or temporal hierarchies and have become the standard architecture in computer vision applications, including image classification, object detection, and semantic segmentation.

Unlike fully connected architectures such as MLP, CNNs exploit local spatial relationships by applying convolutional layers, which consist of learnable filters (also known as kernels) that move across the input data. These filters extract localized features such as edges, textures, or shapes, enabling the network to build a hierarchical understanding of the input.

The convolution operation between an input $\boldsymbol{X}$ and a filter $\boldsymbol{K}$ is mathematically defined as:

$$S(i,j) = (\boldsymbol{X} * \boldsymbol{K})(i,j) = \sum_m \sum_n \boldsymbol{X}(i+m, j+n) \cdot \boldsymbol{K}(m,n)$$

where $*$ denotes the convolution operator, and $S(i, j)$ is the resulting value in the feature map at location (i, j).

A typical CNN is composed of several types of layers, as shown in Figure 4.3.

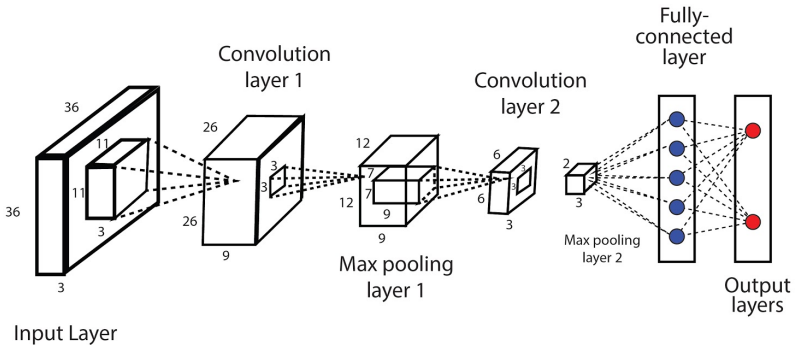


Figure 4.3: CNN architecture [3].

- **Convolutional Layers**: These layers apply multiple filters across the input to generate feature maps. They reduce the number of parameters by sharing weights spatially and are capable of detecting local patterns.
- **Activation Functions**: Non-linear functions such as ReLU ($\text{ReLU}(x) = \max(0, x)$) are applied element-wise to introduce non-linearity into the network.
- **Pooling Layers**: These layers downsample the spatial dimensions of the feature maps using operations like max pooling or average pooling. Pooling reduces computational cost and increases robustness to small translations.
- **Fully Connected Layers**: After several convolution and pooling layers, the resulting high-level features are flattened and passed through one or more fully connected layers for classification or regression.
- **Output Layer**: Produces the final prediction of the model. In classification problems, this layer often uses a softmax function to generate class probabilities.

CNNs are trained using back propagation and gradient-based optimization algorithms. Their shared-weight and sparse connectivity structures make them highly efficient, particularly when dealing with high-dimensional inputs.

One of the key strengths of CNNs is their ability to learn relevant features directly from data, minimizing the need for manual feature extraction. As a result, CNNs have achieved remarkable success not only in image analysis but also in areas such as audio recognition, natural language processing, and medical diagnostics.

4.3.3 Recurrent Neural Network (RNN)

RNNs are a class of neural networks specifically designed to process sequential data. Unlike feed-forward architectures such as MLP or CNN, RNNs include recurrent connections that allow them to retain and use information from previous time steps. This recurrent structure makes them particularly suitable for tasks involving temporal or sequential patterns, such as time series forecasting, speech recognition, and natural language processing.

The core mechanism of a RNN is the propagation of a hidden state vector $\boldsymbol{h}_t$ that evolves over time as it processes each element of the input sequence $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T)$. At each time step t, the hidden state is updated according to:

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b})$$

where:

- $\boldsymbol{h}_{t-1}$ is the previous hidden state,
- $\boldsymbol{x}_t$ is the input at time step t,
- $\boldsymbol{W}_h$ and $\boldsymbol{W}_x$ are learnable weight matrices,
- $\boldsymbol{b}$ is a bias vector,
- σ is a non-linear activation function, typically TanH or ReLU.

Although RNNs are capable of modelling short-term dependencies, they struggle with learning long-range patterns due to the vanishing gradient problem. This limitation impairs their ability to maintain relevant information across many time steps, which led to the development of more advanced recurrent architectures. A schematic representation is shown in Figure 4.4.

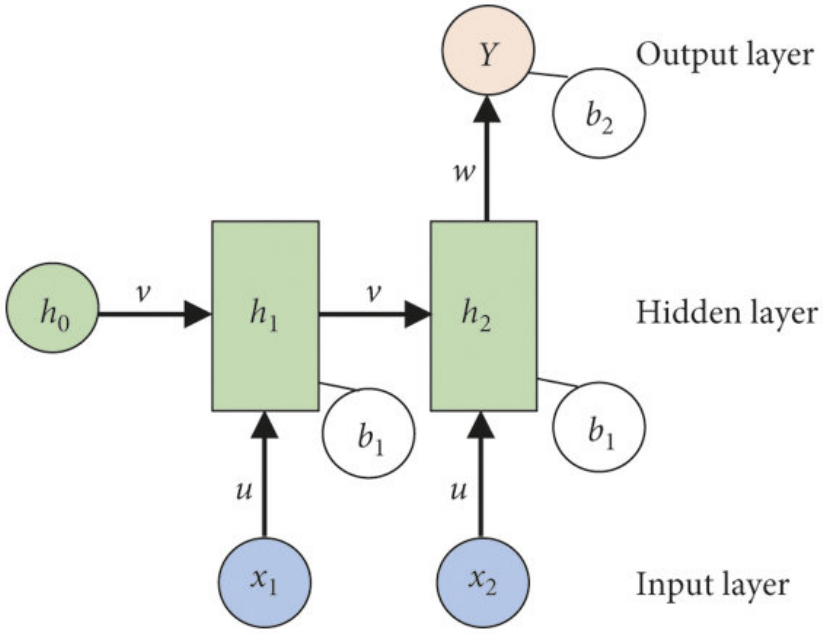


Figure 4.4: RNN architecture [4].

4.3.3.1 Long Short-Term Memory (LSTM)

LSTM networks were introduced to overcome the limitations of standard RNNs, particularly their difficulty in learning long-term dependencies due to vanishing or exploding gradients. LSTMs enhance the recurrent architecture by incorporating a memory cell and a set of gates that control the flow of information through time.

Each LSTM unit maintains a cell state $\boldsymbol{C}_t$ that is updated at every time step through carefully regulated interactions with three gates: the input gate, forget gate, and output gate. These gates allow the model to retain or discard information as needed. The internal operations of an LSTM cell are defined as follows:

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) && \text{(forget gate)} \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) && \text{(input gate)} \\
\tilde{\boldsymbol{C}}_t &= \tanh(\boldsymbol{W}_C[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C) && \text{(cell candidate)} \\
\boldsymbol{C}_t &= \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t && \text{(cell update)} \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) && \text{(output gate)} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t) && \text{(hidden state update)}
\end{aligned}$$

Here, $\odot$ denotes element-wise multiplication, and $\boldsymbol{W}$ and $\boldsymbol{b}$ are learnable parameters. This architecture allows LSTMs to preserve relevant information across long sequences while mitigating the

effect of irrelevant or noisy data. A schematic representation is shown in Figure 4.5.

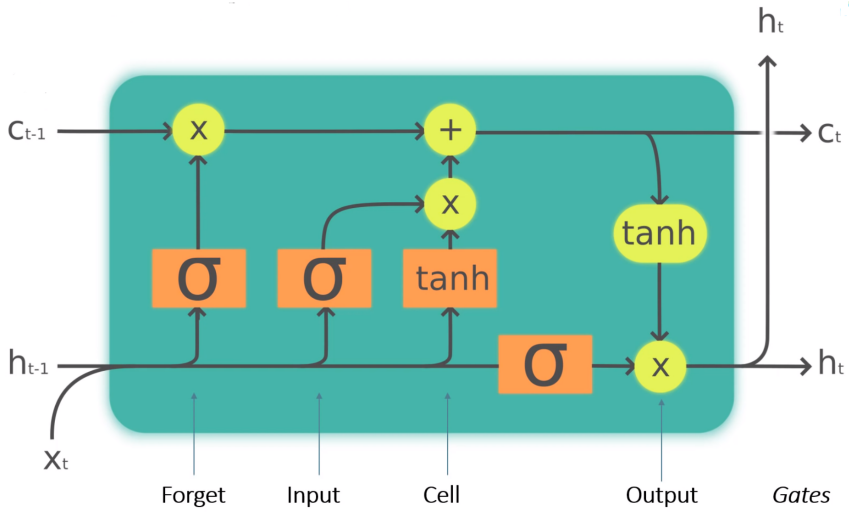


Figure 4.5: LSTM architecture [5].

LSTMs have achieved state-of-the-art results in a wide range of sequential modelling tasks, including machine translation, handwriting generation, video classification, and time series analysis. Despite their success, they can be computationally intensive and may require substantial data and tuning. Simplified alternatives such as Gated Recurrent Units (GRU) or transformer-based models have been proposed to address these limitations in specific contexts.

4.3.4 Graph Neural Network (GNN)

GNNs are a class of deep learning models specifically designed to operate on graph-structured data. Unlike traditional architectures that assume a regular input structure (e.g., vectors or grids), GNNs can handle arbitrary relational data by leveraging the connectivity between elements represented as nodes and edges. This capability makes them well-suited for applications in social networks, molecular chemistry, recommendation systems, and physical simulations.

Formally, a graph is defined as $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Each node $v_i \in V$ is associated with a feature vector $\boldsymbol{x}_i$, and edges may also include attributes such as weights or types. A schematic representation is shown in Figure 4.6.

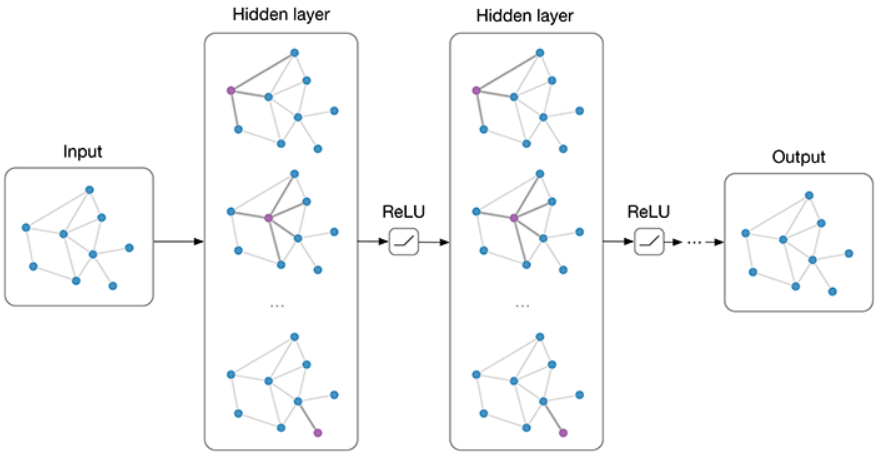


Figure 4.6: GNN architecture [6].

The core mechanism in GNNs is *message passing*, where each node iteratively updates its representation by aggregating information from its neighbours. This process generally involves two steps:

1. **Aggregation**: collecting feature information from a node's neighbourhood.
2. **Update**: combining this aggregated information with the node's own features to compute the new embedding.

This update is applied over multiple layers, allowing each node to gradually incorporate information from more distant parts of the graph.

4.3.4.1 Message Passing Neural Network (MPNN)

MPNNs are a general framework for graph neural networks that unify many existing architectures under a common message-passing paradigm [35]. During each layer, nodes exchange information with their neighbours through a two-step process: message aggregation and state update.

At message passing step t, each node v aggregates messages from its neighbourhood $\mathcal{N}(v)$ using a message function M_t:

$$\boldsymbol{m}_v^{(t+1)} = \sum_{u \in \mathcal{N}(v)} M_t \left(\boldsymbol{h}_v^{(t)}, \boldsymbol{h}_u^{(t)}, \boldsymbol{e}_{vu} \right)$$

Then, node features are updated via an update function U_t:

$$\boldsymbol{h}_v^{(t+1)} = U_t \left(\boldsymbol{h}_v^{(t)}, \boldsymbol{m}_v^{(t+1)} \right)$$

where:

- $\boldsymbol{h}_v^{(t)}$ is the feature of node v at layer t,
- $\boldsymbol{e}_{vu}$ represents the edge feature between nodes v and u,
- M_t and U_t are learnable functions, often implemented as neural networks.

After a fixed number of message passing steps, a final representation is computed using a permutation-invariant *readout* function R:

$$\boldsymbol{y} = R\left(\left\{\boldsymbol{h}_v^{(T)} \mid v \in G\right\}\right)$$

This framework is highly flexible and forms the basis for many popular architectures. For example, Graph Convolutional Network (GCN), Graph Attention Network (GAT) and Graph SAmple and aggreGatE (Graph SAGE) can all be viewed as special cases of MPNN with specific message and update definitions. MPNNs are particularly effective in tasks requiring complex node interactions, such as molecular property prediction or physical simulations.

4.3.4.2 Graph Network Simulator (GNS)

The Graph Network Simulator (GNS) is a learned, graph-based framework designed to model and simulate complex physical systems over time using the principles of GNNs. Instead of relying on hand-crafted equations or computationally expensive numerical solvers (e.g., finite element or finite volume methods), a GNS learns the underlying dynamics directly from data. This allows it to approximate time-dependent physical behaviours with high efficiency and generalization capabilities [36].

In a typical GNS, the physical system is represented as a graph:

- **Nodes** represent discrete elements of the system, such as particles, mesh points, or components of a structure.
- **Edges** capture interactions between entities, such as forces, thermal conductance, or constraints.
- **Node and edge features** include physical quantities like mass, position, velocity, pressure, or temperature.

At each time step, the GNS applies a message-passing mechanism to update the state of each node based on its own properties and those of its neighbours. The updated node features are then interpreted as the system's state at the next time step. This process is repeated to generate full trajectories of the system's temporal evolution.

Mathematically, the forward pass of a GNS follows the structure of a message-passing neural network:

$$\boldsymbol{m}_{ij} = \phi_e(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{e}_{ij}), \quad \boldsymbol{m}_i = \sum_{j \in \mathcal{N}(i)} \boldsymbol{m}_{ij}, \quad \boldsymbol{h}'_i = \phi_v(\boldsymbol{h}_i, \boldsymbol{m}_i)$$

where $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are node features, $\boldsymbol{e}_{ij}$ are edge features, ϕ_e and ϕ_v are learnable neural networks, and $\boldsymbol{h}'_i$ is the updated node representation.

GNS has shown strong performance across a wide range of physical systems, including:

- Fluid and gas dynamics.
- Particle-based systems (e.g., granular flows).
- Soft-body deformation and elasticity.
- Transient heat propagation and coupled multi-physics problems.

Because it is fully data-driven and structured as a GNN, GNS can generalize to unseen geometries, boundary conditions, and physical regimes. This makes it especially appealing for applications where fast surrogate models of physical simulations are needed, such as real-time prediction, control systems, or design optimization.

4.3.4.3 Graph Convolutional Network (GCN)

GCN are a foundational class of GNN that extend the concept of convolution from regular Euclidean domains (like images or sequences) to irregular, non-Euclidean domains such as graphs. They generalize the convolution operation to graph domains by performing neighbourhood averaging in the spectral domain. The layer-wise update rule for a GCN, as introduced by Kipf and Welling [37], is:

$$\mathbf{h_i^{(l+1)}} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{d_i d_j}} \mathbf{W^{(l)}} \mathbf{h_j^{(l)}}\right)$$

where:

- $\boldsymbol{h}_i^{(l)}$ is the node feature at layer l.
- d_i is the degree of node i.
- $\mathcal{N}(i)$ is the set of neighbouring nodes.
- $\boldsymbol{W}^{(l)}$ is a learnable weight matrix.
- σ is a non-linear activation function (e.g., SiLU).

GCNs are computationally efficient but assume uniform weighting of neighbours and are less suitable for very large or dynamic graphs.

A model representation is shown in Figure 4.7. In the GCN network, the self-loop is automatically generated.

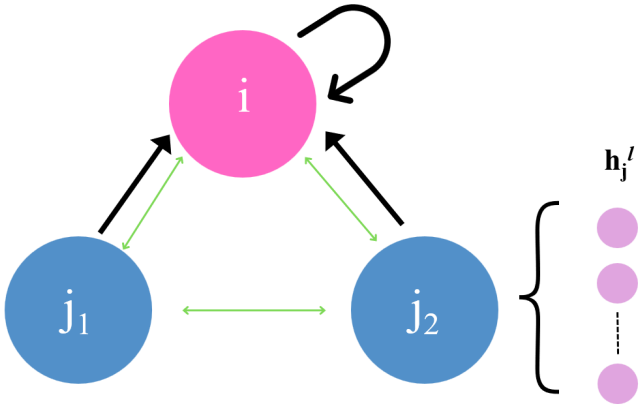


Figure 4.7: GCN model representation.

Where $\mathbf{h_i^{(0)}}$ has a shape of *input dimension* x 1. This input dimension varies between 3, for the PCB 2D problem, and 2, for the 3D model.

4.3.4.4 Graph Attention Network (GAT)

GATs introduce attention mechanisms to learn the relative importance of a node's neighbours [38]. Each neighbour contributes to the target node's update with a learned attention weight α_{ij}:

$$\mathbf{h_i^{(l+1)}} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)} \mathbf{W^{(l)}} \mathbf{h_j^{(l)}}\right)$$

The attention coefficients $\alpha_{ij}^{(l)}$ are computed using:

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\mathbf{W^{(l)}}\mathbf{h_i^{(l)}} \| \mathbf{W^{(l)}}\mathbf{h_j^{(l)}}]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T[\mathbf{W^{(l)}}\mathbf{h_i^{(l)}} \| \mathbf{W^{(l)}}\mathbf{h_k^{(l)}}]\right)\right)}$$

GATs can model heterogeneity in the graph structure and focus more on the most informative neighbours. They are especially powerful in noisy or unstructured graphs.

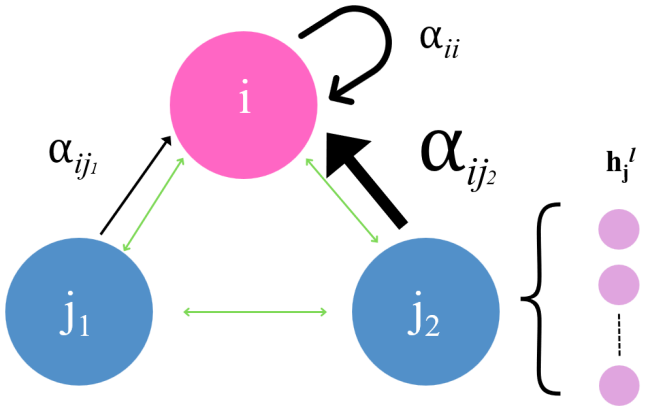


Figure 4.8: GAT model representation.

However, for the GAT network, as the connectivity index has no self-loops, the α_{ii} which is shown in Figure 4.8 is null.

4.3.4.5 Graph SAmple and aggreGatE (Graph SAGE)

Graph SAGE is an inductive framework for learning node embeddings in large-scale or evolving graphs. It works by sampling a fixed-size set of neighbours and aggregating their features using a predefined function [37]:

$$\mathbf{h_i^{(l+1)}} = \sigma\left(\mathbf{W^{(l)}} \cdot \text{AGGREGATE}^{(l)}\left(\left\{\mathbf{h_j^{(l)}}, \forall j \in \mathcal{N}(i)\right\} \cup \mathbf{h_i^{(l)}}\right)\right)$$

Typical aggregation functions include:

- **Mean aggregator**: Average over neighbours. This is the one selected for the Graph SAGE network architecture used.
- **LSTM aggregator**: Sequence-based processing of neighbours.
- **Pooling aggregator**: Apply a neural network to each neighbour and take the element-wise max.

With the mean aggregator function, the mathematical formula is the following:

$$\mathbf{h_i^{(l+1)}} = \sigma\left(\mathbf{W_1 h_i^{(l)}} + \mathbf{W_2} \cdot \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{h_j^{(l)}}\right)$$

Graph SAGE is particularly well-suited for tasks that require generalization to unseen nodes or parts of the graph, such as recommendation systems and streaming networks.

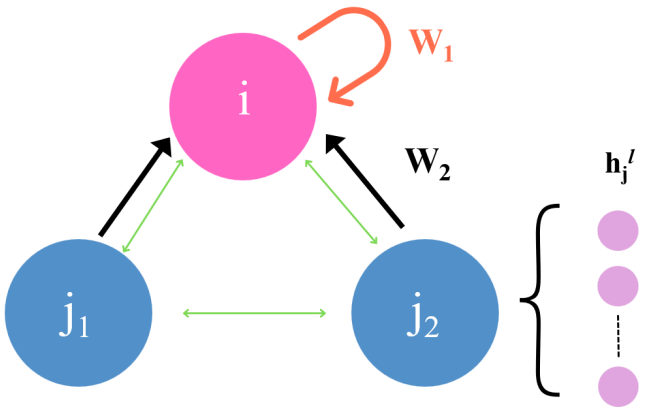


Figure 4.9: SAGE model representation.

However, for the Graph SAGE network, as the connectivity index has no self-loops, the $\mathbf{W_1}$ which is shown in Figure 4.9 is a null matrix.

4.3.4.6 Neural Network Convolution (NNConv)

NNConv is a message-passing-based graph convolution operator introduced as part of the MPNN framework [35]. Unlike GCN, GAT, or Graph SAGE, which use fixed or simple aggregation schemes, NNConv dynamically generates the aggregation weights using a neural network conditioned on edge features. This enables the model to learn complex, edge-dependent relationships in the message-passing process.

The layer-wise update rule for NNConv is given by:

$$\mathbf{h_i^{(l+1)}} = \sigma\left(\frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} \mathbf{\Phi_e(e_{ij})}\cdot \mathbf{h_j^{(l)}}\right)$$

where:

- $\boldsymbol{h}_j^{(l)}$ is the feature vector of neighbour node j at layer l,
- $\boldsymbol{e}_{ij}$ is the edge feature vector between nodes i and j,
- ϕ_e is a neural network (typically a MLP) that maps edge features to a weight matrix,
- σ is a non-linear activation function, such as ReLU.

As the connectivity index has no self-loops, the $\mathbf{\Phi_e(e_{ii})}$ as shown in Figure 4.10 is a null matrix, with $\mathbf{e_{ii}}$ being a null vector.

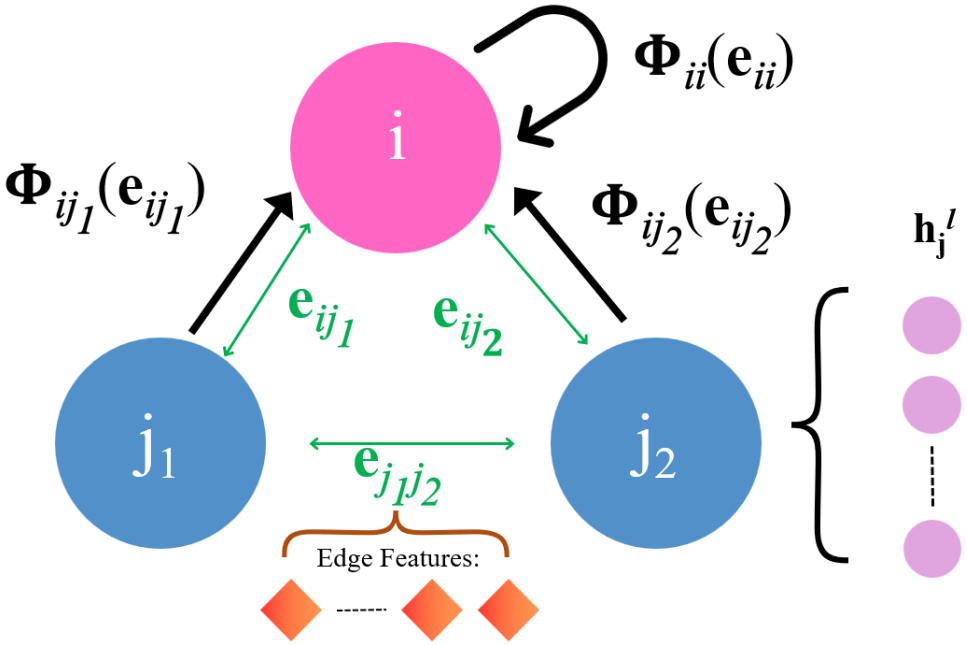


Figure 4.10: NNConv model representation.

