UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AERONÁUTICA Y DEL ESPACIO

GRADO EN INGENIERÍA AEROESPACIAL

TRABAJO FIN DE GRADO

Physics-Informed Neural Networks for Surrogate Modeling of Thermal Behavior in Space Systems

AUTOR: Ismael GALLO LÓPEZ

ESPECIALIDAD: Ciencias y Tecnologías Aeroespaciales

TUTOR DEL TRABAJO: David GONZÁLEZ BÁRCENA

Junio de 2025

Agradecimientos

A mi familia, especialmente a los que ya no están, porque sé lo orgullosos que estarían de ver lo que he conseguido hasta hoy. Sin vuestro apoyo, hoy no estaría donde estoy.

Todo mi esfuerzo va por vosotros.

A la gente que me ha acompañado estos 4 años, sobre todo a las personas que me llevo fuera de la uni y ya no son solo compañeros sino mis mejores amigos.
Nunca imaginé que un cabestrillo y un par de charlas en la academia me pudiesen dar tantas cosas buenas.

A David, por contar conmigo desde hace ya tanto tiempo y por confiar (en muchas ocasiones más que yo) en que este trabajo iba a salir adelante.

A mis compañeros de prácticas del IDR, que nos hemos provocado dependencia del descansito para el café a la hora exacta y que más que trabajar nos dedicábamos a chismear.

Abstract

This thesis explores the use of deep learning, specifically Physics-Informed Neural Networks (PINNs), to model the transient thermal behavior of PCBs in space applications. A Convolutional LSTM architecture is trained to predict the temperature distribution over time on a 13×13 grid, representing a simplified PCB. By incorporating physical constraints into the loss function, the model significantly improves generalization under limited-data scenarios. With as few as 50 training cases, the proposed PINN reduces the Mean Absolute Error (MAE) by more than 96% compared to a purely data-driven baseline.

The impact of the temporal resolution (Δt) and training set size is studied in detail. Results show that intermediate values of Δt, such as 10 s, provide the best trade-off between accuracy and stability. The use of physical loss terms also leads to narrower confidence intervals and improved robustness, particularly in low-data regimes. Finally, training cost and execution time are evaluated, confirming that although PINNs introduce overhead during training, their inference speed remains comparable to classical solvers.

The proposed approach demonstrates the potential of PINNs for thermal modelling in embedded systems, especially when simulation data is scarce. Several future directions are discussed, including generalization to variable time steps, flexible heater configurations, and integration into onboard systems.

Resumen

Este Trabajo de Fin de Grado explora el uso de redes neuronales profundas, en particular las redes neuronales informadas por la física (PINNs), para modelar el comportamiento térmico transitorio de placas de circuito impreso (PCBs) en aplicaciones espaciales. Se entrena una arquitectura basada en ConvLSTM para predecir la distribución de temperaturas en el tiempo sobre una malla de 13×13, que representa una PCB simplificada. Al incorporar restricciones físicas en la función de pérdida, el modelo mejora significativamente su capacidad de generalización en escenarios con datos limitados. Con solo 50 casos de entrenamiento, la PINN propuesta reduce el error absoluto medio (MAE) en más de un 96 % respecto al modelo puramente basado en datos.

Se estudia en detalle el impacto de la resolución temporal (Δt) y del tamaño del conjunto de entrenamiento. Los resultados muestran que valores intermedios de Δt, como 10 s, ofrecen el mejor compromiso entre precisión y estabilidad. El uso de términos físicos en la pérdida también conduce a intervalos de confianza más estrechos y una mayor robustez, especialmente en regímenes con pocos datos. Finalmente, se evalúan el coste de entrenamiento y el tiempo de inferencia, confirmando que, aunque las PINNs introducen sobrecoste en el entrenamiento, su velocidad de inferencia es comparable a la de los programas de cálculo clásicos.

El enfoque propuesto demuestra el potencial de las PINNs para el modelado térmico en sistemas embebidos, especialmente cuando la generación de datos de simulación es costosa. Se discuten varias líneas futuras, incluyendo la generalización a pasos temporales variables, configuraciones flexibles de calentadores y su integración en sistemas embarcados.

Contents

List of Figures

List of Tables

Acronyms

Adam Adaptive Moment Estimation.

ANN Artificial Neural Network.

CAD Computer-aided Design.

CFD Computational Fluid Dynamics.

CNN Convolutional Neural Network.

ConvLSTM Convolutional LSTM.

FC Fully Conected.

FEM Finite Element Method.

GNN Graph Neural Networks.

GPU Graphical Processing Unit.

GRU Gated Recurrent Unit.

ILSVRC ImageNet Large Scale Visual Recognition Challenge.

IR infrared.

LEO Low Earth Orbit.

LPTM Lumped-Parameter Thermal Modeling.

LR Learning Rate.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MLP Multi-layer perceptron.

MSE Mean Squared Error.

ODE Ordinary Differential Equations.

OSS Operational Spacecraft Simulator.

PCB Printed Circuit Board.

PINN Physics-Informed Neural Network.

POD Proper Orthogonal Decomposition.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

ROM Reduced Order Models.

SiLU Sigmoid Linear Unit.

SVD Single Value Decomposition.

TMM Thermal Matematical Model.

1

Introduction

Satellites are exposed to highly variable thermal environments and must be able to withstand extreme temperatures. Depending on their orbit, they may undergo rapid transitions from high temperatures to cryogenic conditions. To ensure the proper functioning of onboard instruments, the thermal control system must maintain component temperatures within safe operating limits. Efficient thermal management is thus essential for the reliability and lifespan of space systems.

Printed Circuit Boards (PCBs) are a core component of satellite electronics. They offer a compact and reliable platform for integrating electronic components and ensuring electrical connectivity. Copper traces on PCBs not only enable efficient current transmission but also contribute to heat conduction, which complicates the prediction of temperature distributions. In the transient regime, temperature evolution is affected by dynamic interactions between conduction, radiation, and boundary conditions, making accurate modelling even more challenging.

Traditional thermal models based on numerical methods can simulate transient thermal behavior with high accuracy, but they are often computationally expensive and time-consuming. Moreover, they require detailed knowledge of material properties and boundary conditions, which may not always be available or easy to update in real-time applications.

These limitations highlight the need for alternative approaches that are both efficient and adaptable, especially in scenarios where rapid thermal estimation is required or physical parameters are uncertain.

This thesis focuses on the use of Convolutional LSTM to predict the transient thermal behavior of PCBs. This architecture combines spatial and temporal processing, making it especially well-suited for modelling temperature maps over time. The model receives as input the initial temperature

distribution, boundary conditions, and other relevant variables, and generates a sequence of temperature maps as output.

Although the current work relies on direct training using synthetic data generated from a numerical simulator, it also lays the groundwork for future integration of physical constraints through Physics-Informed Neural Networks. In addition, alternative architectures such as CNNs, MLPs, and Transformer-based models are also explored and evaluated.

1.1 Structure of the thesis

This thesis is structured into chapters, each covering a key aspect of the work:

- **Chapter 2** introduces the fundamental concepts of deep learning, including neural network architectures, training strategies, and optimization techniques.
- **Chapter 3** presents an overview of the thermal environment in space and the key mechanisms and systems involved in satellite thermal control.
- **Chapter 4** reviews the current state of the art in data-driven thermal modelling, with a focus on the application of neural networks to heat transfer problems.
- **Chapter 5** defines the main objectives of the thesis, both general and specific, and establishes the scope of the work
- **Chapter 6** describes the physical modelling of the transient thermal behavior of PCBs, including heat transfer mechanisms, numerical simulation setup, and generation of training data.
- **Chapter 7** focuses specifically on the ConvLSTM architecture used in this work, detailing its internal structure, input/output format, and expected advantages for spatiotemporal modelling.
- **Chapter 8** describes the model development and training strategy, including data preprocessing, loss functions, training parameters, and evaluation procedure.
- **Chapter 9** presents and analyses the experimental results, comparing different architectures and training setups in terms of accuracy, stability, and generalization.
- **Chapter 10** concludes the thesis by summarizing the main contributions and findings, and outlining possible directions for future research.

Finally, the appendices provide complementary resources and documentation:

- **Appendix A** illustrates the node indexing scheme used in the finite difference solver, clarifying how spatial data is mapped onto a one-dimensional representation.
- **Appendix B** summarizes the alternative architectures explored during the early phase of the project. These include MLP-based regressors and Transformer-based decoders, which were ultimately discarded in favor of the ConvLSTM.
- **Appendix C** includes the complete ranking of all PINN-based models tested during the hyperparameter grid search. Results are ordered by global MAE and extend the summary presented in section 9.2.
- **Appendix D** provides full training time metrics for all tested models, including per-epoch durations and total runtime, to support the evaluation of computational efficiency.

2

Deep Learning fundamentals

Deep learning is a specialized branch of machine learning that uses multilayered Artificial Neural Networks (ANNs) to approximate highly complex functions. At its core, machine learning can be understood as a form of applied statistics—enhanced by computational power—that enables models to learn relationships directly from data without being explicitly programmed.

Within this field, deep learning stands out by leveraging depth (multiple layers of computation) to capture hierarchical patterns in data. This capability makes it particularly well-suited for problems involving nonlinearity, high dimensionality, or noisy measurements—features commonly encountered in thermal modeling of spacecraft systems.

The backbone of any deep learning system is the Artificial Neural Network (ANN), a mathematical framework that maps inputs to outputs through layers of interconnected neurons. To build a functional model, several essential components must be integrated:

- an optimization algorithm (subsection 2.2.5),
- a loss function to guide training (subsection 2.2.4),
- a dataset (subsection 2.2.3), and
- a suitable network architecture (section 2.3).

This chapter provides a comprehensive overview of these elements, as well as a comparison of different network types and their applications in the context of spacecraft thermal control.

2.1 Fundamentals of ANNs

An Artificial Neural Network is a computational model inspired by the structure of biological neural networks in animal brains [8]. It consists of a set of nodes, known as *neurons*, organized in layers and interconnected through weighted connections.

Each neuron receives an input—typically a real-valued vector—applies a linear transformation using learned weights and biases, and then processes the result through a nonlinear activation function. The output is then passed to neurons in subsequent layers.

Formally, an ANN approximates a function by composing multiple transformations of the form:

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

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

ANNs are typically organized into three types of layers:

- **Input layer**: receives the raw data (e.g., temperature, heat flux).
- **Hidden layers**: perform nonlinear transformations to extract patterns and representations.
- **Output layer**: generates the model's final prediction or classification.

The structure of the connections among neurons can vary significantly depending on the task. For example:

- In feedforward networks, information flows strictly from input to output.
- In recurrent networks, feedback loops allow temporal dynamics to be modeled.
- In convolutional networks, neurons are locally connected to exploit spatial structure.

In the following sections, we explore how these structures are implemented, trained, and applied to solve complex problems—specifically, the thermal modeling of spacecraft systems.

2.2 Characteristics and training of ANNs

2.2.1 Architecture

ANNs are typically organized into layers of neurons, as illustrated in Figure 2.1. These layers transform the input data into increasingly abstract representations as it flows from the input to the output.

The three main types of layers are:

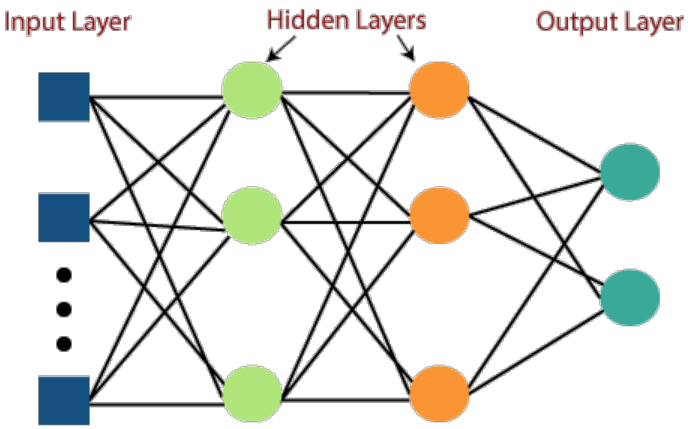


Figure 2.1: Diagram of a MLP [1]

- **Input layer**: receives the raw input data and passes it unchanged to the next layer. The number of neurons equals the dimensionality of the input vector.
- **Hidden layers**: apply linear transformations followed by nonlinear activation functions. Adding more hidden layers increases the model's ability to capture complex nonlinear relationships but also raises the risk of overfitting and increases computational cost.
- **Output layer**: generates the network's final prediction. For classification problems, this is usually a softmax or sigmoid output. In regression tasks—like those considered in this work—the output layer typically contains one neuron per output variable, such as temperature at each node.

Neurons between layers can be connected in different ways:

- **Fully Conected (FC) layers**: each neuron connects to every neuron in the next layer.
- **Pooling layers**: aggregate outputs over spatial neighborhoods, reducing dimensionality and sensitivity to local variations. These are typical in CNNs (see subsection 2.3.2).

Choosing the appropriate architecture—depth, width, and connection pattern—is critical and must be tailored to the problem at hand.

2.2.2 Hidden units and activation functions

Each hidden unit in a neural network applies a nonlinear transformation to its input. This transformation consists of an affine operation followed by an activation function:

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

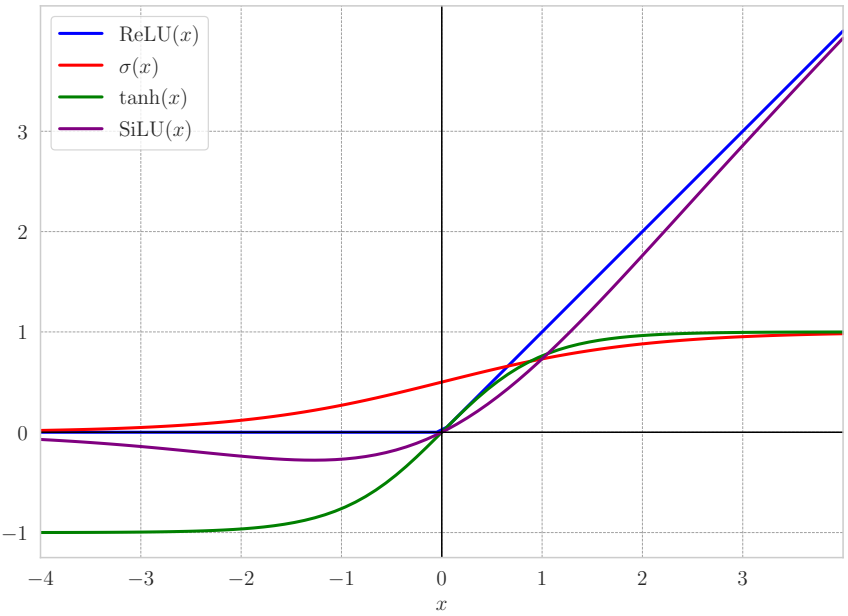


Figure 2.2: Graphical representation of common activation functions

where $\boldsymbol{x}$ 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.

Common activation functions include:

- **Rectified Linear Unit (ReLU)**: $g(z) = \max(0, z)$
- **Sigmoid**: $g(z) = \frac{1}{1+e^{-z}}$
- **Sigmoid Linear Unit (SiLU)**: $g(z) = z \cdot \sigma(z)$
- **Hyperbolic tangent (tanh)**: $g(z) = \tanh(z)$

All of them are represented in Figure 2.2.

While sigmoid and tanh functions are smooth and saturating, they can suffer from vanishing gradients, especially in deep networks. ReLU and its variants have become more common due to better gradient propagation and computational efficiency.

Input normalization is essential, especially when input variables differ in scale (e.g., $T \sim \mathcal{O}(10^2)$, $\dot{Q} \sim \mathcal{O}(10^0)$). Without normalization, learning may stall due to gradient saturation.

2.2.3 Dataset

The dataset forms the foundation of any deep learning application. It contains input–output pairs that the model uses to learn a mapping from inputs to desired targets.

In time-dependent problems, maintaining the temporal order of the data is crucial to preserve the system's dynamics.

Datasets are commonly classified as:

- **Experimental**: collected from real-world sensors or measurements.
- **Synthetic**: generated from physical simulations or numerical models.

To ensure proper training and evaluation, the dataset is typically divided into:

- **Training set**: used to optimize model parameters (70–80% of the data).
- **Validation set**: used to monitor generalization and adjust hyperparameters.
- **Test set**: held out during training; used for final performance evaluation.

Careful dataset splitting is critical to avoid data leakage and overfitting.

2.2.4 Loss function

The loss function measures the discrepancy between the network's predictions and the ground truth. It provides the optimization signal used to adjust the weights during training.

For classification problems, cross-entropy loss is commonly used [9]. In this work, the focus is on regression tasks such as predicting temperatures at specific nodes.

The most widely used loss function in regression is the Mean Squared Error (MSE):

$$L_{\text{MSE}} = \frac{1}{N_{\text{obs}}} \sum_{i}^{N_{\text{obs}}} (T_{i,\text{est}} - T_{i,\text{obs}})^2 \tag{2.1}$$

where $T_{i,\text{est}}$ is the estimated temperature at observation point i, $T_{i,\text{obs}}$ is the observed value, and N_{obs} is the number of observations.

To incorporate physical knowledge, the literature often introduces a physics-informed loss that enforces steady-state thermal balance. A common expression for this loss in Lumped-Parameter Thermal Modeling (LPTM) is given by [10]:

$$L_{\text{physics}} = \frac{1}{N_{\text{node}}} \sum_{i}^{N_{\text{node}}} \left(Q_i - \sum_j C_{ij}(T_{i,\text{est}} - T_{j,\text{est}}) - \sum_j R_{ij}(T_{i,\text{est}}^4 - T_{j,\text{est}}^4) \right)^2 \tag{2.2}$$

where Q_i is the input power at node i, C_{ij} the conductive coupling, R_{ij} the radiative exchange coefficient, and N_{node} the number of nodes.

We also include a penalty on boundary condition errors:

$$L_{\text{bnd}} = \frac{1}{N_{\text{bnd}}} \sum_{i}^{N_{\text{bnd}}} (T_{i,\text{est}} - T_{i,\text{bnd}})^2 \tag{2.3}$$

where $T_{i,\text{bnd}}$ is the imposed temperature at boundary node i, and N_{bnd} is the number of boundary nodes.

The total loss function is:

$$L = \lambda_{\text{MSE}} L_{\text{MSE}} + \lambda_{\text{phy}} L_{\text{physics}} + \lambda_{\text{bnd}} L_{\text{bnd}} \tag{2.4}$$

where λ_{MSE}, λ_{phy}, and λ_{bnd} are the weights for the regression, physics, and boundary losses, respectively.

In this work, we adopt a transient version of the physics loss, based on conservation of energy between time steps:

$$L_{\text{physics}}^{\text{trans}} = \frac{1}{N_{\text{node}}} \sum_{i}^{N_{\text{node}}} \left(\rho c e \Delta x \Delta y \cdot \frac{T_{i,\text{new}} - T_{i,\text{old}}}{\Delta t} - Q_i + \sum_j K_{ij} T_{j,\text{old}} + \sigma \sum_j E_{ij} (T_{j,\text{old}}^4 - T_{\text{env}}^4) \right)^2 \tag{2.5}$$

where $T_{i,\text{old}}$ and $T_{i,\text{new}}$ are the estimated temperatures at two time steps, ρ, c, and e are material density, specific heat, and thickness, Δx, Δy are spatial steps, Δt is the time step, Q_i is the power input, K_{ij} and E_{ij} are the conduction and radiation matrices, and σ is the Stefan–Boltzmann constant.

This transient loss is the one used throughout this thesis, replacing the steady-state term in equation (2.4) as the only modification to the total loss expression.

2.2.5 Gradient-based optimization

Training a neural network involves minimizing a loss function using iterative optimization. Most algorithms are based on gradient descent, which updates the weights in the direction that reduces the loss.

The update rule for a weight w_i at iteration n is:

$$w_i^{(n+1)} = w_i^{(n)} - \eta \frac{\partial L}{\partial w_i}\bigg|_{w_i^{(n)}} \tag{2.6}$$

Here, η is the learning rate, which controls the step size in the parameter space.

One of the most popular optimizers is the Adaptive Moment Estimation (Adam) algorithm [11],

which combines the advantages of momentum and adaptive learning rates. Momentum accelerates convergence by smoothing the update direction using an exponentially decaying average of past gradients, helping to navigate ravines and avoid oscillations. Adaptive learning rates, on the other hand, adjust the step size individually for each parameter based on the magnitude of recent gradients, allowing faster convergence and improved stability in scenarios with sparse or noisy gradients.

2.2.6 Hyperparameters

Hyperparameters are user-defined settings that influence the learning process but are not learned from the data. Key hyperparameters include:

- **Learning rate** (η): controls the size of weight updates. If too high, training may diverge; if too low, it may stall.
- **Epochs**: number of complete passes through the training data.
- **Batch size**: number of samples processed before updating weights.
- **Network architecture**: number of layers and neurons.
- **Activation function**: nonlinearity used in each neuron.

Selecting good hyperparameters is often done via grid search, random search, or expert tuning [12].

2.2.7 Training

Training an ANN is an iterative process in which the model adjusts its parameters to minimize the loss function using the training dataset.

The first distinction is between supervised and unsupervised learning. According to [8]:

- **Supervised learning**: learns to associate inputs with outputs from a labeled dataset.
- **Unsupervised learning**: identifies patterns or structures in unlabeled data.

This work focuses exclusively on supervised learning. The typical workflow of a supervised deep learning model includes:

1. **Parameter initialization**: define the architecture and initialize weights (usually randomly).
2. **Forward pass**: compute the model's output for a batch of inputs from the train set.
3. **Loss computation**: evaluate the error between predictions and ground truth using the loss function.

4. **Backpropagation**: compute gradients of the loss with respect to the weights.
5. **Weight update**: apply the optimization algorithm (e.g., gradient descent) to update weights.
6. **Epoch loop**: repeat for several epochs until convergence or a stopping criterion is met.
7. **Validation**: evaluate the model on the validation set after each epoch to monitor generalization.
8. **Testing**: after training, assess model performance on the test set using unseen data.

The two main goals of training are:

1. Minimize the training error (loss on the training set).
2. Minimize the generalization gap (difference between training and test error).

These objectives relate to two common pitfalls:

- **Underfitting**: the model is too simple to capture the underlying pattern, often due to insufficient capacity.
- **Overfitting**: the model fits the training data well but performs poorly on unseen data, typically due to excessive complexity.

Several techniques help mitigate overfitting [8, 13–15]:

- **Cross-validation**: split the dataset into multiple subsets to ensure generalization.
- **Early stopping**: halt training when validation loss no longer improves (see Figure 2.3).
- **Dropout**: randomly deactivate neurons during training to prevent co-adaptation.
- **Weight decay**: regularizes the model by penalizing large weights:

$$L_{\text{reg}} = L + \lambda \Omega(w) \tag{2.7}$$

  where λ is the regularization coefficient and $\Omega(w)$ is a norm of the weights. Common choices:

  - **L1 regularization**: $\Omega(w) = \|w\|_1 = \sum_i |w_i|$, promotes sparsity.
  - **L2 regularization**: $\Omega(w) = \|w\|_2 = \sqrt{\sum_i w_i^2}$, penalizes large weights uniformly.

- **Data augmentation**: increases dataset diversity, though not always applicable in scientific contexts.

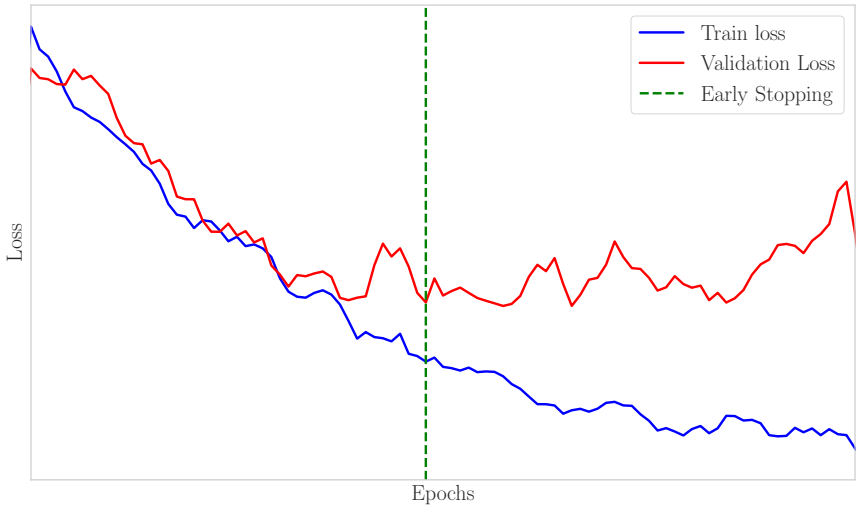


Figure 2.3: Illustration of early stopping

- **Model simplification**: reduce network size to match problem complexity.

Additional training practices include:

- **Learning rate decay** [16]: reduce the learning rate when validation loss plateaus, allowing faster convergence at first and fine-tuning later.
- **Batch normalization** [17]: normalize layer inputs to stabilize learning and prevent vanishing/exploding gradients.
- **Fine-tuning** [18]: adapt a pretrained model to a new task by retraining only selected layers, especially useful when data is scarce.

2.3 Types of ANN

This section presents several common architectures of artificial neural networks. While they share basic components—neurons, layers, activation functions—they are optimized for different data types and problem structures.

The models used in this work are selected based on the need to process spatial and temporal information relevant to spacecraft thermal modeling.

2.3.1 Multi-layer perceptron (MLP)

Multi-layer perceptrons (MLPs), or feedforward neural networks, are among the most basic and widely used architectures in deep learning. They are composed of a sequence of layers: an input

layer, one or more hidden layers, and an output layer. In each layer, every neuron is fully connected to the neurons in the next one.

The objective of an MLP is to learn an approximation of a target function f^* by training a parameterized model:

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$$

where $\boldsymbol{x}$ represents the input, $\boldsymbol{y}$ the predicted output, and $\boldsymbol{\theta}$ encompasses all the trainable parameters (weights and biases).

These networks are referred to as *feedforward* because data flows in a single direction—from the input to the output—without any feedback loops or recurrent connections. As a result, MLPs are not well-suited for tasks that require temporal context or memory of past inputs.