The way in which the NNConv layer generates each Φ_e from the edge features is shown schematically in Figure 4.11, where $\mathbf{p}^{(l+1)}$ denotes the size of the feature vector at each node in the next layer, and $\mathbf{p}^{(l)}$ represents the size of the node features in the current layer.

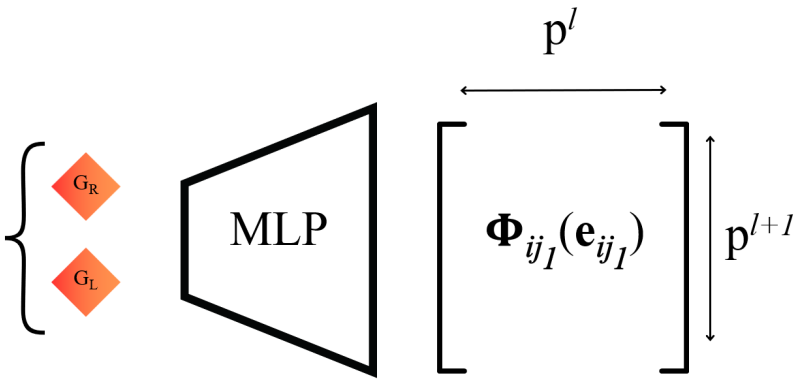


Figure 4.11: Generation of edge matrixes on NNConv.

Unlike traditional convolution operators where the weights are shared across all edges, NNConv allows each edge to produce a custom filter based on its features. This makes it particularly suitable for problems involving rich edge attributes, such as spatial distances, conductances, or material properties [35].

NNConv has shown strong performance in scientific applications where the interactions between nodes are governed by continuous physical quantities. It has been successfully applied to molecular property prediction, particle-based physics simulations, and thermal modelling tasks where edge-level interactions (e.g., thermal conductance) are critical.

4.4 Challenges and Limitations of Artificial Neural Networks

Despite their remarkable success in a wide range of machine learning tasks, ANNs present several well-documented challenges and limitations. Understanding these issues is essential for developing models that are both robust and reliable. This section outlines the most common problems that practitioners may encounter when working with ANNs and provides practical advice on how to mitigate them.

1. **Over-fitting:** Neural networks, especially those with many parameters, are prone to over-fitting the training data. This occurs when the model memorizes specific examples rather than learning generalizable patterns, resulting in poor performance on unseen data. Techniques such as dropout, regularization, data augmentation, and early stopping are commonly used to mitigate this effect [39].

2. **Vanishing and Exploding Gradients:** During back propagation, gradients can become extremely small (vanishing) or excessively large (exploding), particularly in deep architectures. Vanishing gradients slow or halt learning in early layers, while exploding gradients can destabilize training. Solutions include proper weight initialization [40], gradient clipping, normalization layers, and using activation functions such as ReLU.

3. **High Computational Cost:** Training deep neural networks is computationally intensive. It demands significant memory, processing power, and time, often requiring specialized hardware such as GPUs or TPUs. This limitation can restrict the usability of neural networks in resource-constrained environments [29].

4. **Lack of Interpretability:** Neural networks are often described as "black-box" models because their internal decision-making processes are difficult to interpret. This lack of transparency is problematic in sensitive applications such as healthcare, finance, and autonomous systems, where understanding the rationale behind a decision is critical [41].

5. **Data Dependence:** Neural networks generally require large amounts of labelled data to perform well. In domains with limited or imbalanced datasets, the network may struggle to

learn meaningful patterns. Additionally, noisy or mislabelled data can significantly degrade performance [42].

6. **Sensitivity to Input Perturbations:** Neural networks can be highly sensitive to small changes in input data. In particular, adversarial examples, inputs with subtle, often imperceptible modifications, can lead to confident but incorrect predictions. This vulnerability raises concerns in safety-critical applications such as autonomous driving or cybersecurity [43].

7. **Hyper-parameter Sensitivity:** Neural networks involve numerous hyper-parameters, including learning rate, batch size, network depth, and regularization strength. Finding optimal settings requires extensive experimentation and often relies on techniques like grid search, random search, or Bayesian optimization [44].

8. **Catastrophic Forgetting:** In continual or lifelong learning scenarios, neural networks tend to forget previously learned tasks when trained on new ones. This phenomenon, known as catastrophic forgetting, presents a major obstacle for systems intended to learn incrementally over time [45].

4.4.1 Delving into hyper-parameter sensitivity

As previously discussed, the choice of hyper-parameters plays a pivotal role in determining whether an ANN becomes an accurate and efficient model or an ineffective use of computational resources. Since these parameters are typically chosen by practitioners, this stage is particularly prone to human error. This is largely due to the fact that hyper-parameters often interact in complex and unintuitive ways, making their tuning a non-trivial task.

Moreover, given the inherent lack of interpretability in ANNs, even for experts in ML, it is often difficult to predict how a particular change in configuration will affect the model's behaviour. Selecting appropriate hyper-parameter values is therefore essential to achieve both good generalization and convergence. However, manual tuning is not only challenging but also time-consuming and inefficient, especially when dealing with large and high-dimensional search spaces.

To address these challenges, a variety of automated hyper-parameter optimization techniques have been developed. These methods aim to systematically explore the search space and identify high-performing configurations while minimizing computational cost.

- **Grid Search and Random Search:** Traditional baseline approaches that evaluate models on a fixed grid or a set of randomly sampled hyper-parameter combinations. While easy to implement, they scale poorly with the dimensionality of the search space and do not incorporate knowledge from previous trials [44].

- **Bayesian Optimization:** A probabilistic, model-based approach that builds a surrogate function (commonly a Gaussian Process or Tree-structured Parzen Estimator) to approximate

the objective function. It then selects the next set of hyper-parameters to evaluate based on an acquisition function (e.g., Expected Improvement), balancing exploration and exploitation. Bayesian optimization is more sample-efficient than random search and is well-suited to expensive training procedures [46].

- **Hyperopt:** A Python library implementing Bayesian optimization using the Tree-structured Parzen Estimator (TPE) algorithm. It supports conditional search spaces and has been widely adopted for deep learning experiments, especially when combined with distributed computing frameworks [47].
- **Optuna:** A modern hyper-parameter optimization framework designed for flexibility and efficiency. Optuna employs a define-by-run interface, allowing dynamic search space definitions, and supports early stopping through pruning unpromising trials. It also facilitates distributed optimization and integration with deep learning libraries such as PyTorch and TensorFlow. Its combination of speed, simplicity, and support for complex search strategies makes it particularly attractive for academic and industrial research [48].

These methods enable researchers to automate the search for optimal configurations, reduce human bias in model selection, and significantly improve model performance and training efficiency. In practical machine learning workflows, hyper-parameter optimization is now considered a standard step, often consuming as much time as the training itself.

5

Objectives and Scope of the Thesis

This chapter outlines the main goals and the scope of the present Bachelor's Thesis. The work aims to explore the application of GNNs to thermal modelling problems by leveraging the structural similarities between graphs and the discretization schemes used in classical thermal analysis methods such as the LPTM, FEM and the CFD. Building upon existing work with CNNs, the study proposes adapting the approach to GNN architectures, which are better suited to the topological nature of thermal models. The objectives span from adapting and optimizing the architecture to validating its performance on both 2D and 3D synthetic datasets derived from dedicated thermal solvers. Additionally, the thesis investigates the amount of data required to achieve accurate predictions with the best-performing model.

- The scope of this Bachelor's Thesis is to take advantage of the similarities between graphs and the way LPTM and FEM models are discretized and solved, as it happens with ESATAN models.
- Adapt the CNN thermal modelling developed by A. Barbosa [49] to a more suitable architecture that reassembles to the way thermal models are discretized such as GNNs.
- Optimize and train different GNN architectures with synthetic data provided by a PCB thermal model solver designed on python.
- Once the feasibility of Graph Neural Network (GNN) has been tested on this 2D problem, the same GNN architectures will be tested on a full 3D model of a satellite. Beginning with a simplified model to first demonstrate that the architectures are capable of learning complex connections between nodes. This testing will also be done with synthetic data provided from an ESATAN model solver developed on python.

- To conclude, a study of how much synthetic data does the best architecture need for accurately solve the problem will be done

6

Comparison of several GNN architectures solving a steady 2D PCB

This thesis is a continuation of the work developed by Alejandro Barbosa in his bachelor's thesis [49], where a CNN was implemented to solve the same two-dimensional PCB thermal problem addressed here. In his work, a convolutional autoencoder was also tested, achieving a RMSE of 0.18 K, demonstrating promising performance for this task.

In Barbosa's thesis, it has been explored the feasibility of using neural networks as surrogate models to replace traditional thermal solvers in predicting temperature distributions in space systems, specifically in PCBs. CNNs have shown the ability to produce results comparable to theoretical models, while significantly reducing processing time, making them a promising tool for real-time applications.

The implementation of Physics-Informed Neural Network (PINN) has further improved prediction accuracy by incorporating physical principles directly into the training process. This approach not only reduces the number of training samples required but also ensures that the predictions remain physically consistent.

Additionally, the use of autoencoders for dimensionality reduction of the temperature distributions has been evaluated. Autoencoders have proven effective in compressing and reconstructing data with high fidelity and have also been successfully applied to predict temperature distributions given boundary conditions. Compared to Singular Value Decomposition (SVD), optimized autoencoders have the potential to outperform SVD by capturing non-linear relationships in the data.

His work has also addressed the ability of neural networks to perform fine-tuning, allowing them to adapt to new experimental data using a reduced number of training samples. This strategy has

proven effective, demonstrating that pre-trained networks can be adjusted to new conditions with limited additional data while maintaining acceptable performance.

In this thesis, several GNN architectures will be tested to see if these architectures are suitable for solving thermal problems. The reasons behind choosing GNNs rather than delving deeper on CNNs with autoencoders and fine-tuning is because the main scope is to solve real 3D models, what is impossible for CNNs and MLPs given the non-regular geometry of real 3D models.

6.1 Thermal Model PCB

The thermal model used in this work allows calculating the steady-state temperature distribution for a 13×13 node PCB plate, given a specific set of boundary conditions. This model is employed to generate a dataset consisting of 30,000 synthetic steady state cases used to train various Graph Neural Network (GNN) architectures [49]. For each case, the temperatures of the four interface nodes, located at the corners of the plate and in contact with the satellite structure, are randomly selected within the range of 260 K to 310 K. Similarly, the environment temperature, used solely for radiative calculations, is also randomly chosen within the same range. Additionally, the model includes four heaters with randomly assigned dissipation powers ranging from 0.1 to 1.25 W. This setup enables the simulation of a wide variety of thermal scenarios, whether to reflect different operating conditions of a fixed component layout or to represent different PCB configurations altogether. Once these parameters are specified, the thermal model outputs a temperature distribution such as the one shown in Figure 6.1.

Given the moderate size of the domain, the neural network architectures designed for this problem do not require excessive complexity. The relatively low number of nodes and simpler spatial features result in shorter training times and allow for the use of smaller-scale models. Training was carried out on a laptop equipped with an RTX 2060 GPU, and all reported performance metrics, such as average time per epoch, are referenced to this hardware configuration.

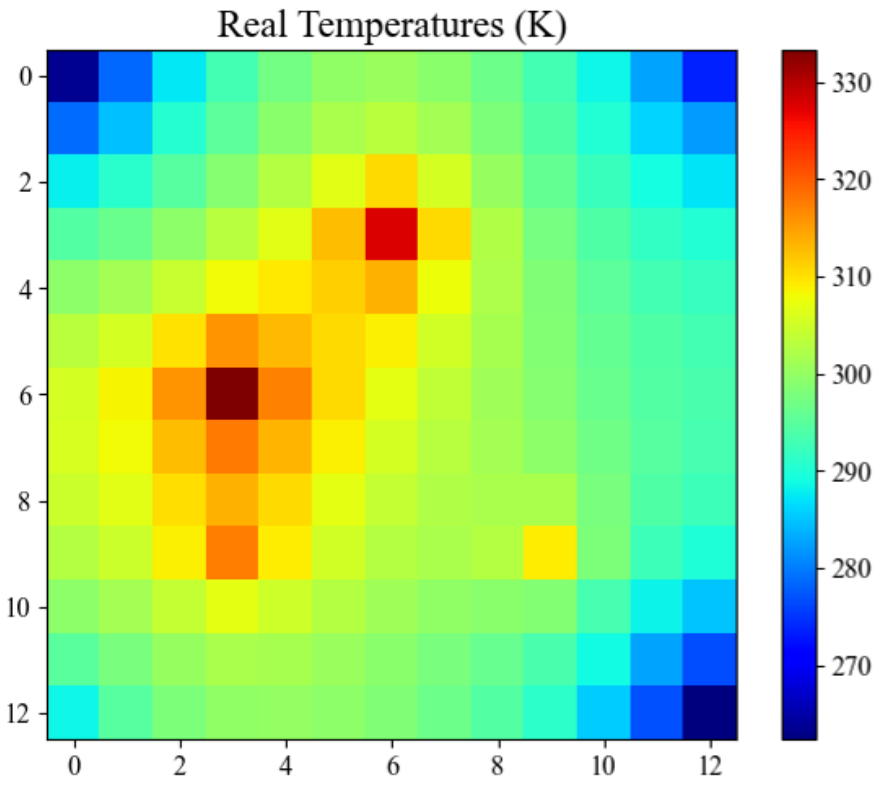


Figure 6.1: Temperature distribution of a PCB with random boundary conditions.

6.1.1 Model Simplifications

To simplify the model, certain assumptions are made:

One-dimensional Thermal Conduction

It is assumed that the heat conduction in the PCB is one-dimensional, which significantly simplifies the calculations. This model allows each node to exchange heat only with its neighbouring nodes, storing all temperature information in a single vector of 169 components (13x13). The labelling of each node in a sample 6x5 plate is as follows [49]:

	25	26	27	28	29
	20	21	22	23	24
	15	16	17	18	19
	10	11	12	13	14
↑	5	6	7	8	9
y	0	1	2	3	4
	x →				

Radiative Environment

It is assumed that the PCB is completely surrounded by a single environment node, implying a view factor to the environment of $F_{p,a} = 1$. Furthermore, the environment is treated as a blackbody at a temperature $T = T_{env}$. It is also assumed that the nodes only exchange heat radiatively with the environment, not with each other.

Material Properties and Dimensions

The thermal properties of the PCB material are constant and homogeneous throughout the plate. Additionally, the length and thickness of the plate do not change:

- Side length: $L = 0.1$ m
- Thickness: $e = 0.001$ m
- Thermal conductivity: $k = 10$ W/(m*K)
- Emissivity: $\varepsilon = 0.8$

6.1.2 Boundary Conditions

The boundary conditions implemented in the model are as follows:

Power Dissipated by Electronic Components

Throughout the work, four electronic components are considered, dissipating power within the range of 0.1 to 1.25 W. Their positions may vary as required, but for all the training, the 4 heaters were displayed in the same configuration as shown in Figure 6.2.

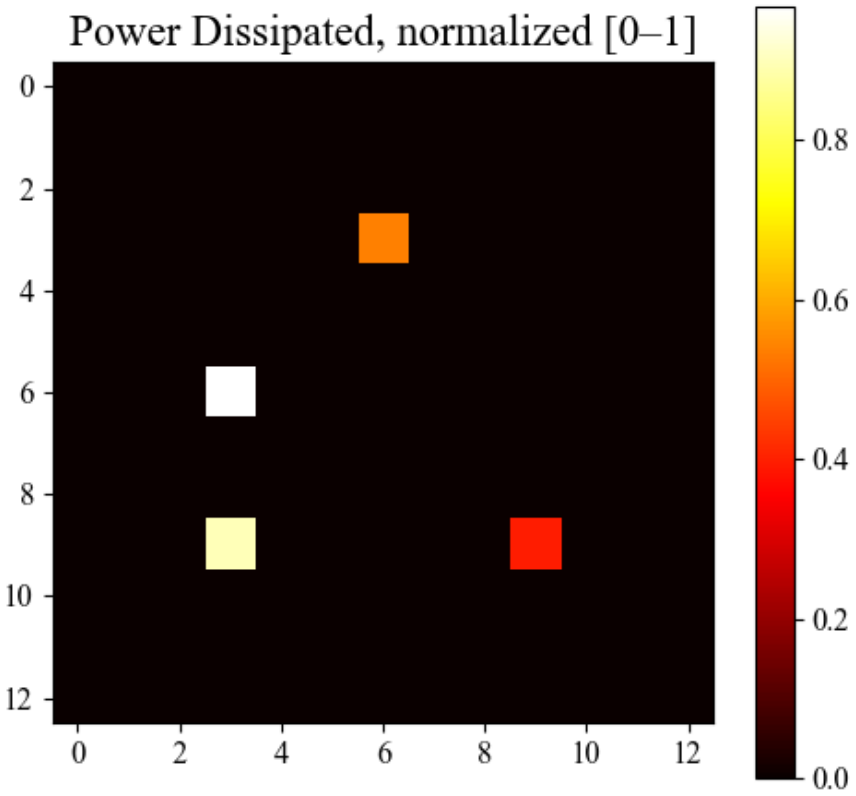


Figure 6.2: Location of the power dissipation in a regular 13×13 plate.

Interface Temperatures

The PCB interfaces with the rest of the satellite is considered to be located at the corners, as shown in Figure 6.3, with a fixed and constant temperature. Each corner may be at a different temperature from the others, and during the entire process, they are constrained between 260 and 310 K.

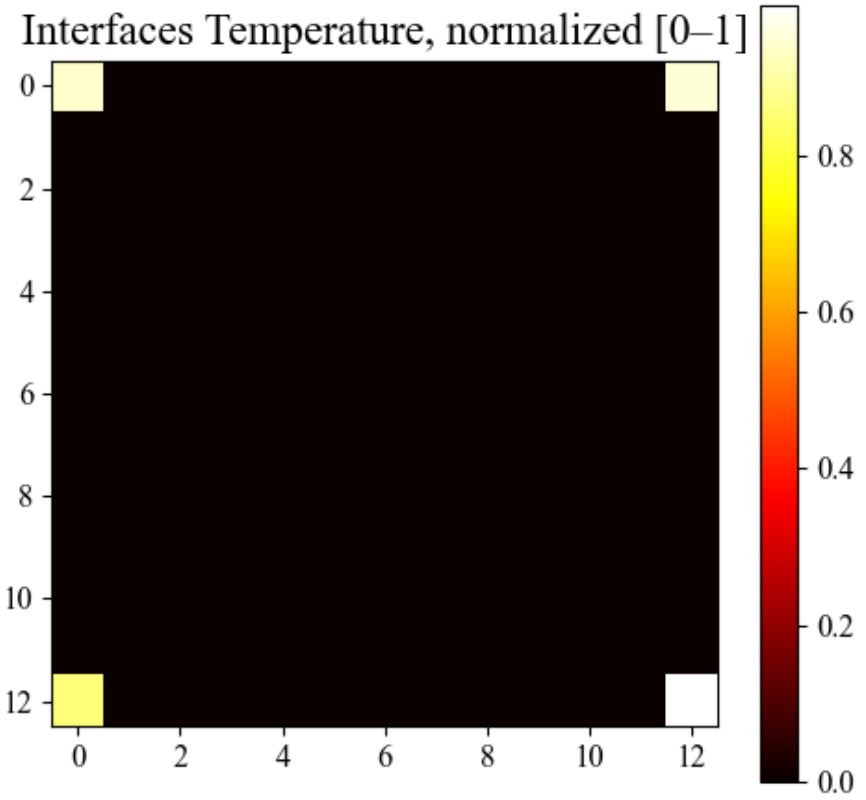


Figure 6.3: Example of interface temperatures located at the corners.

Environment Temperature

The environment temperature directly affects the heat radiated. It remains constant throughout the iteration process and is constrained between 260 and 310 K.

6.1.3 Lumped Parameter Method

The Lumped Parameter Method is used to calculate the temperature distribution on a PCB. This method simplifies thermal modelling by representing the PCB as a set of discrete nodes, each with its corresponding thermal capacitance and thermal resistances connecting the nodes.

For transient analysis, it is necessary to solve a system of ordinary differential equations (ODEs) that describe how the temperature in each node changes over time. This system of ODEs accounts for the temporal variations in temperature due to internal heat sources and external environmental conditions.

On the other hand, in the steady-state case, where the temperature distribution at equilibrium is sought, a system of non-linear equations must be solved. These non-linear equations arise from terms that represent thermal radiation, which depends on the fourth power of the temperature.

This is the case studied in the present work.

The steady state problem is solved using the energy balance equation:

$$\mathbf{K} \cdot \vec{T} + \sigma \cdot \mathbf{E} \cdot (\vec{T}^4 - T_{env}^4) = \vec{Q}$$

where $\mathbf{K}$ is the conductivity coupling matrix of all nodes, $\mathbf{E}$ is the radiative coupling matrix, $\sigma = 5.67 \times 10^{-8}$ is the Stefan-Boltzmann constant, $\vec{Q}$ is the vector of heat absorbed by each node, and $\vec{T}$ is the vector of temperatures we aim to obtain.

6.1.4 Heat Flux Vector

To construct this vector, it is necessary to introduce the dissipated powers by each electronic component at its corresponding position. Additionally, to apply the boundary conditions for the interface temperatures, the temperatures at the interfaces must be incorporated at the positions of the corner nodes. This is because, subsequently, a 1 is inserted into these positions in the conductivity coupling matrix, so that when multiplying the vector $\vec{T}$ by the matrix $\mathbf{K}$, the temperature will always match the boundary condition temperature:

$$1 \cdot \vec{T_0} = T_{interfaces}$$

To satisfy this equation, the rows corresponding to these positions in the radiative coupling matrix are nullified, ensuring they do not influence the result. For example, if an electronic component dissipates 5 W of power at node 6, and at positions 0 and 4 there are interfaces with a temperature of 300 K, the heat flux vector would be expressed as:

$$\vec{Q} = \begin{pmatrix} 300 \\ 0 \\ 0 \\ 0 \\ 300 \\ 0 \\ 5 \\ \vdots \end{pmatrix}$$

6.1.5 Conductivity Coupling Matrix

To obtain the matrix $\mathbf{K}$, it is first necessary to calculate the conductances of the plate G_L, which in this case are constant and homogeneous due to the geometric symmetry of the nodes:

$$G_L = \frac{kA}{L} = e \cdot k$$

where e is the thickness of the PCB. The matrix can be classified as a correlation matrix between nodes and their neighbours, where each row corresponds to a node. In each row, the conductances between the node and its neighbours are indicated, and the diagonal of the matrix contains the sum of all conductances connected to that node.

$$\mathbf{K} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ -G_L & \sum G_L & -G_L & 0 & 0 & 0 & -G_L & \cdots & 0 \\ 0 & -G_L & \sum G_L & -G_L & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -G_L & \sum G_L & -G_L & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ -G_L & 0 & 0 & 0 & 0 & \sum G_L & -G_L & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & 1 \end{pmatrix}$$

6.1.6 Radiative Coupling Matrix:

To obtain the matrix E, it is necessary to calculate the radiative conductances G_R:

$$G_R = 2\Delta x \cdot \Delta y \cdot \varepsilon = 2l^2\varepsilon$$

where the factor of 2 accounts for the two faces of the PCB, and Δx and Δy represent the space between each node. In this case:

$$l = \Delta x = \Delta y = \frac{L}{13}$$

Since each node only exchanges heat with the environment, the resulting matrix is simpler, as it only contains the G_R values on the diagonal, except where the interface boundary conditions must be fulfilled, represented by rows of zeros:

$$\mathbf{E} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & G_R & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & G_R & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & G_R & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & G_R & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & G_R & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & 0 \end{pmatrix}$$

6.1.7 Equation Resolution

To avoid solving the non-linear system of equations, a fixed-point iteration method is used:

$$\mathbf{A} \cdot \Delta\vec{T_i} = \vec{b}$$

where:

$$\mathbf{A} = \mathbf{K} + 4\sigma \cdot \mathbf{E} \cdot \vec{T_i}^3$$

$$\vec{b} = \vec{Q} - \mathbf{K} \cdot \vec{T_i} - \mathbf{E} \cdot (\vec{T_i}^4 - T_{env}^4)$$

This equation is solved to obtain $\Delta\vec{T_i}$, allowing the cycle to be completed:

$$\vec{T_{i+1}} = \vec{T_i} + \Delta\vec{T_i}$$

The loop repeats until $\Delta\vec{T_i} < 0.01$ or until 1000 iterations are reached, in which case convergence is considered not achieved.

6.2 Insights of some decisions taken

The reason behind using a Dataset with 30,000 cases lies on an study made in the GCN network in which the evolution of the MSE was evaluated training the network for 100 epochs. In this study it was determined that increasing greatly the number of cases does not correlate with such a big reduction on MSE, as it is shown in Figure 6.4, and it really increases the time needed for training. To identify better the evolution of the MSE, as it is shown in Figure 6.5, the X axis was truncated. So on, selecting 30,000 cases is also because the previous research done to solve this problem with

a CNN trained the network with such amount of data, so in order to see how well it performed compared to a CNN, we need to train it on the same number of cases [49].

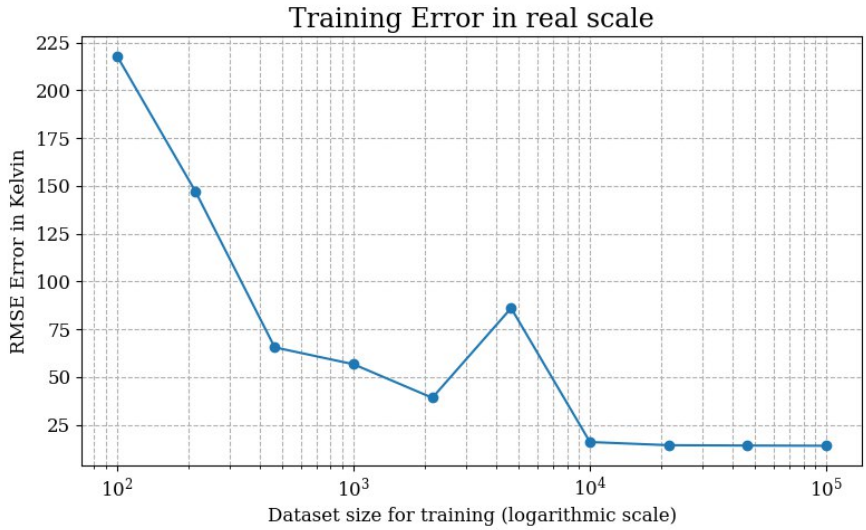


Figure 6.4: Evolution of the MSE of a GCN 2D model with the number of cases.

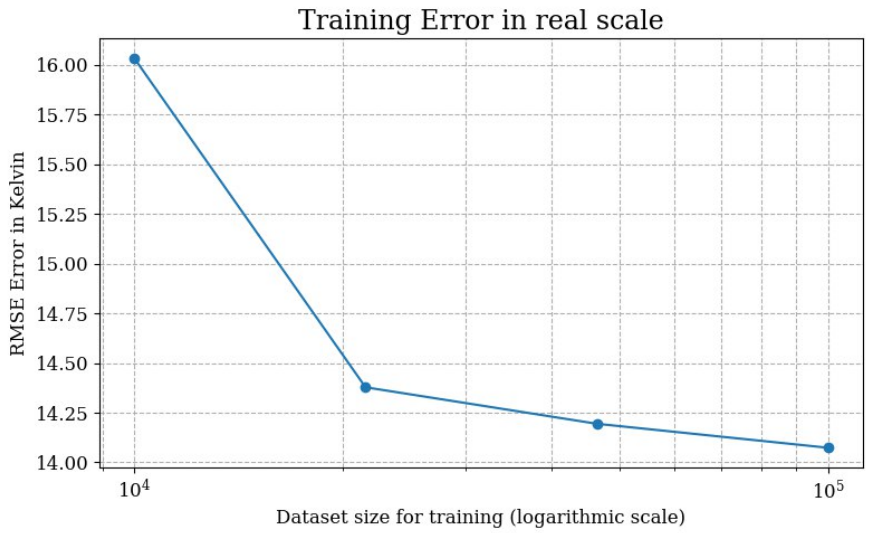


Figure 6.5: Evolution of the MSE of a GCN 2D model with the number of cases, truncated.

In order to get better results and avoid problems such as gradient vanishing or exploding gradients, the three inputs were normalised with their respective maximum value from all the Dataset. Furthermore, batch normalization, dropout and residual were used to obtain better results and avoid the problems presented above.

Residual Connections

Residual connections are a technique used to improve the training of deep neural networks by allowing the model to learn residual functions with respect to the layer inputs. Instead of learning a direct mapping $\mathcal{F}(x)$, the network learns a residual mapping $\mathcal{F}(x) = H(x) - x$, such that the final output becomes:

$$H(x) = \mathcal{F}(x) + x$$

This shortcut connection enables the propagation of gradients across many layers, helping to alleviate the vanishing gradient problem and facilitating the training of deep architectures [50].

Batch Normalization

Batch normalization is a regularization and optimization technique that normalizes the input of each layer to have zero mean and unit variance across the current mini-batch. Given a mini-batch $\{x_1, x_2, ..., x_m\}$, the normalized output is computed as:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where μ_B and σ_B^2 are the mean and variance of the batch, and ϵ is a small constant for numerical stability. Batch normalization accelerates training, reduces sensitivity to weight initialization, and can act as an implicit regularizer [51].

Dropout

Dropout is a regularization technique designed to prevent over-fitting by randomly disabling a subset of neurons during each training iteration. For a given layer, a binary mask $\boldsymbol{r} \sim \text{Bernoulli}(p)$ is applied to the activations:

$$\tilde{\boldsymbol{h}} = \boldsymbol{r} \odot \boldsymbol{h}$$

where $\boldsymbol{h}$ is the vector of activations and $\odot$ denotes element-wise multiplication. During inference, all neurons are used, and their outputs are scaled by the dropout probability p to maintain consistency with training [39].

Activation function

The activation function used was the SiLU function, due to the fact that this function avoids the problem of `dead` neurons, which happens when using ReLU. The SiLU function introduces non-linearity and prevents vanishing gradient for positive values, focusing on smoothness and computational efficiency [52].

6.3 Graph Convolutional Network

Following an initial investigation into the various existing GNN architectures, the first model selected was based on the MPNN framework. Specifically, a GCN was chosen due to its relatively simple structure, making it a suitable starting point for addressing the two-dimensional PCB thermal problem. In this architecture, node feature updates during forward propagation depend on the features of the node itself and its neighbouring nodes. However, a notable limitation of the GCN is its inability to natively incorporate edge features. While it is possible to manually assign weights to edges, representing the relative importance of each neighbouring node, this process requires human input, which can introduce bias or errors and potentially lead to suboptimal predictions.

6.3.1 Hyper-parameter optimization for GCN and results obtained

To better understand and explore optimal configurations, Optuna was used. This library is very useful for ANN optimizations. Although, this requires a lot of computational resources and to find a solution it need several cases, so it requires even more time than training or developing the network itself.

Number of Layers

When selecting the number of layers in an MPNN, it is essential to understand how these networks propagate information: the number of layers directly determines the maximum distance over which nodes can exchange features. Therefore, choosing more than 24 layers is not particularly meaningful in this case, as the maximum distance between any two nodes on the 13×13 PCB plate is 24, and the influence of one corner on the opposite corner is minimal. However, increasing the number of layers also allows the network to learn more complex patterns, which can improve predictive performance. This introduces a trade-off between network depth and practical relevance. To explore this, a GCN was trained for 100 epochs using 30,000 graphs from the dataset, and the variation of MSE with respect to the number of layers was analysed, as shown in Figure 6.6.

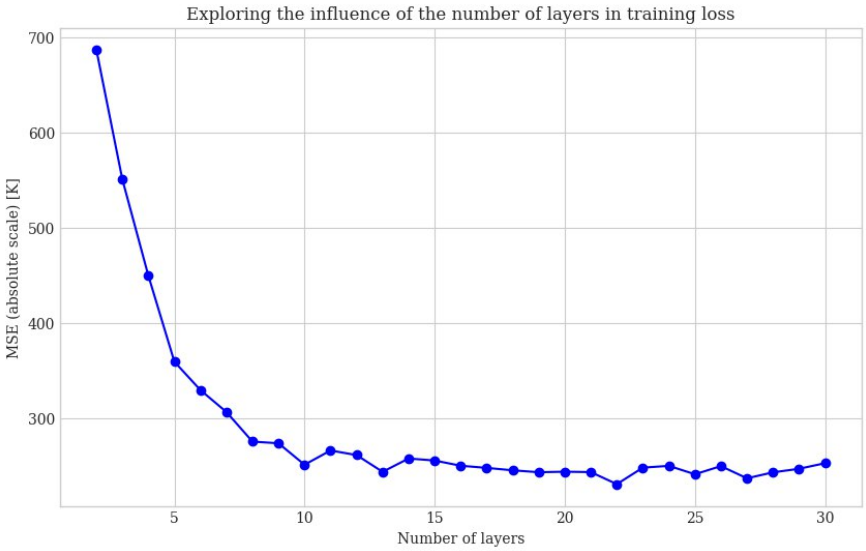


Figure 6.6: Evolution of the MSE with the number of layers in GCN 2D architecture.

As shown in Figure 6.6, the MSE reaches a point beyond which it no longer decreases, indicating no apparent benefit in using more than 13 layers. This behaviour suggests that the ANN becomes excessively complex for the given problem, limiting its ability to learn meaningful patterns and potentially leading to over-fitting. To better highlight the performance threshold of the GCN, the axes were truncated in Figure 6.7.

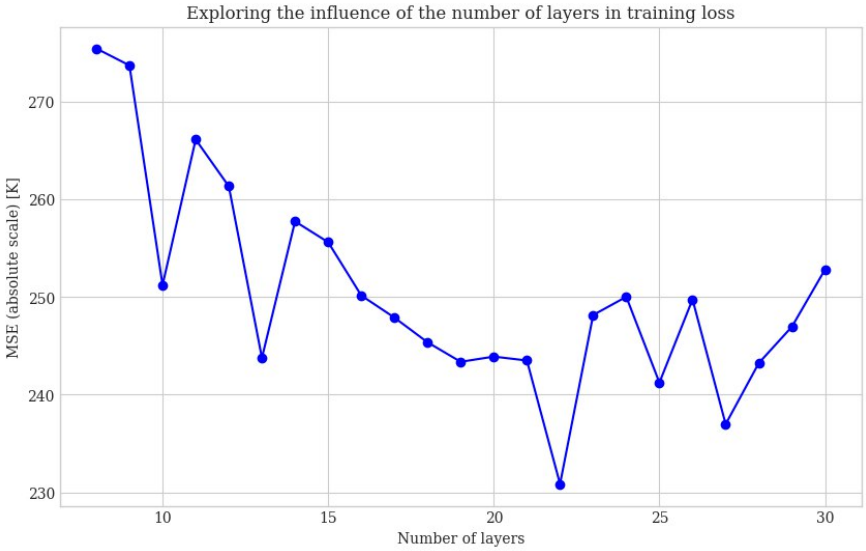


Figure 6.7: Evolution of the MSE with the number of layers in GCN 2D architecture, truncated.

Having this in mind, an optimization study was done with Optuna, with the number of layers ranging from 4 to 12. This range was selected due to the fact that more layers do not have to increase the performance based on Figure 6.6, but it increases computational cost. After the study

done combining the hyper-parameters described below, the optimal number of layers was found to be 8.

Number of Neurons (*Hidden dimension*)

To obtain an optimal hidden dimension for the GCN model, configurations of 8, 9, 16, 32, 64, 128, 256 and 512 neurons were tested in an Optuna study to find the optimal setup. It was observed that beyond 256 neurons per node, the model began to over-fit by learning complex patterns unrelated to the thermal problem, leading to slower training and worse results. This happens because in GNN the hidden dimension is the number of features that each node has, an so is the information passed to their neighbours. As it is a temperature prediction problem, increasing excessively the number of features leads to less accurate models. After the study was done, the optimal hidden dimension in combination with the other hyper-parameters was found to be 64.

Batch Size

For this network, the batch size in the Optuna study ranged between 16, 32 and 64, not being greater than 64 for the inability of the network to update its weights effectively so it could learn meaningful patterns. Finally, a batch size of 32 was found to be the best value of this hyper-parameter in combination with the other hyper-parameters.

Number of Epochs

In the network, an Early Stopping was implemented, so if after 100 epochs the validation total loss, which includes all the MSE losses, did not reduce, the Early Stopper would be triggered. This is done for both optimize computational resources, and avoid over fitting. For any of the trainings, the amount of epochs needed to trigger that Early Stopper were always below 450 epochs.

Learning Rate

As with the other hyper-parameters, the initial learning rate was optimized with Optuna, with this initial learning rate ranging from 1e-2 to 1e-4. Since a learning rate decay is implemented, the learning rate is halved every 10 epochs in which the MSE does not decrease. After the study was done, it was found that the optimal initial learning rate was 0.00454.

Dropout Rate

A dropout probability ranging from 0% to 20% was used at the Optuna study. This maximum value was chosen because many networks employ dropout rates of similar magnitude. Nevertheless, after the study, the optimal dropout rate was found to be 5,38%.

6.3.1.1 Results

The combination of hyper-parameters that Optuna found to be the best for this problem is the following Table 6.1.

Table 6.1: Hyper-parameters used for GCN 2D training.

Hyper-parameter	**Value**
Number of Layers	8
Hidden Dimension	32
Dropout Rate	0.0539
Learning Rate	0.00454
Batch Size	32

With the loss function being only the MSE

Once the network was trained using the aforementioned hyper-parameters, its feasibility for thermal modelling in this problem became apparent as it is shown in Figure 6.8, offering an initial look at its capability solving other thermal complex problems.

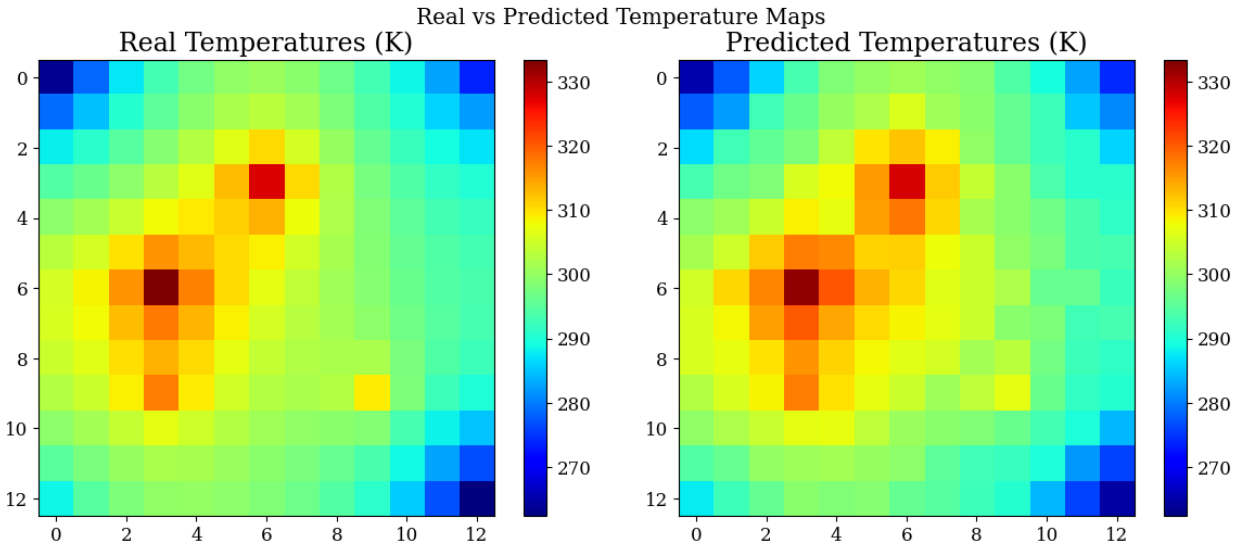


Figure 6.8: Target and predicted temperatures by the GCN 2D.

To ensure suitability for space environments, an accuracy threshold was defined: a node's predicted temperature is considered acceptable if it falls within ±3 K of the ground truth. This threshold is more stringent than the 5 K maximum absolute error specified in the ECSS-E-ST-31C standard [53], as required by the European Space Agency (ESA).

After training this GCN with the hyper-parameters discussed above in Table 6.1, the amount of nodes of the whole Dataset that were within that threshold were 83.63% . Though, as it can be seen in the Figure 6.9, the error mostly accumulates next to the heaters and interfaces. Mainly due to the fact that the temperature gradient is larger than in other nodes.

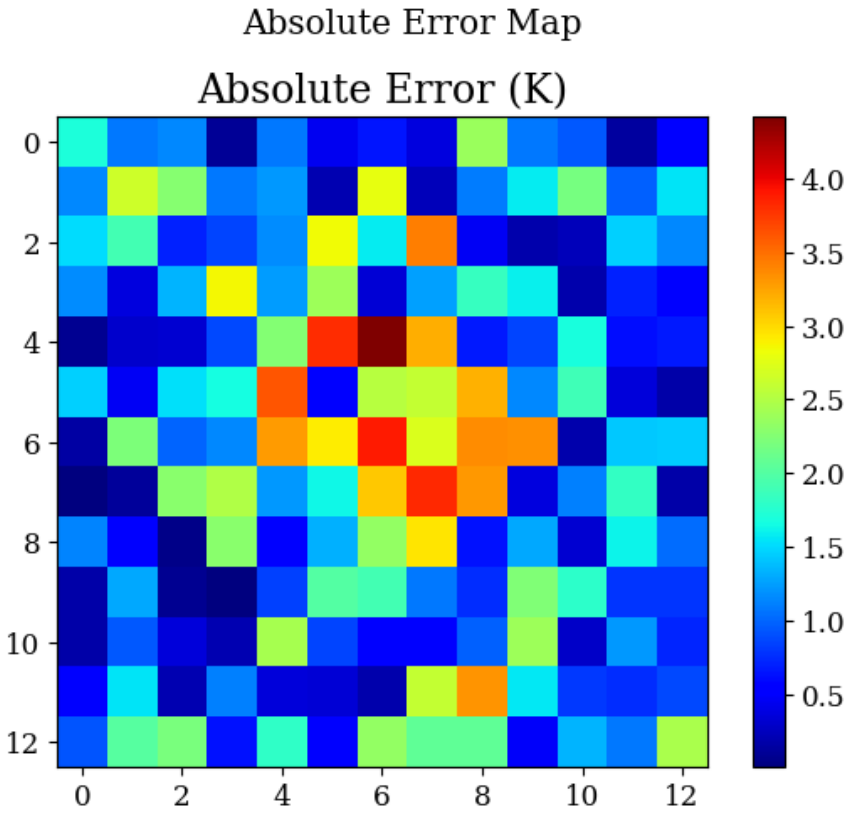


Figure 6.9: Absolute error in Kelvin per node. GCN 2D only with MSE.

As it can be seen in the Figure 6.10, this network stops improving after epoch $\sim$ 180. The Early stopper, triggered at epoch $\sim$ 280, saves the best model found, with the minimum validation loss among all epochs.

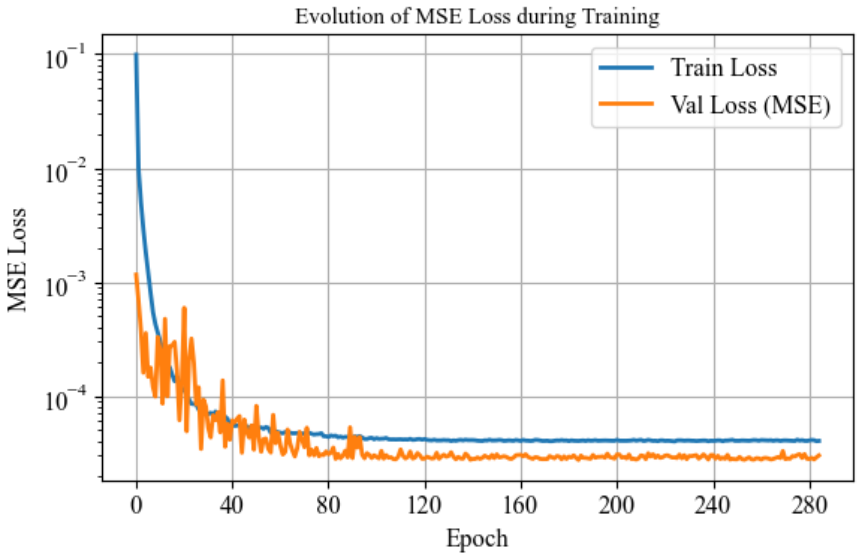


Figure 6.10: Evolution of the training and validation loss, all normalized. GCN 2D, only with MSE.

The results obtained with this loss function are showed in Table 6.2, with a maximum error of 12.82 K, being 83.63% of nodes within the 3 K threshold and a RMSE of 2.10 K.

Table 6.2: Evaluation metrics and loss components of the GCN 2D trained model using the MSE as the loss function.

Metric	**Value**
MSE	4.41 K^2
RMSE	2.10 K
MAE	1.69 K
Coefficient of Determination (R^2)	0.9684
Accuracy (%)	83.63%
Maximum error	12.82 K
Mean Maximum error	8.87 K

With Physics Implemented in a combined loss function

To better predict the temperatures at nodes with higher errors, the total loss used during training and validation included not only the MSE, but also additional terms: the MSE of the boundary conditions at interfaces, the heaters, and a physics-based loss term derived from the first law of thermodynamics. All these additional terms are weighted by factors selected based on where the GCN tends to make larger errors. These weights were determined empirically by observing which configurations resulted in more nodes falling within a 3 K threshold. However, the maximum error

of the GCN with this new loss function (12.87 K) Table 6.3 is slightly higher than when using only the MSE (12.82 K) Table 6.2.