Some of the defining characteristics of MLPs are their layered architecture, composed of a well-defined sequence of input, hidden, and output layers, and their use of dense connections, meaning that each neuron in a given layer is connected to all neurons in the next. To enable the network to model nonlinear relationships, each linear transformation is followed by a nonlinear activation function, as discussed in subsection 2.2.2. Training is typically performed using gradient-based optimization techniques, most commonly through backpropagation combined with stochastic gradient descent. Additionally, MLPs operate with fixed-size inputs and outputs, requiring the dimensionality to be known in advance. Finally, they are stateless models: each input is processed independently, without any memory of previous inputs or temporal context.

The term "depth" in this context refers to the number of layers in the network. Increasing the depth allows the model to learn more complex and abstract patterns. However, deeper architectures also bring challenges such as overfitting and vanishing gradients, which require strategies like normalization, dropout, or careful initialization to mitigate.

2.3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) [19] are a type of neural network specifically designed to process data with a grid-like topology, such as images, time series, or structured spatial maps. The term "convolutional" comes from the use of convolution operations—mathematical filters applied to local input regions—to extract relevant features [8].

Unlike MLPs, which use fully connected layers where each neuron interacts with every input, CNNs typically perform sparse interactions. This is achieved by using kernels (filters) that are smaller than the input itself and slide across it to detect local patterns. This results in fewer parameters and more efficient computation.

A *kernel* is a small matrix of weights designed to extract specific features from the input (e.g., edges, corners, textures). By stacking multiple convolutional layers, the network builds a hierarchical representation: low-level features are captured in the initial layers, while deeper layers learn complex

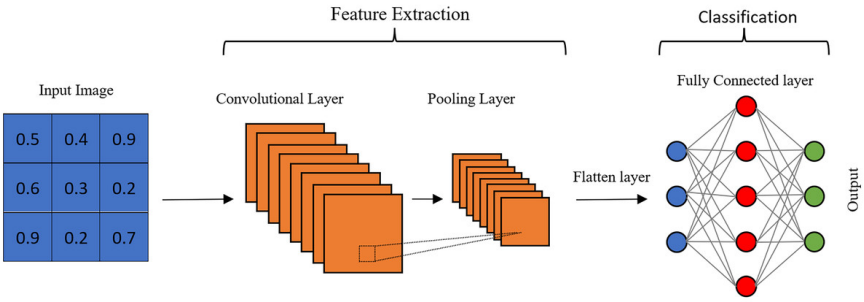


Figure 2.4: Schematic representation of a CNN [2]

structures.

For instance, in an image processing task, early kernels may detect vertical or horizontal edges, while later layers can recognize shapes or entire objects. This allows CNNs to work directly with raw data such as pixels, reducing the need for manual feature engineering.

In addition to convolutions, CNNs include *pooling layers*, which reduce the spatial resolution of feature maps while preserving the most salient information. Pooling provides robustness to small shifts or distortions in the input. The most commonly used pooling strategies are [20]:

- **Max pooling**: selects the maximum value in a region, emphasizing dominant features.
- **Average pooling**: computes the average value, smoothing the output.

Pooling contributes to translation invariance and further reduces the number of parameters, allowing CNNs to generalize better to slightly misaligned inputs.

CNNs can also handle variable-sized inputs, making them more flexible than traditional MLPs that require fixed input dimensions. However, a FC layer is usually included at the end of the CNN to aggregate features and produce a final decision or prediction.

As shown in Figure 2.4, the final layer in typical CNN architectures is a dense (FC) layer that maps the extracted features to the target variable—either a class label or a continuous value.

CNNs gained widespread attention after their success in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [21], where they dramatically outperformed traditional computer vision techniques. This marked the beginning of the modern deep learning era, demonstrating the potential of CNNs to solve high-dimensional tasks efficiently using GPUs.

Due to their ability to process structured data while reducing complexity and computation time, CNNs have become a standard tool in fields such as: image and video recognition, biomedical imaging, environmental and geospatial modeling, and scientific computing.

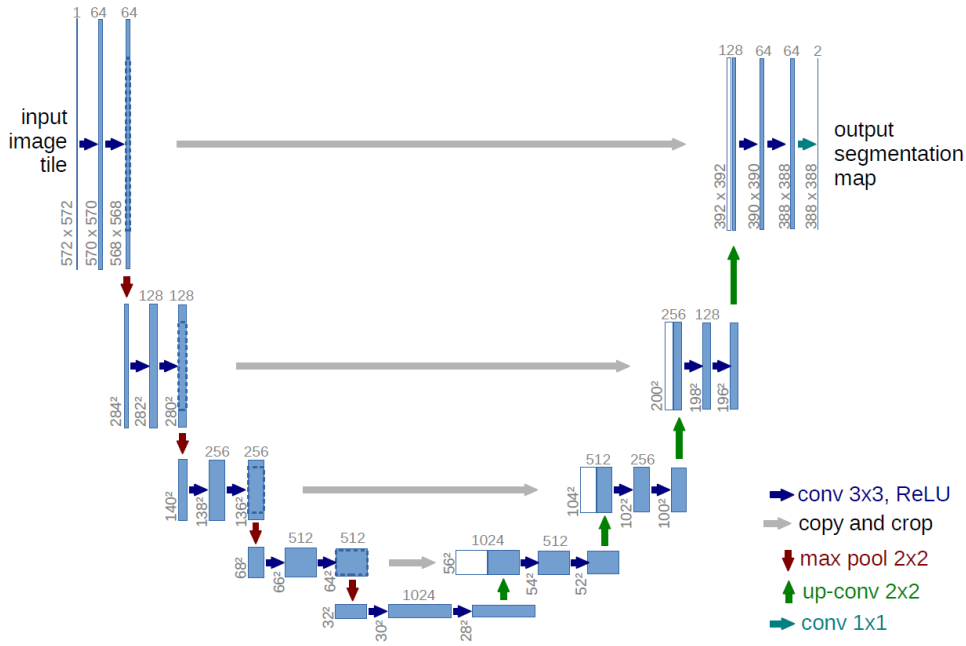


Figure 2.5: Example of a U-Net architecture [3]

2.3.2.1 U-Net

The U-Net architecture [3] is a specialized form of CNN originally developed for biomedical image segmentation. Its name derives from its symmetric, U-shaped design–as the one shown in Figure 2.5–, which features a characteristic encoder–decoder structure with skip connections.

The architecture consists of three main components:

- **Encoder**: progressively reduces the spatial resolution of the input while increasing the number of feature maps using convolution and pooling layers. This downsampling process captures increasingly abstract representations of the input data, similar to traditional CNNs.
- **Decoder**: symmetrically upsamples the encoded features using transposed convolutions (or upsampling layers). The goal is to recover the spatial resolution while refining feature localization.
- **Skip connections**: directly link layers of the encoder to their counterparts in the decoder. These connections enable the reuse of fine-grained spatial information that would otherwise be lost during the downsampling process, significantly improving output accuracy.

This design allows the U-Net to be both **efficient** and **accurate**, even when trained on relatively small datasets. Skip connections are especially important for preserving details, which is critical in segmentation tasks where spatial precision is paramount.

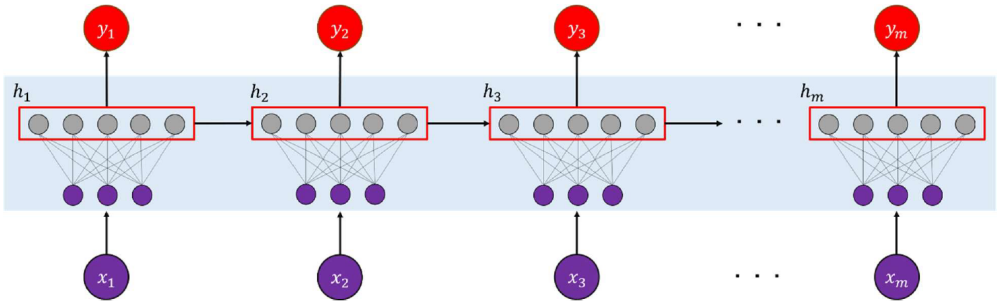


Figure 2.6: Schematic representation of an RNN [4]

Originally proposed for medical applications (e.g., tumor segmentation in microscopy images), the U-Net has since been successfully applied across a wide range of domains, including: biomedical imaging, satellite and aerial image analysis, geophysical modeling and thermal simulations of space systems.

In the context of this work, U-Nets are used to model spatially distributed temperature fields, taking advantage of their ability to localize and propagate information across multiple resolution levels. Their compact and symmetric structure makes them well-suited for scientific applications with strong spatial regularities and limited data availability, as shown in [22].

2.3.3 Recurrent Neural Network

Recurrent Neural Networks (RNNs) [23] are a type of neural network specifically designed to handle sequential data, such as time series, natural language, or audio. Unlike feedforward networks like MLPs or CNNs, RNNs possess an internal memory, which allows them to process and retain temporal information across input sequences.

At each time step t, the RNN receives an input $\boldsymbol{x}_t$ and updates its hidden state $\boldsymbol{h}_t$ based on both the current input and the previous hidden state $\boldsymbol{h}_{t-1}$:

$$\boldsymbol{h}_t = f(\boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_t \boldsymbol{h}_{t-1} + \boldsymbol{b}) \tag{2.8}$$

where f is a nonlinear activation function (typically tanh or ReLU), and $\boldsymbol{W}_x$, $\boldsymbol{W}_t$, and $\boldsymbol{b}$ are trainable parameters of the network.

This recursive structure allows the RNN to carry information forward over time. The process can be described in three steps:

1. The RNN receives a new input $\boldsymbol{x}_t$ at time t.
2. It uses its memory (previous state $\boldsymbol{h}_{t-1}$) to update the current hidden state.
3. It produces an output $\boldsymbol{y}_t$ based on the updated state.

Thanks to their memory mechanism, RNNs are particularly useful for modeling time-dependent processes. However, standard RNNs struggle to capture long-term dependencies due to the vanishing and exploding gradient problem during backpropagation through time (BPTT). To overcome this limitation, more advanced variants like LSTM and GRU have been developed.

2.3.3.1 Long Short-Term Memory

Long Short-Term Memorys (LSTMs) [24] are an advanced type of RNN specifically designed to address the vanishing and exploding gradient problems encountered in long sequences. They are capable of learning long-term dependencies by introducing a memory cell and a set of gates that regulate the information flow over time.

Each LSTM unit maintains a memory cell $\boldsymbol{c}_t$, which stores relevant information throughout the sequence. The flow of information into, out of, and within the cell is controlled by three gates:

- **Forget gate** $\boldsymbol{f}_t$: determines which part of the previous memory should be forgotten.
- **Input gate** $\boldsymbol{i}_t$: decides what new information should be stored in the memory.
- **Output gate** $\boldsymbol{o}_t$: regulates which part of the memory is sent to the output.

Mathematically, the update rules for an LSTM cell at time t are:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f\right) \tag{2.9a}$$
$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i\right) \tag{2.9b}$$
$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o\right) \tag{2.9c}$$
$$\tilde{\boldsymbol{c}}_t = \tanh\left(\boldsymbol{W}_c \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c\right) \tag{2.9d}$$

The memory cell is then updated by combining the retained memory and the new candidate values:

$$\boldsymbol{c}_t = \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + \boldsymbol{i}_t * \tilde{\boldsymbol{c}}_t \tag{2.10}$$

Finally, the hidden state $\boldsymbol{h}_t$ is computed using the updated memory:

$$\boldsymbol{h}_t = \boldsymbol{o}_t * \tanh(\boldsymbol{c}_t) \tag{2.11}$$

This architecture enables the LSTM to selectively retain or discard information and to adapt its internal memory dynamically. An important enhancement introduced by later studies is the use

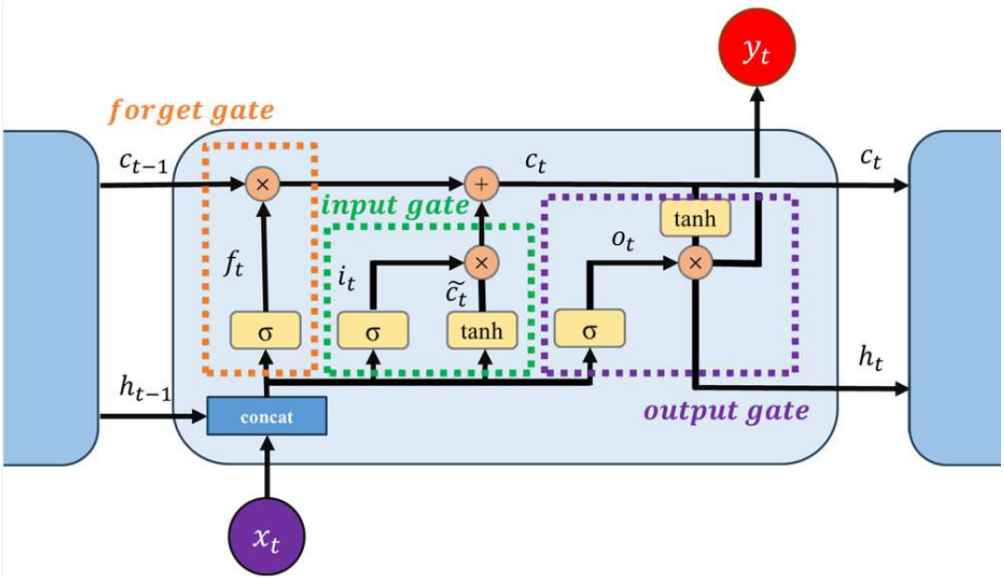


Figure 2.7: Functioning diagram of an LSTM [4]

of context-adaptive recurrence weights [25], which allow the model to adjust the integration step dynamically, offering greater flexibility and accuracy in temporal modeling.

2.3.4 Physics-Informed Neural Network

Physics-Informed Neural Networks [26, 27] are not a fundamentally new type of neural network architecture. Instead, they represent a training paradigm in which known physical laws are embedded into the learning process via modifications to the loss function.

Importantly, PINNs can be implemented using any standard architecture—such as MLPs, CNNs, or RNNs. Their defining characteristic lies in how they enforce the underlying physics of the problem during training, typically by including residuals of differential equations or conservation laws as additional loss terms.

By incorporating prior knowledge of governing equations, PINNs regularize the training process, effectively constraining the solution space and improving generalization. This becomes especially valuable in scientific and engineering domains, where data acquisition is costly or scarce, the underlying physical model is known (typically in the form of differential equations), and traditional numerical solvers are computationally expensive.

In essence, PINNs offer the ability to learn from both data and theory simultaneously, combining the strengths of data-driven models with the interpretability and robustness of physics-based approaches.

This synergy allows PINNs to produce accurate results with significantly fewer training samples compared to conventional ANNs, which may require large datasets to learn implicit patterns from

scratch. This advantage is particularly relevant in engineering fields where the system's behavior is well-characterized mathematically, but high-fidelity simulations or experiments are costly.

Recent studies have demonstrated the effectiveness of PINNs in a range of applications, such as:

- **Structural mechanics** [28]
- **Fluid dynamics** [29, 30]
- **Power systems** [31]

In the context of spacecraft thermal modeling, PINNs offer a promising new direction. By incorporating the steady-state or transient energy balance equations directly into the loss function (as shown in equation (2.2)), the network can better predict temperature distributions while maintaining physical consistency.

Although the use of PINNs in thermal space applications is still in its early stages [10, 32], current research suggests a significant potential to improve both accuracy and data efficiency in real-time thermal prediction models.

3

Thermal control in space systems

The focus of this work is the thermal modeling of spacecraft systems in Earth orbit. As a preliminary step, it is essential to characterize the thermal environment in which a body operates while orbiting the Earth.

While there is a wide range of mission profiles for space systems (e.g., planetary orbiters, deep space probes, interplanetary missions), this chapter is limited to Earth-orbiting spacecraft. With respect to mission type—such as scientific, telecommunications, or Earth observation—the impact on thermal analysis is generally less critical than the orbital environment itself.

The importance of thermal control lies in meeting the strict operational requirements of spacecraft subsystems, particularly the payload. All components must be maintained within specific temperature limits throughout the mission. Typical thermal requirements for various subsystems are summarized in Table 3.1.

Maintaining these thermal requirements is a challenging task, especially under the tight constraints of mass, volume, and power available to spacecraft subsystems. Therefore, to validate compliance during design, an uncertainty margin of ± 15 K is generally accepted in preliminary phases, and ± 5 K after correlation with experimental data [33]. These will be the thresholds used for model validation throughout this work.

3.1 Heat transfer mechanisms in space

In general, heat transfer occurs via three mechanisms: conduction, convection, and radiation [34]. However, in the space environment—where there is no atmosphere—convection can be neglected

Table 3.1: Thermal requirements for spacecraft subsystems [5]

Temperature range	$T_{\min}$ [°C]	$T_{\max}$ [°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	$\Delta T < 5$°C
High-resolution cameras	$\Delta T < 0.1$°C
Detectors (CCD)	$\Delta T < 0.01$°C

Temperature stability	
Electronics (housing)	$\frac{dT}{dt} < 5$°C/hour
CCD detectors during observation	$dT < 0.1$°C

due to the near-vacuum conditions [5]. Therefore, only conduction and radiation are considered in spacecraft thermal analysis.

3.1.1 Conductive heat transfer

Conduction is the transfer of heat within a solid or between solids in direct contact, driven by temperature gradients. At the microscopic level, it occurs through lattice vibrations and electron motion. The general behavior is governed by Fourier's law [34]:

$$\boldsymbol{q} = -k\nabla T, \tag{3.1}$$

where $\boldsymbol{q}$ is the heat flux vector (W/m^2), k the thermal conductivity of the material (W/m $\cdot$ K), and T the temperature field (K).

In Lumped-Parameter Thermal Modeling (LPTM), conduction between nodes i and j is expressed through a lumped conductance:

$$Q_{ij} = G_{Lij}(T_j - T_i), \tag{3.2}$$

where G_{Lij} (W/K) is the thermal conductance between nodes, and T_i, T_j the temperatures at those nodes. For homogeneous media, G_{Lij} is computed geometrically as:

$$G_{Lij} = \frac{kA}{L}, \tag{3.3}$$

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

When two solids are in contact, heat flow across the interface is reduced due to microscopic surface roughness and imperfect contact. This effect is modeled using the thermal contact conductance, h_c ($\mathrm{W/m^2 \cdot K}$), defined as:

$$h_c = \frac{\dot{Q}}{A\,\Delta T_c}, \tag{3.4}$$

where $\dot{Q}$ is the steady-state heat transfer rate across the interface, A is the nominal contact area, and ΔT_c is the temperature discontinuity at the contact surface.

The inverse quantity is the thermal contact resistance:

$$R_c = \frac{1}{h_c}. \tag{3.5}$$

This resistance can become the limiting factor in heat flow paths, especially in systems with low-pressure contacts or thermally resistive materials between layers. Accurate modeling of h_c is therefore essential for realistic simulation of conduction-dominated thermal networks.

3.1.2 Radiative heat transfer

Thermal radiation is the emission of electromagnetic waves by matter as a result of its temperature. It is a purely surface phenomenon that does not require a medium to propagate, which makes it the dominant heat transfer mechanism in the vacuum of space. For spacecraft thermal control, radiation governs both energy exchange with the environment and internal redistribution between components.

All bodies at a temperature $T > 0\,\mathrm{K}$ emit thermal radiation. The spectral distribution of this emission is described by Planck's law:

$$I_b(\lambda, T) = \frac{2hc^2}{\lambda^5 \left(\exp\left(\frac{hc}{\lambda kT}\right) - 1\right)} \tag{3.6}$$

In this expression, $I_b(\lambda, T)$ represents the spectral intensity of radiation emitted by a blackbody at temperature T and wavelength λ. The parameter h is Planck's constant ($6.626 \times 10^{-34}\,\mathrm{J \cdot s}$), c is the speed of light in vacuum ($2.998 \times 10^{8}\,\mathrm{m/s}$), and k is Boltzmann's constant ($1.381 \times 10^{-23}\,\mathrm{J/K}$). This formulation describes how the intensity of emitted radiation varies with wavelength and temperature.

The total power radiated over all wavelengths is given by integrating Planck's law, which leads to the Stefan–Boltzmann law for an ideal blackbody:

$$E_b = \sigma T^4, \tag{3.7}$$

where E_b is the emissive power ($\mathrm{W/m^2}$), T the surface temperature (K), and $\sigma = 5.669 \times 10^{-8}\,\mathrm{W/m^2 \cdot K^4}$ is the Stefan–Boltzmann constant.

A blackbody is an idealized surface that absorbs all incident radiation and emits the maximum possible radiation at each wavelength for a given temperature. Real materials are characterized by their emissivity $\varepsilon \in [0, 1]$, which quantifies the ratio of actual emission to that of a blackbody at the same temperature:

$$E = \varepsilon \sigma T^4. \tag{3.8}$$

Similarly, the absorptivity α quantifies the fraction of incident radiation that a surface absorbs. According to Kirchhoff's law of thermal radiation, at thermal equilibrium and for a given wavelength and direction, emissivity equals absorptivity:

$$\varepsilon(\lambda, \theta) = \alpha(\lambda, \theta). \tag{3.9}$$

In spacecraft applications, gray, diffuse surfaces are typically assumed, where emissivity and absorptivity are independent of wavelength and direction. For external surfaces, the balance between absorbed solar radiation and emitted thermal radiation determines the equilibrium temperature.

When two surfaces exchange radiation, the net transfer depends not only on their temperatures and emissivities, but also on their geometric relationship. This is captured by the view factor $F_{i \to j}$, which represents the fraction of radiation leaving surface i that directly reaches surface j. The view factor satisfies the reciprocity relation:

$$A_i F_{i \to j} = A_j F_{j \to i}, \tag{3.10}$$

where A_i, A_j are the surface areas. The full treatment of radiation exchange between multiple surfaces—including self-viewing, mutual shielding, and multiple reflections—is further addressed in section 3.3.

Although the exact computation of radiative exchange in spacecraft is complex and nonlinear, the fundamental idea remains that radiative energy loss grows rapidly with temperature and becomes the primary cooling mechanism in vacuum environments [5, 34].

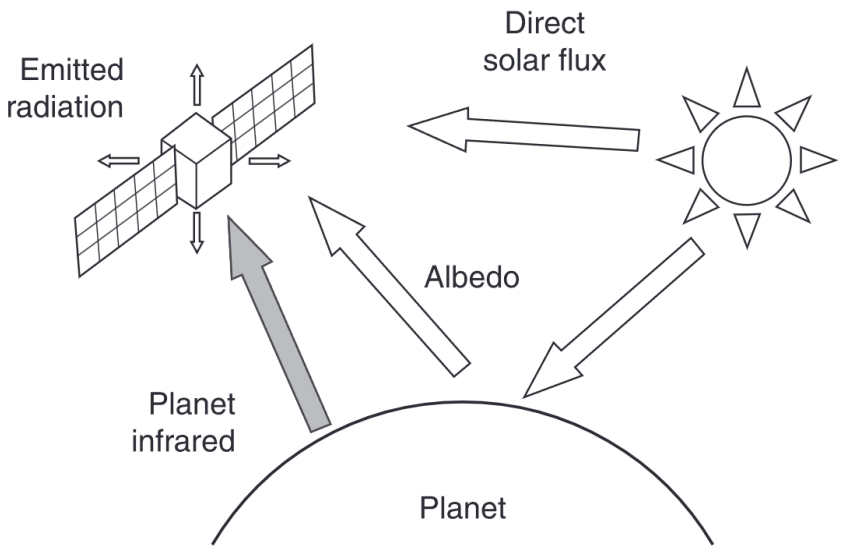


Figure 3.1: Heat fluxes in low Earth orbit [5]

3.2 Thermal environment in Low Earth Orbit

As discussed earlier, the space environment varies significantly depending on whether the mission is in planetary orbit, interplanetary space, or deep space. For spacecraft in Earth orbit, the dominant thermal loads originate from three sources: direct solar radiation, reflected solar radiation (albedo), and infrared radiation emitted by the planet itself [5]. An overview of these heat fluxes is shown in Figure 3.1.

Several unique properties of the space environment must be considered. First, the absence of an atmosphere means that conduction and convection with the surroundings are negligible. Second, the deep space background temperature is extremely low—around 2.73 K [35]—and is often modeled as a blackbody at this temperature.

3.2.1 Solar radiation

At a distance of 1 astronomical unit (AU) from the Sun—corresponding to Earth's orbit—solar radiation is the primary heat source for satellites. According to equation (3.7), the Sun can be approximated as a blackbody radiator at a temperature of 5762 K [36].

The nominal value of direct solar radiation incident on a surface normal to the Sun's rays at 1 AU is known as the **solar constant**, with a value of:

$$G_s = 1366.1 \text{ W/m}^2 1001 [37]$$

Although it is termed a "constant," this value actually fluctuates slightly due to solar activity cycles and the Earth's elliptical orbit around the Sun [38].

The thermal load absorbed by a planar surface from solar radiation can be estimated by:

$$\dot{Q}_{\text{Sun}} = \alpha G_s A \cos\theta \tag{3.11}$$

where α is the surface solar absorptance [5], A is the area of the surface, and θ is the angle between the surface normal and the incident solar rays.

3.2.2 Albedo

The second most significant thermal load comes from the **albedo**, defined as the fraction of incident solar radiation reflected by the Earth's surface or atmosphere. This contribution is shown in Figure 3.1 and is quantified by the albedo coefficient a.

This coefficient varies with surface type: oceans typically have $a \approx 0.05$–0.10, while snow-covered regions may reach up to 0.95 [5]. Cloud coverage also plays a significant role, with an average value of $a \approx 0.80$ in cloudy regions. For Earth orbit applications, a representative average value of $a = 0.3$ is commonly used [36].

Albedo only contributes when the portion of Earth visible to the satellite is illuminated by the Sun. The resulting thermal load is estimated by [5]:

$$\dot{Q}_{\text{alb}} = a\, G_s\, A\, F_{\text{SC-P}} \cos\phi \qquad -\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2} \tag{3.12}$$

where:

- a is the albedo coefficient.
- G_s is the solar constant.
- A is the surface area.
- ϕ is the solar zenith angle.
- $F_{\text{SC-P}}$ is the view factor between the spacecraft and the planet.

Albedo effects are more relevant for Low Earth Orbit, and become negligible at higher altitudes.

3.2.3 Planetary radiation

The third major heat flux is planetary infrared (IR) radiation, emitted by the Earth's surface and atmosphere. Like albedo, it varies throughout the orbit.