The combined loss function minimised was:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{mse}} \cdot \mathcal{L}_{\text{mse}} + \lambda_H \cdot \mathcal{L}_{\text{heaters}} + \lambda_B \cdot \mathcal{L}_{\text{boundary}} + \lambda_{\text{PI}} \cdot \mathcal{L}_{\text{physics}}$$

Where:

- $\mathcal{L}_{\text{total}}$: Total loss used during training.
- $\mathcal{L}_{\text{mse}}$: Mean squared error between predicted and target temperatures.
- $\mathcal{L}_{\text{heaters}}$: Heater power loss (e.g., error in applied or inferred heater input).
- $\mathcal{L}_{\text{boundary}}$: Boundary condition loss (e.g., enforcing fixed temperatures or fluxes).
- $\mathcal{L}_{\text{physics}}$: Physics-informed loss enforcing thermal laws (e.g., energy conservation, Fourier's law).
- $\lambda_{\text{mse}}, \lambda_h, \lambda_b, \lambda_{\text{pi}}$: Weighting factors controlling the relative contribution of each term.

With $\lambda_B = 1$, $\lambda_H = 10$, and $\lambda_{\text{PI}} = 0.001$, the best result was obtained, with a training time per epoch of 31.26 seconds. As shown in Figure 6.11, which displays the maximum absolute error across the domain, the largest discrepancies are concentrated near the heaters and the interfaces. This suggests that the combined loss function is mostly achieving its intended purpose: prioritizing accuracy in regions with significant thermal gradients and boundary influences. However, it is not as effective as it was supposed, due to the maximum error being located near these nodes with boundary conditions.

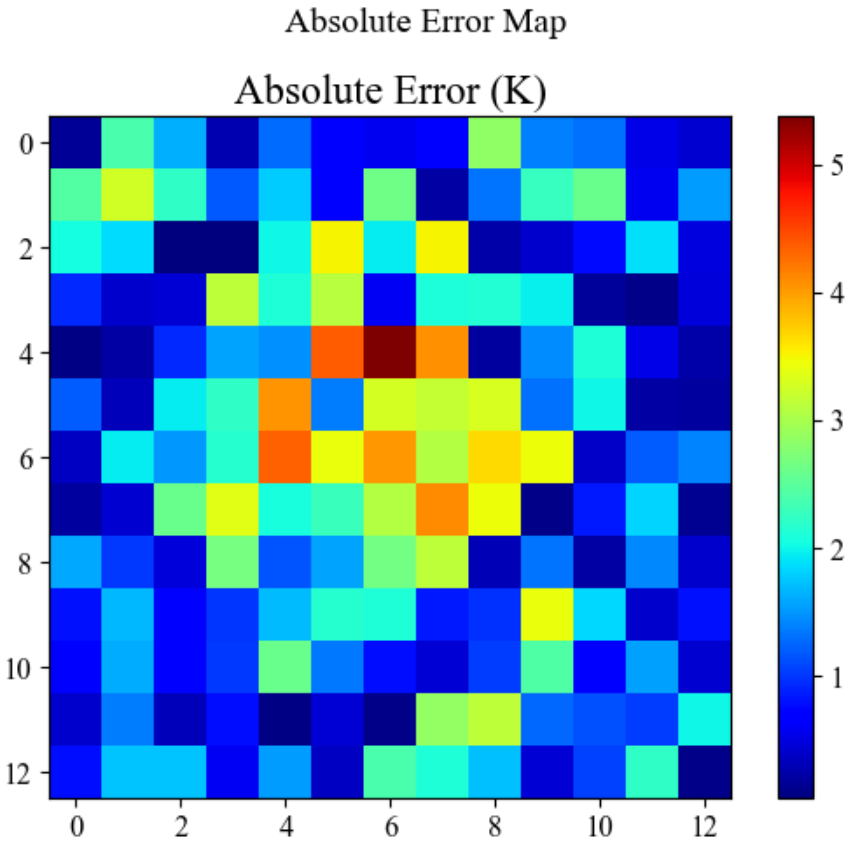


Figure 6.11: Absolute error in Kelvin per node. GCN 2D with the combined loss function.

With the accuracy percentage within the threshold of 86.57 %, slightly higher than when using only the MSE, the RMSE obtained was 2.02 K and with a maximum error in the whole Dataset of 12.87, a bit higher than the one obtained using the MSE loss function. The rest of the metrics obtained are shown in Table 6.3.

Table 6.3: Evaluation metrics and loss components of the GCN 2D trained model with the combined loss function.

Metric	**Value**
MSE	4.06 K^2
RMSE	2.02 K
MAE	1.62 K
Coefficient of Determination (R^2)	0.9707
Accuracy (%)	86.57%
Boundary Condition Loss	1.48 K
Heaters Loss	2.01 K
Maximum error	12.87 K
Mean Maximum error	9.35 K
Physics-Informed Loss	1.221 W
Time for training	154 minutes

6.4 Graph Attention Network (GAT)

After using the GCN architecture, the GAT was explored to assess whether it could provide any improvement. The key feature of this architecture is that it allows the network to learn the relative importance of each node's neighbours. This was particularly promising for this problem, as there are boundary conditions at interfaces and heat sources where temperatures vary significantly, resulting in steep gradients. The GCN appeared to struggle with capturing these sharp variations, so the idea was to enable the model to assign greater importance to neighbours that exhibit significant temperature differences relative to the target node.

6.4.1 Hyper-parameter optimization for GAT and results obtained

For this architecture, as it was done with the GCN architecture, an Optuna study was conducted. As most of the hyper-parameters were previously explained, only those that had significant importance on the network training will be mentioned again.

Number of Layers

As this architecture is more complex than the GCN, increasing the number of layers is more costly than in the previous one. This is something that can also be inferred from the way it calculates the importance of neighbours.

To see how it affected training, the network was trained using three layers, which yielded better

results than the GCN. However, as the number of layers increased, more issues arose compared to the GCN. In this type of architecture, hyper-parameters must be carefully selected to avoid instability. Otherwise, the network may fail to converge, resulting in temperature predictions that deviate significantly from the ground truth.

To find an optimal value for this hyper-parameter, a new Optuna study was conducted, exploring a range of 3 to 10 layers. This narrower range, with a lower maximum than in the GCN Optuna study, was primarily chosen to reduce computational cost and ensure convergence.

Number of Attention Heads

In GATs, attention heads refer to independent, parallel attention mechanisms that operate within a single GAT layer. Each attention head computes its own attention coefficients and applies a separate linear transformation to the features of a node's neighbours. The outputs of all heads are then either concatenated or averaged to produce the final node representation for that layer [38].

The motivation behind using multiple attention heads is to increase the model's expressiveness and robustness. Each head can focus on different aspects of the local neighbourhood, such as feature similarity, spatial orientation, or graph connectivity. Allowing the network to learn richer and more diverse representations.

In general, intermediate layers of the GAT use concatenation to increase model capacity, while the final output layer uses averaging to keep the output dimensionality fixed. The number of attention heads is a tunable hyper-parameter and typically ranges from 4 to 16 in most of the literature.

This multi-head attention mechanism enhances the GAT's ability to learn meaningful representations from graph-structured data, especially in heterogeneous or noisy environments.

Nevertheless, as it happened with the number of layers, the attention heads also require a precise selection of the appropriate numbers, so that it does not over-fit or learn any meaningless information.

To find the optimal value of these attention heads per node, an Optuna study was conducted. The number of heads in this study ranged from 2 to 8, being this maximum value because, while training, more than 8 attention heads seemed to only add computational time with no significant improvement in prediction. Finally, the Optuna study concluded that 2 attention heads were the optimal number in combination with the other hyper-parameters.

Number of Neurons

The range for the hidden dimension in the Optuna study for this architecture was reduced, being 8, 9, 16, 32, 64 and 128, mainly because using 256 caused long training times and the network was unable to predict correctly, exploding from time to time. The optimal hidden dimension for this architecture was found to be 16

Dropout Rate

As this hyper-parameter might be seen as negligible, in this type of architecture, it is essential to tune it correctly. After the Optuna study was conducted, it was found that the optimal dropout rate was 2.28%.

6.4.1.1 Results

The combination of optimal hyper-parameters found by Optuna after the study is showed in Table 6.4.

Table 6.4: Hyper-parameters used for GAT 2D training

Hyper-parameter	**Value**
Number of Layers	7
Hidden Dimension	16
Dropout Rate	0.0228
Learning Rate	0.00071
Batch Size	8
Number of Attention Heads	2

With Physics Implemented in a combined loss function

With these hyper-parameters and with the following weights for the total train loss; $\lambda_B = 1$, $\lambda_H = 10$, $\lambda_{\text{PI}} = 0.0001$ the network obtained worse results than the GCN, having a maximum error overall of 12.99 K, a RMSE of 2.38 K and with the accuracy being 78.96%. With an average time for epoch in training of 142.29 seconds, which is also greater than the one obtained in the GCN training. The rest of metrics obtained are shown in Table 6.5.

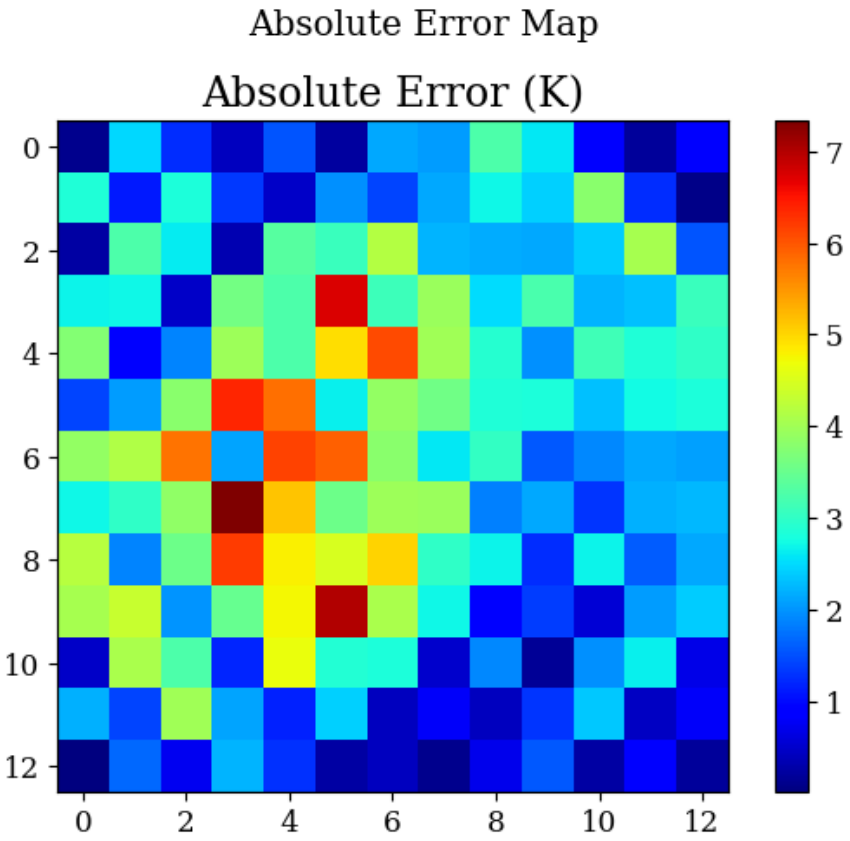


Figure 6.12: Absolute error in Kelvin per node. GAT 2D with the combined loss function.

As we can see in the Figure 6.12, if it is true that the interfaces and heaters nodes are not the ones with the maximum error, the error is still bigger than in the GCN overall. This error is distributed mostly around the heaters.

Table 6.5: Evaluation metrics and loss components of the GAT trained model with the combined loss function.

Metric	**Value**
MSE	5.68 K^2
RMSE	2.38 K
MAE	1.91 K
Coefficient of Determination (R^2)	0.9550
Accuracy (%)	78.96%
Boundary Condition Loss	1.14 K
Heaters Loss	2.38 K
Maximum error	12.99 K
Mean Maximum error	5.56 K
Physics-Informed Loss	1.225 W
Time for training	749 minutes

With the loss function being only the MSE

To determine if using the combined loss function really improves the results, the same network was trained but only using the MSE as the function to minimize.

As it is shown in Figure 6.13, the maximum error is located on the interfaces, and has the same error pattern in the left part of the PCB, but with a slightly bigger error.

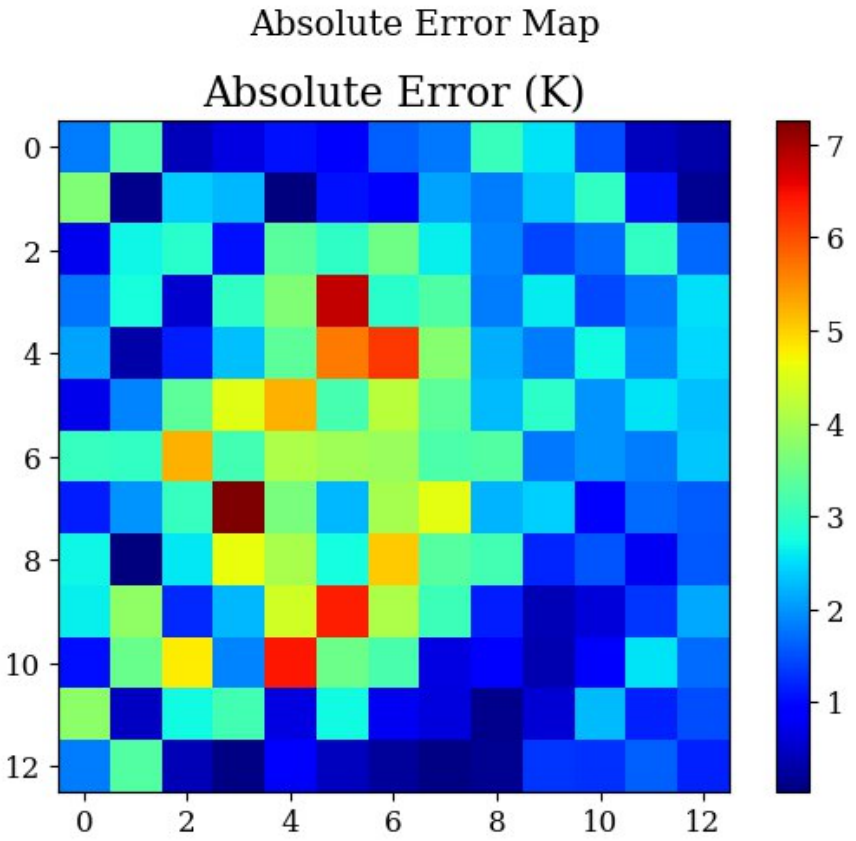


Figure 6.13: Absolute error in Kelvin per node. GAT 2D only with MSE.

It was found that the maximum error was almost the same, but a bit larger than with the combined loss function. Additionally the number of nodes within the 3K threshold was fewer than when using the combined loss function, being this value of 76.79%. The RMSE obtained was a bit bigger (2.54 K). The rest of the results are shown in Table 6.6.

Table 6.6: Evaluation metrics and loss components of the GAT trained model using the MSE as the loss function.

Metric	**Value**
MSE	6.43 K^2
RMSE	2.54 K
MAE	2.02 K
Coefficient of Determination (R^2)	0.9493
Accuracy (%)	76.79%
Maximum error	13.36 K
Mean Maximum error	5.56 K

6.5 Graph SAmple and aggreGatE (Graph SAGE)

The drive to use this architecture is based on what the ANN does with the information that has been aggregated, usually by sum or mean operations. This architecture aggregates the information from the nodes averaging it, what it is suspected to be more useful in this PCB problem than what the GCN does, because almost every node, except the ones located at the edges, have the same connectivity degree.

This architecture has the same number of hyper-parameters as the GCN. Nevertheless, the GCN takes into account the self-loop in each node, what the Graph SAGE does not with the connectivity index generated.

6.5.1 Hyper-parameter optimization for SAGE and results obtained

As it was done with the other architectures, Optuna was used, with the following range for hyper-parameters:

- Number of Layers: The study was done from 2 to 10 layers.
- Hidden dimension: The study was done between 8, 9 , 16, 32, 64 and 128 neurons per node.
- Dropout rate: It ranged from 0% to 20%
- Learning rate: It ranged from 1e-2 to 1e-4.
- Batch size: It was done between 8, 16, 32 and 64.

The combination of hyper-parameters that Optuna found to be the best for this problem is shown in Table 6.7.

Table 6.7: Hyper-parameters used for SAGE 2D training.

Hyper-parameter	**Value**
Number of Layers	10
Hidden Dimension	64
Dropout Rate	0.0172
Learning Rate	0.00384
Batch Size	16

With Physics Implemented in a combined loss function

With these hyper-parameters and with the following weights for the total train loss; $\lambda_B = 0.01$, $\lambda_H = 0.01$, $\lambda_{\text{PI}} = 0.0001$ the network obtained better results than the GCN, having a maximum

error overall of 8.52 K, what is noticeably better than the maximum error in the GCN (12.87 K). The RMSE was also the lowest among the architectures previously tested (1.24 K) and the accuracy threshold obtained was also the best (98.09%). With an average time for epoch in training of 42.04 seconds, which is slightly higher than the average time per epoch in the GCN, but noticeably lower than in the GAT. The other metrics obtained are shown in Table 6.8.

Table 6.8: Evaluation metrics and loss components of the SAGE trained model with the combined loss function.

Metric	**Value**
MSE	1.53 K^2
RMSE	1.24 K
MAE	0.98 K
Coefficient of Determination (R^2)	0.9885
Accuracy (%)	98.09%
Boundary Condition Loss	0.86 K
Heaters Loss	1.24 K
Maximum error	8.52 K
Mean Maximum error	4.77 K
Physics-Informed Loss	1.220 W
Time for training	117 minutes

As it is shown in Figure 6.14, the maximum error is located at the centre of the PCB, between the four heaters, and the distribution of the error is mostly uniform, with two zones differenced, the one with highest error (top left) and the one with lower error (interfaces and bottom right).

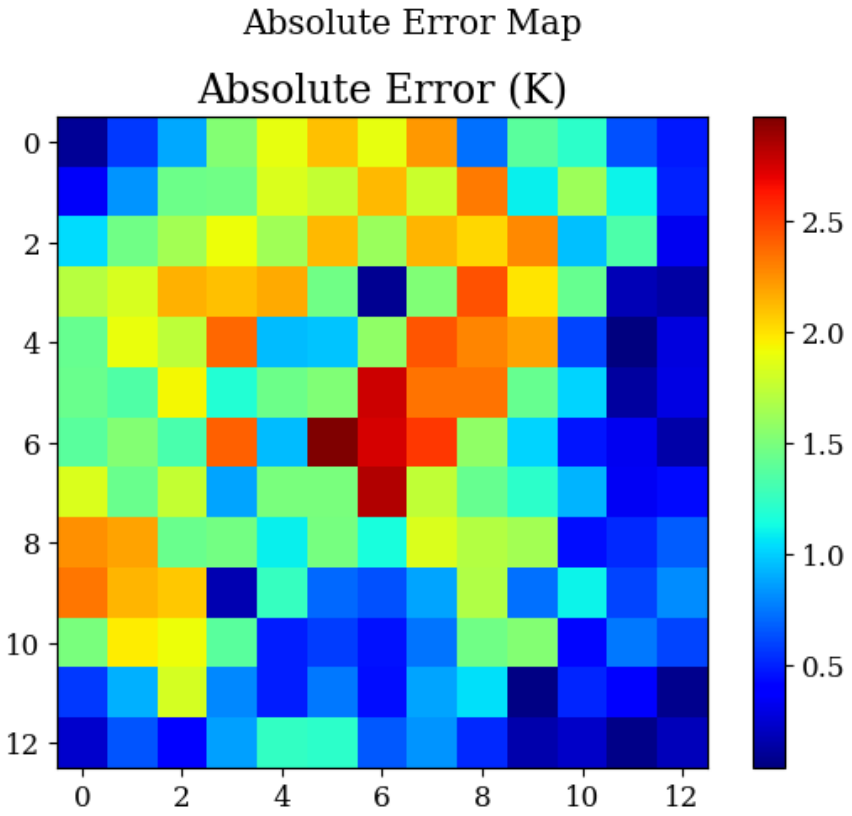


Figure 6.14: Absolute error in Kelvin per node. SAGE 2D with the combined loss function.

With the loss function being only the MSE

To see if the combined function loss improved the model, a training using the MSE as the loss function was ran. The pattern of the error is similar to the one with the combined loss function, as it is shown in Figure 6.15.

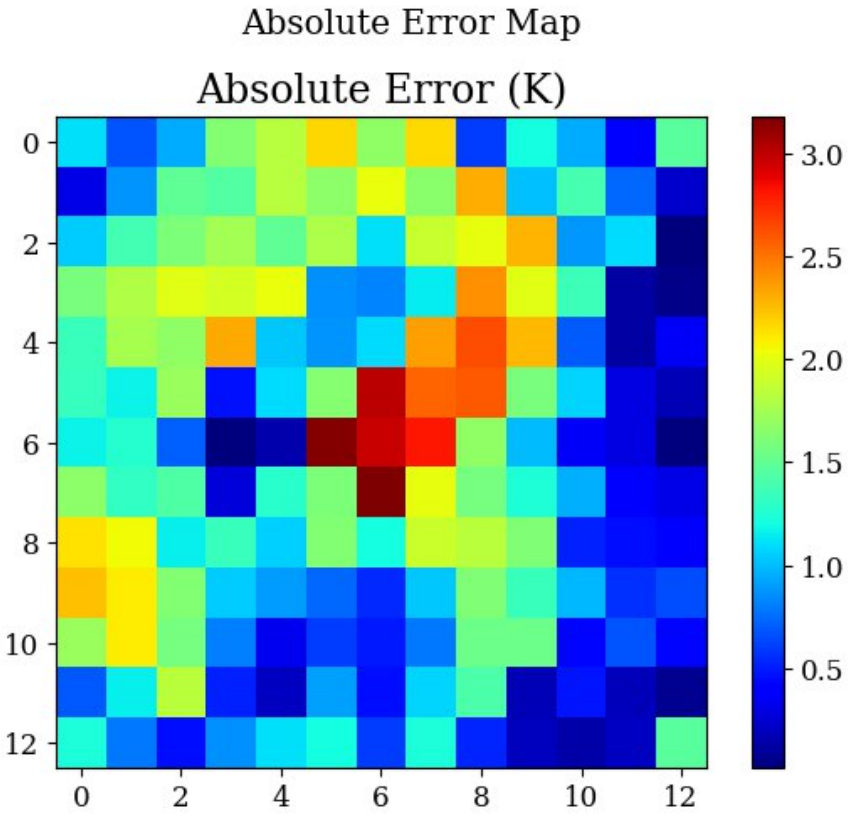


Figure 6.15: Absolute error in Kelvin per node. SAGE 2D only with MSE.

It was found that more nodes were within the 3K threshold (98.67%) despite that this increase is almost negligible. Also the maximum error was smaller (6.15 K) but, this time it is noticeable, and the RMSE followed the same tendency (1.20 K). The other metrics obtained are shown in Table 6.9. So it can be stated that for this architecture, the combined loss function does not offer any advantage.

Table 6.9: Evaluation metrics and loss components of the SAGE trained model using the MSE as the loss function.

Metric	**Value**
MSE	1.45 K^2
RMSE	1.20 K
MAE	0.95 K
Coefficient of Determination (R^2)	0.9895
Maximum error	6.15 K
Mean Maximum error	4.20 K
Accuracy (%)	98.67%

6.6 Neural Network Convolution (NNConv)

To conclude with, the NNConv architecture was used due to the fact that it is the only architecture of the ones listed before that accepts a vector as features on its edges. This is particularly relevant and convenient given that between nodes, the LPTM uses the linear conductance and the radiative conductance between two neighbour nodes to calculate the amount of heat exchanged between them. So, with this in mind, having an architecture that weights the importance of a node's neighbours based on a physics parameter is very convenient and can enhance the interpretation on how does the NNConv predict. Both conductances represents the importance of the radiative activity between nodes and the importance of the conduction phenomenon independently, so it can reach solutions based more on physical traits.

For this architecture, the number of hyper-parameters is the same as the GCN and Graph SAGE, with the only difference that each of the edges has 2 attributes as inputs, G_L and G_R. Nevertheless, in this PCB, all are the same among the nodes, so this feature is not really useful for solving this 2D problem, but, when scaling it to a real 3D model, it would certainly make a difference to take into account those parameters.

6.6.1 Hyper-parameter optimization for NNConv and results obtained

As it was done with the other architectures, Optuna was used, with the following range for hyper-parameters:

- Number of Layers: The study was done from 2 to 10 layers.
- Hidden dimension: The study was done between 8, 9 , 16, 32 and 64 neurons per node.
- Dropout rate: It ranged from 0% to 20%
- Learning rate: It ranged from 1e-2 to 1e-4.
- Batch size: It was done between 8, 16, 32 and 64.

The combination of hyper-parameters that Optuna found to be the best for this problem is shown in Table 6.10.

Table 6.10: Hyper-parameters used for NNConv 2D training

Hyper-parameter	**Value**
Number of Layers	7
Hidden Dimension	9
Dropout Rate	0.0147
Learning Rate	0.00023
Batch Size	16

With Physics Implemented in a combined loss function

With these hyper-parameters and with the following weights for the total train loss; $\lambda_B = 1$, $\lambda_H = 10$, $\lambda_{\text{PI}} = 0.0001$ the network obtained better results than the GCN and GAT , having a maximum error overall of 11.23 K, which is the smallest among the three.However, it is still far from the 6.15 K obtained with the Graph SAGE architecture using the MSE as the loss function. Despite having a smaller maximum error, the percentage of nodes within the threshold is 79.22%, which is slightly better than GAT. Nevertheless, both are the architectures with the worst performance in this metric. Furthermore, this architecture obtained the worst RMSE, being 2.92 K. The average time for epoch in training was of 56.27 seconds. The other metrics are shown in Table 6.11.

Table 6.11: Evaluation metrics and loss components of the NNConv trained model with the combined loss function.

Metric	**Value**
MSE	5.63 K^2
RMSE	2.37 K
MAE	1.90 K
Coefficient of Determination (R^2)	0.9579
Accuracy (%)	79.22%
Boundary Condition Loss	1.12 K
Heaters Loss	2.37 K
Maximum error	11.23 K
Mean Maximum error	7.91 K
Physics-Informed Loss	1.228 W
Time for training	139 minutes

In Figure 6.16, the error is primarily concentrated in a horizontal band across the middle of the PCB. This pattern is noteworthy, as it may suggest that the network is not effectively learning the underlying distribution. It is also noticeably that the network predicted very well the temperature

on its interfaces, as they are among the nodes with lower error.

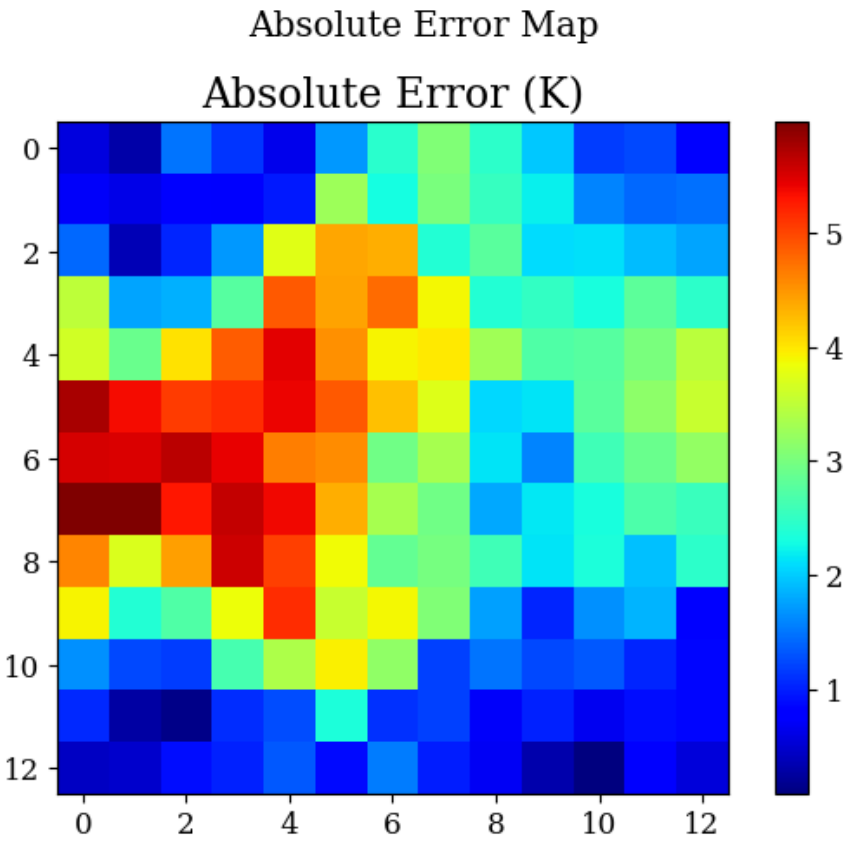


Figure 6.16: Absolute error in Kelvin per node. NNConv 2D with the combined loss function.

With the loss function being only the MSE

Also, a training using the MSE as the loss function was done, with the results being worse than with the combined loss function. The maximum error overall was of 10.68 K, which is slightly better than the one obtained with the combined loss function. However, both the RMSE and the nodes within the accuracy threshold were worse than with the combined loss function, with values of 2.78 K and 72.36% respectively. The other results obtained are shown in Table 6.12.

Table 6.12: Evaluation metrics and loss components of the NNConv trained model using the MSE as the loss function.

Metric	**Value**
MSE	7.72 K^2
RMSE	2.78 K
MAE	2.18 K
Coefficient of Determination (R^2)	0.9466
Maximum error	10.68 K
Mean Maximum error	7.92 K
Accuracy (%)	72.36%

As it is showed in Figure 6.17, the error mostly accumulates on the interfaces, what is the reason behind using a combined loss function. However, the horizontal band across the middle of the PCB is narrower than with the combined loss function, with a little expansion around the central nodes. This pattern could also suggest that the network is not effectively learning the underlying distribution either.

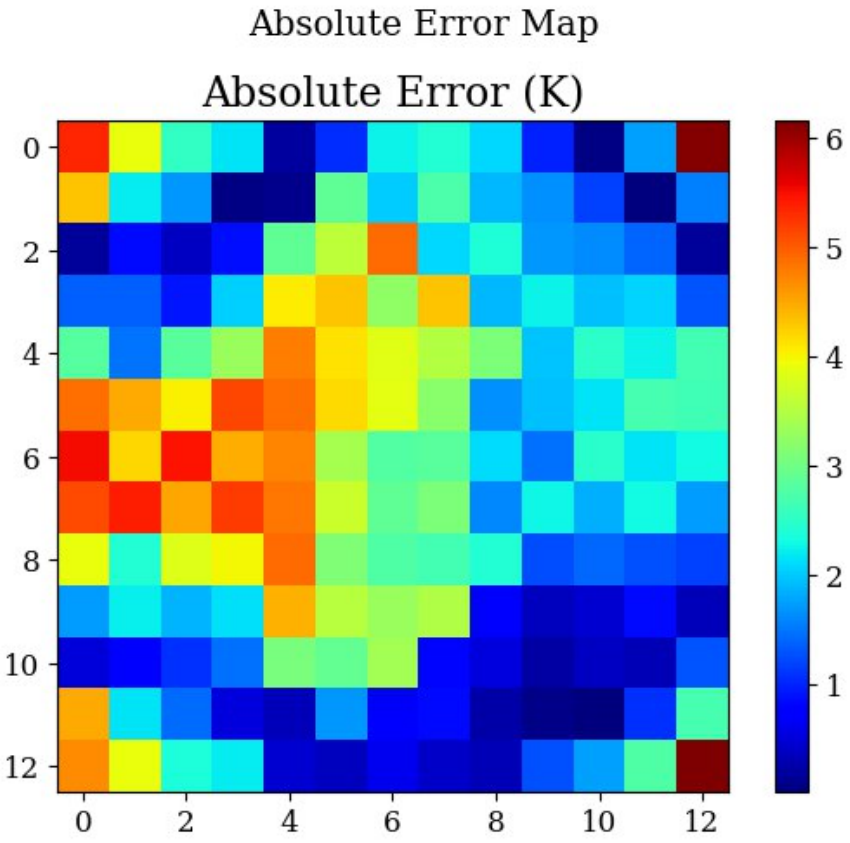


Figure 6.17: Absolute error in Kelvin per node. NNConv 2D only using the MSE.

6.7 Comparison between the GNN architectures

For this comparison, a table summarizing the main hyper-parameters and performance metrics has been created to facilitate understanding and comparison of the previously discussed architectures Table 6.13. All models were selected based on their performance. Notably, three out of the four architectures used the combined loss function, as it significantly improved their results. The only exception was the Graph SAGE model, which performed slightly worse in accuracy and RMSE with the combined loss. However, it achieved a noticeably lower maximum error when trained solely with the MSE.

Table 6.13: Comparison of hyper-parameters and performance across GNN architectures

Metric	**GCN**	**GAT**	**GraphSAGE**	**NNConv**
Number of Layers	8	7	10	7
Hidden Dimension	32	16	64	9
Batch Size	32	8	16	16
Learning Rate	0.00454	0.00071	0.00384	0.00023
Dropout Rate	0.0539	0.0228	0.0172	0.0147
Time per Epoch (s)	31.26	142.29	42.04	56.27
Root Mean Squared Error [K]	2.02	2.38	1.20	2.92
Accuracy (%)	86.57	78.96	98.67	79.22
Maximum error [K]	12.87	12.99	6.15	11.23
Mean Maximum error [K]	9.35	5.56	4.20	7.91
λ_H	10	10	N/A	10
λ_B	1	1	N/A	1
λ_{PI}	0.001	0.0001	N/A	0.0001

As it can be observed in Table 6.13, the architectures with the worst results were the NNConv and the GAT, with a maximum error of 11.23 K and 12.99 K respectively, and not as many nodes within the 3 K threshold compared to the GCN and Graph SAGE, being 86.57% and 98.67% respectively. In addition, the RMSE follows the same pattern, with the GCN and the Graph SAGE with the lower values, being 2.02 K and 1.20 K respectively. Furthermore, the time needed per epoch in training was smaller in those who performed better, which is also an advantage to take into account.

Nevertheless, this comparison may overlook factors that are crucial for achieving better performance in a 3D model. Among all the listed architectures, only the NNConv can incorporate edge features as vectors, an important capability when these features represent properties such as thermal conductivity or the radiative conductance between two surfaces (nodes). With this in mind, no architecture can be definitively ruled out at this stage. However, the most promising candidates appear to be the Graph SAGE and GCN. Additionally, the GAT, with its attention heads, could

prove highly valuable in a 3D context, as these heads are capable of learning relevant geometric patterns and other spatial information.

7

Description of the 3D ESATAN-TMS model to use in training

As it has been demonstrated, GNNs are very convenient and accurate solving problems that they are typically solved through the Lumped Parameter Thermal Method (LPTM), Finite Element Method (FEM) and Computational Fluid Dynamics (CFD). In order to demonstrate its feasibility for solving complex problems, a satellite was created in ESATAN-TMS.

This satellite was specifically developed to be the benchmark in the next step of implementing GNN for solving thermal problems in space. However, this satellite has been over-simplified to test first if changing from 2D to 3D has a noticeable impact on the network.

Whilst it is true that problems in 3 dimensions are completely different from the typically 2 dimensional problems, both thermal problems can be discretized on interconnected nodes, in which the spatial relations translates into nodes with more connections.

This model consist of:

- **Solar panels**: Both solar panels are connected to the body structure through user-defined conductances, not to the MLI. Each panel is modelled as a single node, with different materials assigned to each surface to resemble real configurations.
- **Multi-Layer Insulation (MLI)**: The MLI envelops the body structure and is modelled with one node per surface to simulate thermal gradients across its layers.
- **Camera**: The camera is divided into several components. First, the camera box, defined in the model as (camera elec), which dissipates between 5 and 14 W. Secondly, the camera

cylinder, which protects the internal lens and penetrates the MLI. Finally, the camera cylinder is enclosed by a protective cover.

- **Structure**: The satellite is constructed from Al-6061 and consists of a box formed by four plates. Instruments are mounted on these plates and thermally connected to the satellite structure.
- **Payloads**: The satellite includes two power-dissipating boxes that simulate scientific payloads and instruments. Those two boxes dissipate between 5 and 14 W.
- **Battery**: To emulate a realistic satellite configuration, a battery is included to store and supply energy to the camera and other components. The battery also dissipates heat, ranging from 5 to 14 W, which is not so accurate, but it was done to have the same criteria.
- **Electronic Box**: This component dissipates power ranging from 5 to 14 W.
- **Radiator**: Primarily used to evacuate heat generated internally and absorbed through albedo and other external sources.

The geometry of the satellite modelled can be observed in Figure 7.1.

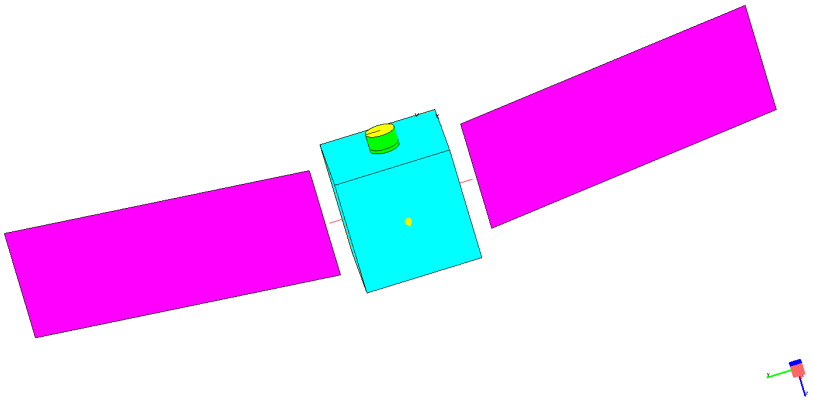


Figure 7.1: Geometry of the satellite.

For further details, the MLI and one wall of the structure are hidden, revealing the payloads and the plates to which the components are attached, as shown in Figure 7.2.

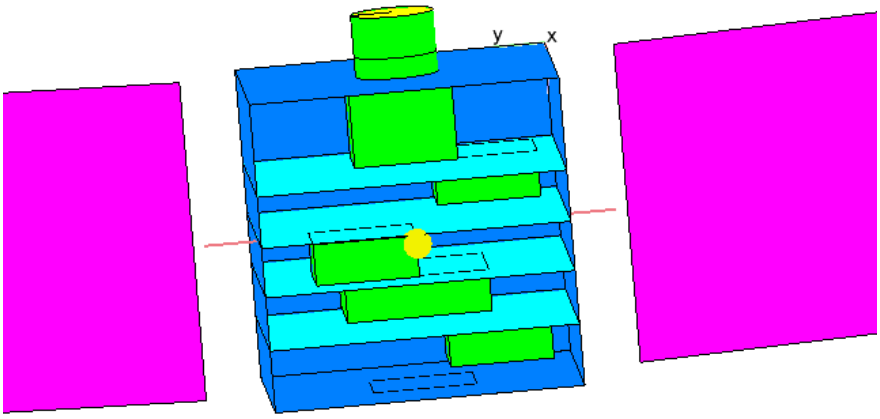


Figure 7.2: Geometry of the satellite, with the MLI and one wall hidden.

Where the components that dissipate power are located as shown in Figure 7.3.

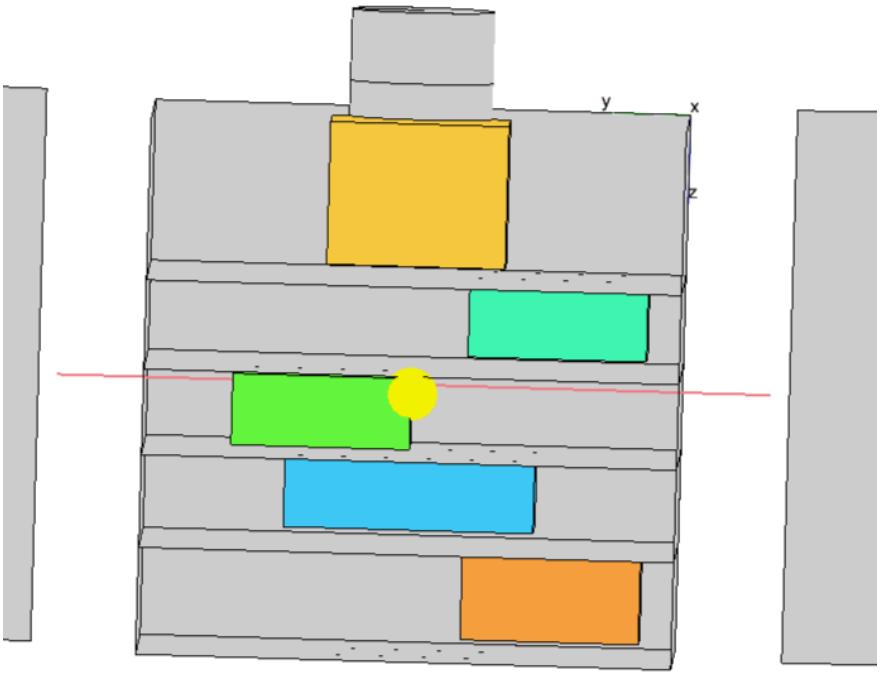


Figure 7.3: Elements that dissipate power in the satellite.

In Table 7.1 a more detailed information is given to better understand what instrument is represented on which colour and what internal node numeration is followed. The order followed is from the instrument at the top of the Figure 7.3 to the bottom.

Table 7.1: Node and colour assignment for each instrument.

Instrument	**Colour**	**External Node ID**	**Internal Node ID**
Camera box	Yellow	41000	16
Box 1	Turquoise	31000	11
Battery	Green	30000	10
Electronic Box	Blue	32000	12
Box 2	Orange	34000	14

Delving more into some properties of the materials used, as it can be seen in the Figure 7.4, the emissivity of the satellite varies from 0.9 to 0.1. In order to see most of the optical properties, some elements were hidden.

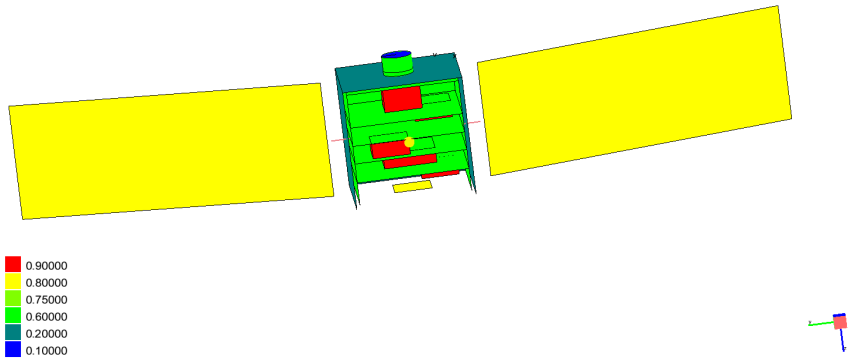


Figure 7.4: IR emissivity of the satellite.

Finally, as shown in Figure 7.5, the thermal conductivity of the different materials is illustrated

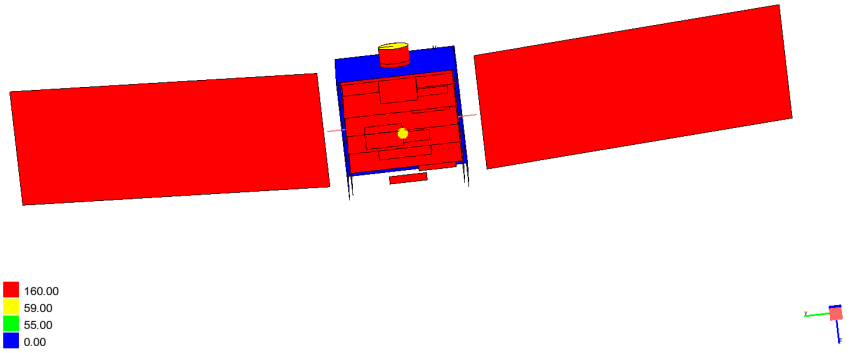


Figure 7.5: Conductivity of the different materials used in the satellite.

The satellite described above was transformed into a graph and represented as such, as shown in Figure 7.6. Each node corresponds to a thermal entity, and each edge represents a connection, either via linear conductance (due to physical contact) or through radiative interaction.

For clarity, Table 7.2 presents the correspondence between each thermal identity name, its external node ID, and its internal node ID, providing a clearer understanding of the component mapping and node numbering.

Table 7.2: Component classification and associated ID ranges.

Component	**Level**	**Group**	**ID Range**	**Local ID**	Power
CAMERA	2	STRUCTURE	#40000–41000	15–16	-
INSTRUMENTS	2	STRUCTURE	#30000–34000	10–14	-
MLI	2	STRUCTURE	#50000–60100	19–30	-
SOLAR_PANEL_1	3	SOLAR_PANELS	#70000	31	-
SOLAR_PANEL_2	3	SOLAR_PANELS	#72000	32	-
BODY_PANEL	3	BODY	#10000–10005	0–5	-
BODY_PLATE_INF	3	BODY	#12000	6	-
BODY_PLATE_MED	3	BODY	#13000	7	-
BODY_PLATE_MED2	3	BODY	#14000	8	-
BODY_PLATE_SUP	3	BODY	#15000	9	-
CAMERA_CYL	3	CAMERA	#40000	15	-
CAMERA_ELEC	3	CAMERA	#41000	16	True
CAMERA_LENS	3	CAMERA	#42000	17	-
CAMERA_TAPE	3	CAMERA	#43000	18	-
BATTERY	3	INSTRUMENTS	#30000	10	True
BOX_1	3	INSTRUMENTS	#31000	11	True
ELECTRONIC_BOX	3	INSTRUMENTS	#32000	12	True
RADIATOR	3	INSTRUMENTS	#33000	13	-
BOX_2	3	INSTRUMENTS	#34000	14	True
MLI_minus_X	3	MLI	#50000–50100	19–20	-
MLI_minus_Y	3	MLI	#50200–50300	21–22	-
MLI_minus_Z	3	MLI	#50400–50500	23–24	-
MLI_plus_X	3	MLI	#50600–50700	25–26	-
MLI_plus_Y	3	MLI	#50800–50900	27–28	-
MLI_plus_Z	3	MLI	#60000–60100	29–30	-
Environment	-	-	#99999	33	-

To see how does the network plots the satellite, Nx library from python [54] was used. In Figure 7.6 the thermal identities are numbered with the external IDs and the colour represents the temperature obtained solving through ESATAN-TMS (LPTM) a generic case.

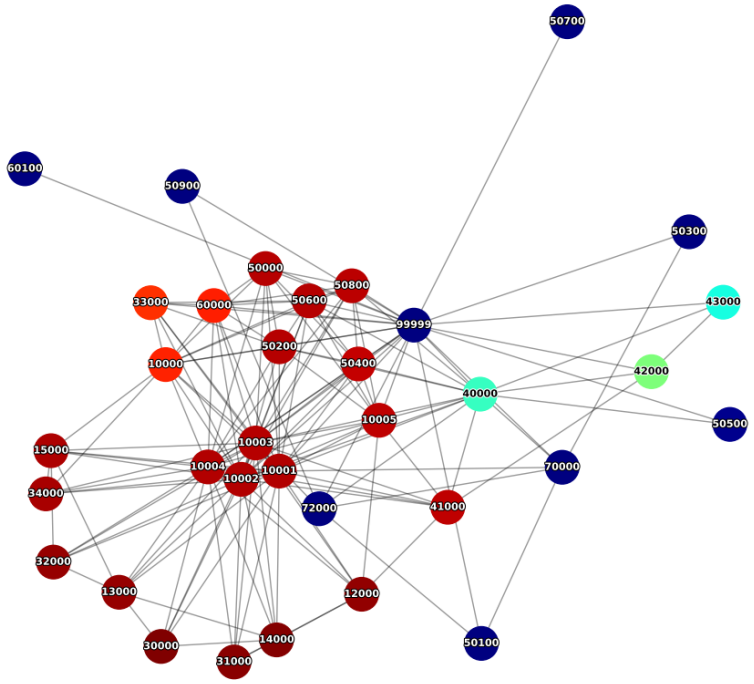


Figure 7.6: Model representation as a graph, with external ID numeration.