This contribution can be approximated by modeling the planet as a blackbody with an effective temperature T_p. A first-order estimate of T_p is obtained by balancing the absorbed and emitted radiation:

$$G_s \, \pi R_p^2 (1 - a) = 4 \, \pi R_p^2 \, \sigma T_p^4 \tag{3.13}$$

For Earth, with $a = 0.3$, the equivalent blackbody temperature is $T_p = 255$ K [36]. The corresponding radiative flux varies from 150 to 350 W/m^2 depending on the spacecraft's position, with a typical average of 230 W/m^2 at the surface.

The total heat received from planetary IR can be estimated as:

$$\dot{Q}_{\text{planet}} = \varepsilon \, A \, F_{\text{SC}-\text{P}} \, \sigma \, T_p^4 \tag{3.14}$$

where:

- ε is the IR emissivity of the spacecraft surface,
- A is the exposed area,
- $F_{\text{SC}-\text{P}}$ is the spacecraft–planet view factor.

3.3 Thermal Matematical Model

Developing a Thermal Matematical Model is a fundamental step in the thermal analysis of spacecraft systems. The goal is to construct a mathematical representation of the system that accurately captures its thermal behavior under expected operational conditions.

The process begins with the construction of a geometric model of the spacecraft using a CAD tool or equivalent. This model provides key geometrical information such as areas exposed to heat fluxes and view factors between surfaces. It does not need to be fully detailed—only the geometry relevant to thermal behavior must be preserved.

Once the geometry is established, the model is discretized into a network of thermal nodes. Each node i is treated as an isothermal element characterized by its temperature T_i and thermal capacitance C_i. This approach, known as the *lumped parameter method*, replaces the continuous thermal distribution with a finite set of discrete elements [5].

The general energy balance equation for each node is:

$$C_i \frac{dT_i}{dt} = \dot{Q}_{\text{sun},\,i} + \dot{Q}_{\text{alb},\,i} + \dot{Q}_{\text{planet},\,i} + \dot{Q}_{\text{dis},\,i} + \sum_{j=1}^{n} K_{ij} \, (T_j - T_i) + \sum_{j=0}^{n} R_{ij} \, (T_j^4 - T_i^4) \tag{3.15}$$

where:

- $\dot{Q}_{\text{sun},\,i}$, $\dot{Q}_{\text{alb},\,i}$, and $\dot{Q}_{\text{planet},\,i}$ are the external thermal loads on node i due to direct solar radiation, albedo, and planetary radiation, respectively.
- $\dot{Q}_{\text{dis},\,i}$ is the internally dissipated power in the node (e.g., electronics).
- K_{ij} and R_{ij} are the conductive and radiative couplings between nodes i and j, respectively.[1]
- The index $j = 0$ refers to deep space, treated as a blackbody at 2.73 K.

With all relevant properties (geometry, material, dissipations, environment) and coupling matrices known, equation (3.15) defines a system of Ordinary Differential Equationss. Solving this system yields the temperature evolution of all nodes over time, enabling both steady-state and transient thermal analysis.

[1] Note that $K_{ij} = K_{ji}$ and $R_{ij} = R_{ji}$ due to reciprocity.

4

State of the art

This chapter reviews the current state of the art in the use of ANNs for thermal modeling, with a particular focus on space applications. The review addresses the types of networks employed, their typical architectures, implementations as surrogate models, their integration with Lumped-Parameter Thermal Modeling (LPTM), and their performance in both steady-state and transient thermal problems.

Empirical modeling techniques face significant limitations when dealing with noisy or uncertain data, and their computational cost tends to increase rapidly when solving non-linear systems. In contrast, ANNs have proven to be a powerful alternative for addressing such issues, particularly in thermal modeling tasks.

Among various application domains, heat transfer has emerged as a particularly promising field for ANN implementation. Since most thermal modeling problems are governed by the laws of thermodynamics, they often share similar solution strategies [39]. Over the past two decades, numerous researchers have proposed ANN-based approaches for thermal problems. For instance, Souayeh et al. developed an ANN model for heat transfer and fluid flow [40]; Yadav and Chandel focused on modeling solar radiation using ANN techniques [41]; and Kalogirou provided an early application of ANN in combustion process modeling [42].

For transient thermal problems, various ANN-based solutions have been explored. One notable example is presented by Heng et al. [43], who applied the superposition principle to compute the thermal response of a parabolic solar collector. They employed a feedforward ANN to estimate the system's response to individual heat flux pulses, subsequently summing each contribution to obtain the overall daily thermal response. The ANN model achieved a mean absolute deviation of less than 2 K in less than one minute, in stark contrast to the traditional FEM simulation, which required

three to four hours. Training the network took approximately 12 hours using a dataset comprising 29,884 samples.

Sikka et al. [44] demonstrated a practical application of feedforward ANNs for steady-state thermal prediction in chip-on-heat spreader geometries, a domain typically addressed using FEM. Their regression model included between one and three hidden layers, with five to twenty-five neurons per layer, and was trained on 3,500 data points generated from a combination of analytical solutions and ANSYS simulations. Input features were derived from physically meaningful parameters and embedded into a compact, non-dimensional space.

Two training approaches were evaluated: conjugate-gradient descent with L2 regularization, and the Levenberg–Marquardt algorithm with Bayesian regularization. The latter yielded better accuracy and generalization performance, achieving a test MSE as low as 2.9×10^{-3} while requiring significantly fewer training data.

Importantly, the authors emphasized the impact of carefully designed input embeddings and network depth on predictive accuracy, while also acknowledging the risks of over-parameterization. Their work stands as a compelling example of PINN design that balances computational efficiency with predictive precision in thermal modeling tasks.

Across the reviewed literature, it has been consistently shown that ANNs offer several advantages in thermal modeling. These include the ability to capture complex nonlinear relationships, handle high-dimensional input spaces, and approximate solutions to problems where the governing physical equations may be partially unknown or too costly to solve using conventional methods. Additionally, ANNs significantly reduce computational time compared to traditional numerical simulations, making them highly attractive for real-time or large-scale applications.

However, these approaches also present a number of limitations. Chief among them is the need for extensive, high-quality datasets for training. Moreover, ANNs often lack physical interpretability, are highly sensitive to network architecture and hyperparameter choices, and their predictive accuracy tends to degrade when extrapolating beyond the training range. These challenges highlight the importance of integrating domain knowledge—such as through physics-informed approaches—to enhance robustness and generalization.

In spacecraft thermal control, ANN applications have been studied since the early 2000s [45], and have gained renewed interest in recent years due to the increased availability of computational power and the development of open-source libraries tailored for ANN modeling [46].

Reis et al. [47] proposed the use of ANNs to enable real-time thermal prediction onboard spacecraft, supporting the Operational Spacecraft Simulator (OSS) in planning in-orbit maneuvers. Traditionally, thermal computations performed by the OSS rely on complex and computationally intensive models, which necessitate simplifications—such as reducing the number of time steps or using coarse approximations—to operate in real-time.

The same research group later applied this methodology to a small *CubeSat* [48]. In that work,

29 scenarios were simulated using a thermal solver, including worst-case Hot and Cold cases, extreme parameter values (maximum, minimum, and average), and two additional arbitrary cases to evaluate ANN generalization. A classical MLP architecture was implemented, using 7 input features (time, power from three components, solar radiation, albedo, and Earth radiation), two hidden layers with 30 to 50 neurons each, and 9 output neurons corresponding to the temperatures at 6 external surfaces and 3 internal components.

The training was carried out using two distinct datasets generated from 10 orbits of each scenario: the first consisting of the last orbit from 14 scenarios, and the second including the last 5 orbits of all scenarios except the two arbitrary ones. The hyperparameters were set as follows: a learning rate of 0.01, momentum constant of 0.5, error tolerance of 10^{-4}, and a maximum of 10^6 epochs to prevent non-convergence.

With the first dataset, the ANN achieved an error of less than 1 °C for the Hot and Cold scenarios. However, for the unseen cases A and B, the maximum error increased to 3.04 °C. As discussed in chapter 3, errors below 5 °C are considered acceptable, and thus, these results were deemed satisfactory. With the second dataset, while the ANN maintained good accuracy on trained scenarios, the mean error in the A and B cases rose to 3.59 °C and 5.59 °C, respectively. Moreover, the maximum error observed exceeded 20 °C, well beyond the acceptable threshold, which undermined the model's generalization performance.

In subsequent work, the same authors explored the application of ANN modeling to larger spacecraft, specifically the Amazonia-1 satellite, a 650-kg Earth observation platform [49]. In this study, 12 scenarios were simulated using a thermal solver: two representing worst-case Hot and Cold conditions, and ten additional cases selected randomly from intermediate parameter ranges (labeled A to J).

For this case, the chosen architecture was a Recurrent Neural Network (see subsection 2.3.3) with 95 input features, one hidden layer containing 20 neurons, and 50 output neurons. Several training datasets were used, beginning with only the Hot and Cold scenarios and progressively incorporating scenarios A through E. The largest dataset, referred to as HotColdABCDE, included all these scenarios presented in random order to reduce overfitting. The hyperparameters were set to a tolerance of 10^{-5}, a learning rate of 0.01, and a momentum constant of 0.5. Training was performed over 1,000 epochs.

The evaluation of model performance revealed that the mean generalization error[1] decreased from 9.4 °C when trained only on the HotCold dataset to 3.9 °C with the HotColdABCDE dataset. However, in none of the training scenarios did the model achieve a maximum generalization error per equipment below 5 °C. As a result, the proposed approach was deemed not viable for operational deployment in its current form.

Later work in this field by Tanaka and Nagai [10] introduced the use of Physics-Informed Neural

[1]Computed as the average error across cases F, G, H, I, and J.

Network (PINN)s for spacecraft thermal analysis. Their approach aimed to estimate the actual temperature distribution of a satellite by linking observational data to the system's underlying TMM. In this framework, the loss function incorporates three components: a term based on observed data (typically the MSE), a term derived from the governing heat balance equations in steady-state, and a contribution accounting for boundary condition violations, as detailed in subsection 2.3.4.

The study considered a pseudo small-satellite measuring 50 cm and discretized into 100 thermal nodes. Each node was assigned heat dissipation characteristics corresponding to onboard devices, along with physical parameters such as thermal conductance and emissivity. The ANN's hyperparameters were selected via grid search, resulting in an architecture with two hidden layers of 40 neurons each. Training was carried out using the Adam optimizer with a learning rate of 0.0005, a stopping tolerance of 5×10^{-4}, and a total of 200,000 epochs. The SiLU activation function was employed throughout, and implementation was done using PyTorch.

Training results showed consistent convergence after approximately 100,000 epochs. While both the physics-based and MSE loss components decreased steadily, the boundary condition term remained unstable and failed to improve significantly. The average prediction error across three different sensor configurations ranged from 3.7 K to 5.5 K, with maximum errors exceeding 8.0 K in all cases—above the commonly accepted threshold of 5 K. The authors noted one key advantage of PINNs: their ability to train effectively with smaller datasets compared to conventional ANNs. However, the study did not specify the number of training samples used, nor did it provide precise timing data for the training process. POD modes were computed using singular value decomposition (SVD) on a prior dataset of size m.

Continuing along the same line of research, Tanaka and Nagai proposed the development of surrogate models for Physics-Informed Neural Networks using POD-based data reduction techniques [32]. Building upon the system and TMM described in their earlier work [10], this study introduced the use of Proper Orthogonal Decomposition (POD)—a mathematical technique for extracting low-dimensional features from complex data [50]. The goal was to capture the essential characteristics of spacecraft temperature distributions in a reduced-order form.

Through this approach, the temperature distribution of an n-dimensional model, $T_n \in \mathbb{R}^n$, can be approximated with sufficient accuracy using a set of r POD coefficients, $\alpha \in \mathbb{R}^r$, where $r \ll n$. To incorporate POD into the physics-informed loss function, the authors trained an ANN that outputs the POD mode coefficients. These coefficients were then transformed back into the full temperature field in order to compute the residuals of the heat balance equation, independently of any observational data. For comparison purposes, the study also included a standard ANN model using POD without the physics-based loss formulation.

To evaluate the proposed methodology, two satellite models were developed. Model A represented a small satellite consisting of 100 thermal nodes—similar to the configuration used in the previous study—while Model B corresponded to a larger satellite with 1,464 nodes. Heat input loads were assigned using Gaussian-distributed values to simulate the power dissipation of onboard devices.

The SiLU activation function and the Adam optimizer were selected for both models. The learning rate was set to 10^{-4}, and training was conducted over 500 epochs. Several architectural configurations were tested, including networks with between one to four hidden layers and varying numbers of neurons per layer (60, 90, 120, 150, and 180). All other hyperparameters followed the default settings provided by PyTorch.

The ANN architecture was optimized through a parametric study involving 1,600 training cases, 100 prior samples for SVD decomposition, and 40 POD modes for Model A. The optimal configuration selected for Model A consisted of a single hidden layer with 150 neurons, while Model B performed best with four hidden layers of 150 neurons each. A grid search over the number of prior data samples and POD modes revealed that results stabilized for $m \geq 100$; thus, $m = 100$ was chosen. Regarding the number of POD modes, $r = 40$ was selected since it preserved 99.75% of the cumulative singular values for Model A and 91.92% for Model B. The final training dataset size was set to 3,200.

Under the configuration $(m, r, N_{\text{train}}, N_{\text{test}}) = (100, 40, 3200, 500)$, the average nodal standard deviation for Model A was 0.6 K, with the most significant deviation reaching 1 K in one of the panels. For Model B, which featured a more complex architecture and a higher node count, the average error standard deviation increased to 2.0 K, with a maximum deviation of 12.7 K. The largest errors were localized in isolated elements of the structure.

Despite the higher overall error in Model B, the temperature prediction errors remained within acceptable bounds for critical regions. Specifically, the error in the inner decks—where onboard devices were located—remained below 2 K for Model A and below 4 K for Model B, thus satisfying the typical accuracy requirement of 5 K for thermal analysis in spacecraft systems.

Lastly, the computational cost of training and inference was evaluated. For Model A, training the POD-PINN model required 130.6 minutes, while the POD-ANN model took significantly longer—approximately 677.2 minutes. For Model B, the corresponding training times were 676.6 minutes and 934.2 minutes, respectively.

In terms of inference, the execution time for 500 thermal simulation cases using the full thermal solver was 5,755.8 seconds for Model A. In comparison, the POD-PINN and POD-ANN models completed the same task in just 0.13 seconds. For Model B, the solver required 7,562.4 seconds, whereas both reduced-order models completed the simulation in only 0.17 seconds.

These results clearly demonstrate the potential of surrogate modeling techniques, particularly when coupled with POD and physics-informed frameworks, to dramatically reduce computation time in large-scale or real-time thermal analysis applications.

More recently, Yamashita et al. combined POD mode reduction with RNN and LSTM architectures to address transient thermal analysis challenges [4]. The fundamental principles of POD remained consistent with prior studies, serving as a reduced subspace that captures the dominant spatial features of temperature distributions in spacecraft.

The satellite thermal model used in this work was the same 100-node CubeSat configuration as Model A in previous studies. The networks were tasked with predicting temperature evolution over a full orbital period of 6,052 s, using a time step of 62.5 s.

To account for uncertainties in the TMM, the authors introduced variability in the heat generation of all 18 onboard devices. The input to the neural networks consisted of time-series data for the power dissipation of these 18 devices, along with the average heat input for each of the six external panels. Monte Carlo simulations were used to generate both the prior and training datasets based on the defined uncertainty parameters.

The same hyperparameters were used for both RNN and LSTM models: hyperbolic tangent activation functions, Adam optimizer, a learning rate of 0.0001, and 300 training epochs. The datasets included 1,600 training samples, 200 validation samples, 100 prior samples, and 14 POD modes. Both models used one hidden layer with 150 neurons and an additional Fully Conected layer to compute the temperature outputs. The only architectural difference was that the FC layer employed the SiLU activation function.

In terms of performance, the POD-LSTM model outperformed the POD-RNN across all evaluated metrics. The maximum temperature error was 10.7 K for the LSTM and 21.2 K for the RNN, while the average errors were 1.16 K and 1.42 K, respectively. Moreover, 93.6% of the nodes in the LSTM model exhibited absolute errors below 3 K, compared to 89.6% in the RNN model.

Regarding computational cost, total training time was 16.2 hours for the POD-RNN and 20.7 hours for the POD-LSTM. However, inference times were negligible for both models: only 0.025 s and 0.031 s per simulation case, respectively, in stark contrast with 17.6 s required by the traditional TMM.

An increasingly popular strategy in the field of computational modeling is the use of surrogate models [51], also referred to as metamodels or Reduced Order Models. These models aim to approximate the behavior of complex and computationally expensive simulations by learning from a limited set of high-fidelity data. Rather than replacing the underlying physical models, surrogate models act as efficient stand-ins that can be rapidly evaluated, making them particularly valuable for tasks involving optimization, uncertainty quantification, or real-time decision-making.

Due to their ability to identify complex patterns in data, ANNs are well-suited for surrogate modeling, as they can approximate nonlinear input–output relationships with high accuracy. This capability enables them to significantly reduce simulation costs while maintaining reasonable predictive performance. Representative applications of ANN-based surrogate models can be found in [52–54].

Beyond neural networks, several other techniques are also used to construct surrogate models. One of those methods is Proper Orthogonal Decomposition (POD), already discussed in [32], which reduces the dimensionality of complex systems by extracting dominant spatial or temporal modes via SVD. Other notable applications of POD-based model reduction techniques can be found in [55–57].

As described in section 3.3, most thermal models used in the analysis of space systems rely on the lumped-parameter approach, and in particular on the Lumped-Parameter Thermal Modeling (LPTM). In addition to spacecraft thermal analysis, this method has been widely adopted for other types of thermal problems [58, 59], primarily due to its lower computational cost compared to CFD or FEM methods. This makes LPTM-based formulations especially compatible with PINN architectures, as the governing equations are well defined and relatively straightforward to evaluate.

In most of the reviewed works, LPTM or similar TMMs serve as the foundation for generating training data, with ANNs functioning as fast approximators. In some cases—particularly those involving PINNs—the heat balance equations from LPTM models are directly embedded into the loss function, enabling hybrid architectures that preserve physical consistency while drastically reducing computational time.

To summarize, ANN-based approaches have been successfully applied to a wide range of thermal modeling problems, especially within the space domain. From feedforward networks predicting onboard temperatures in CubeSats to POD-LSTM models accelerating transient thermal analysis in complex spacecraft, these methods have demonstrated strong potential. Nevertheless, challenges remain regarding generalization to unseen scenarios and robust integration with physically grounded models such as LPTMs. Addressing these limitations will be essential for the broader adoption of intelligent thermal modeling frameworks in future space missions.

5

Objectives

5.1 Motivation and context

Thermal control is a critical subsystem in any space mission, ensuring that onboard components operate within safe temperature limits throughout the mission lifecycle. As spacecraft designs grow increasingly complex and mission profiles more demanding, there is a growing need for more agile, intelligent, and autonomous thermal management strategies. Traditional methods, while robust, often lack the flexibility and computational efficiency required for fast decision-making and real-time operations.

Recent advances in artificial intelligence, particularly in the field of Artificial Neural Networks (ANNs), present new opportunities for data-driven modeling of complex physical systems, including thermal behavior. These methods can complement or even replace traditional approaches when accurate modeling is needed under time constraints or limited telemetry conditions.

This work is situated within this emerging paradigm shift, aiming to explore how deep learning architectures—specifically, convolutional recurrent networks—can enhance the prediction and control of transient temperature distributions in spacecraft components.

5.2 Application of ANNs to spacecraft thermal control

ANNs offer valuable capabilities for spacecraft thermal control, especially in scenarios where traditional methods face limitations in accuracy, speed, or adaptability.

A relevant application consists in the reconstruction of faulty or missing sensor readings. When

a temperature sensor fails during flight, a trained ANN can infer its value using correlations learned from the remaining sensors and prior operational data [60]. This contributes to system redundancy and robustness.

The primary focus of this work, however, lies in a more ambitious application: using ANNs to predict the full temperature distribution of a spacecraft in real time. This capability enables improved onboard thermal monitoring and facilitates faster simulation and control decisions during mission operations.

Traditionally, two main approaches have been used in the context of the Operational Spacecraft Simulator (OSS) [61] to approximate thermal behavior in real time:

1. **Case interpolation**: estimates the system's thermal response by interpolating between a finite set of precomputed cases under known conditions.
2. **Reduced thermal models**: simplifies the thermal network by reducing the number of nodes, allowing for faster real-time integration of the heat balance equations.

Each approach comes with trade-offs. Interpolation may fail under off-nominal or untrained conditions, while model reduction sacrifices spatial fidelity, particularly in regions of thermal interest.

ANNs offer a promising alternative to these approaches. By training on detailed simulation or experimental data, they can learn complex thermal dynamics while preserving high-resolution spatial information. Once trained, they enable fast and accurate predictions—even in conditions not explicitly covered by the training set—thus addressing the key limitations of traditional interpolation and model reduction techniques.

5.3 Challenges and strategy

One of the major challenges in applying ANNs to thermal problems is the cost of generating training data. High-fidelity simulations, such as those based on Finite Element Method (FEM), are computationally expensive. Moreover, the diversity of boundary conditions, operational environments, and internal configurations results in an enormous design space to explore.

To address this, this thesis investigates the integration of physical priors via PINNs. By incorporating energy conservation laws and boundary conditions directly into the learning objective, we aim to reduce the number of required training samples and improve generalization to unseen scenarios.

This hybrid approach combines the expressiveness of neural networks with the reliability of physics-based modelling, enabling the development of compact, accurate, and data-efficient models for thermal prediction. Such models are particularly well-suited for onboard integration, where fast and reliable performance is essential for autonomous thermal control.

5.4 Specific objectives of this work

The overarching goal of this work is to demonstrate the feasibility and advantages of convolutional recurrent neural networks for modeling transient thermal dynamics in spacecraft components. To this end, the following specific objectives are addressed:

- Develop and evaluate several neural network architectures, including feedforward networks, convolutional encoders, and ConvLSTM-based models, for predicting full temperature fields over time.
- Propose an input encoding strategy that incorporates physical structure and masking mechanisms to improve learning efficiency and model interpretability.
- Quantitatively compare the performance on a PCB model using synthetic data using metrics such as MAE, error standard deviation, and maximum deviation across time steps.
- Demonstrate the advantages of ConvLSTM architectures in capturing spatiotemporal thermal patterns.
- Explore the integration of physical loss terms and lay the foundation for hybrid PINN-ConvLSTM architectures.

Together, these objectives aim not only to demonstrate the technical viability of the proposed approach, but also to provide a practical framework that could be extended to real-world onboard thermal systems in future missions.

6

Thermal modeling of Printed Circuit Boards

6.1 Introduction to Printed Circuit Boards

A Printed Circuit Board (PCB) is defined as an electrical circuit in which the components and conductors are integrated within a mechanical structure. The conductive elements of a PCB include copper traces, terminals, heat sinks, and planar conductors. The mechanical support is provided by an insulating laminated material interleaved with conductive layers. The final structure is typically finished with a non-conductive solder mask and a silkscreen layer, which is used to indicate the placement and identification of electronic components.

PCBs play a critical role in space systems due to their capacity to provide compact, lightweight, and reliable integration of electronic components. In the highly constrained environment of space missions—where volume, mass, and power consumption must be minimized—PCBs enable the implementation of complex electronic subsystems within tight design envelopes. Their layered architecture, shown in Figure 6.1, allows for high-density routing of signals and power distribution while maintaining mechanical robustness and thermal stability. Furthermore, PCBs used in space applications are subject to stringent requirements regarding radiation tolerance, thermal cycling, outgassing, and vibration resistance, making their design and manufacturing processes pivotal to mission success.

To support both analysis and experimental testing, PCBs are often equipped with thermal elements such as heaters and temperature sensors. An example of this configuration is shown in

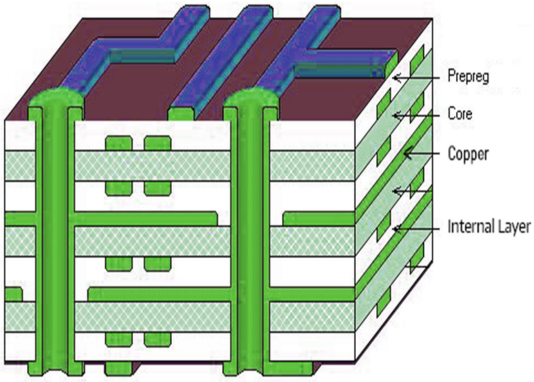


Figure 6.1: PCB stackup [6].

Figure 6.2, where heating elements and wired probes are installed on a planar test board for thermal characterization. This kind of setup is representative of the thermal scenarios modeled in this thesis.

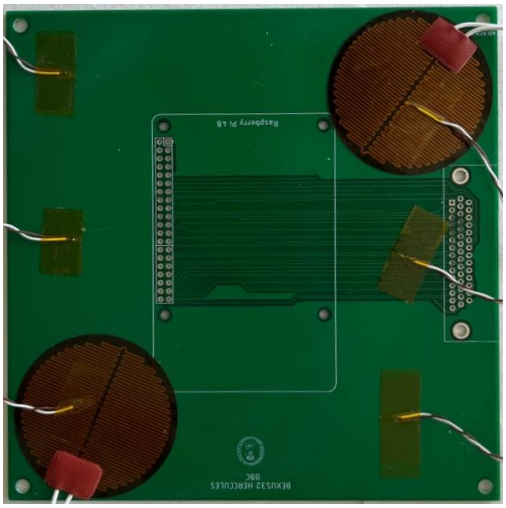
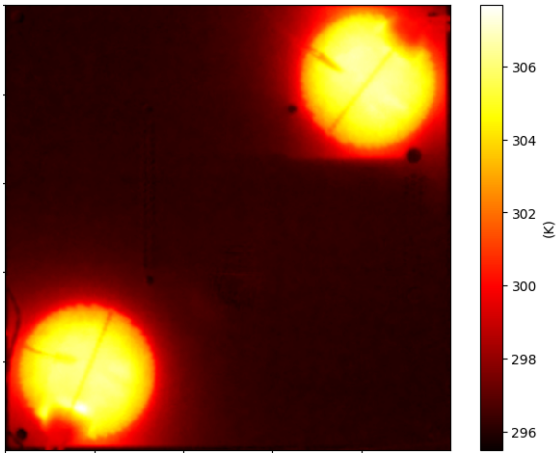


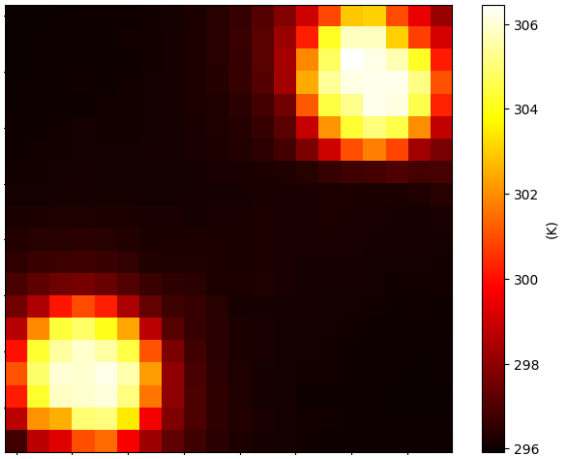
Figure 6.2: Example of a test PCB.

To illustrate the relationship between the measured thermal behavior of the real PCB and the spatial discretization used in this work, Figure 6.3 presents two thermographic images of the same test board. Both were acquired experimentally under steady heating conditions, but differ in the level of spatial detail.

The left image (Figure 6.3a) shows the original thermographic measurement, capturing the full spatial detail of the thermal distribution. The right image (Figure 6.3b) is a downsampled version of the same measurement, reduced to a 20×20 grid. While this resolution is still finer than the 13×13 discretization used in the numerical model, it serves as an intermediate reference to evaluate how spatial resolution affects the ability to capture thermal gradients. The comparison confirms that the dominant features—such as localized hot spots at heater locations and thermal gradients



(a) Original high-resolution thermographic measurement.



(b) Thermographic image downsampled to a 20×20 grid.

Figure 6.3: Effect of spatial discretization on the temperature representation of the PCB.

toward the edges—remain identifiable even with reduced resolution. This observation supports the use of relatively coarse spatial discretizations in data-driven surrogate models when the objective is to reproduce global thermal behavior rather than fine local details. However, this assumption holds primarily when the physical size of the numerical node is significantly smaller than that of the dissipative component, as it is in the case of Figure 6.3b.

6.2 Thermal Matematical Model of the PCB

To generate the data required to train the network, as well as to evaluate its accuracy, a physics-based thermal solver of a simplified PCB model is used. This solver, part of [62], computes the temperature distribution over a square PCB subjected to internal heat dissipation, conductive heat transfer within the board, and radiative heat loss to the environment.

The plate is modeled as a homogeneous solid with constant material properties. Thermal conduction is restricted to two dimensions within the plane of the board, under the assumption that temperature gradients across the thickness are negligible. The discretized domain consists of a regular grid of $n \times n$ nodes ($n = 13$ in this study), each representing a thermal control volume.

Each node can exchange heat with its immediate neighbors through conduction, and all nodes exchange energy with a radiative environment modeled as a blackbody at temperature T_{env}. Heat sources, representing electronic components, are modeled as localized power inputs at designated interior nodes. The four corner nodes of the domain are treated as thermal interfaces with the satellite structure, with fixed temperatures that may vary between simulations.

6.2.1 Solver regimes

Steady-state regime. In steady-state conditions, the temperature field $T(x, y)$ satisfies the nonlinear energy balance:

$$\nabla \cdot (k \nabla T) + \varepsilon \sigma (T^4 - T_{\text{env}}^4) = q \tag{6.1}$$

where k is the thermal conductivity, ε is the surface emissivity, σ is the Stefan–Boltzmann constant, and q is the internal heat generation per unit area.

The differential operators in equation (6.1) are approximated using a finite-difference scheme. The Laplacian is discretized by central differences between adjacent nodes, transforming the PDE into a system of nonlinear algebraic equations:

$$\mathbf{K} \cdot \vec{T} + \sigma \cdot \mathbf{E} \cdot (\vec{T}^4 - T_{\text{env}}^4) = \vec{Q} \tag{6.2}$$

where:

- $\vec{T} \in \mathbb{R}^{n \times n}$ is the temperature vector of all nodes.
- $\mathbf{K}$ is the conductivity coupling matrix, built from the conductances between neighboring nodes.
- $\mathbf{E}$ is a diagonal matrix representing radiative exchange with the environment (since no view factor between the PCB nodes is taken into account).
- $\vec{Q}$ contains the power dissipated by heaters and the imposed temperatures at interface nodes.

To enforce Dirichlet conditions at the interface nodes, their corresponding rows in $\mathbf{K}$ are set to identity and those in $\mathbf{E}$ are zeroed out. This guarantees that the matrix product yields the imposed temperatures:

$$T_i = T_{\text{interface}} \tag{6.3}$$

Because of the nonlinearity of the radiation term, equation (6.2) is solved using a fixed-point iterative scheme. At each iteration i, the linearized update is computed by solving:

$$\mathbf{A} \cdot \Delta \vec{T}_i = \vec{b} \tag{6.4}$$

with:

$$\mathbf{A} = \mathbf{K} + 4\sigma \cdot \mathbf{E} \cdot \text{diag}(\vec{T}_i^3), \qquad \vec{b} = \vec{Q} - \mathbf{K} \cdot \vec{T}_i - \sigma \cdot \mathbf{E} \cdot (\vec{T}_i^4 - T_{\text{env}}^4)$$

The temperature is then updated as $\vec{T}_{i+1} = \vec{T}_i + \Delta \vec{T}_i$, and the procedure is repeated until convergence (typically defined as $\max |\Delta \vec{T}_i| < 0.01$) or a maximum number of iterations is reached.

Transient regime. For time-dependent simulations, the solver integrates the transient heat equation:

$$\rho c \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + \varepsilon \sigma (T^4 - T_{\text{env}}^4) + q \tag{6.5}$$

The spatial terms are discretized using the same finite-difference method as in the steady-state case. The time derivative is approximated using an explicit Euler scheme:

$$\vec{T}_{n+1} = \vec{T}_n + \Delta t \cdot \frac{d\vec{T}}{dt} \tag{6.6}$$

This results in a forward time integration, where the temperature at each step depends on the current thermal flux and radiative losses. The discretization of this equation results in equation (3.15).

The time step Δt must respect the numerical stability condition based on the thermal diffusivity $\alpha = \frac{k}{\rho c}$:

$$\Delta t < \frac{1}{2} \cdot \frac{(\Delta x)^2}{\alpha} \tag{6.7}$$

This condition ensures that heat does not propagate unrealistically fast through the domain.

6.2.2 Physical parameters and configuration

The reference PCB used in this study is modeled as a square plate with constant and homogeneous material properties, representative of typical aerospace electronics. The default configuration assumes the following values:

- Plate dimensions: $L = 0.1\,\text{m}$.
- Thickness: $e = 0.001\,\text{m}$.
- Thermal conductivity: $k = 15\,\text{W}/(\text{m} \cdot \text{K})$.
- Specific heat capacity: $c = 900\,\text{J}/(\text{kg} \cdot \text{K})$.
- Density: $\rho = 2700\,\text{kg}/\text{m}^3$.
- Emissivity: $\varepsilon = 0.8$.

The computational grid uses a mesh refinement level resulting in $n = 13$ nodes in each direction (13×13 total). The positions of the heaters and interface nodes remain fixed and are defined as follows:

These node IDs correspond to a row-wise flattened indexing of the 13×13 mesh, where nodes are numbered from left to right and bottom to top. The complete indexing layout is provided in **??**. This configuration defines a single, physically coherent reference model of a PCB, which is used to generate data under a variety of boundary conditions and power inputs, as described in the next section.

Table 6.1: Interface nodes used in the default configuration.

Node ID	**Grid coordinates** (i, j)
0	(0, 0)
12	(12, 0)
168	(12, 12)
156	(0, 12)

Table 6.2: Heater nodes used in the default configuration.

Node ID	**Grid coordinates** (i, j)
66	(3, 6)
31	(6, 3)
129	(3, 9)
134	(9, 9)

6.2.3 Application

This physics-based solver provides a tunable and interpretable model to simulate realistic thermal maps of a PCB. The ability to vary heater positions, power inputs, interface conditions, and material properties allows for the generation of synthetic datasets with diverse thermal profiles—making it a robust reference for training and validating data-driven surrogate models.

An example of the temperature field computed by the steady-state solver is shown in Figure 6.4. The temperature increases around the heater locations, while remaining constrained at the fixed interface nodes. This configuration is representative of typical satellite operating conditions, where internal heat dissipation and boundary constraints define the thermal behavior of the board.

6.3 Convergence analysis

A simulation is considered to have reached steady state when the absolute error between the current temperature and the final stationary value at every node remains below 1 K for at least 10 consecutive seconds. This threshold ensures both numerical stability and physical plausibility in detecting equilibrium conditions across the domain.

The convergence to the steady state behavior of the transient solver was assessed using statistical methods to ensure that the selected simulation time (`time_sim` = 650 s) captures the full thermal response for nearly all scenarios in the dataset.

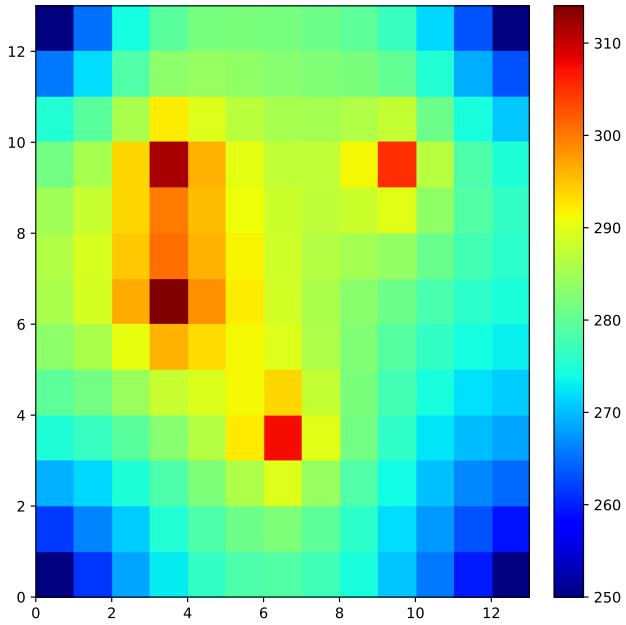


Figure 6.4: Example of the resulting temperature field.

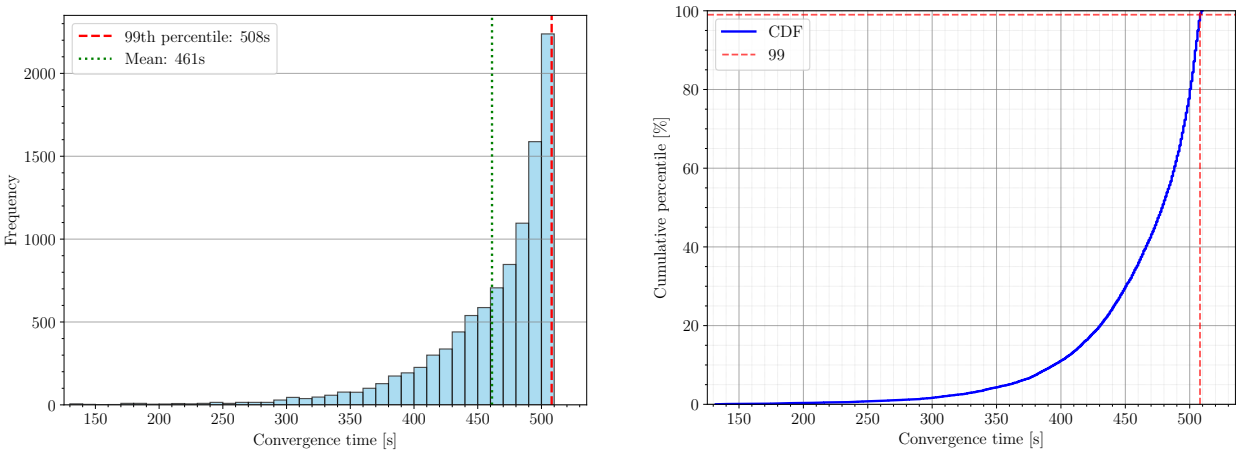
6.3.1 Statistical convergence analysis

Figure 6.5a shows the distribution of convergence times over 10,000 randomized simulations. On average, cases stabilize around 461 s, while 99% of them converge before 508 s. This observation is confirmed by the cumulative distribution function in Figure 6.5b, where the 99^{th} percentile is used as a conservative reference to define the minimum simulation duration.

6.3.2 Final justification of selected parameters

Based on these results, the chosen simulation duration of `time_sim` = 650 s is conservative and ensures convergence across virtually all test cases, including those with slower thermal dynamics. This duration guarantees that the generated temperature profiles are complete and physically reliable, supporting the training and evaluation of data-driven models.

In addition, the selected value allows for a short window of quasi-steady behavior after convergence is achieved. This inclusion enriches the dataset by providing the network with examples from both the transient regime and the stabilized state. At the same time, the chosen duration avoids excessive computational costs associated with simulating unnecessarily long periods during which no significant physical changes occur. This balance ensures physical representativeness while maintaining computational efficiency and data diversity.



(a) Histogram of convergence times.

(b) Cumulative distribution function (CDF).

Figure 6.5: Statistical distribution of convergence times across 10,000 randomized cases.

6.4 Dataset generation procedure

To train the ANN with physically consistent data, a dataset was generated using the transient thermal solver described in the previous section. Each sample corresponds to a simulation of the same reference PCB under varying thermal boundary conditions. The temperature field is computed over time, producing temporal sequences of thermal maps that reflect how heat propagates across the board under different scenarios.

All simulations use a fixed board configuration, with a discretization of 13×13 nodes, initial uniform temperature $T_{\text{init}} = 298\,\text{K}$, time step $\Delta t = 1\,\text{s}$, and total simulation time of $650\,\text{s}$. These values ensure stability of the integration scheme while capturing the relevant thermal dynamics over a sufficiently long horizon.

Three types of physical input parameters are varied across simulations:

- The power dissipated by each of the four heaters, representing different operational states.
- The temperatures at the four fixed thermal interfaces located at the corners.
- The temperature of the radiative environment surrounding the board.

The spatial positions of heaters and interface nodes remain unchanged throughout the dataset. This allows the network to focus on learning how the system responds to variations in boundary values rather than geometric changes. The input values are sampled uniformly within the physically meaningful ranges shown in Table 6.3.

Each simulation produces a sequence of temperature fields, which can be downsampled at different temporal intervals (e.g., every 2, 5, 10, or 20 steps) to study the effect of sampling frequency

Table 6.3: Range of physical input parameters used for dataset generation.

Input	**Unit**	**Range**
Heater power (per node)	W	[0.5, 1.5]
Interface temperatures	K	[270, 320]
Environment temperature	K	[270, 320]

on model performance. This enables the creation of multiple training sets from the same core data while preserving physical consistency. Crucially, all downsampling is performed after the simulation has been computed at a sufficiently small and stable time step ($\Delta t = 1\,\mathrm{s}$). This approach avoids the physical inconsistencies that would arise from solving the transient heat equation directly with large time steps. Since the solver uses an explicit Euler integration scheme, increasing the time step beyond the stability limit would lead to numerical instability and unphysical results. Downsampling from high-resolution data allows for coarser temporal inputs while maintaining the integrity of the underlying thermal dynamics.

Importantly, the ANN does not receive any internal material properties—such as thermal conductivity, heat capacity, or emissivity—as input. These values are kept constant across all simulations to define a single reference PCB. Consequently, the network learns to predict temperature evolution based solely on external conditions, boundary temperatures, and power inputs, making it suitable for real-world deployment where material properties may be unknown or difficult to obtain.

7

ConvLSTM-based architecture for transient thermal prediction

The results presented in the previous chapter revealed that among all the evaluated architectures, the ConvLSTM-based model was the only one capable of consistently maintaining prediction errors within an acceptable threshold across the entire temporal sequence. Given its strong performance and relevance to the thermal modeling problem under consideration, this chapter provides a comprehensive analysis of the underlying mechanisms, design choices, and structural advantages of the ConvLSTM architecture. The goal is to understand in depth how this model processes spatiotemporal information and to prepare the foundation for the physics-informed extensions introduced in the following chapter.

7.1 Motivation and core structure

Recurrent Neural Network such as fully connected LSTMs (FC-LSTM) have shown remarkable success in modeling temporal sequences. However, their dense connectivity makes them ill-suited for spatiotemporal data, as they flatten spatial structure and discard important locality information. To address this limitation, Shi et al. [7] introduced the Convolutional LSTM (ConvLSTM), a recurrent unit designed to handle spatiotemporal dynamics by embedding convolutional operations directly into the input-to-state and state-to-state transitions.

ConvLSTM units replace the dense matrix multiplications of traditional LSTMs with convolutional layers. As illustrated in Figure 7.1, each gate and memory update is computed through spatial convolutions, maintaining the 2D structure of the input data (e.g., temperature maps in

this work). This structure can be the predecessor of the Recurrent GNN [63] that allows a 3D representation of a thermal model. All internal states—inputs $\mathcal{X}_t$, hidden states $\mathcal{H}_t$, cell states $\mathcal{C}_t$, and the input i_t, forget f_t, and output o_t gates—are represented as 3D tensors (channels, height, width), preserving spatial correlations across time.

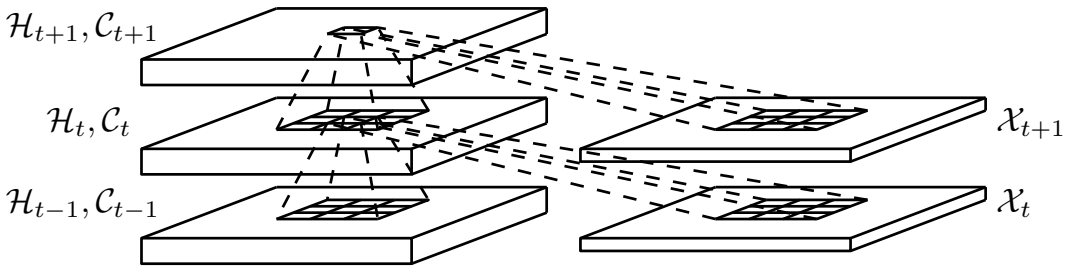


Figure 7.1: Internal structure of a ConvLSTM cell [7].

The recurrent update equations for a ConvLSTM cell are as follows:

$$\begin{aligned} i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\ f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \\ \mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\ o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\ \mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t) \end{aligned} \tag{7.1}$$

Here, '$*$' denotes convolution and '$\circ$' the element-wise product. In these equations, $\mathcal{X}_t$ is the input map at time step t, and $\mathcal{H}_t$, $\mathcal{C}_t$ are the hidden and cell states, which retain the spatial dimensions of the data. The gates i_t, f_t, and o_t control the flow of information into, through, and out of the memory cell, and are computed from both the current input and the previous hidden and cell states.

The convolutional weights W_{x*} and W_{h*} are learned filters applied to the input and hidden states, while the peephole weights W_{c*} allow each gate to access the previous or current memory state directly. The b_* terms are bias tensors. Activations are computed using the sigmoid function σ for gating and the hyperbolic tangent tanh for candidate state updates.

This formulation preserves the locality of spatial features while enabling temporal recurrence, making the ConvLSTM a natural choice for simulating dynamic physical fields such as thermal diffusion. The construction of the input tensor $\mathcal{X}_t$—which includes physical parameters, masks, and the previous temperature state—is detailed in section 7.4.

7.2 Encoding-forecasting architecture

In practice, ConvLSTM layers are typically stacked into deeper structures to enhance their modeling capacity. A widely used setup is the *encoding-forecasting* framework, shown in Figure 7.2. It consists of two symmetric parts: an encoder that reads the input sequence and compresses it into a latent state, and a decoder that unfolds this state to predict future frames autoregressively.

In our case, we simplify this structure to a one-to-one setup, using the same recurrent block across the entire sequence and producing the full temperature trajectory in a single pass. Nevertheless, the internal logic of state propagation and memory retention remains consistent with this encoder-forecasting view.

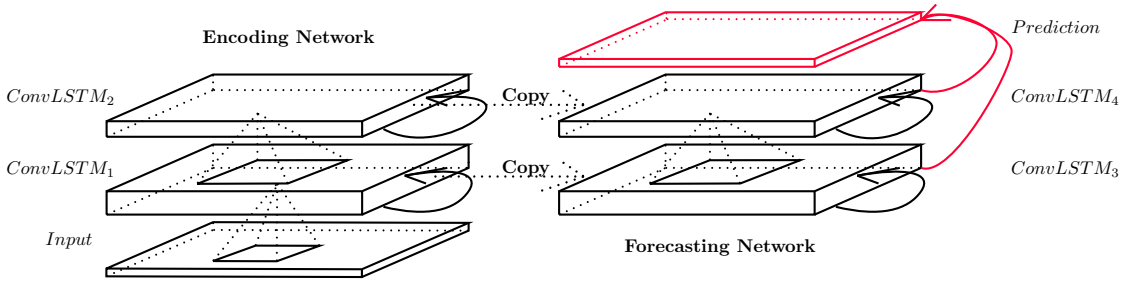


Figure 7.2: Typical encoder-forecasting ConvLSTM structure as proposed by [7].

7.3 Spatiotemporal processing characteristics

Unlike fully connected models that treat each input map as a vector, ConvLSTM layers operate directly on spatial grids. This has several key advantages:

1. **Locality preservation:** Convolutions restrict the influence of each pixel to its neighborhood, which mirrors physical processes like thermal diffusion.
2. **Parameter sharing:** Filters are shared across space, reducing the total number of parameters and improving generalization.
3. **Boundary handling:** Padding strategies (typically zero-padding) allow the model to differentiate inner and border regions, which is important in thermal models with interface boundaries.

These properties make ConvLSTM an ideal candidate for modeling complex thermophysical systems, where spatiotemporal consistency and local interactions play a central role.

7.4 Input encoding strategy

The success of the ConvLSTM architecture for transient thermal modeling depends critically on how the input information is encoded at each time step. In this work, we adopt a six-channel input representation that explicitly distinguishes between physical parameters, boundary conditions, and internal state evolution.

This design deviates from previous approaches such as [22], where only two channels are used: one for interface temperature and another for heater power. In that work, the ambient temperature is introduced externally and remains constant throughout the simulation. While this strategy simplifies the network input, it implicitly assumes the model can internalize the environmental influence during training.

In contrast, our six-channel input provides a richer and more structured representation, aiming to improve generalization and facilitate learning of the physical interactions that govern the system. The design process was incremental: the model was originally conceived for steady-state simulations using only three channels corresponding to boundary conditions—namely, interface temperature, heater power, and ambient temperature. To extend the model to the transient regime, a fourth channel was added to represent the temperature distribution at the previous time step, introducing autoregressive dynamics into the input.

However, this four-channel configuration was not sufficient to guide the model effectively, particularly in distinguishing between nodes that are externally constrained and those whose evolution must be predicted. This motivated the inclusion of two additional binary masks—one for interface nodes and one for heater locations—bringing the total to six input channels.

This extension was inspired by the masking mechanisms introduced in partial convolution networks [64], where the convolution operation is conditioned on the availability of valid input values. While our problem domain is different from image inpainting, the conceptual parallel holds: we guide the network to focus on physically meaningful regions while ignoring spatial zones where no external input is present. This structured separation allows the network to better learn the distinction between enforced boundary conditions and regions governed solely by internal thermal dynamics.

Each input tensor at a given time step is a six-channel map that reflects both the physical configuration and the thermal state of the system. These channels are constructed from the raw simulation data as follows:

1. **Interface temperature** (Figure 7.3a): The normalized temperature values applied at interface nodes. Values outside these nodes are masked to zero.

2. **Heater power** (Figure 7.3b): The normalized heating power map applied at heater locations. Non-heater nodes are masked to zero.

3. **Ambient temperature** (Figure 7.3c): A spatially uniform map where every node contains the normalized ambient temperature.
4. **Interface mask** (Figure 7.3d): A binary map (1 at interface nodes, 0 elsewhere) that guides the model to distinguish constrained boundary values.
5. **Heater mask** (Figure 7.3e): A binary map indicating the positions of the heaters within the domain.
6. **Previous temperature** (Figure 7.3f): The normalized temperature at the previous time step.

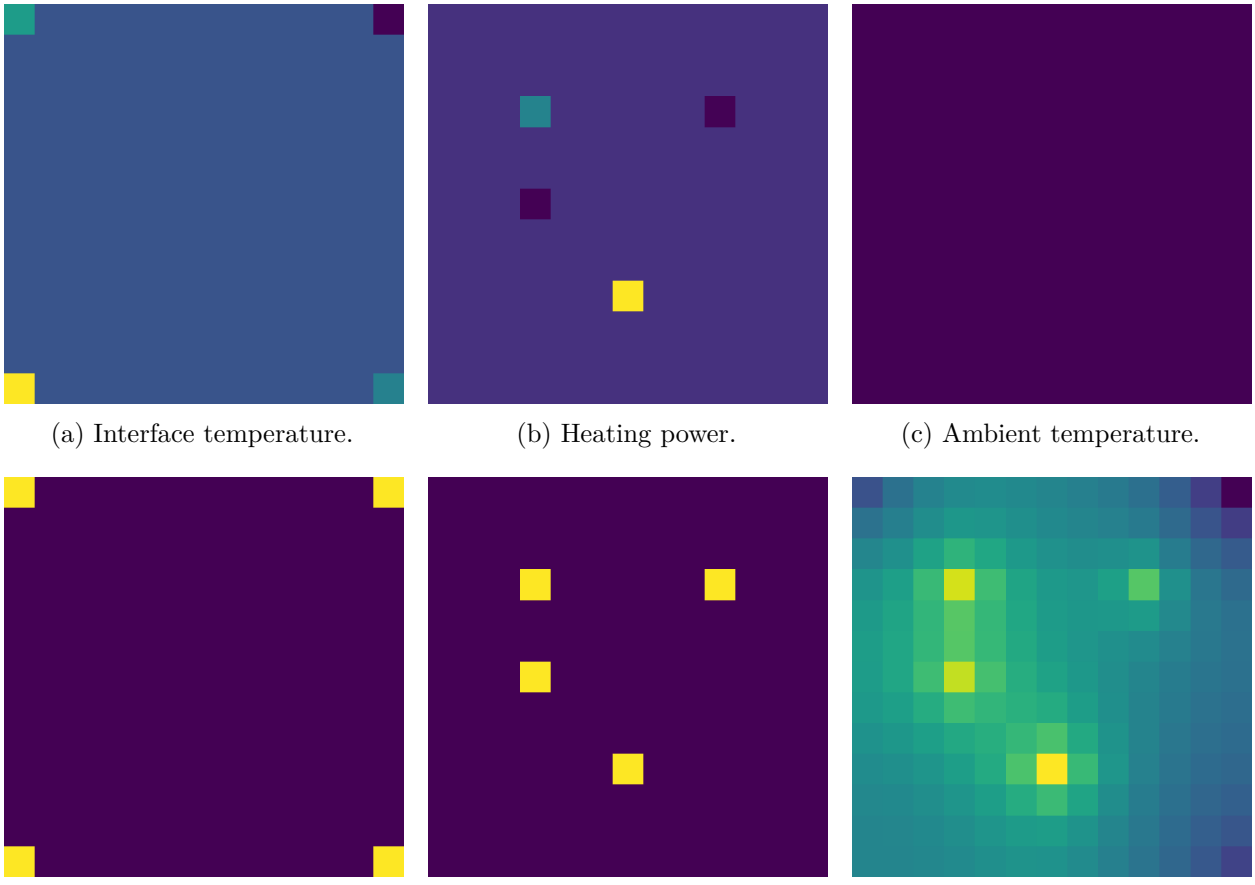
This separation allows the model to identify which elements are physically active at each location and disentangle the roles of heating, boundary enforcement, and thermal propagation. In particular, by including a dedicated channel for the previous temperature state, the model is explicitly informed of the thermal evolution, reinforcing the internal memory mechanism of the ConvLSTM. This design enables the network to leverage both spatial context and temporal recurrence for accurate forecasting. Importantly, the use of binary masks allows the model to distinguish between zero values that carry physical meaning—such as normalized temperatures or heating powers—and those that represent the absence of input information at specific locations.