The nodes that dissipate power are highlighted in Figure 7.7, with the colour indicating the amount of heat dissipated.

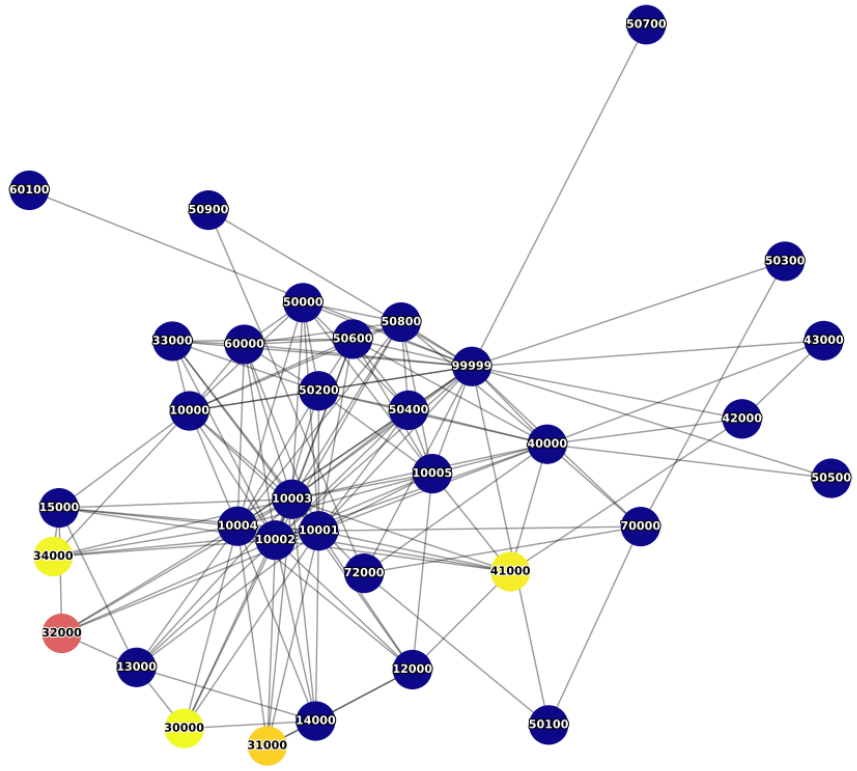


Figure 7.7: Model representation as a graph, displaying nodes that dissipate power with external ID numeration.

Reasons behind using an enclosure

The rationale for modelling the environment as a node lies in the fact that, in space, the surroundings can be represented by an equivalent blackbody temperature that accounts for an enclosure. This enclosure combines various contributions such as albedo, solar radiation, and other radiative phenomena. As a result, the satellite in orbit can be approximated as being within an enclosed environment, not deep space itself, but rather a composite of these radiative effects, whose temperature can be adjusted accordingly.

This approach is particularly convenient, as it enables the simulation of realistic scenarios using a simplified model that still yields accurate results. However, it is more suitable for steady-state problems, since transient modelling of these phenomena is more complex and often better approached through estimation. With this in mind, it is also possible to estimate worst-case scenarios in the satellite's orbit by modelling them as steady-state cases with an enclosure defined by an equivalent temperature.

7.1 Generating syntethic data

As in the 2D PCB, to test if the network can learn effectively the patterns that lead to an accurate temperature prediction, a python solver for ESATAN-TMS steady state models was used.

This solver first generates randomly the power dissipated by all the components and the temperature of the enclosure. The first one ranges from 5 to 14 Watts, and the latter one from 260 to 310 K, the same range as in the 2D PCB.

The solver gets all the data from the .TMD file that ESATAN-TMS generates, including GL, GR, areas, node ID, etc... and solves iterating until a convergence criterion is met.

The dataset stores specific information from the ESATAN-TMS model. Each node has two input features: the boundary temperature, which is non-zero only for the node representing the enclosure, and the dissipated power. The output dimension corresponds to the predicted temperature. The dataset also contains the G_L and G_R values, which serve as edge input features for the NNConv. Additionally, the connectivity information is stored so that the network knows how the nodes are linked. The connection between node number 99999 (the enclosure node) and the other nodes is defined as one-directional, ensuring that the network does not alter the enclosure temperature but instead learns how it influences the temperatures of the other nodes. All of the data stored is normalized with its maximum value respectively.

As it was stated previously, the satellite model is over-simplified, so that we can take a glimpse of the suitability for solving 3D models. Thanks to this, the ESATAN-TMS solver does not need a lot of time to generate the solution. However, as the approach to be followed is to solve real complex 3D models, we can expect that, to generate each solution for a real model, we will need much more time. With this in mind, the number of cases is limited to 10,000. This number is still big, but more feasible to generate.

7.2 Results

The four architectures tested were the same as in the 2D PCB case, with each GNN model being an exact copy of those used in the 2D trials, except for the NNConv, in which a MLP was used after the last layer. Batch normalization and residual connections were still implemented, and the activation function remained SiLU for all models. All input data were normalized by their respective maximum values, including the target temperatures.

All of the following hyper-parameters were obtained through an optimization study using Optuna, with the exception of the weights in the combined loss function.

The new combined loss function differs slightly from the one used in the 2D case. This is because, in the 3D configuration, there are no fixed temperatures at the interfaces. Although the environment node has a fixed temperature, it was expected that the network could accurately predict this value.

One potential solution would be to implement one-way connections between the environment and the other nodes. However, to define a physically consistent loss function, bidirectional connectivity is required.

At present, the combined loss function only incorporates the heater loss term in addition to the MSE. Implementing a physics-informed loss remains a task for future work.

With this in mind, both types of connectivity indices were generated and tested across all architectures.

The combined loss function minimised was:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{mse}} \cdot \mathcal{L}_{\text{mse}} + \lambda_H \cdot \mathcal{L}_{\text{heaters}}$$

7.2.1 Graph Convolutional Network (GCN)

Even though one connectivity was built asymmetrical, this is the only architecture that does not accept asymmetrical connections, but builds symmetrical connections, so, it is expected that the network does make errors while predicting this fixed temperature.

The Optuna study involved training 100 different hyper-parameter combinations for 50 epochs each. The number of layers varied from 3 to 10, while the hidden dimension was chosen from 32, 64, or 128. The dropout rate ranged from 0.0 to 0.2, and the learning rate was selected between 1e-2 and 1e-4. Finally, the batch size was set to 16, 32, or 64.

The best combination of hyper-parameters found by Optuna are shown in Table 7.3.

Table 7.3: Hyper-parameters used for GCN 3D training.

Hyper-parameter	**Value**
Number of Layers	3
Hidden Dimension	64
Dropout Rate	0.1506
Learning Rate	0.00296
Batch Size	16

With the loss function being only the MSE

As it was done in the PCB 2D problem, the training was done using only the MSE as the loss function. Later, it will be tested with a combined loss function. With the aforementioned hyper-parameters the average time per epoch was of 5.97 seconds and the results obtained are shown in Table 7.4.

Table 7.4: Evaluation metrics and loss components of the GCN 3D trained model only using the MSE.

Metric	**Value**
MSE	5.29 K^2
RMSE	2.29 K
MAE	1.68 K
Coefficient of Determination (R^2)	0.9705
Accuracy (%)	83.68%
Maximum error [K]	10.88
Mean Maximum error [K]	5.25
Time for training	28 minutes

This network achieved a maximum error overall of 10.88 K, with 83.68% of nodes within the 3 K accuracy threshold and 2.29 K as the RMSE, what seems very promising after the experience obtained with the PCB 2D problem.

As it can be observed in Figure 7.8 the network is far from over-fitting, which indicates that the complexity of the network suits well the problem.

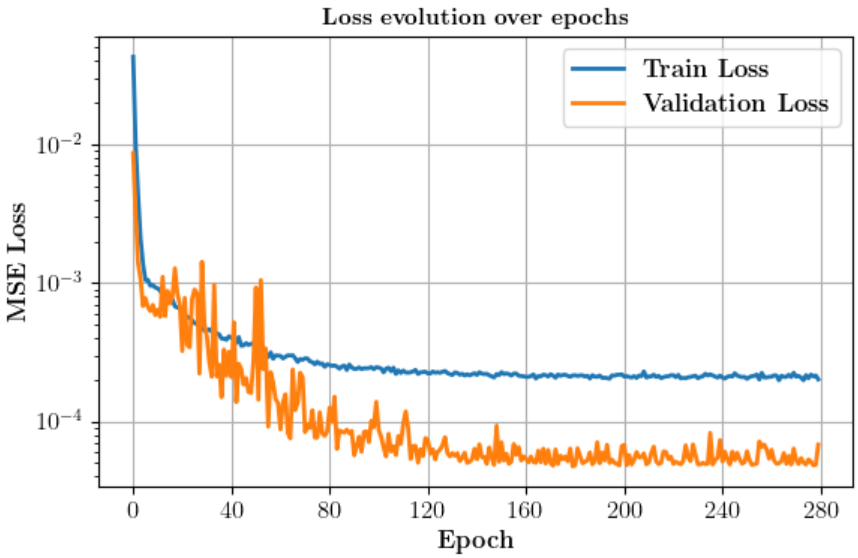


Figure 7.8: Evolution of training loss and validation loss, all normalized. GCN 3D, only with MSE.

As shown in Figure 7.9, the largest errors are primarily located on the heaters, while the rest of the nodes are predicted more accurately. It is also notable that the environment node (33) is among those with the lowest error.

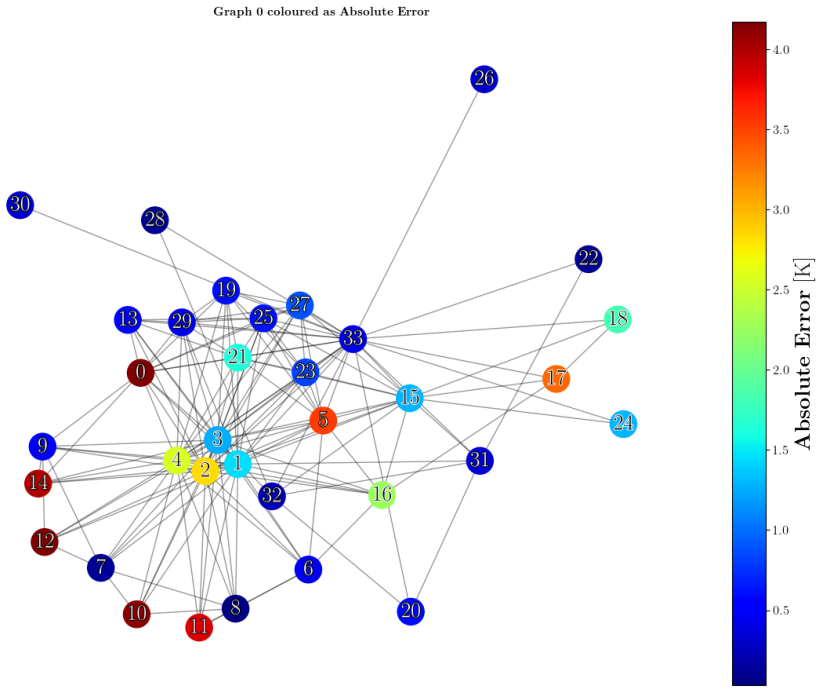


Figure 7.9: Absolute error in Kelvin per node. GCN 3D only with MSE.

With the combined loss function without physics

To see if there is any improvement, the combined loss function was tested with $\lambda_H = 1$ and the results are shown in Table 7.5.

Table 7.5: Evaluation metrics and loss components of the GCN 3D trained model with the combined loss function.

Metric	**Value**
MSE	5.47 K^2
RMSE	2.33 K
MAE	1.80 K
Coefficient of Determination (R^2)	0.9789
Accuracy (%)	80.49%
Heaters Loss	2.05 K
Maximum error [K]	10.73
Mean Maximum error [K]	5.33

Though the maximum error in Figure 7.10 is almost 6 K, which is truly worsen than in Figure 7.9 with the MSE alone, the maximum error obtained was slightly lower, with 10.73 K. Nevertheless, the accuracy threshold and the RMSE obtained were worse, with 80.49% and 2.33 K respectively.

In a nutshell, for this architecture, the combined loss function does not offer any improvements.

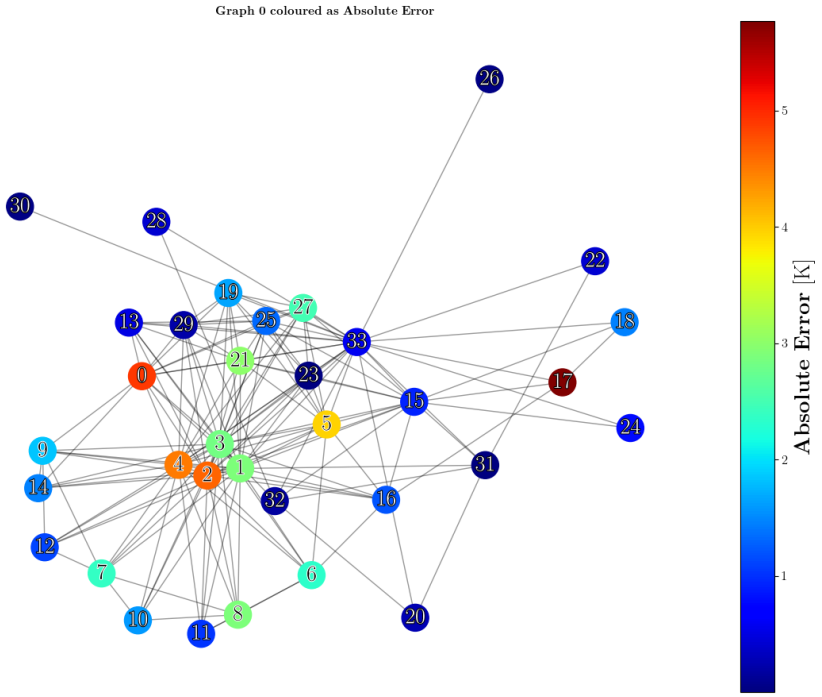


Figure 7.10: Absolute error in Kelvin per node. GCN 3D with the combined loss function.

7.2.2 Graph Attention Network (GAT)

For the Optuna study, the same range of hyper-parameters were set, with the number of attention heads ranging from 2 to 8.

The best combination of hyper-parameters found by Optuna are shown in Table 7.6.

Table 7.6: Hyper-parameters used for GAT 3D training.

Hyper-parameter	**Value**
Number of Layers	4
Hidden Dimension	32
Dropout Rate	0.0048
Learning Rate	0.00147
Batch Size	16
Number of Attention Heads	2

With the loss function being only the MSE and asymmetrical connectivity

First, the Dataset with asymmetrical connections was used, obtaining the results shown in Table 7.7, with an average time per epoch of 11.50 seconds.

Table 7.7: Evaluation metrics and loss components of the trained GAT 3D model only with the MSE and the asymmetrical Dataset.

Metric	Value
MSE	1.61 K^2
RMSE	1.27 K
MAE	0.94 K
Coefficient of Determination (R^2)	0.9937
Accuracy (%)	96.79%
Maximum error [K]	5.70
Mean Maximum error [K]	2.69
Time for training	68 minutes

This network achieved outstanding results when compared to GCN, with a maximum error of 5.70 K, 96.97% nodes within the accuracy threshold and a RMSE of 1.27 K.

As it happened in the 2D PCB problem, the network sometimes has spikes in validation loss, but does not over-fit as it can be observed in Figure 7.11 and predicts very well the temperatures as showed in Table 7.7.

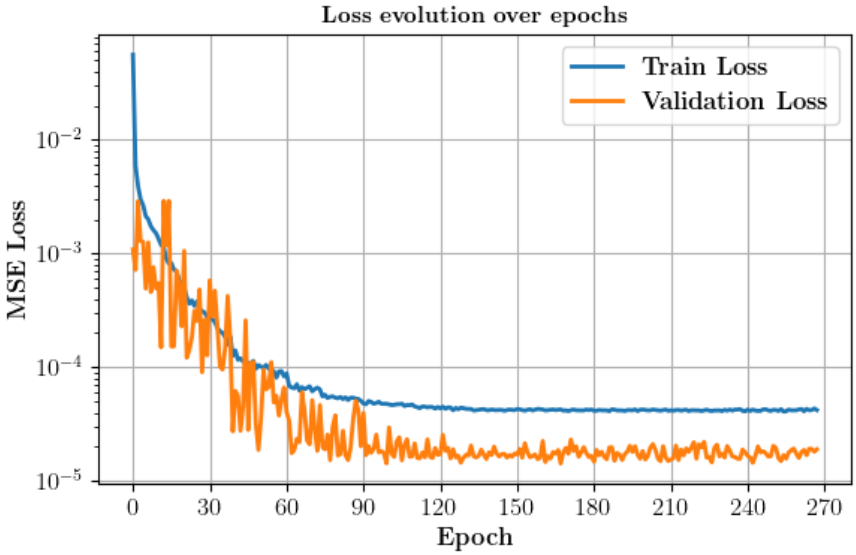


Figure 7.11: Evolution of training loss and validation loss, all normalized. GAT 3D asymmetrical, only with MSE.

With the maximum error in theFigure 7.12 of $\sim$ 4.5 K, and its located on the camera cylinder, while the heaters are well predicted and among those with lowest error.

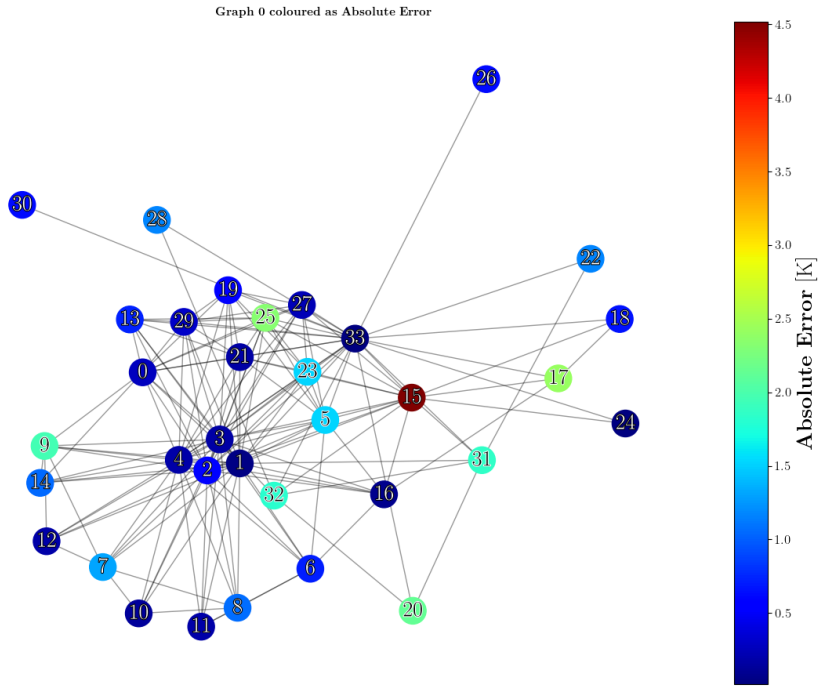


Figure 7.12: Absolute error in Kelvin per node. GAT 3D asymmetrical, only with MSE.

With the loss function being only the MSE and symmetrical connectivity

Nevertheless, when the symmetrical connectivity graph was used, the results improved even further, as shown in Figure 7.13.

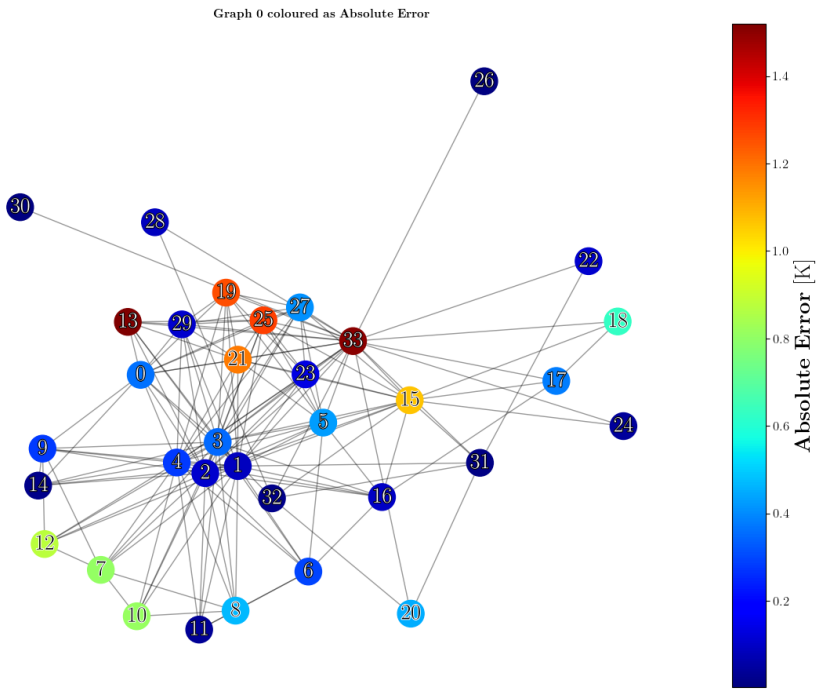


Figure 7.13: Absolute error in Kelvin per node. GAT 3D symmetrical, only with MSE.

The network performs better when all nodes are connected symmetrically, matching the physical model. However, the node with the highest error is the environment node, with an error of approximately 1.4 K as shown in Figure 7.13.

The results were noticeably better with this symmetrical connectivity, as shown in Table 7.8. Achieving noticeably better score in RMSE with 0.70 K, and improving slightly the maximum error overall (5.29 K). Additionally the accuracy threshold is also better with 99.88%.

Table 7.8: Evaluation metrics and loss components of the GAT 3D trained model only with the MSE and the symmetrical Dataset.

Metric	**Value**
MSE	0.49 K^2
RMSE	0.70 K
MAE	0.53 K
Coefficient of Determination (R^2)	0.9981
Accuracy (%)	99.88%
Maximum error [K]	5.29
Mean Maximum error [K]	1.78

With the combined loss function and symmetrical connectivity

To see if the network performed better with the combined loss function, a new training was ran with $\lambda_H = 1$, with the symmetrical connectivity. The results obtained are shown in Table 7.9.

Table 7.9: Evaluation metrics and loss components of the GAT 3D trained model with the combined loss function and the symmetrical Dataset.

Metric	**Value**
MSE	0.68 K^2
RMSE	0.82 K
MAE	0.55 K
Coefficient of Determination (R^2)	0.9937
Accuracy (%)	99.11%
Heaters Loss	0.36 K
Maximum error [K]	10.32
Mean Maximum error [K]	2.54

The results obtained are worse than when using only the MSE, with a RMSE of 0.82 K and an accuracy of 99.11%, both of them being slightly worse, but there was a noticeable increase in the maximum error, achieving 10.32 K.