Figure 7.3 illustrates these six input channels. The top row corresponds to the physical boundary conditions applied at each time step, while the bottom row provides structural and state-related context. This visual decomposition highlights the interpretability of our input strategy and the clear semantic role played by each channel.

7.5 Preliminary comparison with alternative models

Before settling on the ConvLSTM as the primary architecture, several alternative models were explored, including a fully connected MLP, a spatio-temporal regressor with explicit time conditioning, and a convolutional encoder combined with a Transformer decoder. While each of these approaches offered some potential advantages, none were able to combine accuracy, stability, and recursive prediction over long horizons. In particular, all relied on uniform initial conditions and could not be autoregressively fed with their own outputs—an essential requirement for deployment in real-time or onboard scenarios.

A detailed evaluation of these early models, including their configuration and performance metrics, is provided in Appendix B.



(a) Interface temperature. (b) Heating power. (c) Ambient temperature.

(d) Interface node mask. (e) Heater node mask. (f) Previous step temperature.

Figure 7.3: Six-channel input encoding used by the ConvLSTM model.

8

Model development and training strategy

8.1 System specifications

Two different computing environments were used throughout the project: a local desktop computer for data generation and debugging, and a cloud-based GPU instance on Google Colab [65] for model training and optimization. The specifications of each system are detailed in Table 8.1.

Table 8.1: Hardware and software specifications of development and training environments

Component	**Desktop (local)**	**Google Colab (cloud)**
Operating system	Windows 10 (build 19041)	Linux (kernel 6.1.123+)
Python version	3.9.21	3.11.13
PyTorch version	2.5.1	2.5.1+cu124
CUDA version	11.8	12.4
CUDA available	Yes	Yes
GPU configuration	6 physical / 6 logical cores	6 physical / 12 logical cores
RAM	15.92 GB	52.96 GB
GPU model	NVIDIA GeForce GTX 1050	NVIDIA L4
GPU memory	2.0 GB	22.16 GB
Compute capability	6.1	8.9
CUDA cores	5	58

Data generation and pre-processing were performed locally using the desktop setup, while the

training and hyperparameter tuning of neural networks were conducted in the cloud environment, taking advantage of the higher GPU memory and processing capacity of the NVIDIA L4 card.

8.2 Implementation details

The code developed for this project is organized within a GitHub repository named `ia_thermal` [62]. Although the repository contains several folders, the core implementation is located in the `ismaelgallo/` subdirectory. Additionally, a `requirements.txt` file is included at the root of the repository to facilitate the installation of all required dependencies.

The typical workflow consists of two stages:

1. **Dataset generation.** Executing the script `dataset_generation_convlstm.py` generates all dataset variants used throughout the project.
2. **Model training.** Once the datasets are available, different models can be trained using the corresponding Jupyter notebooks. Each notebook is designed for a specific architecture, for example:
   - `PCB_ConvLSTM_O2O.ipynb` for one-to-one ConvLSTM training.
   - `PCB_MLP_PINN.ipynb` for training a physics-informed MLP.
   - `PCB_Thermal_UNet_multistep.ipynb` for multistep training of a model.

To ensure reproducibility, random seeds are set explicitly during both dataset generation and model training. Dataset splits and normalization statistics are also fixed and reused consistently across different experiments.

The code is compatible with both CPU and GPU execution. When available, GPU acceleration is enabled automatically.

8.3 Dataset generation

The dataset used to train and evaluate the model is generated through a custom numerical simulator that reproduces the transient thermal behavior of a square PCB under varying boundary and heating conditions. To ensure comparability across samples, the simulation parameters—such as the grid resolution, total simulation time, time step, and initial temperature—are kept constant. Specifically, all cases are solved on a fixed 13×13 node grid over a simulation window of 650 seconds with time steps of 1 second and an initial temperature of 298.0 K.

Each data sample corresponds to a unique thermal scenario, created by randomly sampling a combination of heating powers, interface temperatures, and environmental temperature within

predefined physical ranges. These values are carefully chosen to ensure diversity across cases while avoiding repetition. Once the boundary and heating conditions are defined, the numerical solver computes the temperature distribution at each time step, producing a sequence of thermal maps that serve as ground truth.

In parallel, the input tensors required by the model are constructed to represent all relevant information at each time step. These include the heater configuration, interface temperatures, environmental temperature, and the previous temperature map—either the initial condition or the model's own output at the prior step. This structure enables the model to perform step-by-step predictions in an autoregressive fashion.

To study the model's performance under different temporal resolutions, each generated sequence is also downsampled at regular intervals (e.g., every 1, 2, 5, 10, 20, 50, or 100 steps), allowing us to evaluate the impact of temporal sparsity. For each downsampling level and for both cases—with and without boundary condition information—a separate dataset variant is created, resulting in a rich set of training configurations tailored to different experimental needs.

All datasets are normalized independently, using feature-wise statistics (mean and standard deviation) computed from the entire dataset. The full set of 3050 sequences is then split into training, validation, and test sets using a simple heuristic: the validation set comprises one fifth of the training set size, and the test set one tenth of the validation set. This yields 2500 training sequences, 500 validation sequences, and 50 test sequences.

Table 8.2 summarizes the key parameters used during dataset generation.

Table 8.2: Summary of dataset generation parameters

Parameter	**Value / Description**
Grid resolution	13×13 nodes
Total simulation time	650 seconds
Time step (Δt)	1.0 s
Initial temperature	298.0 K
Heater power range	[0.5, 1.5] W (4 heaters)
Interface temperature range	[270, 320] K (4 interfaces)
Environmental temperature range	[270, 320] K
Training set size	2500 sequences
Validation set size	500 sequences
Test set size	50 sequences
Step intervals tested	{1, 2, 5, 10, 20, 50, 100}

The total time required to generate all cases on a standard desktop computer was approximately 679.06 seconds, with an average time of 0.222 seconds per case. These numbers provide a rough estimate of the computational cost associated with producing realistic, physics-based training data for thermal modeling tasks.

8.4 Model training

8.4.1 Scheduled sampling

Scheduled sampling is a technique designed to improve the training of autoregressive sequence models by mitigating the discrepancy between training and inference, known as *exposure bias* [66]. During training, models traditionally use the ground truth (*teacher forcing*) at each time step, which does not reflect the inference scenario, where the model must rely on its own predictions.

Scheduled sampling addresses this by introducing a stochastic mechanism: at each time step t, the model decides whether to use the true target value $\mathbf{T}_t$ or its own previous prediction $\hat{\mathbf{T}}_t$ as the input for the next step. This choice is governed by a probability p_t, which gradually decreases during training.

Formally, at each time step t, we sample a Bernoulli random variable:

$$\epsilon_t \sim \text{Bernoulli}(p_t),$$

and define the input to the model at time t as:

$$\tilde{\mathbf{T}}_t = \epsilon_t \cdot \mathbf{T}_t + (1 - \epsilon_t) \cdot \hat{\mathbf{T}}_t.$$

Here:

- $\mathbf{T}_t$ is the true target at time t (ground truth).
- $\hat{\mathbf{T}}_t$ is the model prediction from the previous time step.
- $\epsilon_t = 1$ means we use the ground truth, $\epsilon_t = 0$ means we use the model prediction.

The probability p_t of using the ground truth is scheduled to decay over time, according to a chosen schedule. In our experiments, we use a linear schedule defined by three hyperparameters:

- `p0`: initial probability of using the ground truth at the beginning of training.
- `p_min`: minimum probability of using the ground truth at the end of training.
- `decay_epochs`: number of epochs over which the probability decays from p_0 to $p_{\min}$.

A simple linear decay as the one used in this thesis can be written as:

$$p_e = \max\left(p_{\min},\ p_0 - \frac{e}{\text{decay\_epochs}}(p_0 - p_{\min})\right),$$

where e is the current epoch.

In our experiments, we set $p_0 = 1.0$, $p_{\min} = 0.0$, and decay_epochs $= 200$, which means the model starts by fully relying on the ground truth and gradually transitions to using its own predictions.

This gradual transition improves robustness by teaching the model to generate sequences even when its own predictions deviate from the ground truth, as highlighted in Bengio et al. [67].

8.4.2 Training pipeline

The model is trained using normalized datasets, already divided into training and validation sets. Each training epoch consists of iterating through batches of data and predicting the temporal evolution of temperature distributions. The model processes the sequence step by step, using its own output from the previous step or the true value, depending on a gradually decreasing probability (see subsection 8.4.1).

The loss is computed across the entire sequence, measuring the deviation between predicted and reference temperatures at each time step. This global loss is then used to update the model's parameters through gradient-based optimization. Training is accelerated using GPU resources and mixed-precision arithmetic, which reduces memory usage and improves performance.

At the end of each epoch, the model is evaluated on a separate validation set to monitor generalization. A learning rate scheduler adapts the optimization process based on validation performance, and early stopping is applied if no improvement is observed over several consecutive epochs.

To ensure reproducibility and facilitate post-processing, all relevant training information—including loss curves, training time, and hardware specifications—is automatically logged and saved.

The overall training pipeline is summarized in Figure 8.1, which visually outlines the sequence of steps from data loading to model evaluation. It starts by feeding the current batch into the ConvLSTM network in an autoregressive fashion. At each time step, the scheduled sampling mechanism decides whether to use the ground truth or the model's own prediction as input ($\mathcal{X}_t$) for the next step, according to the probability schedule described in subsection 8.4.1. The model performs a forward pass through the ConvLSTM layers to generate predicted temperature maps for the entire sequence. After the sequence is generated, the loss is computed by comparing the predictions to the reference data. When using physics-informed training, the loss includes not only the standard MSE but also contributions from the physical residuals and boundary conditions, as defined in equation (2.4). Finally, the gradients of the total loss are computed and used to update the model's parameters through backpropagation, and the model is evaluated on the validation set to monitor performance.

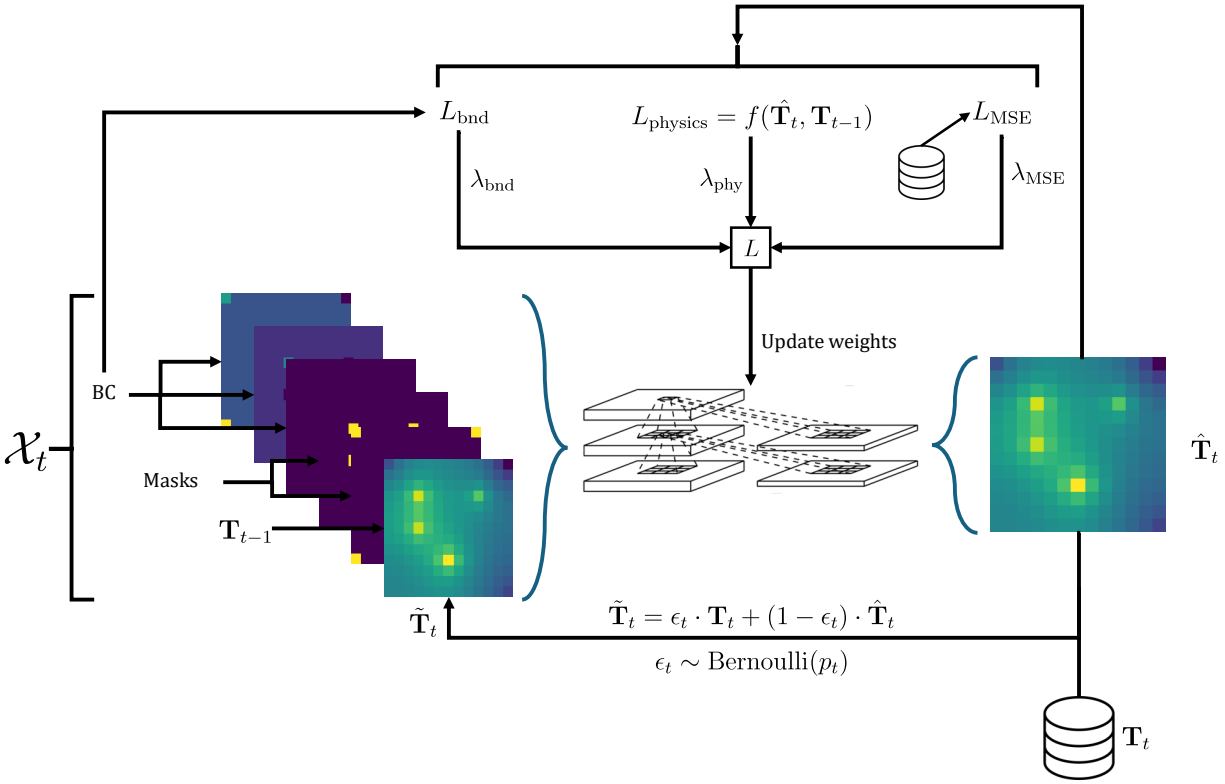


Figure 8.1: Flowchart of the PINN-ConvLSTM training pipeline, showing the sequence of operations.

8.4.3 Hyperparameter optimization

Although hyperparameter optimization frameworks such as Optuna [68] offer efficient sampling strategies for exploring high-dimensional spaces, fully leveraging their capabilities requires running a large number of training trials, which was not feasible within the resource and time constraints of this project. This challenge is well known in the literature [69], where random and adaptive search strategies are shown to outperform grid search only when the number of trials is sufficiently large.

In our case, however, the model exposes only a small number of tunable hyperparameters: the number of input channels, the dimensions of the hidden state, and the kernel size in the convolutional layers. This low-dimensional search space makes simpler strategies like grid search more viable.

Preliminary experiments were conducted using reduced training and validation sets and a limited number of epochs, following early-stopping heuristics as proposed in [70], but the added implementation complexity outweighed the benefits in this context.

A small grid search was ultimately carried out to identify configurations that offered a good balance between performance and simplicity. While the final setup may not represent a global optimum, it was empirically effective and aligned with the practical goals of the project. In many cases, once a model performs consistently well, it's often wise to follow the pragmatic rule: "if it

works, don't try to fix it".

8.5 Evaluation metrics

Three different metrics are used in this work: one for training the models, and two for assessing their predictive performance.

During training, the objective function is either the standard MSE or a custom physics-informed loss defined in equation (2.4), which incorporates physical constraints derived from the thermal model. The choice of loss function depends on the specific architecture and the training objective, as described in Appendix B.

For the evaluation and comparison of trained models, we report both the MAE and the maximum error observed in the test set, along with the standard deviation of the MAE. The MAE provides an interpretable measure of the average deviation between predicted and reference temperatures, while the maximum error highlights the worst-case discrepancy in the predictions. These complementary metrics offer a more complete characterization of model accuracy under consistent conditions.

8.6 Summary of final configuration

The final configuration of the model and training procedure is detailed in the following tables, grouped into four categories: model architecture, general training setup, scheduled sampling strategy, and learning rate scheduling with early stopping.

Table 8.3 summarizes the architectural choices. The model receives six input channels and outputs a single-channel temperature prediction. Two convolutional layers with 64 hidden units each and a 3×3 kernel size are used, with ReLU activation to introduce non-linearity.

Table 8.3: ConvLSTM model architecture hyperparameters

Parameter	**Value**
Input channels	6
Hidden dimensions	[64, 64]
Kernel size	3x3
Activation function	ReLU
Output channels	1

Table 8.4 lists the core training parameters. The batch size is set to 64, and training is performed using the Adam optimizer with an initial learning rate of 0.01. Although the maximum number of epochs is set to 1000, training rarely reaches this limit due to the use of early stopping criteria.

The configuration for scheduled sampling, shown in Table 8.5, gradually reduces the model's

Table 8.4: Training configuration

Parameter	Value
Batch size	64
Initial learning rate	0.01
Optimizer	Adam
Max epochs	1000

dependence on the ground truth as training progresses. The transition from fully guided training (probability 1.0) to fully autoregressive training (probability 0.0) occurs linearly over the first 200 epochs. This technique helps the model adapt to the conditions it will face at inference time.

Table 8.5: Scheduled sampling configuration

Parameter	Value
Initial probability	1.0
Final probability	0.0
Decay duration	200 epochs
Decay schedule	Linear

Finally, Table 8.6 outlines the parameters used for learning rate adjustment and early stopping. The learning rate scheduler monitors the validation loss and reduces the learning rate by a factor of 0.1 if no improvement is observed for 10 consecutive epochs, helping model convergence as shown in [16]. In parallel, early stopping halts training entirely if the validation loss stagnates for 100 epochs, preventing overfitting and saving computational resources.

Table 8.6: Learning rate scheduler and early stopping

Parameter	Value
Scheduler type	ReduceLROnPlateau
Patience (LR)	10 epochs
Reduction factor	0.1
Patience (stopping)	100 epochs
Monitoring metric	Validation loss

These hyperparameters define the complete training strategy used in this work and reflect a balance between empirical performance and practical feasibility. The effectiveness of this configuration is assessed in the next chapter, where the predictive results of the model are presented and discussed.

9

Results

9.1 Impact of temporal resolution

One of the most relevant decisions when training autoregressive models for transient simulation is the choice of the time step Δt. A smaller Δt leads to longer sequences and potentially more accurate modeling of the dynamics, but also increases computational cost and the risk of error accumulation during inference. Conversely, larger time steps reduce sequence length and inference time but may degrade performance due to larger state transitions between steps.

To study this trade-off, we conduct a systematic evaluation of a fixed ConvLSTM architecture (detailed in section 8.6) trained with three different durations: 50, 200, and 650 seconds. Each model is evaluated using its respective training rollout durations, as well as a 650-second rollout to assess robustness and generalization (as justified in section 6.3).

9.1.1 Evaluation protocol

All models are trained using a reduced dataset with 1000 sequences for training and 200 for validation. Performance is assessed on a separate test set of 1000 unseen sequences using an autoregressive strategy, where the model uses its own previous predictions as inputs for the next steps.

For each training duration, we evaluate:

- A rollout matching the training duration (50, 200, or 650 s), used to measure in-distribution performance.

- A 650-second rollout to analyze extrapolation beyond the training window up to the desired timeframe.

While all plots show the temporal evolution of errors over a 600-second window for consistency with time divisions, the reported metrics in the tables correspond to rollouts of 650 seconds. This simplification is justified by the strong similarities between the results obtained for both rollouts, which show almost no differences.

9.1.2 Different models trained

9.1.2.1 Model trained for 650 s

Since the main objective of this work is to accurately predict 650-second evolutions, the simplest approach is to train the network for this time window. By varying Δt, different models were trained and their results are shown in Table 9.1 and Figure 9.1.

Table 9.1: Error metrics for 650 s rollout with varying Δt (model trained for 650 s)

Δt [s]	Sequence length	MAE [K]	Max Error [K]	Std. [K]	Initial MAE [K]	Final MAE [K]
1	651	8.754	64.583	6.937	5.401	8.406
2	326	4.493	57.307	3.955	3.794	4.838
5	131	1.104	33.681	0.964	0.641	1.634
10	66	0.261	12.478	0.243	0.437	0.286
20	33	0.765	11.840	0.595	0.379	0.746
50	14	1.207	10.578	0.828	0.622	0.980
100	7	2.374	11.771	1.399	1.462	2.612

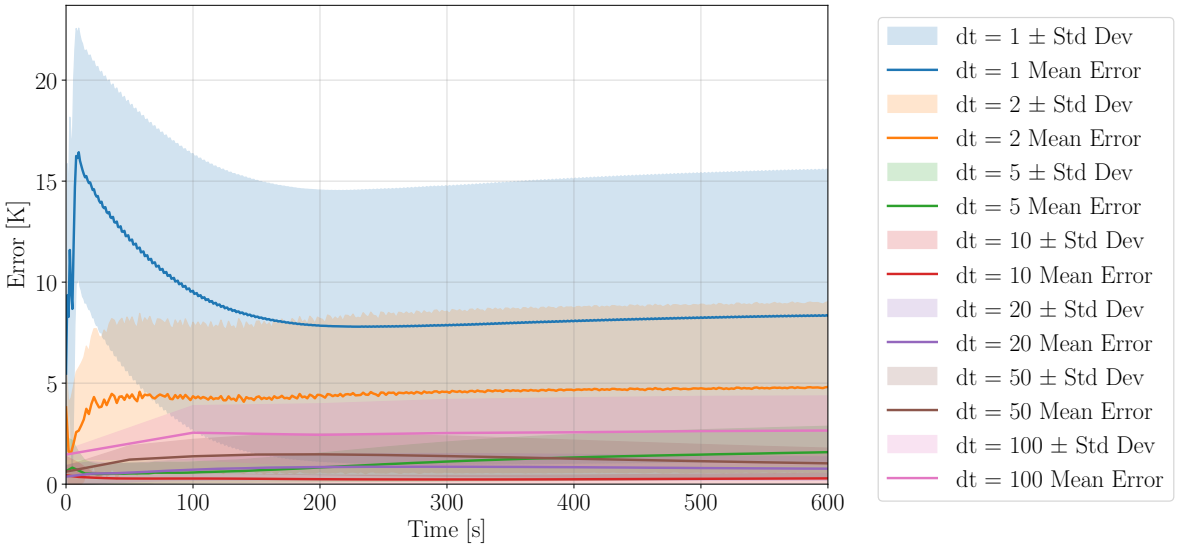


Figure 9.1: Temporal evolution of the MAE for models trained for 650 s and evaluated over 600 s.

It is clearly observed both in Figure 9.1 and Table 9.1 that, in general, a smaller time step—and therefore a longer sequence—worsens the network's performance. However, also a bigger time step causes a higher MAE, therefore choosing an optimum value requires a study, as will be discussed later.

9.1.2.2 Model trained for 200 s

Although the primary objective is to predict a 650-second sequence, it has been shown that models can extrapolate beyond their training horizons [71]. Therefore, a 200-second training window is used to assess performance both within and beyond this range.

Table 9.2: Error metrics for 200 s rollout with varying Δt (model trained for 200 s)

Δt [s]	Sequence length	MAE [K]	Max Erorr [K]	Std. [K]	Initial MAE [K]	Final MAE [K]
1	201	1.607	35.249	1.427	1.182	1.960
2	101	1.374	45.771	2.225	1.384	1.580
5	41	0.125	13.678	0.125	0.118	0.186
10	21	0.178	10.056	0.155	0.167	0.163
20	11	0.397	8.163	0.315	0.315	0.442
50	5	0.753	4.948	0.490	0.625	0.827
100	3	1.749	8.678	0.826	1.541	1.863

Table 9.3: Error metrics for 650 s rollout with varying Δt (model trained for 200 s)

Δt [s]	Sequence length	MAE [K]	Max Error [K]	Std. [K]	Initial MAE [K]	Final MAE [K]
1	651	4.646	44.931	2.798	1.182	10.180
2	326	3.260	75.620	3.953	1.384	6.332
5	131	1.303	48.788	2.371	0.118	4.514
10	66	0.628	21.401	0.571	0.167	1.537
20	33	1.664	22.488	1.298	0.315	4.861
50	14	3.509	75.918	3.931	0.625	8.445
100	7	7.367	76.138	5.226	1.541	19.500

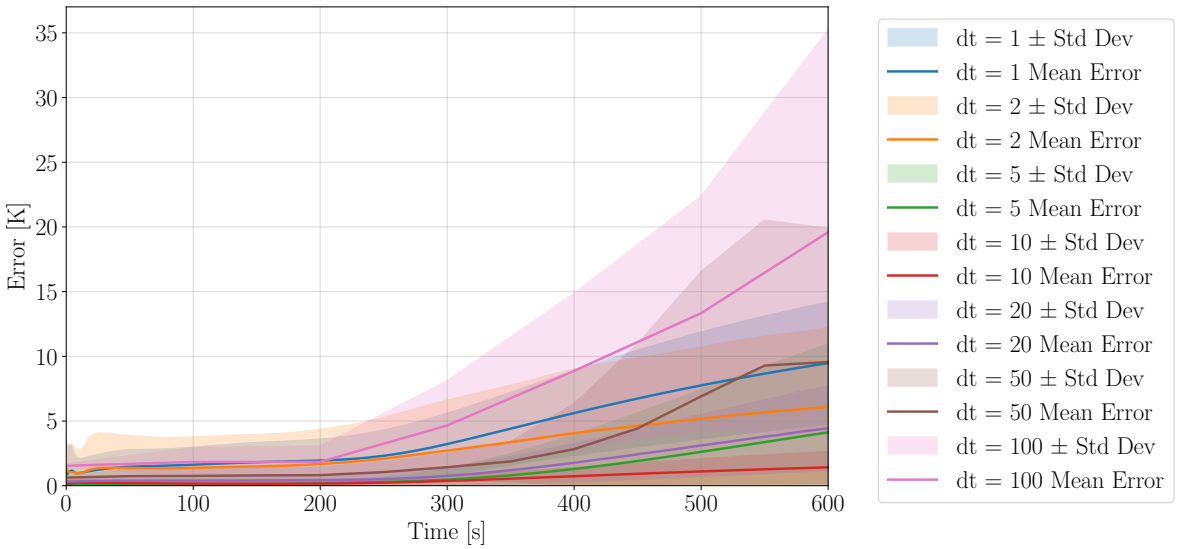


Figure 9.2: Temporal evolution of the MAE for models trained for 200 s and evaluated over 600 s.

However, the significant differences between the error values in Table 9.2 and Table 9.3—especially in the final MAE—as well as the sharp trend change in Figure 9.2, indicate that propagation using the 200-second-trained model is generally not feasible—at least not without implementing a PINN—with the exception of the case with $\Delta t = 10\,s$, which shows an adequate response but remains an isolated result.

9.1.2.3 Model trained for 50 s

As with the previous model, extending the rollout far beyond the training horizon leads to significant error, as clearly illustrated by Table 9.5 and Figure 9.3, in contrast to the generally low MAE in the 50-second rollout in Table 9.4.

Table 9.4: Error metrics for 50 s rollout with varying Δt (model trained for 50 s)

Δt [s]	Sequence length	MAE [K]	Max Error [K]	Std. [K]	Initial MAE [K]	Final MAE [K]
1	51	0.065	5.222	0.074	0.092	0.113
2	26	0.109	5.405	0.231	0.062	0.158
5	11	0.082	3.135	0.094	0.083	0.085
10	6	0.182	5.971	0.236	0.157	0.234
20	3	0.335	6.411	0.273	0.321	0.333
50	2	0.652	5.352	0.379	0.648	0.656

Table 9.5: Error metrics for 650 s rollout with varying Δt (model trained for 50 s)

Δt [s]	Sequence length	MAE [K]	Max Error [K]	Std. [K]	Initial MAE [K]	Final MAE [K]
1	651	7.889	65.930	4.531	0.091	10.914
2	326	6.592	60.212	5.899	0.061	12.253
5	131	5.153	47.863	3.707	0.082	9.026
10	66	7.035	65.221	5.144	0.155	8.966
20	33	19.022	77.014	8.951	0.321	26.190
50	14	14.854	105.394	10.038	0.647	21.532

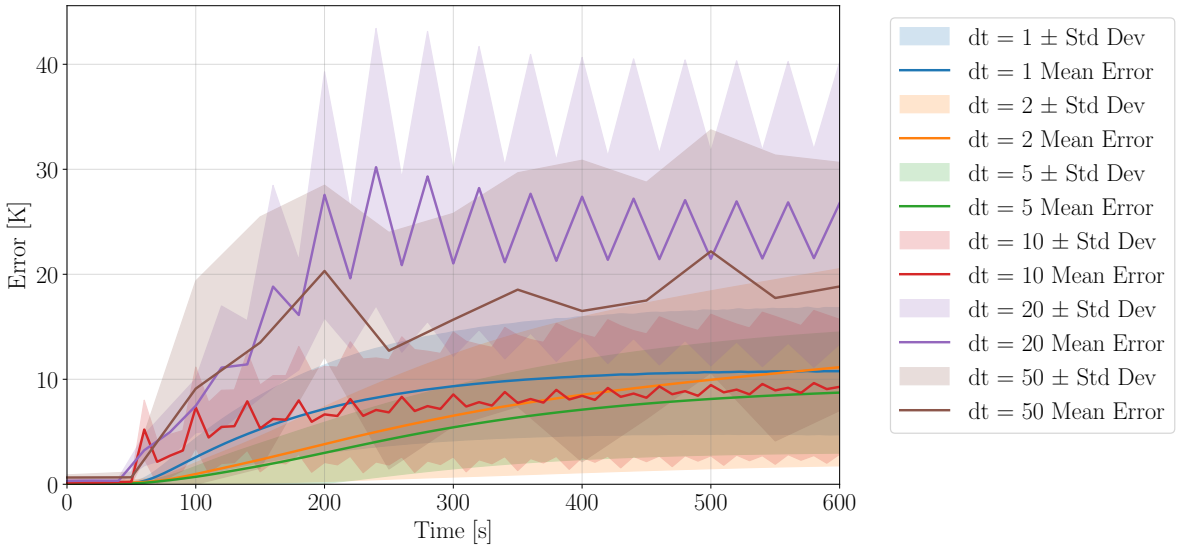


Figure 9.3: Temporal evolution of the MAE for models trained for 50 s and evaluated over 600 s.

9.1.3 Discussion and conclusion

To summarize the results, the MAE versus Δt for same-window rollouts is shown in Figure 9.4. It is clearly evident that, especially for longer sequences, shorter time steps lead to higher errors. However, larger time steps are not always better, as they lack the detail needed for accurate predictions.

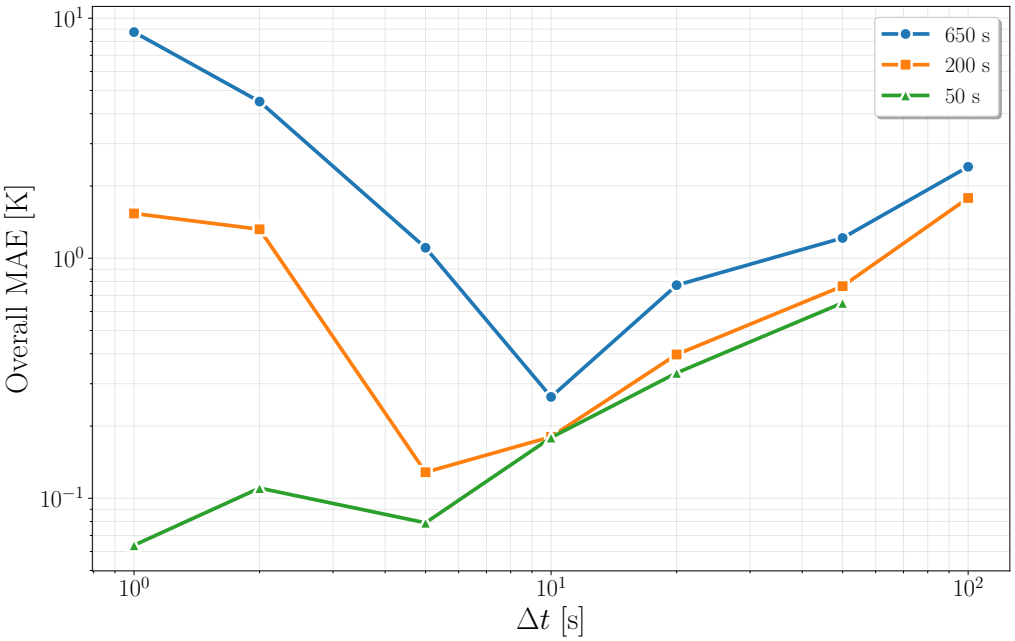


Figure 9.4: Overall MAE versus time step (all configurations)

Moreover, the results suggest that the time step itself is not the only critical parameter, but rather the sequence length, as illustrated in Figure 9.5. The figure shows a scatter plot of the overall MAE against sequence length for all configurations tested, with different colors indicating models trained for 50 s, 200 s, and 650 s, and larger markers corresponding to larger time steps (Δt). Most low-error models are concentrated in the short-sequence region of the plot, where sequence lengths remain moderate, with the exception of some configurations using larger Δt (indicated by larger dots) that also achieve good performance despite their shorter sequences.

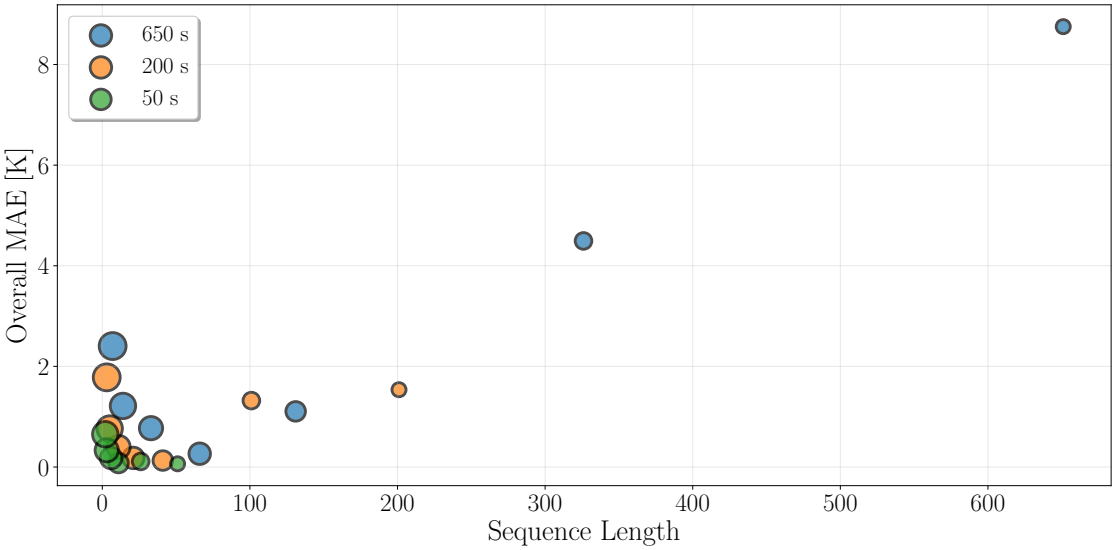


Figure 9.5: Overall MAE versus sequence length (all configurations).

The findings confirm that the choice of Δt significantly affects the accuracy and stability of autoregressive models. Very small values such as 1 or 2 seconds result in long sequences with higher computational demands and error propagation. In contrast, very large values such as 50 or 100 seconds create large interpolation gaps, decreasing prediction accuracy.

Across all training durations, the most balanced performance is achieved with $\Delta t = 10$ s. This configuration consistently minimizes both overall and final errors while maintaining reasonably short inference sequences. For instance, the model trained for 650 s with $\Delta t = 10$ s achieves the best overall error metrics, outperforming even smaller-Δt configurations despite having fewer time steps.

These results align with previous studies [72], which reported enhanced stability and generalization in recurrent networks when using coarser temporal resolutions. They suggest that moderate Δt values (e.g., 5–10 s) provide a robust trade-off between fidelity and computational cost in transient thermal prediction tasks using ConvLSTM models.

9.2 PINN-parameter tuning

Following the discussion on time step selection, a value of $\Delta t = 10\,\mathrm{s}$ is chosen. The next step involves tuning the weights of the loss function, recalled here from equation (2.4):

$$L = \lambda_{\mathrm{MSE}} L_{\mathrm{MSE}} + \lambda_{\mathrm{phy}} L_{\mathrm{physics}} + \lambda_{\mathrm{bnd}} L_{\mathrm{bnd}}$$

The parameters λ_{MSE}, λ_{phy}, and λ_{bnd} are real numbers that can take arbitrary positive values. Since each loss term typically exhibits a different order of magnitude, an unbalanced choice of

weights may cause one term to dominate and saturate the optimizer. To avoid this, the weights must be carefully adjusted so that all components contribute comparably to the total loss.

For instance, Figure 9.6 shows the evolution of the training loss when all weights are set to 1, which leads to a significant imbalance due to the differences in scale—especially when one term, such as the boundary component, is orders of magnitude larger than the others. In contrast, the training loss in Figure 9.7 evolves more favorably when a more appropriate weighting scheme is applied.

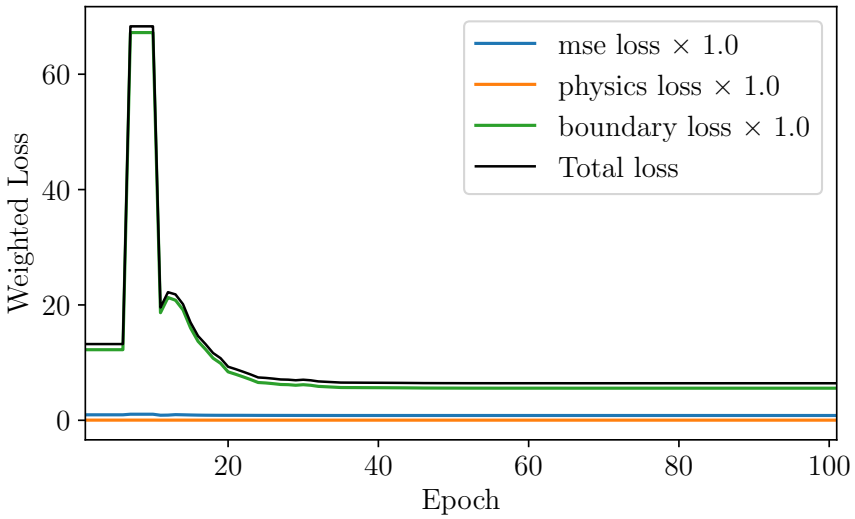


Figure 9.6: Training loss evolution for a sample case with all weights set to 1.

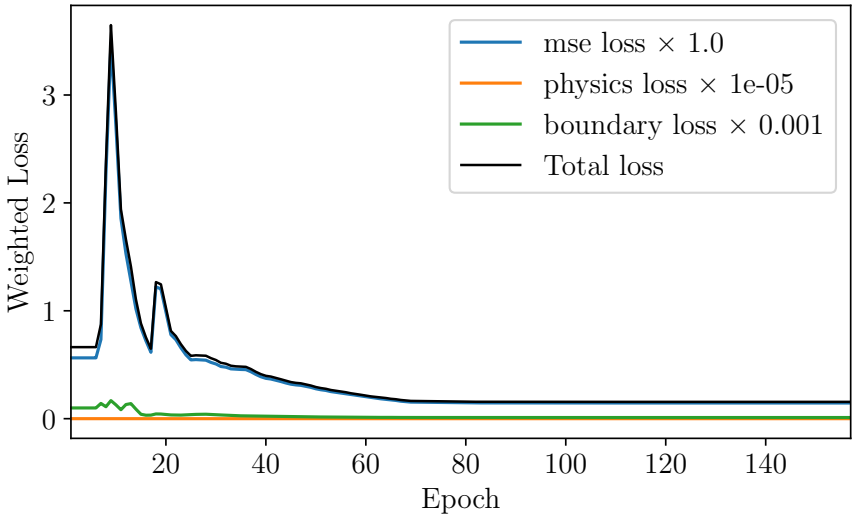


Figure 9.7: Training loss evolution for a sample case with differentiated weights.

To address this, a grid search optimization is performed [73]. Since the absolute values of the

weights are less important than their relative magnitudes [74], the values explored are:

$$\begin{aligned} \lambda_{\text{MSE}} &= 1 \\ \lambda_{\text{phy}},\ \lambda_{\text{bnd}} &\in \{0,\ 10^{-5},\ 10^{-4},\ 10^{-3},\ 10^{-2},\ 10^{-1},\ 1\}^{1} \end{aligned} \tag{9.1}$$

Although more advanced strategies for initializing weights exist [75], grid search is selected for its simplicity and effectiveness in low-dimensional hyperparameter spaces [76].

Since the main motivation behind Physics-Informed Neural Network models is to reduce the amount of data required for training [77], the following analysis focuses on assessing their performance in low-data regimes. Specifically, the grid search is repeated for different dataset sizes: 10, 50, 200, 400 and 500 training cases.

9.2.0.1 $n_{\text{train}} = 10$

In the most extreme scenario, only 10 training cases are used to assess the model's capacity in ultra-low data regimes. For all the following cases, a rule of thumb is performed to select the number of validation cases: $n_{\text{val}} = n_{\text{train}}/5$ This setting is particularly relevant for high-fidelity simulations, where generating large datasets may be computationally prohibitive.

The top configurations from the grid search are shown in Table 9.6, ranked by increasing overall MAE. Despite the limited supervision, multiple models significantly outperform the baseline—reaching up to 80% reduction in mean prediction error.[2]

These results support the hypothesis that embedding physical constraints promotes generalization in data-scarce regimes. The best-performing model uses moderate regularization weights: $\lambda_{\text{phy}} = 10^{-6}$, $\lambda_{\text{bnd}} = 10^{-3}$. Its results are analyzed in more detail below.

[1] Due to computational cost, only this subset was fully tested. The complete grid search is shown in Appendix C.
[2] Improvement is calculated as $\frac{\text{MAE}_{\text{baseline}} - \text{MAE}}{\text{MAE}_{\text{baseline}}} \times 100$. A positive value indicates a lower MAE than the baseline.

Table 9.6: Top 10 physics-informed models for $n_{\text{train}} = 10$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
1	1	10^{-6}	10^{-3}	1.6846	17.5397	+79.5%
2	1	1	10^{-2}	1.6854	17.4132	+79.5%
3	1	10^{-2}	10^{-3}	1.7480	15.5926	+78.7%
4	1	10^{-3}	10^{-2}	1.7963	19.6237	+78.2%
5	1	10^{-4}	10^{-2}	1.8199	22.5164	+77.9%
6	1	10^{-2}	10^{-4}	1.9471	23.8640	+76.3%
7	1	10^{-4}	10^{-4}	1.9961	20.9336	+75.7%
8	1	10^{-1}	10^{-2}	2.0524	20.0407	+75.0%
9	1	10^{-7}	0	2.0570	30.3938	+75.0%
10	1	10^{-5}	10^{-4}	2.2317	25.1944	+72.9%
Baseline	1	0	0	8.2242	55.4656	–

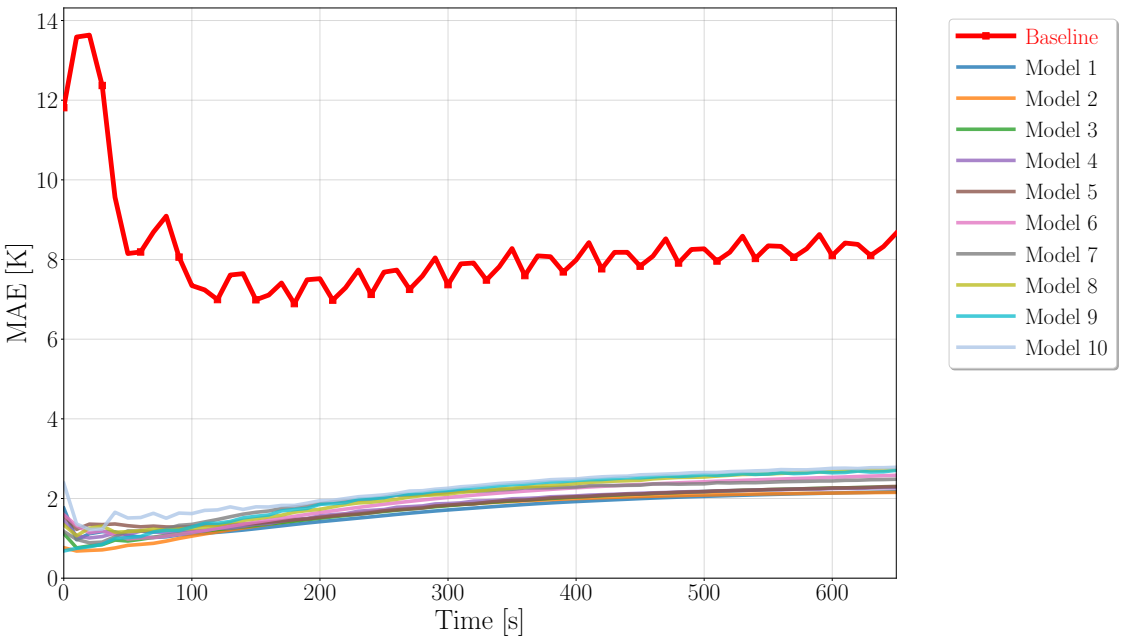
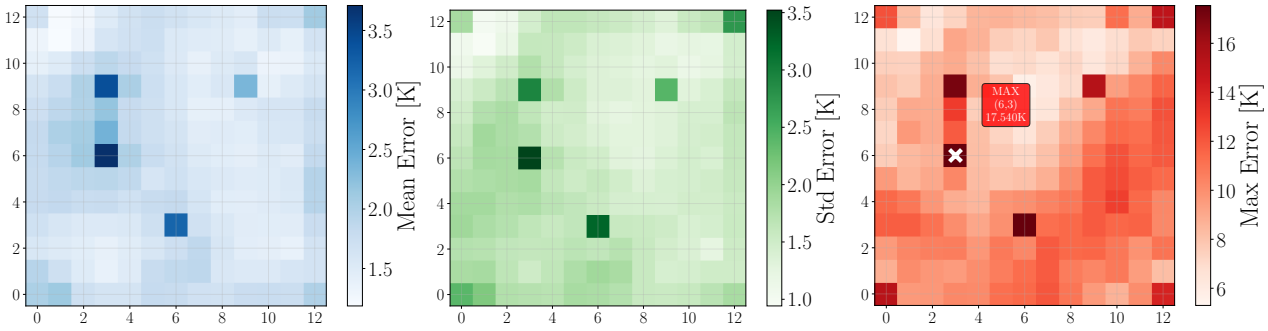


Figure 9.8: MAE vs. time for the top 10 physics-informed models and the baseline, with $n_{\text{train}} = 10$.

To gain further insight into the spatial error distribution of the best model, Figure 9.9a, Figure 9.9b, and Figure 9.9c show the mean, standard deviation, and maximum error across 100 test cases. Errors concentrate around heated regions: node (6,3), an active heater, is the most variable, while node (9,3)–also a heater– reaches the global maximum error.



(a) Mean error (b) Standard deviation (c) Maximum error

Figure 9.9: Spatial error distribution across all time steps for the best model at $n_{\text{train}} = 10$.

Temporal behavior is evaluated in Figure 9.14, which shows the error evolution over 650 s for three representative nodes: (6,6) in the center, (6,3) on a heater, and (0,0) at the interface. These nodes were selected to reflect different physical regions. It is appreciated that the error is generally higher at the beginning of the simulation, a trend maintained in most of the cases. This phenomenon may be explained because the network memory is not yet `loaded` (still has not enough information).

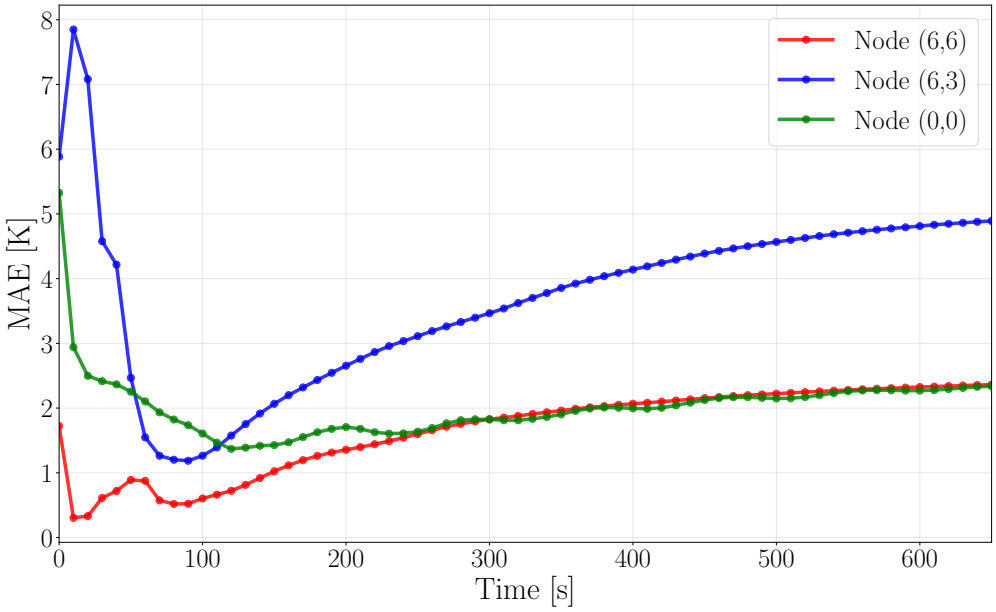
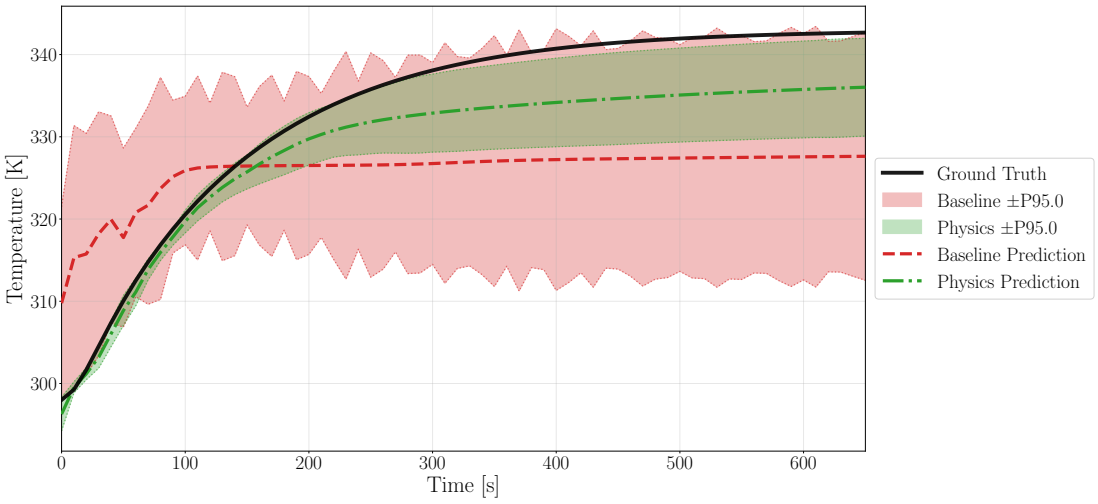


Figure 9.10: Temporal evolution of the prediction error at selected nodes, for $n_{\text{train}} = 10$.

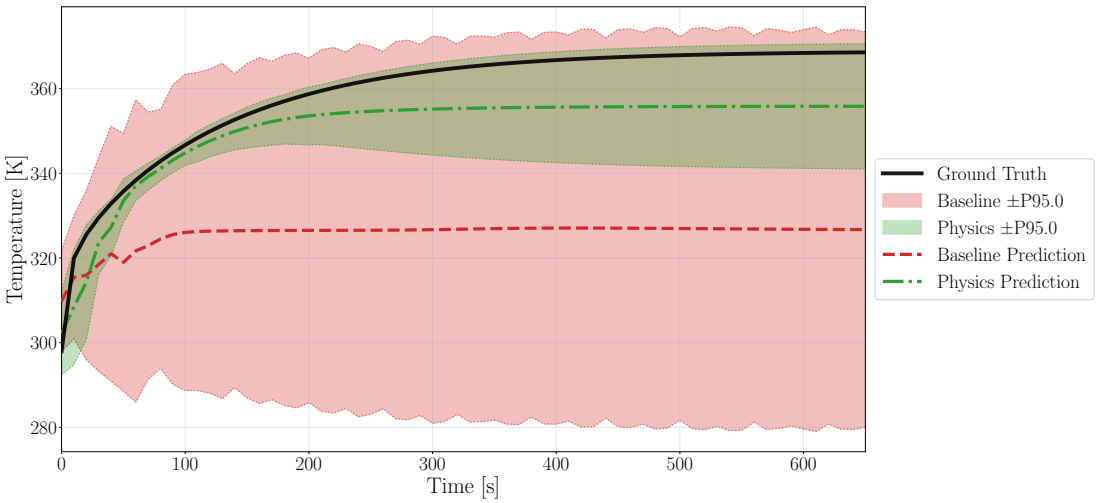
Finally, Figure 9.11 displays the predicted temperature with 95% confidence bands for the same three nodes in a random case selected from the validation subset. These bands are computed from 100 independent test cases and illustrate the improved precision of the PINN. For example, the mean confidence span at node (6,3) shrinks from 83.6 K in the baseline to just 20.9 K with physical regularization, as shown in Figure 9.11b.