This can be perceived in the Figure 7.14, with the maximum error being of ~ 1.6 K, which is slightly more than with the MSE all alone.

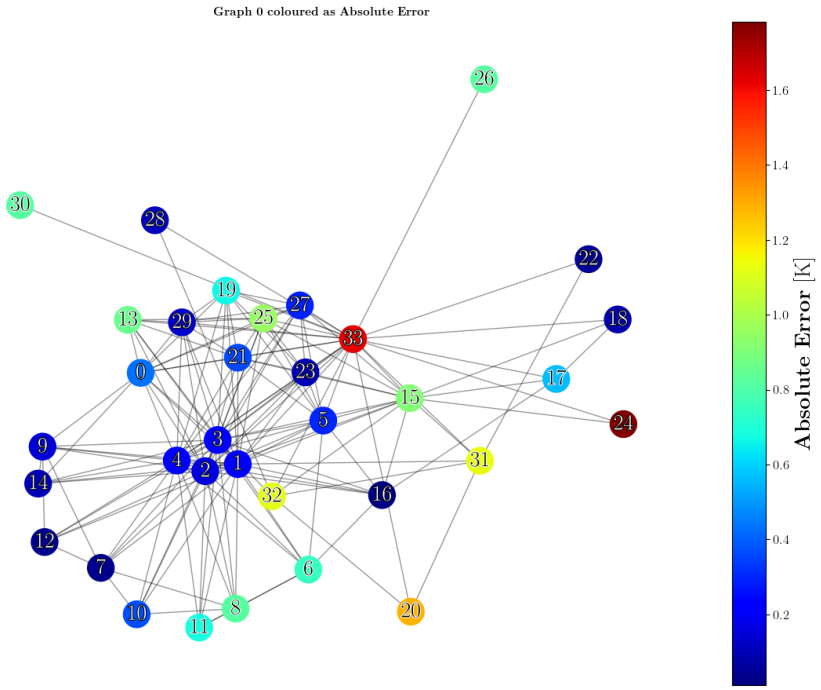


Figure 7.14: Absolute error in Kelvin per node. GAT 3D symmetrical, with the combined loss function.

Despite this, in this architecture, the largest error is primarily located in the environment node, regardless of the connectivity type or loss function used.

7.2.3 Graph SAmple and aggreGatE (Graph SAGE)

The best combination of hyper-parameters found by Optuna is shown in Table 7.10.

Table 7.10: Hyper-parameters used for SAGE 3D training.

Hyper-parameter	Value
Number of Layers	4
Hidden Dimension	128
Dropout Rate	0.0578
Learning Rate	0.00137
Batch Size	64

With the loss function being only the MSE and asymmetrical connectivity

As with the GAT, the first training was set with the asymmetrical connectivity, obtaining an average training time per epoch of 2.55 seconds, with the results obtained shown in Table 7.11.

Table 7.11: Evaluation metrics and loss components of the SAGE 3D trained model only with the MSE and the asymmetrical Dataset.

Metric	**Value**
MSE	6.62 K^2
RMSE	2.49 K
MAE	1.97 K
Coefficient of Determination (R^2)	0.9764
Accuracy (%)	79.14%
Maximum error [K]	10.17
Mean Maximum error [K]	6.43
Time for training	7 minutes

With the results being worse than with the GAT architecture, obtaining a maximum error of 10.17 K, an accuracy of 79.14%, which is the lowest of all networks tested in 3D, and a RMSE of 2.49 K.

Despite these results, which are not bad, these hyper-parameters made the architecture excessively complex, ending with over-fitting in training as shown in Figure 7.16, which is the first time in all the trainings made, in 2D and 3D that happened. However, for this reason an Early stopper was implemented, so that the best model was trained before over-fitting began, at epoch 52.

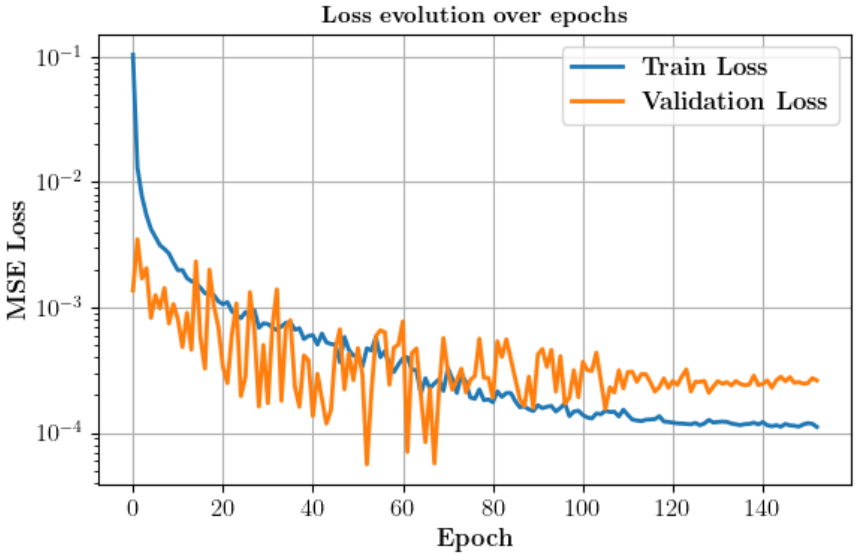


Figure 7.15: Evolution of training loss and validation loss, all normalized. SAGE 3D asymmetrical, only with MSE.

As shown in Figure 7.16, the maximum error is located at the environment node, with an absolute error of approximately 7 K, despite the fact that asymmetrical connections were explicitly introduced

to prevent the network from incorrectly predicting this node's temperature.

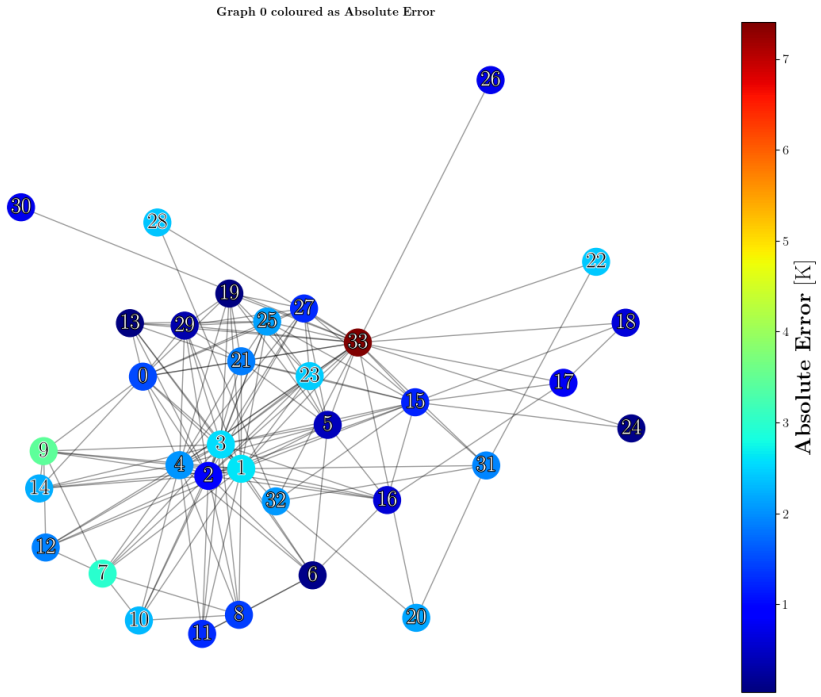


Figure 7.16: Absolute error in Kelvin per node. SAGE 3D asymmetrical, with the MSE.

With the loss function being only the MSE and symmetrical connectivity

To see if it happens the same as with the GAT, the training was set with the symmetrical connectivity graph. With the metrics being shown in Table 7.12.

Table 7.12: Evaluation metrics and loss components of the SAGE 3D trained model only with the MSE and the symmetrical Dataset.

Metric	**Value**
MSE	2.62 K^2
RMSE	1.62 K
MAE	1.32 K
Coefficient of Determination (R^2)	0.9901
Accuracy (%)	93.87%
Maximum error [K]	5.83
Mean Maximum error [K]	3.45

Surprisingly, with the symmetrical connectivity the network improved noticeably, obtaining a maximum error of 5.83 K, an accuracy of 93.86% and a RMSE of 1.62 K. In a nutshell, it performed

better in all aspects.

As it can be seen in the Figure 7.17, this network is on the verge of over-fitting, with noticeably spikes on the validation loss, what could be that the network is excessively complex for the problem to solve, or that simply, this architecture is not well fitted for this problem.

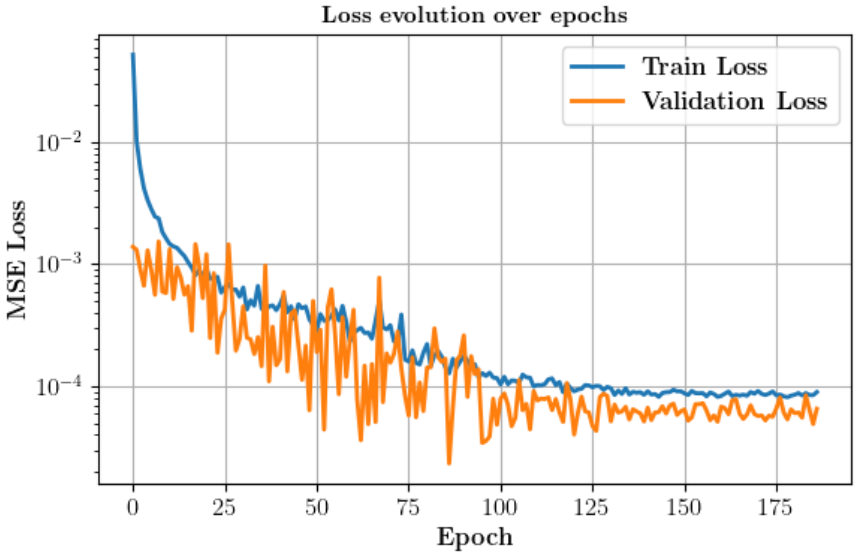


Figure 7.17: Evolution of training loss and validation loss, all normalized. SAGE 3D symmetrical, only with MSE.

Also the error pattern differs from the previous network, as it is shown in Figure 7.18, the largest error is not located in the environment node, but in the ones only connected to this node.

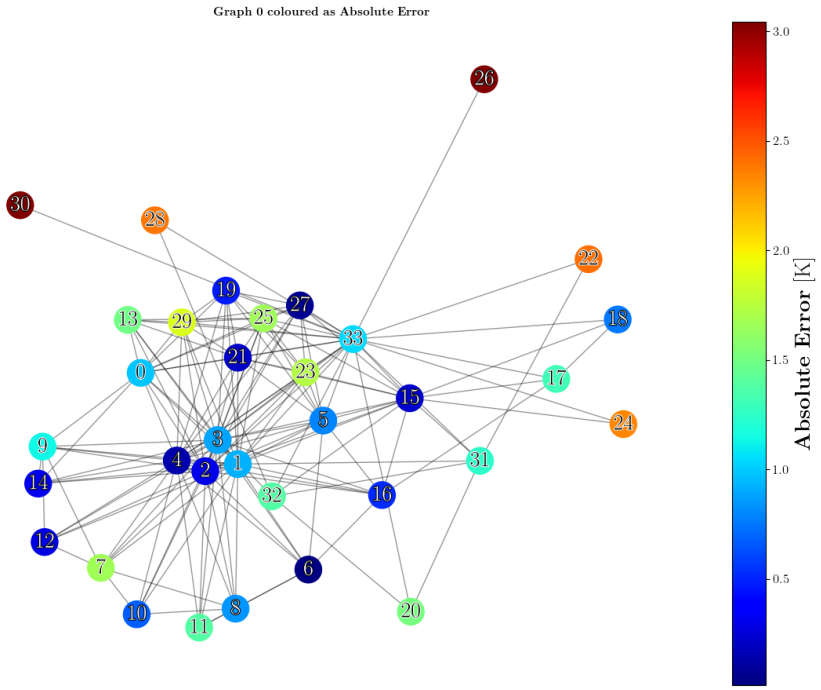


Figure 7.18: Absolute error in Kelvin per node. SAGE 3D symmetrical, with the MSE.

With the combined loss function and symmetrical connectivity

Although in the previous networks the largest error was located in nodes that do not dissipate power, a training run using the combined loss function was conducted, with the results shown in Table 7.13.

Table 7.13: Evaluation metrics and loss components of the SAGE 3D trained model with the combined loss function and the symmetrical Dataset.

Metric	**Value**
MSE	5.50 K^2
RMSE	2.35 K
MAE	1.88 K
Coefficient of Determination (R^2)	0.9791
Accuracy (%)	79.68%
Heaters Loss	1.86 K
Maximum error [K]	7.86
Mean Maximum error [K]	5.25

Achieving similar results than with the asymmetrical Dataset, being both far from the best results possible for this architecture. The maximum error obtained was of 7.86 K, the accuracy 79.68%

and the RMSE of 2.35 K.

The error pattern is similar to the one obtained using the MSE, with the maximum error in the Figure 7.19 below 6 K.

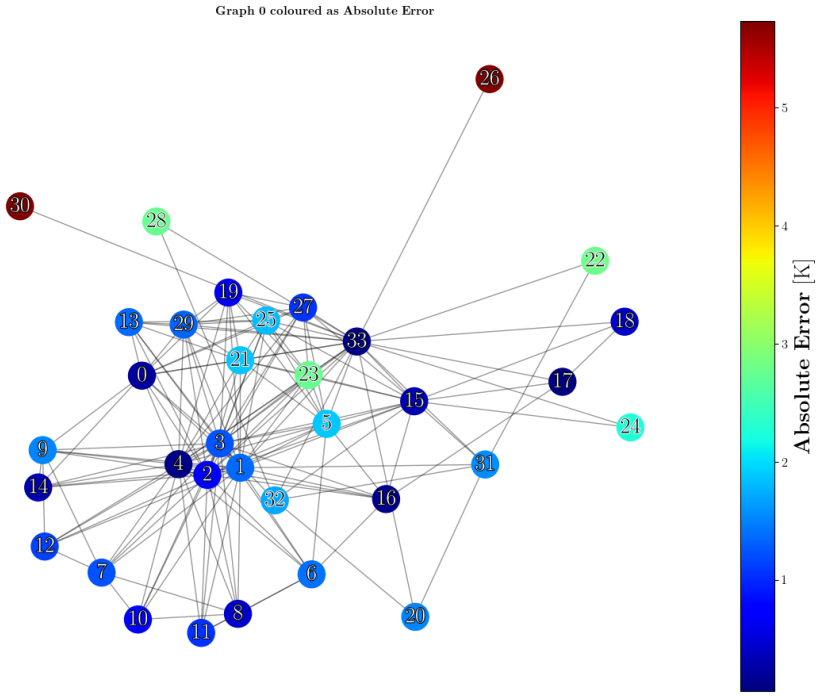


Figure 7.19: Absolute error in Kelvin per node. SAGE 3D symmetrical, with the combined loss function.

As shown in Figure 7.20, this is the Graph SAGE model that is furthest from over-fitting.

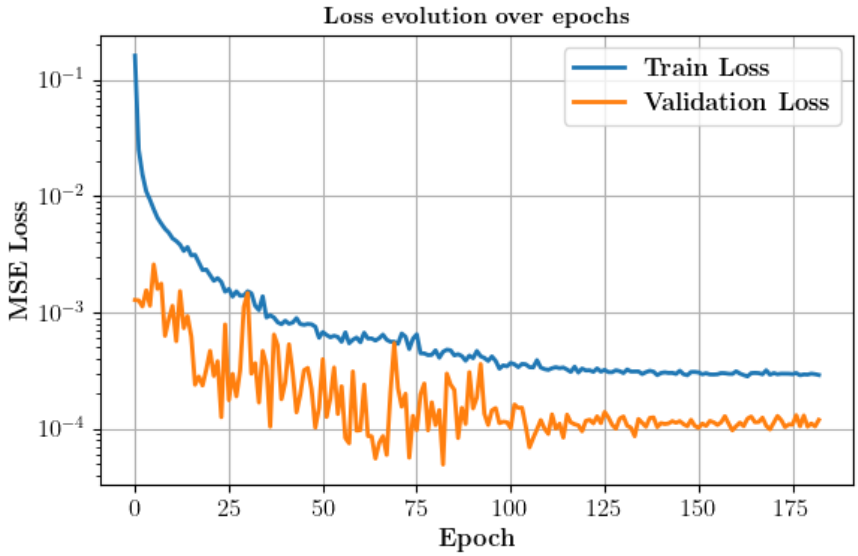


Figure 7.20: Evolution of training loss and validation loss, all normalized. SAGE 3D symmetrical, with the combined loss function.

7.2.4 Neural Network Convolution (NNConv)

Several modifications were made to the 2D NNConv architecture, including the addition of a final MLP at the output of the last layer. This MLP consists of two linear layers with a SiLU activation function in between, followed by a sigmoid activation function after the second linear layer. The network learns the weights of each edge using a SiLU activation function applied to the edge features, while the activation function used between layers is ReLU.

Since this architecture supports edge features, G_L and G_R were introduced as input features on the edges, as in the 2D PCB NNConv model. Because the network predicts normalized temperatures, these edge features also had to be normalized to prevent instabilities caused by mixing normalized node features with denormalized edge features, which could lead to exploding values or other training issues.

With these changes, the best combination of hyper-parameters found by Optuna are shown in Table 7.14, with an average time per epoch of 19.81 seconds.

Table 7.14: Hyper-parameters used for NNConv 3D training.

Hyper-parameter	**Value**
Number of Layers	9
Hidden Dimension	64
Dropout Rate	0.0009
Learning Rate	0.00169
Batch Size	64

With the loss function being only the MSE and asymmetrical connectivity

As it was done previously, this architecture was tested first with asymmetrical connectivity, with the results being showed in Table 7.15.

Table 7.15: Evaluation metrics and loss components of the NNConv 3D trained model only using the MSE and the asymmetrical Dataset.

Metric	**Value**
MSE	0.06 K^2
RMSE	0.24 K
MAE	0.18 K
Coefficient of Determination (R^2)	0.9994
Accuracy (%)	100.00%
Maximum error [K]	2.13
Mean Maximum error [K]	0.58
Time for training	97 minutes

This network achieved best results than any other model of the three aforementioned architectures, with an accuracy of 100%, a maximum error of 2.13 K, halving the best previous result, and a RMSE of 0.24 K.

Similar to what happened to the Graph SAGE with the symmetrical connectivity, the largest error is located in the nodes connected only with the environment node, as shown in Figure 7.21, with a maximum error of $\sim 0.7\,\mathrm{K}$.

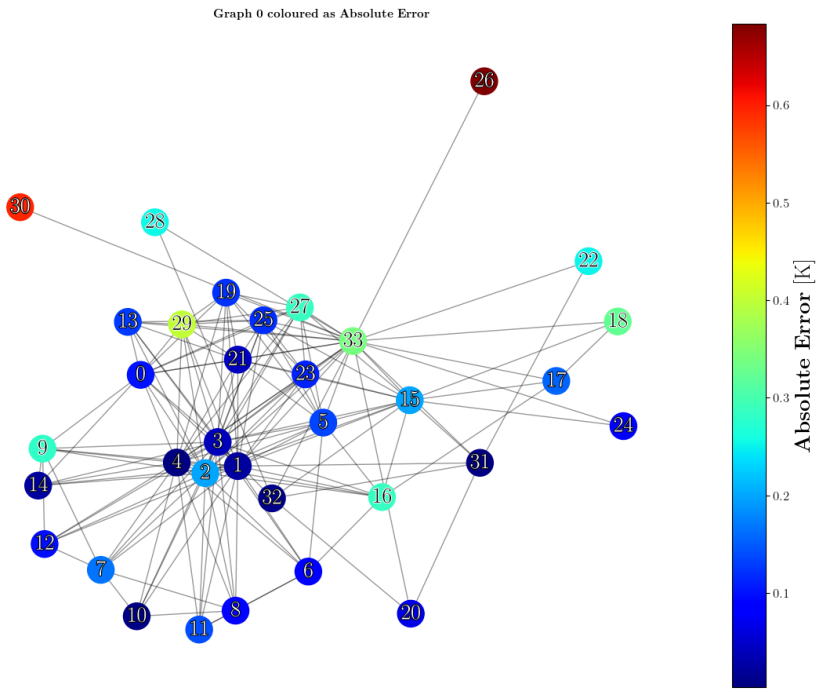


Figure 7.21: Absolute error in Kelvin per node. NNConv 3D asymmetrical, with the MSE.

With the loss function being only the MSE and symmetrical connectivity

As in the other architectures, the network was tested with the symmetrical connectivity graph, obtaining the results shown in Table 7.16.

Table 7.16: Evaluation metrics and loss components of the NNConv 3D trained model only with the MSE and the symmetrical Dataset.

Metric	**Value**
MSE	0.03 K^2
RMSE	0.18 K
MAE	0.13 K
Coefficient of Determination (R^2)	0.9999
Accuracy (%)	100.00%
Maximum error [K]	1.32
Mean Maximum error [K]	0.43

Once again, with the symmetric connectivity, the network performed better, nearly halving the previously best maximum error from 2.13 K to 1.32 K, while also reducing the RMSE to 0.18 K and maintaining the same accuracy (100%).

The error pattern varies slightly, but still follows the pattern of the largest error being in the nodes mainly connected with the environment node, with a maximum error in the Figure 7.22 of $\sim 0.35\,\mathrm{K}$.

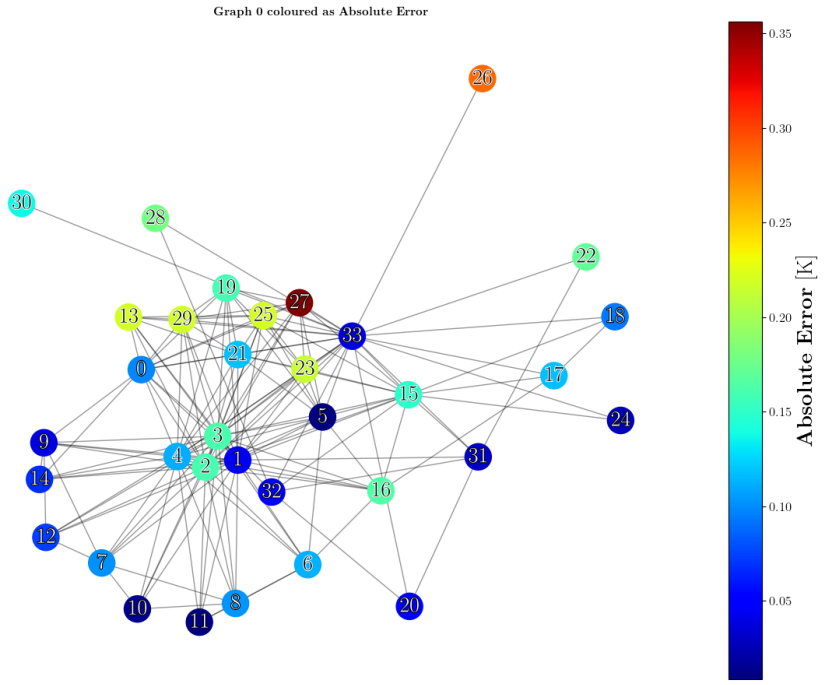


Figure 7.22: Absolute error in Kelvin per node. NNConv 3D symmetrical, with the MSE.

With the combined loss function and symmetrical connectivity

To conclude this exhaustive study, one last training was done, now with the combined loss function, obtaining the following results.