Although great improvements are shown from the baseline to the physics-trained model, this one still shows some inaccuracies, as it can be clearly shown in Figure 9.11a.

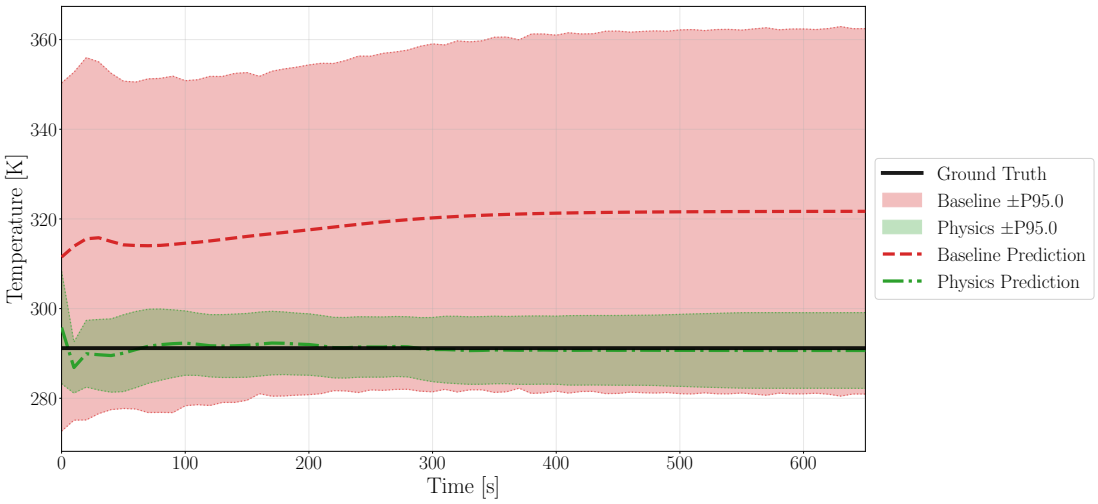
Confidence bands are constructed by adding and subtracting the 95th percentile MAE from the prediction at each node and time step. Thus, a narrower band implies higher predictive precision.



(a) (6,6)



(b) (6,3)



(c) (0,0)

Figure 9.11: Temperature evolution and 95% confidence bands at selected nodes, for $n_{\text{train}} = 10$.

9.2.0.2 $n_{\text{train}} = 50$

The next experiment increases the dataset to 50 training cases, still within a data-limited regime but more permissive than before. The goal is to assess whether the trend observed for $n_{\text{train}} = 10$ persists and how model performance scales with more data.

Table 9.7 confirms that physics-informed training continues to significantly outperform the baseline. In this case, improvements exceeding 90% in overall MAE are observed, indicating strong generalization with relatively little supervision. As before, the most successful models combine small but nonzero values of λ_{phy} and λ_{bnd}.

Table 9.7: Top 10 physics-informed models for $n_{\text{train}} = 50$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-2}	10^{-2}	0.2923	8.2837	+96.4%
2	1	10^{-4}	10^{-3}	0.3524	7.3460	+95.7%
3	1	10^{3}	10^{-2}	0.4055	22.4204	+95.0%
4	1	10^{-1}	10^{-2}	0.4259	11.6117	+94.8%
5	1	10^{-5}	10^{-4}	0.4270	12.8592	+94.8%
6	1	10^{-5}	10^{-3}	0.4438	9.7711	+94.6%
7	1	10^{2}	10^{-2}	0.5307	15.4444	+93.5%
8	1	10^{-3}	10^{-4}	0.5377	19.2638	+93.4%
9	1	1	10^{-3}	0.5521	14.2679	+93.2%
10	1	10^{-1}	0	0.5693	16.8806	+93.0%
Baseline	1	0	0	8.1499	53.8969	–

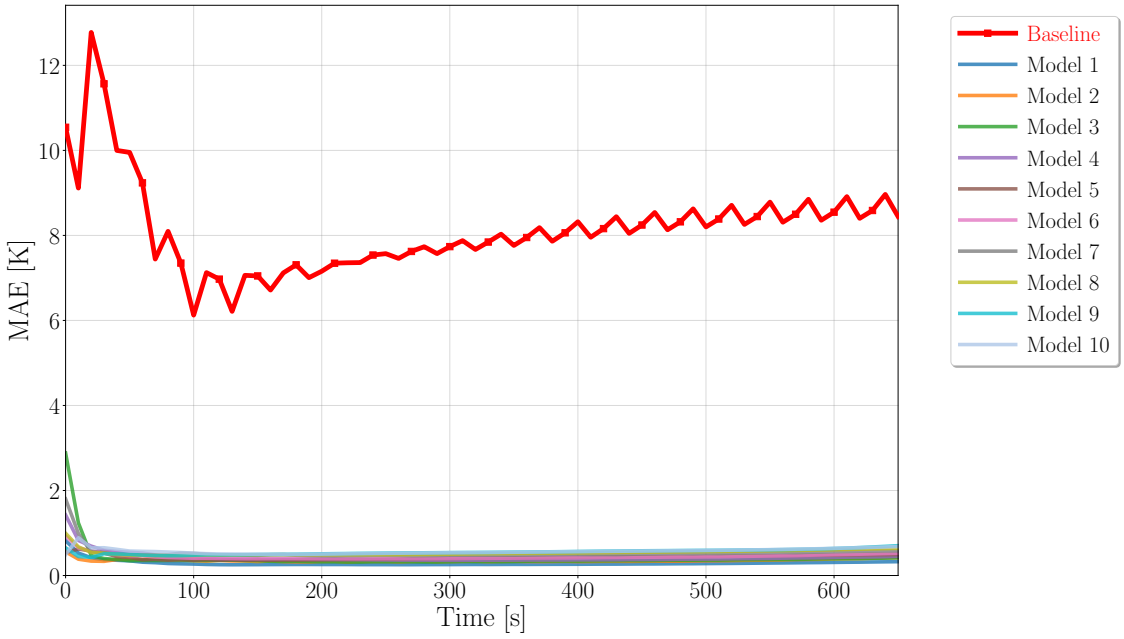
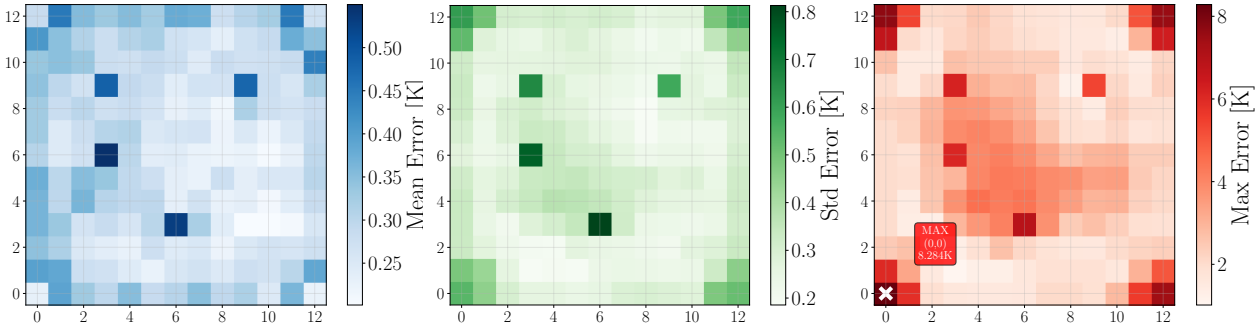


Figure 9.12: MAE vs. time for the top 10 physics-informed models and the baseline, with $n_{\text{train}} = 50$.

To understand the spatial distribution of errors in the best model, Figure 9.13 shows the mean, standard deviation, and maximum error per node across the 100 test cases. The worst-case error is located at node (0,0), at a corner interface, while the most variable region appears near node (3,6), a heater. The average error remains low in most of the domain.



(a) Mean error (b) Standard deviation (c) Maximum error

Figure 9.13: Spatial error distribution across all time steps for the best model at $n_{\text{train}} = 50$.

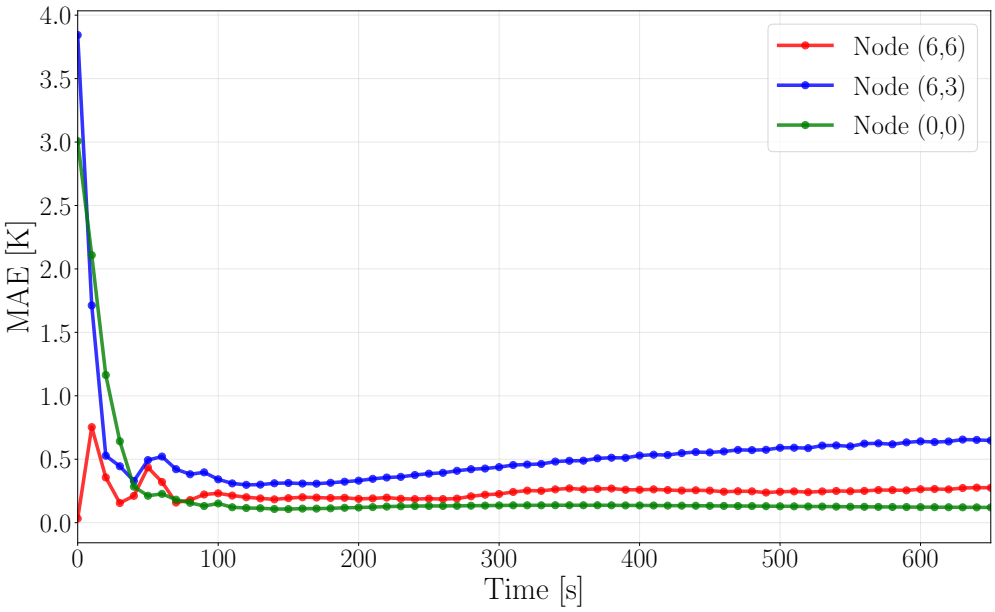
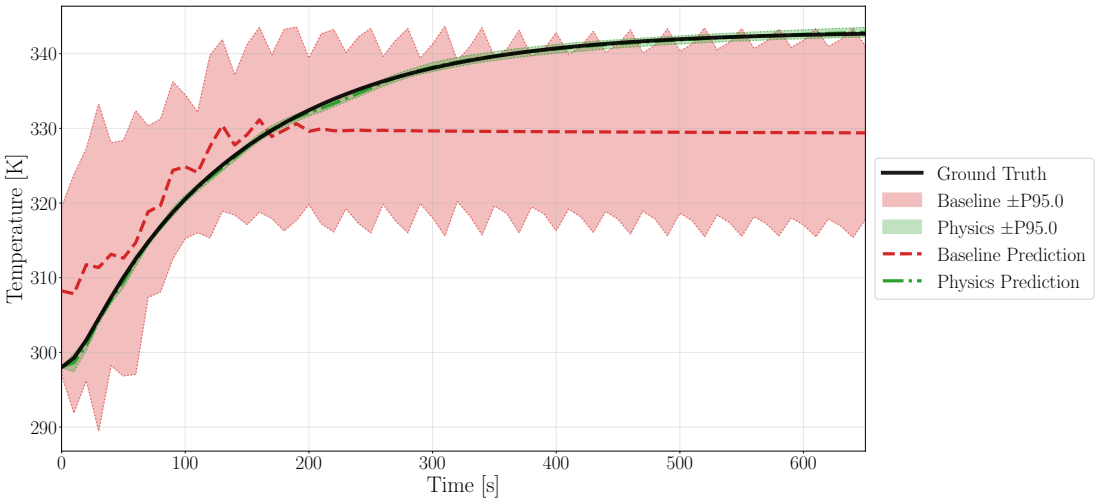


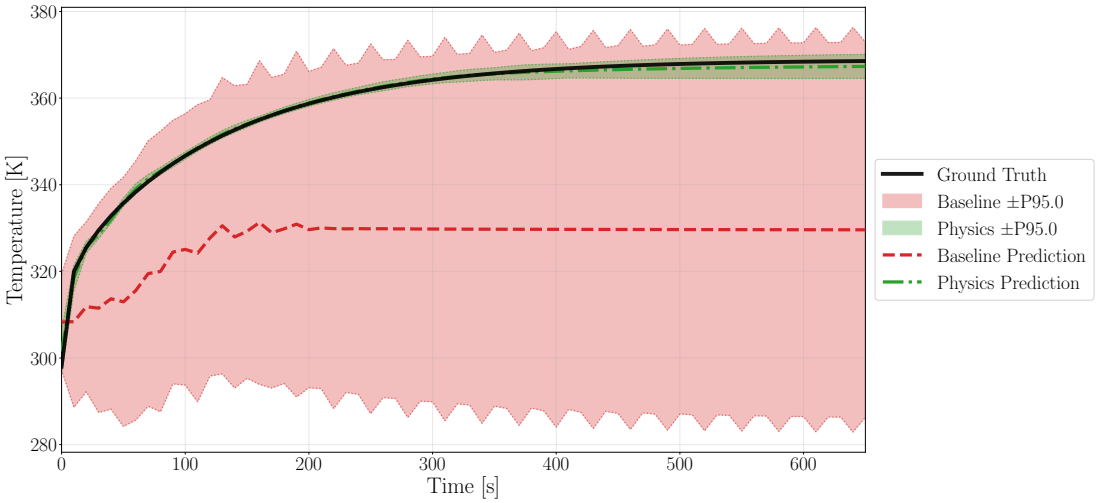
Figure 9.14: Temporal evolution of the prediction error at selected nodes, for $n_{\text{train}} = 50$.

As in the previous analysis for $n_{\text{train}} = 10$, the temporal behavior and prediction confidence are assessed for three representative nodes: the center (6,6), a heater (6,3), and an interface (0,0). The results in Figure 9.15 show that physics-informed training leads to significantly narrower 95% confidence bands—up to 95% tighter than the baseline.

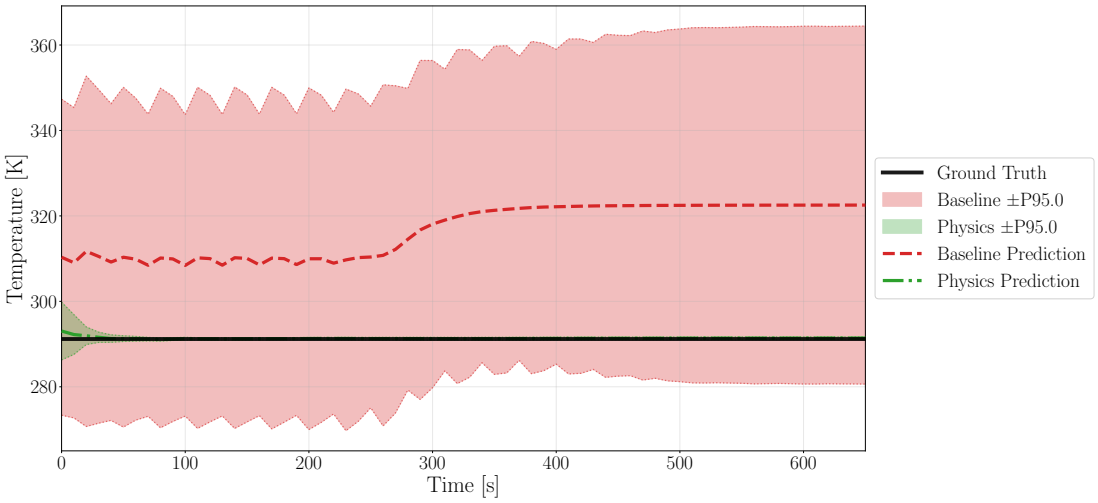
A big improvement can be observed from Figure 9.11–specially in the physics model–where the confidence bands are really narrow, hence showing a very precise model.



(a) (6,6)



(b) (6,3)



(c) (0,0)

Figure 9.15: Temperature evolution and 95% confidence bands at selected nodes, for $n_{\text{train}} = 50$.

9.2.0.3 $n_{\text{train}} = 200$

This experiment increases the training set to 200 cases, entering a more standard data regime. The objective is to assess whether the advantages of physics-informed losses persist at this scale and how the model's behavior evolves as more data becomes available.

As shown in Table 9.8, PINNs still considerably outperform the baseline. Surprisingly, the relative improvement keeps greater than 90%. The best model achieves an overall MAE of just 0.188 K, using a very small physics weight $\lambda_{\text{phy}} = 10^{-6}$ and a moderate boundary weight $\lambda_{\text{bnd}} = 10^{-3}$. This same model gets a maximum error of 4.94 K, just below the 5 K threshold mentioned along the whole work. For that, it can be shown that with just 200 training cases–just over 36 hours [3] of data– can predict the model behavior with the desired accuracy.

Table 9.8: Top 10 physics-informed models for $n_{\text{train}} = 200$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-6}	10^{-3}	0.1880	4.9379	+95.6%
2	1	10^{-5}	10^{-3}	0.1897	7.2890	+95.6%
3	1	10^{-2}	10^{-3}	0.1979	6.1811	+95.4%
4	1	10^{-6}	10^{-2}	0.2053	7.3129	+95.2%
5	1	10^{-3}	10^{-2}	0.2060	8.3923	+95.2%
6	1	10^{-6}	10^{-4}	0.2145	6.2305	+95.0%
7	1	1	0	0.2238	12.3674	+94.8%
8	1	10^{-2}	10^{-4}	0.2246	6.1144	+94.8%
9	1	10^{-4}	10^{-2}	0.2258	5.3254	+94.7%
10	1	10^{-5}	0	0.2278	6.7252	+94.7%
Baseline	1	0	0	4.2818	57.7394	–

[3] 650 seconds/case $\times$ 200 cases $\times \frac{1 \text{ hour}}{3600 \text{ seconds}} = 36.11$ hours

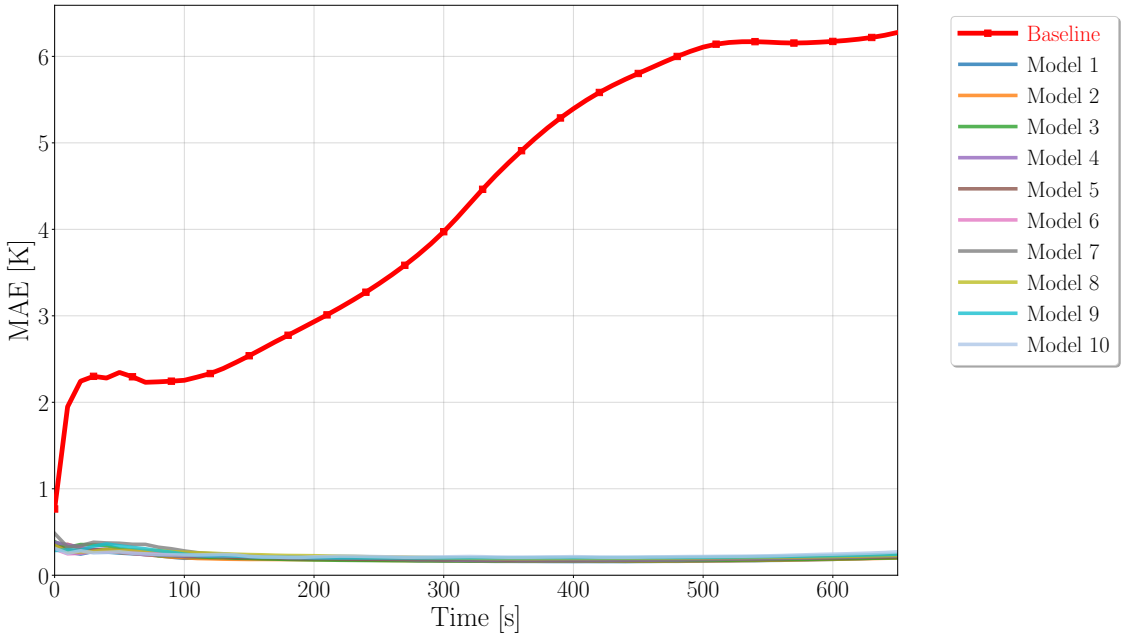
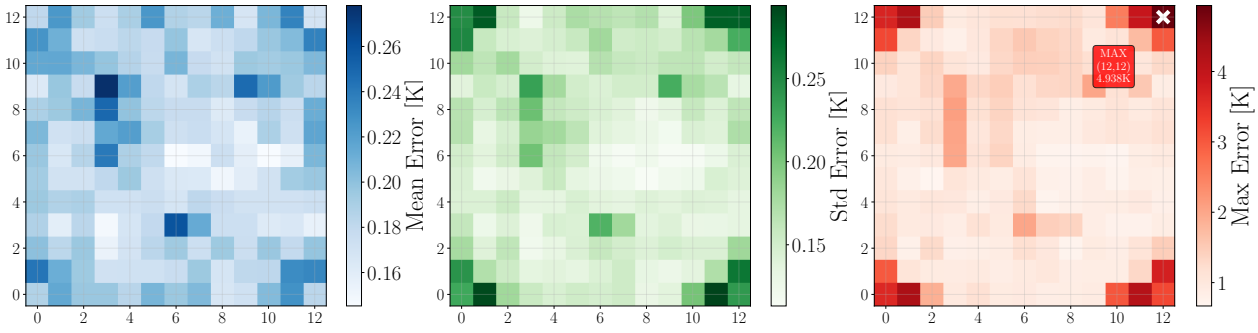


Figure 9.16: MAE vs. time for the top 10 physics-informed models and the baseline, with $n_{\text{train}} = 200$.

The spatial distribution of errors for the best model is shown in Figure 9.17, with maps of the mean, standard deviation, and maximum error across all nodes and time steps. The node with the highest standard deviation and global maximum error is located at (13, 13), with a peak of 4.94 K.



(a) Mean error (b) Standard deviation (c) Maximum error

Figure 9.17: Spatial error metrics for the best model with $n_{\text{train}} = 200$.

As in previous sections, the temporal evolution of the prediction error is evaluated at nodes (6,6), (6,3), and (0,0). The results in Figure 9.18 confirm that errors remain low and stable over time.

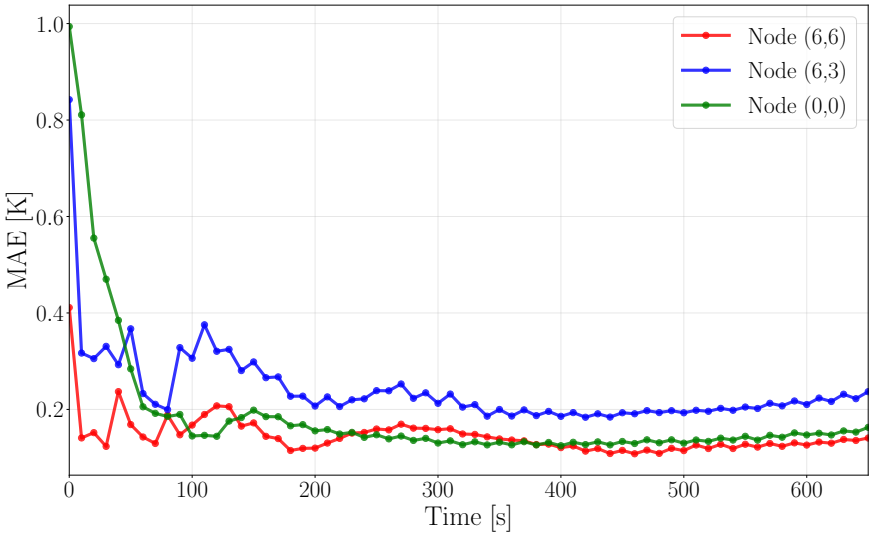
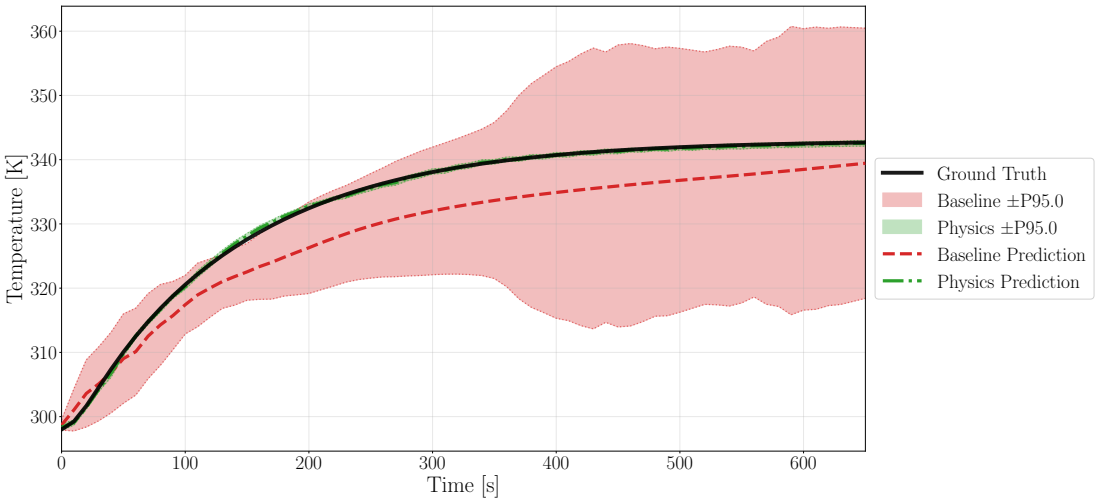


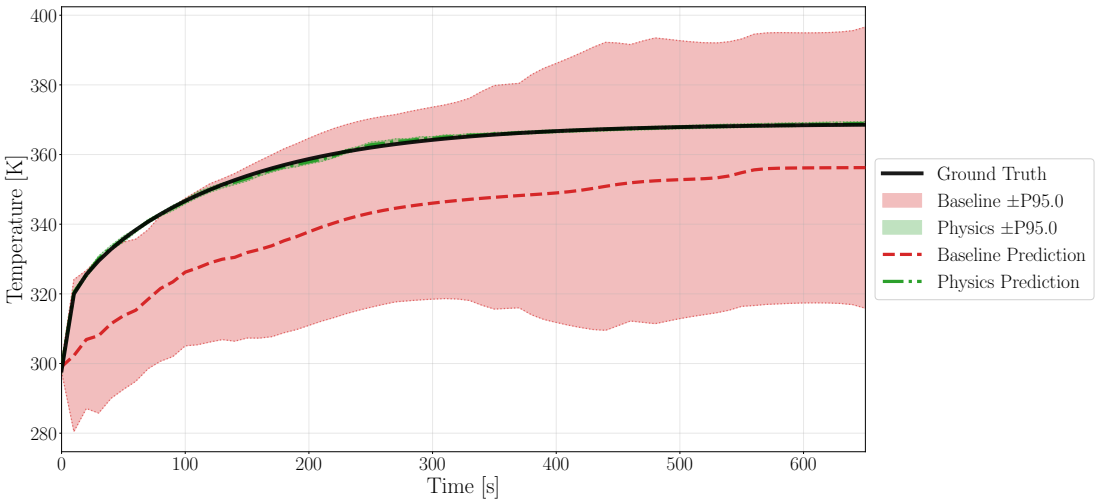
Figure 9.18: Temporal evolution of the prediction error at selected nodes, for $n_{\text{train}} = 200$.