Table 7.17: Evaluation metrics and loss components of the NNConv 3D trained model with the combined loss function and the symmetrical Dataset

Metric	**Value**
MSE	0.06 K^2
RMSE	0.25 K
MAE	0.19 K
Coefficient of Determination (R^2)	0.9998
Accuracy (%)	99.99%
Heaters Loss	0.13 K
Maximum error [K]	3.58
Mean Maximum error [K]	0.63

This network performed the worst among the NNConv architectures, falling short of 100% accuracy by just 0.01%. It also exhibited the highest maximum error within the NNConv models (3.58 K) and an RMSE of 0.25 K, which is slightly higher than that obtained with asymmetrical connectivity.

The pattern differs slightly, as shown in Figure 7.23, with the node exhibiting the largest error not being primarily connected to the environment. However, the error still tends to accumulate near this region, with one exception where it extends slightly farther. The maximum error is approximately 0.5 K.

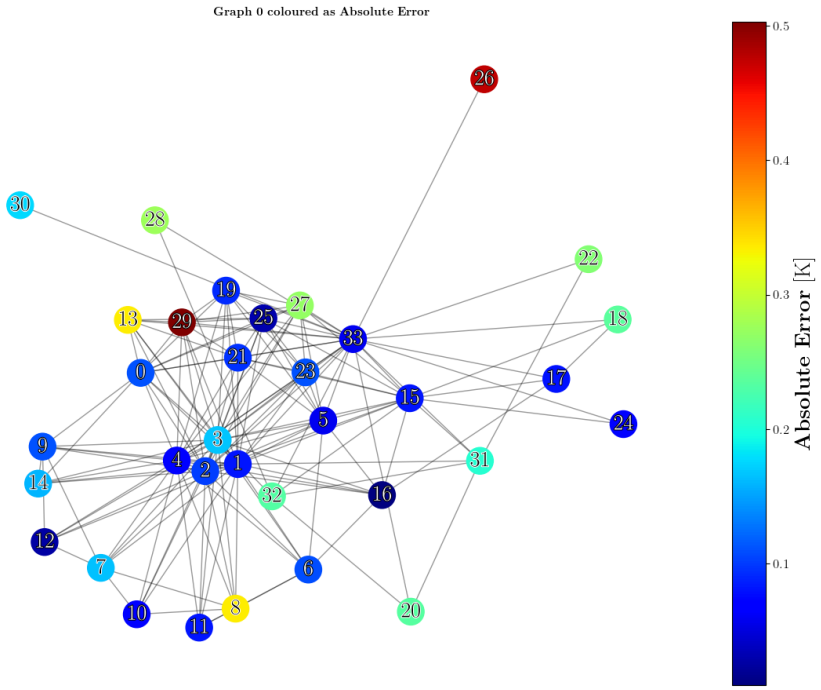


Figure 7.23: Absolute error in Kelvin per node. NNConv 3D symmetrical, with the combined loss function.

7.3 Comparison between the four different architectures

Unlike in the PCB problem, the combined loss function appears to degrade performance across all architectures. Additionally, implementing one-directional connectivity between the environment and the other nodes had the opposite effect of what was intended, leading to worse predictions compared to using symmetrical connections, as in the LPTM. In Table 7.18, the best results are presented, all of which were obtained using the MSE as the loss function, without the combined loss. All selected networks were trained with symmetrical connectivity between the environment and the other nodes. These two design choices were made with the aim of achieving the best possible network performance.

Table 7.18: Comparison of hyper-parameters and performance across GNN architectures, tested on the 3D ESATAN-TMS model.

Metric	**GCN**	**GAT**	**GraphSAGE**	**NNConv**
Number of Layers	3	4	4	9
Hidden Dimension	64	32	128	64
Batch Size	16	16	64	64
Learning Rate	0.00296	0.00147	0.00137	0.00169
Dropout Rate	0.1506	0.0048	0.0578	0.0009
Time per Epoch (s)	5.97	11.50	2.55	19.81
Root Mean Squared Error [K]	2.29	2.38	1.62	0.18
Maximum error [K]	10.88	5.29	5.83	1.32
Mean Maximum error [K]	5.25	1.78	3.45	0.43
Accuracy (%)	83.68	99.88	93.87	100.00
λ_H	N/A	N/A	N/A	N/A

Now that all of the architectures have been tested with a real 3D model, the flaws of the simpler architectures have reached the surface. Contrary to what happened in the 2D problem, the GAT and mostly the NNConv, have set a high standard in achieving good results, with an astonishing 99.88% and 100% of nodes between the threshold respectively. This is more remarkable due to the fact that the heat dissipated varies much more than in the PCB, and the 3D graph has more complex connections between nodes.

It is suspected that the reason the NNConv architecture achieves such strong performance is due to its ability to incorporate edge features. By leveraging these features, the network can learn the relative importance or weight of each connection. In this case, only G_L and G_R were used as edge features; however, these two values encapsulate several physical properties, such as emissivity, conductivity, and the distance between thermal identities, providing the network with rich and informative input.

Additionally, a key advantage of using only the MSE as the loss function is that models can be trained exclusively on real data, without the need to assume physical properties such as G_L or G_R.

Since it was demonstrated that the NNConv architecture performs best when trained on the symmetrical dataset using MSE as the loss function, a study was conducted to evaluate the evolution of training and validation loss over 100 epochs with varying dataset sizes. The motivation behind this analysis lies in the fact that solving a single case of a real 3D complex model is computationally expensive; therefore, reducing the number of training samples becomes a practical necessity.

As shown in Figure 7.24, the validation loss when the Dataset has 464 graphs is near the validation loss obtained with 10,000 graphs.

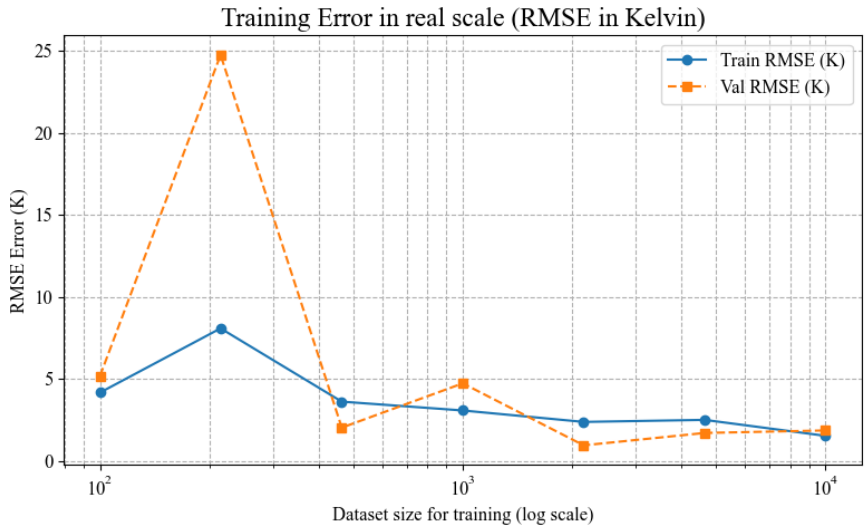


Figure 7.24: Evolution of the validation and training loss of a NNConv 3D model with the number of cases.

To evaluate whether the network can maintain accurate predictions when the dataset is reduced to 464 graphs, a full training was performed. The results are presented in Table 7.19, with an average training time per epoch of 0.96 seconds.

Table 7.19: Evaluation metrics and loss components of the NNConv 3D trained model only using the MSE and the reduced symmetrical 464 cases Dataset.

Metric	**Value**
MSE	0.58 K^2
RMSE	0.73 K
MAE	0.58 K
Coefficient of Determination (R^2)	0.9949
Accuracy (%)	99.87%
Maximum error [K]	3.71
Mean Maximum error [K]	1.73
Time for training	6 minutes

The maximum error is 3.71 K, which is particularly impressive considering that the dataset was reduced by more than a factor of ten. The network also achieved an accuracy of 99.87% and an RMSE of 0.73 K. These results surpass those of the best models from the other three architectures, suggesting that, for this particular problem, the NNConv is the most suitable architecture.

Finally, the error pattern seem to be similar to the one achieved with the 10,000 size Dataset, as it is shown in Figure 7.25.

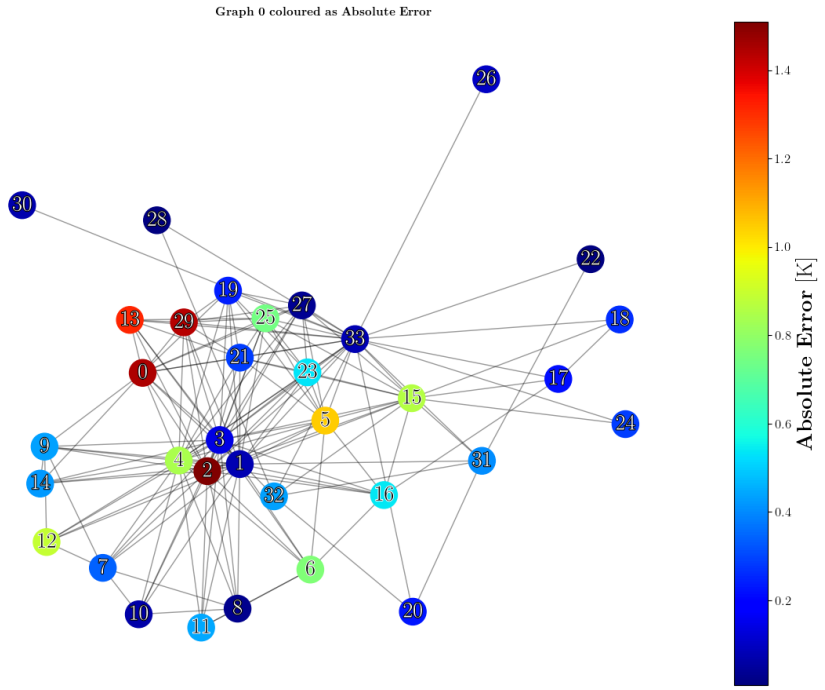


Figure 7.25: Absolute error in Kelvin per node. NNConv 3D symmetrical reduced Dataset, with the MSE.

To conclude, given that the error with 100 cases shown in Figure 7.24 is not so far, a last training was conducted, with the results being shown in Table 7.20.

Table 7.20: Evaluation metrics and loss components of the NNConv 3D trained model only using the MSE and the reduced symmetrical 100 cases Dataset.

Metric	**Value**
MSE	2.07 K^2
RMSE	1.33 K
MAE	1.08 K
Coefficient of Determination (R^2)	0.9691
Accuracy (%)	95.59%
Maximum error [K]	4.93
Mean Maximum error [K]	2.91
Time for training	6 minutes

These results are remarkable, considering that the RMSE (1.33 K) and the maximum error (4.93 K) surpass those of any other architecture trained with 100 times more data. Although the accuracy (95.59%) is lower than that achieved by the GAT and Graph SAGE models, the performance obtained with the reduced dataset remains highly impressive.

8

Conclusions and Future Steps

8.1 Conclusions

In this thesis, four of the many available Graph Neural Network architectures were evaluated. The initial tests were conducted on a two-dimensional PCB problem. Previous research on this 2D PCB employed Convolutional Neural Network models with auto encoders and fine-tuning; however, due to the similarity between graph-based approaches and the Lumped Parameter Thermal Method, GNNs present a more promising alternative.

After obtaining the initial results and identifying the regions where the GNN exhibited the highest prediction errors, a combined loss function was introduced. This loss function included three additional terms alongside the standard MSE: an MSE applied to the interface nodes, another to the heaters, and a physics-based term derived from the squared residual of the first law of thermodynamics. The relative weights of each term were empirically adjusted based on the performance of each architecture.

The results showed that the Graph SAGE and GCN architectures achieved the best performance among the four evaluated, with Graph SAGE emerging as the most promising for solving 2D problems. Using only the MSE as the loss function, the Graph SAGE model reached a maximum error of 6.15 K, an accuracy of 98.67%, and an RMSE of 1.20 K. It required an average training time of 42.04 seconds per epoch, which was the second lowest among the tested architectures. Overall, this architecture not only delivered the best predictive performance but also maintained a reasonable computational cost.

Following these encouraging results, a simplified ESATAN-TMS thermal model was developed to test scalability to more complex problems. While the model included fewer nodes and subsystems

than typical configurations, it retained key satellite features such as solar panels, payloads, batteries, and heat-dissipating instruments.

For this 3D model, the same four GNN architectures were tested. Among them, the NNConv and GAT demonstrated the highest accuracy in temperature prediction. The superior performance of GAT is attributed to its attention mechanisms, which allow the network to capture complex patterns related to geometry and spatial interactions—features that are essential in 3D thermal models. In contrast, NNConv benefits from its ability to incorporate edge features, such as conductivity and conductance, enhancing its predictive capability as demonstrated in the previous chapter.

The best results were obtained with the NNConv architecture, achieving 100% accuracy, a maximum error of 1.32 K, and an RMSE of 0.18 K. These metrics represent a significant improvement over all other architectures evaluated. Due to this strong performance, a dataset reduction study was conducted. Even when trained with only 100 cases, the NNConv model achieved an accuracy of 95.59%, a maximum error of 4.93 K, and an RMSE of 1.33 K. Notably, these results outperform the best models of the other architectures trained with 10,000 cases, and the accuracy remains comparable to that of the GAT and Graph SAGE models.

8.2 Future Work

Based on the outcomes of this thesis, several directions for future work are proposed:

- **Physics-Informed Network:** As demonstrated in the 2D PCB problem, incorporating the first law of thermodynamics into the loss function may improve prediction accuracy. Future models should explore the inclusion of this physical term, appropriately weighted, in more complex scenarios.
- **Complex 3D Models:** Given the proven ability of these architectures to accurately predict temperatures, future work should focus on increasing model complexity. Testing the networks on more realistic and detailed satellite configurations will help assess their scalability and generalization. This is particularly relevant since solvers like ESATAN-TMS require significantly more time to simulate each case compared to real-time GNN predictions.
- **Transient Simulations:** To make GNN-based models viable for real-world space applications, they must be extended to handle transient thermal simulations. This will require deeper exploration of GNN architectures capable of learning temporal dynamics. If necessary, a new architecture should be developed specifically to handle time-dependent behaviour.
- **Unified Architecture:** Ideally, a single architecture should be developed that integrates the strengths of the best steady-state and transient models. This unified network would not only handle both regimes but also support various complex geometries and topologies, enabling it

to serve as a general-purpose solver for thermal prediction problems where the domain can be discretized into a graph.

Bibliography

[1] R. Siegel, J. R. Howell, Thermal Radiation Heat Transfer, 3rd Edition, Hemisphere Publishing Corporation, 1992.

[2] A. Khusro, S. Husain, M. S. Hashmi, A. Q. Ansari, Small signal behavioral modeling technique of gan high electron mobility transistor using artificial neural network: An accurate, fast, and reliable approach, International Journal of RF and Microwave Computer-Aided Engineering 30 (4) (2020) e22112. doi:10.1002/mmce.22112.
URL https://doi.org/10.1002/mmce.22112

[3] Y. Geng, Q. Li, G. Yang, W. Qiu, Computer Vision and Natural Language Processing, Springer Nature Singapore, Singapore, 2024, pp. 275–304. doi:10.1007/978-981-97-3954-7_10.
URL https://doi.org/10.1007/978-981-97-3954-7_10

[4] Y. Hong, Q. Zhang, Indicator selection for topic popularity definition based on ahp and deep learning models, Discrete Dynamics in Nature and Society 2020 (2020) 1–11. doi:10.1155/2020/9634308.

[5] Diagrama de una célula lstm con puerta de olvido, https://es.wikipedia.org/wiki/Memoria_larga_a_corto_plazo, imagen vectorial de Wikipedia con licencia CC BY-SA 3.0; consultada: 2025-06-30 (–).

[6] K. Osanlou, C. Guettier, T. Cazenave, E. Jacopin, Planning and learning: A review of methods involving path-planning for autonomous vehicles, arXiv preprint arXiv:2207.13181, accessed: 2025-07-01.
URL https://arxiv.org/abs/2207.13181

[7] J. Meseguer, Spacecraft thermal control, Woodhead Publishing, Oxford, 2012.

[8] Z. Yang, A. D. Gaidhane, J. Drgoňa, V. Chandan, M. M. Halappanavar, F. Liu, Y. Cao, Physics-constrained graph modeling for building thermal dynamics, Energy and AI 16 (2024) 100346. doi:https://doi.org/10.1016/j.egyai.2024.100346.
URL https://www.sciencedirect.com/science/article/pii/S2666546824000120

[9] W. Kirchgässner, O. Wallscheid, J. Böcker, Thermal neural networks: Lumped-parameter thermal modeling with state-space machine learning, Engineering Applications of Artificial Intelligence 117 (2023) 105537. doi:https://doi.org/10.1016/j.engappai.2022.105537.
URL https://www.sciencedirect.com/science/article/pii/S0952197622005279

[10] X. Wang, Z. Long, K. Zhou, H. Xiao, X. Che, J. Liao, C. Li, Study on the temperature uniformity of workpieces inside an annealing furnace, International Journal of Thermal Sciences 212 (2025) 109798. doi:https://doi.org/10.1016/j.ijthermalsci.2025.109798.
URL https://www.sciencedirect.com/science/article/pii/S1290072925001218

[11] J. C. Simões, G. Carrilho da Graça, Experimental validation of neural network-based prediction of natural ventilation bulk airflow rate, Energy and Buildings 342 (2025) 115871. doi:https://doi.org/10.1016/j.enbuild.2025.115871.
URL https://www.sciencedirect.com/science/article/pii/S0378778825006012

[12] P. Purwono, A. Ma'arif, W. Rahmaniar, H. Imam, H. I. K. Fathurrahman, A. Frisky, Q. M. U. Haq, Understanding of convolutional neural network (cnn): A review, International Journal of Robotics and Control Systems 2 (2023) 739–748. doi:10.31763/ijrcs.v2i4.888.

[13] Geoelements gns theory, https://www.geoelements.org/gns/#/theory, accessed: 2025-06-30 (2025).

[14] Q. Hernández, A. Badías, F. Chinesta, E. Cueto, Thermodynamics-informed graph neural networks, IEEE Transactions on Artificial Intelligence 5 (3) (2024) 967–976. doi:10.1109/TAI.2022.3179681.

[15] M. Grmela, H. C. Öttinger, Dynamics and thermodynamics of complex fluids. i. development of a general formalism, Physical Review E 56 (6) (1997) 6620–6632. doi:10.1103/PhysRevE.56.6620.

[16] H. Sanchis-Alepuz, M. Stipsitz, Towards real time thermal simulations for design optimization using graph neural networks (2022). arXiv:2209.13348.
URL https://arxiv.org/abs/2209.13348

[17] Instituto Universitario de Microgravedad Ignacio Da Riva, Control térmico, https://www.idr.upm.es/es/upm-sat2?view=article&id=33:control-termico&catid=12, consultado: 6 de marzo de 2025.

[18] tec-science, Thermal conduction in solids, accessed: 2025-05-19 (n.d.).
URL https://www.tec-science.com/thermodynamics/heat/thermal-conduction-in-solids/

[19] J. Holman, Heat Transfer, 8th Edition, McGraw-Hill, 1997.

[20] Y. A. Çengel, Heat transfer : a practical approach, 2nd Edition, Mc Graw-Hill, Boston ; Madrid [etc], 2003.

[21] Sagredo, Umbra, penumbra, and antumbra cast by an opaque object, https://en.wikipedia.org/wiki/File:Umbra,_penumbra,_and_antumbra.svg, accessed: 2025-06-30 (2007).

[22] A. Kontaxoglou, S. Tsutsumi, S. Khan, S. Nakasuka, Towards a digital twin enabled multifidelity framework for small satellites, PHM Society European Conference 6 (1) (2021) 10. doi:10.36001/phme.2021.v6i1.2801.
URL https://doi.org/10.36001/phme.2021.v6i1.2801

[23] National Centers for Environmental Information (NCEI), Outgoing longwave radiation (olr) daily, accessed: 2025-03-07 (2025).
URL https://www.ncei.noaa.gov/products/climate-data-records/outgoing-longwave-radiation-daily

[24] European Space Agency, ESATAN-TMS Engineering Manual 2: Background Material, section 2.1: Lumped Parameter Method (2020).

[25] R. V. Southwell, Relaxation methods in theoretical physics, Reports on Progress in Physics 1 (1) (1934) 37–67.

[26] H. W. Emmons, Heat transfer in the electrical analogy, Transactions of the ASME 65 (1943) 51–58.

[27] G. M. Dusinberre, Analogies for heat transfer problems, Mechanical Engineering 76 (3) (1954) 221–226.

[28] J. F. Samaniego, 'deep learning' y 'machine learning': ¿en qué se diferencian?, publicado en el blog de Orange (oct 2021).
URL https://blog.orange.es/innovacion/deep-learning-vs-machine-learning/

[29] I. Goodfellow, Deep learning, Adaptive computation and machine learning, The MIT Press, Cambridge, Massachusetts, 2016.

[30] S. N, Understanding mean squared error in machine learning, accessed: 2025-04-25 (jan 2025).
URL https://futureagi.com/blogs/mean-squared-error-2025

[31] M. W. Ahmed, Understanding mean absolute error (mae) in regression: A practical guide, accessed: 2025-04-25 (aug 2023).
URL https://medium.com/@m.waqar.ahmed/understanding-mean-absolute-error-mae-in-regressio

[32] GeeksforGeeks, What is cross-entropy loss function?, accessed: 2025-04-25 (jan 2024).
URL https://www.geeksforgeeks.org/what-is-cross-entropy-loss-function/

[33] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359–366.

[34] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[35] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, International conference on machine learning (2017) 1263–1272.

[36] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. Battaglia, Learning to simulate complex physics with graph networks, Proceedings of the 37th International Conference on Machine Learning (ICML) (2020) 8459–8468.

[37] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, CoRR abs/1706.02216. arXiv:1706.02216.
URL http://arxiv.org/abs/1706.02216

[38] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks (2018). arXiv:1710.10903.
URL https://arxiv.org/abs/1710.10903

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014) 1929–1958.

[40] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 2010, pp. 249–256.

[41] F. Doshi-Velez, B. Kim, Towards a rigorous science of interpretable machine learning, arXiv preprint arXiv:1702.08608.

[42] X. Zhu, Semi-supervised learning literature survey, Computer Sciences Technical Report 1530, University of Wisconsin-Madison.

[43] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: International Conference on Learning Representations (ICLR), 2014.

[44] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, in: Journal of Machine Learning Research, Vol. 13, 2012, pp. 281–305.

[45] M. McCloskey, N. J. Cohen, Catastrophic interference in connectionist networks: The sequential learning problem, Psychology of Learning and Motivation 24 (1989) 109–165.

[46] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: Advances in Neural Information Processing Systems (NeurIPS), 2012.

[47] J. Bergstra, D. Yamins, D. D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, International Conference on Machine Learning (ICML) (2013) 115–123.

[48] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2623–2631.

[49] A. Barbosa Lizarbe, Análisis térmico estacionario de sistemas espaciales y generación de modelos surrogados mediante redes neuronales, presentado el 4 de julio de 2024 (julio 2024).

[50] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[51] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015, pp. 448–456.

[52] V. Mishra, Activation functions in focus: Understanding relu, gelu, and silu, `https : / / medium . com / @varun _ mishra / activation-functions-in-focus-understanding-relu-gelu-and-silu-841ed1c6df0c`, accessed: 2025-06-16 (2023).

[53] European Cooperation for Space Standardization (ECSS), Ecss-e-st-31c – thermal control, `https://ecss.nl/standard/ecss-e-st-31c-thermal-control/`, accessed: 2025-07-03 (2008).

[54] A. A. Hagberg, D. A. Schult, P. J. Swart, Networkx, `https://networkx.org/`, version 2.x (2008).