Finally, Figure 9.19 shows the predicted temperature and corresponding 95% confidence bands for the same three nodes. As in earlier experiments, the incorporation of physics constraints leads to narrower uncertainty bands and improved predictive reliability across the domain.

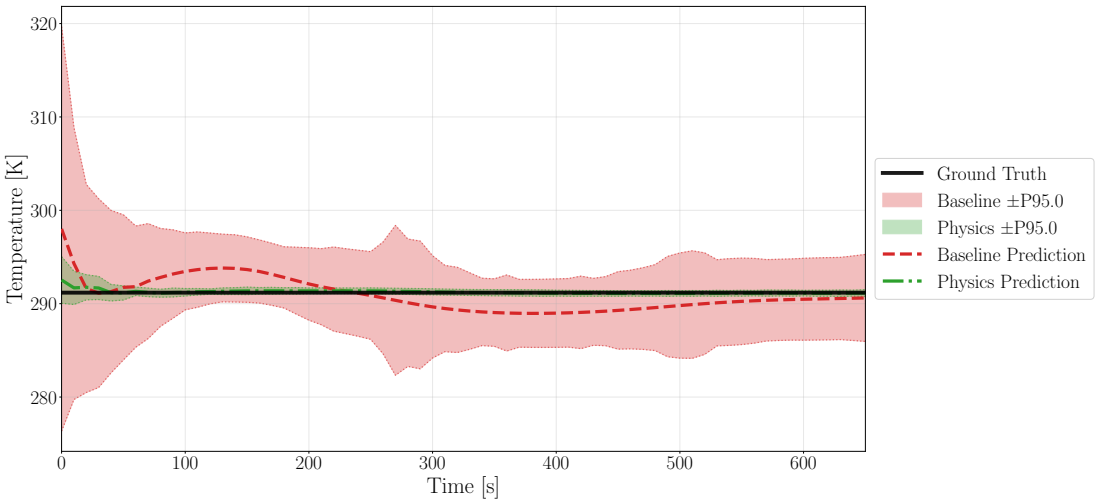
Once again, the physics model clearly outperforms the baseline one, with such precision that the band can hardly be distinguished from the ground truth. Although the baseline model performs fairly well, there are errors that show that it is not yet ready for trustworthy predictions.



(a) (6,6)



(b) (6,3)



(c) (0,0)

Figure 9.19: Temperature evolution and 95% confidence bands at selected nodes, for $n_{\text{train}} = 200$.

9.2.0.4 $n_{\text{train}} = 400$

This experiment evaluates the model when 400 training cases are available. As shown in Table 9.9, the performance gap between different configurations becomes narrower, yet physics-informed training continues to provide a substantial improvement over the MSE-only baseline. This time, many physic models achieve predictions all below the threshold of 5 K.

Table 9.9: Top 10 physics-informed models for $n_{\text{train}} = 400$.

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
1	1	1	0	0.1215	4.2932	+91.0%
2	1	10^{-2}	10^{-3}	0.1258	3.6793	+90.7%
3	1	0	10^{-3}	0.1283	3.6538	+90.5%
4	1	10^{-7}	0	0.1350	4.3961	+90.0%
5	1	10^{-5}	0	0.1357	4.0503	+90.0%
6	1	10^{-4}	10^{-3}	0.1366	3.2666	+89.9%
7	1	10^{-7}	10^{-4}	0.1428	4.7796	+89.4%
8	1	10^{-6}	10^{-3}	0.1476	3.1357	+89.1%
9	1	10^{-6}	0	0.1544	5.7311	+88.6%
10	1	10^{-1}	10^{-4}	0.1584	5.6473	+88.3%
Baseline	1	0	0	1.3527	30.4930	–

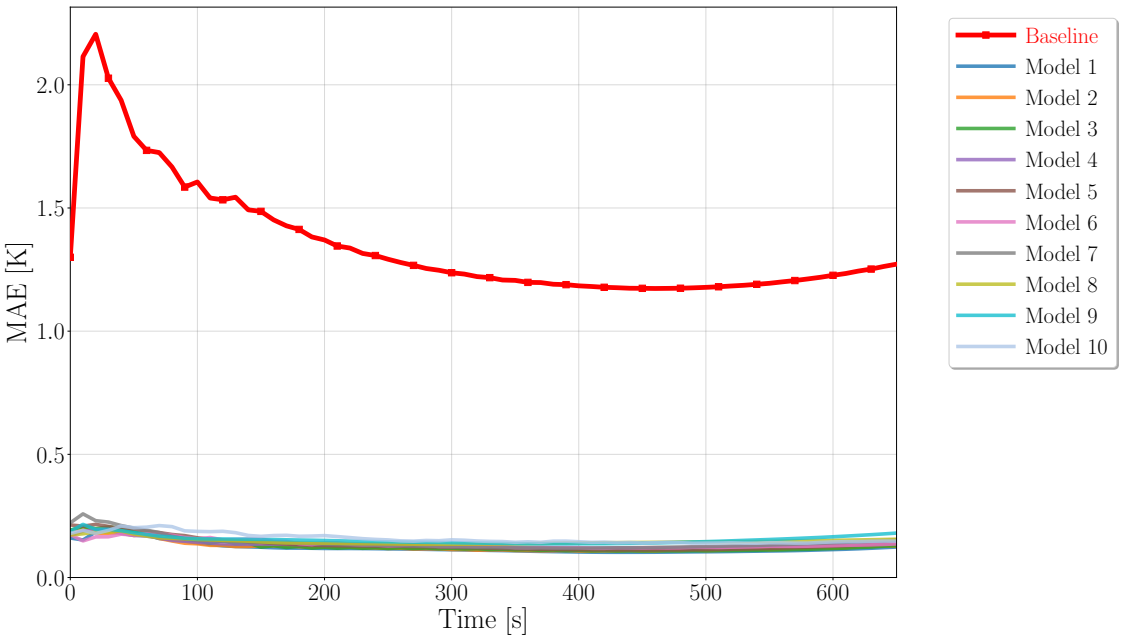
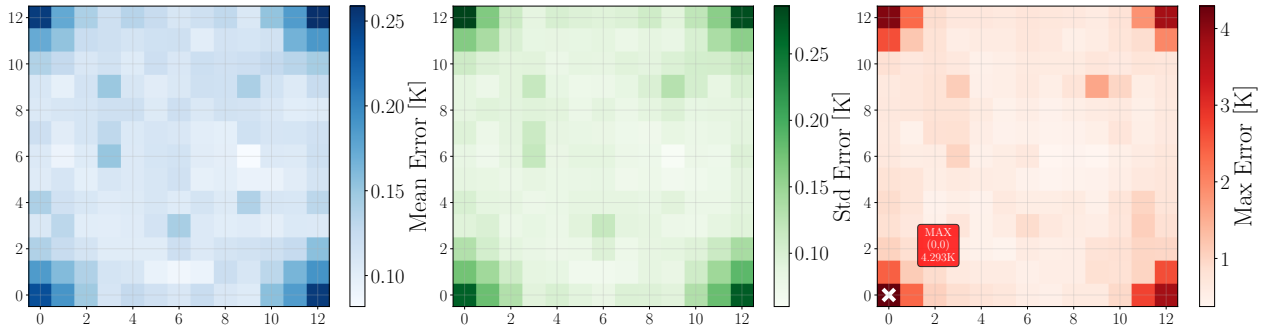


Figure 9.20: MAE vs. time for the top 10 physics-informed models and the baseline, with $n_{\text{train}} = 400$.

The best model corresponds to a configuration with $\lambda_{\text{phy}} = 1.0$ and no boundary loss. It achieves both the lowest overall and final MAE, with a 90% improvement over the baseline. Interestingly, the lowest maximum error is achieved by a different configuration (rank 8), highlighting that optimal settings may vary depending on the metric of interest.

To analyze the spatial distribution of errors, Figure 9.21 shows the mean, standard deviation, and maximum error across all 100 test cases. The model achieves a global average error of 0.1215 K and a maximum one of 4.29 K, placed at node (0,0). Node (12,0) shows the greatest variability.



(a) Mean error (b) Standard deviation (c) Maximum error

Figure 9.21: Spatial error metrics for the best model with $n_{\text{train}} = 400$.

Temporal prediction errors at the same three representative nodes, which are shown in Figure 9.22. These were selected consistently across all experiments to represent the central, heater, and boundary zones of the domain.

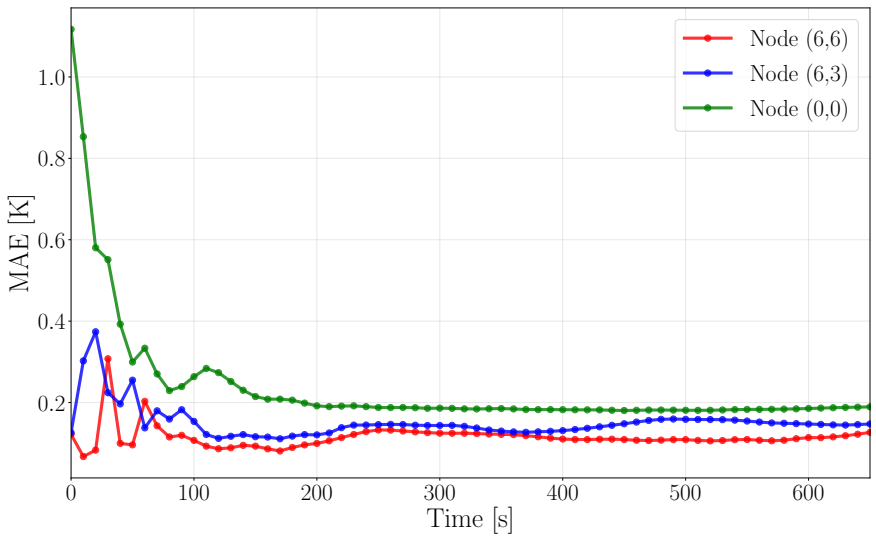
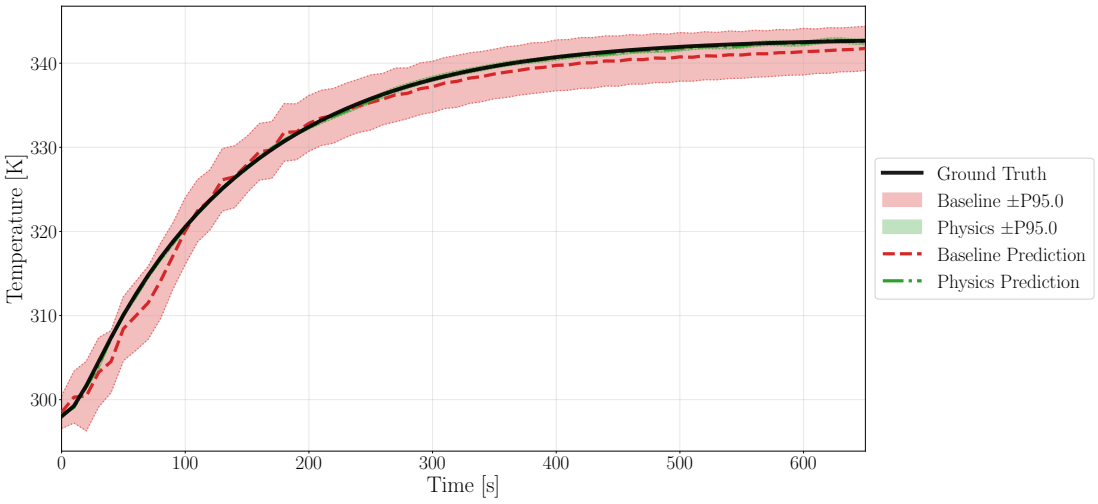


Figure 9.22: Temporal evolution of the prediction error at selected nodes, for $n_{\text{train}} = 400$.

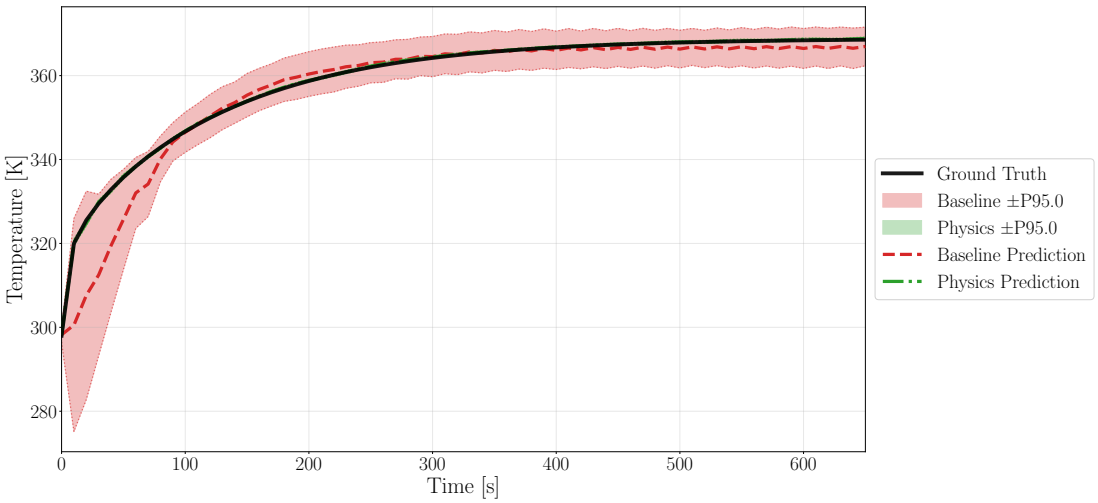
Finally, Figure 9.23 shows the predicted temperature and 95% confidence bands for the same

nodes. As observed in prior experiments, incorporating physics constraints leads to tighter confidence intervals, particularly at nodes with stronger thermal gradients, such as heaters.

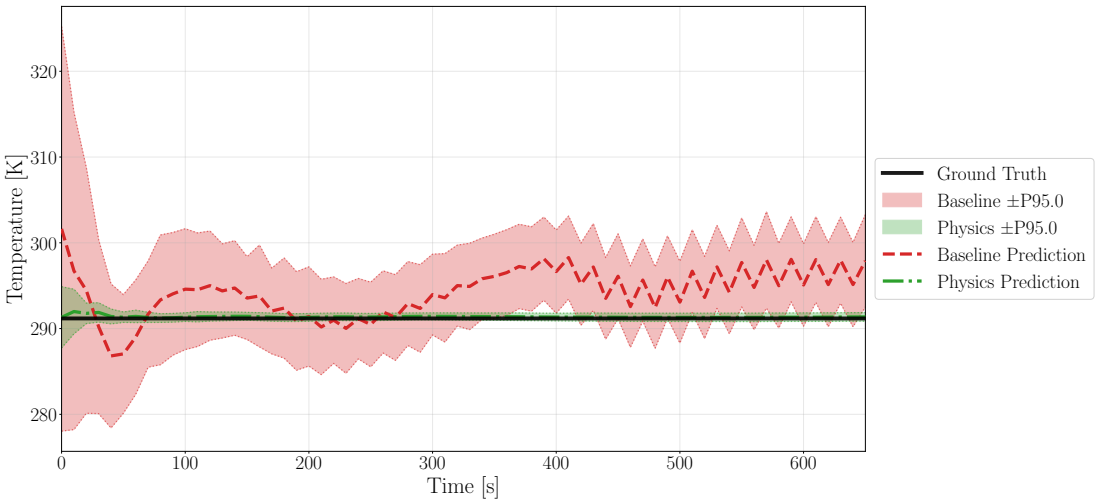
This time, the PINN model shows outstanding performance once again in all representations. Even though Figure 9.23c appears to be much less accurate, it is due to the scale factor in the y-axis. The mean band withs for Figure 9.23a, Figure 9.23b, and Figure 9.23c is 0.5317 K, 0.6983 K, and 1.1570 K, respectively. However, this slight difference could have been predicted from Figure 9.21 and Figure 9.22, where the node (0,0) showed the biggest error.



(a) (6,6)



(b) (6,3)



(c) (0,0)

Figure 9.23: Temperature evolution and 95% confidence bands at selected nodes, for $n_{\text{train}} = 400$.

9.2.0.5 $n_{\text{train}} = 500$

In the next experiment, the training dataset is increased to 500 cases, entering a data-rich regime. The goal here is to assess whether adding more data continues to improve performance, and whether physics-informed models maintain their advantage.

As shown in Table 9.10, the top physics-informed model achieves a 67.1% improvement over the baseline, despite all models sharing the same architecture and training setup. Nearly all top-ranked configurations include a nonzero physics weight λ_{phy}, reinforcing its consistent regularization effect across different training sizes.

Table 9.10: Top 10 physics-informed models for $n_{\text{train}} = 500$.

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-6}	10^{-3}	0.1156	4.0574	+67.1%
2	1	0	10^{-3}	0.1169	5.1126	+66.7%
3	1	1	0	0.1209	4.3138	+65.6%
4	1	10^{-1}	10^{-4}	0.1222	3.5352	+65.2%
5	1	10^{-2}	0	0.1232	6.4490	+64.9%
6	1	10^{-4}	0	0.1300	3.0866	+63.0%
7	1	10^{-1}	10^{-3}	0.1308	3.3396	+62.7%
8	1	10^{-7}	10^{-4}	0.1315	4.0293	+62.5%
9	1	10^{-7}	0	0.1319	5.2489	+62.4%
10	1	10^{-3}	10^{-4}	0.1340	5.4654	+61.8%
Baseline	1	0	0	0.3512	17.1021	–

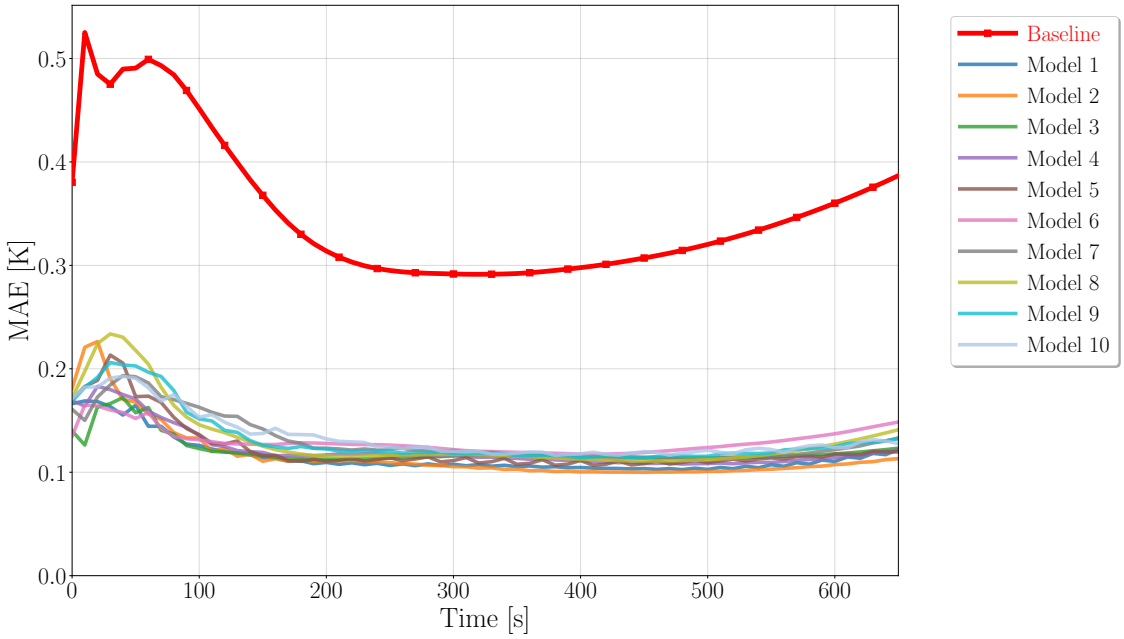
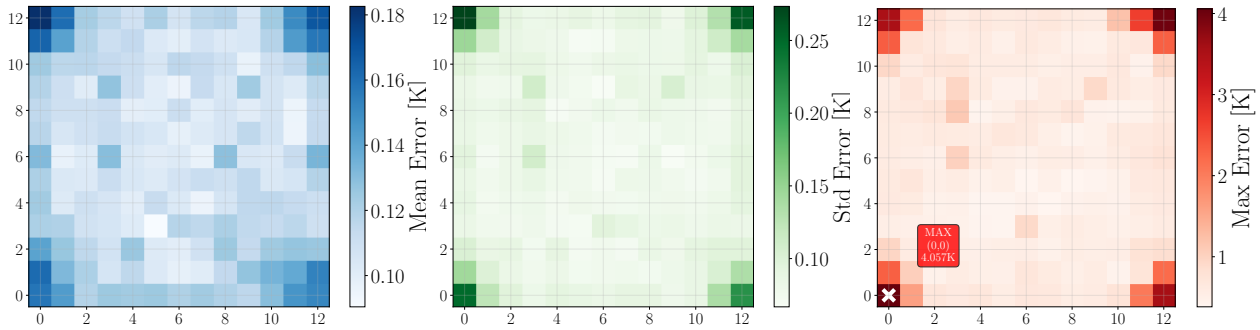


Figure 9.24: MAE vs. time for the top 10 physics-informed models and the baseline, with $n_{\text{train}} = 500$.

To better understand the spatial error distribution of the best model, Figure 9.25 displays the domain-wide mean, standard deviation, and maximum error across all 100 test cases. As in previous experiments, errors concentrate near heater nodes such as (6,3), and specially around boundary nodes like (0,0), while the interior remains more accurate.



(a) Mean error (b) Standard deviation (c) Maximum error

Figure 9.25: Spatial distribution of error metrics across all time steps for the best model at $n_{\text{train}} = 500$.

Temporal error evolution at the three representative nodes is shown in Figure 9.26. This selection matches the setup used in earlier sections, enabling consistent cross-comparison.

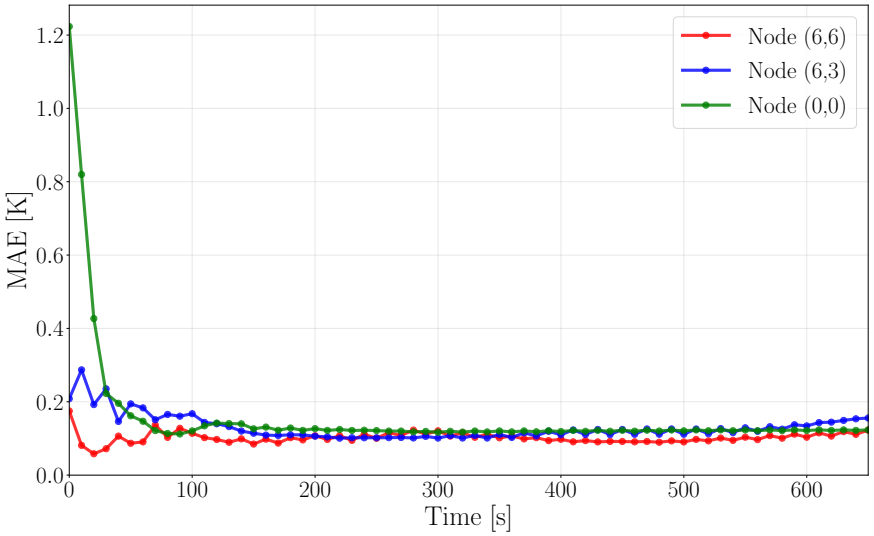
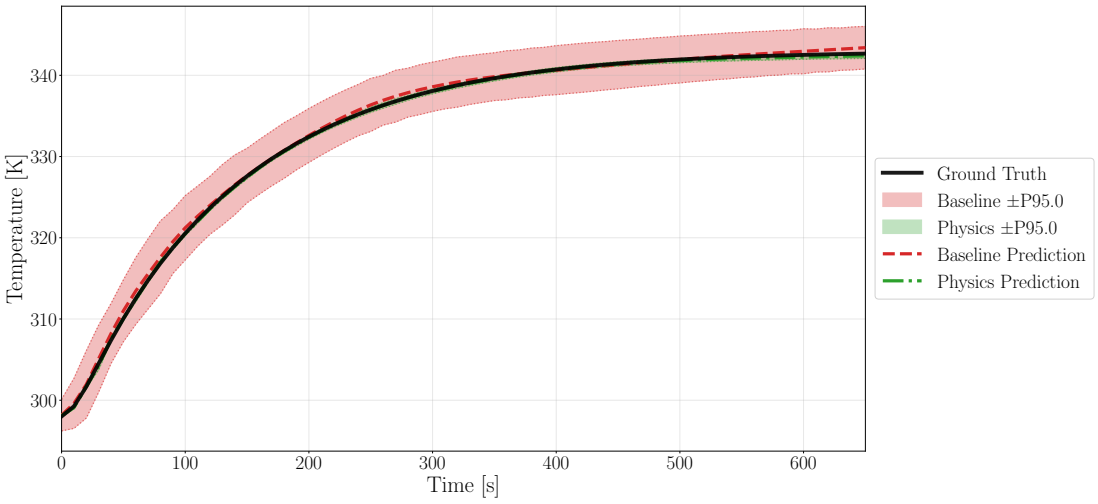
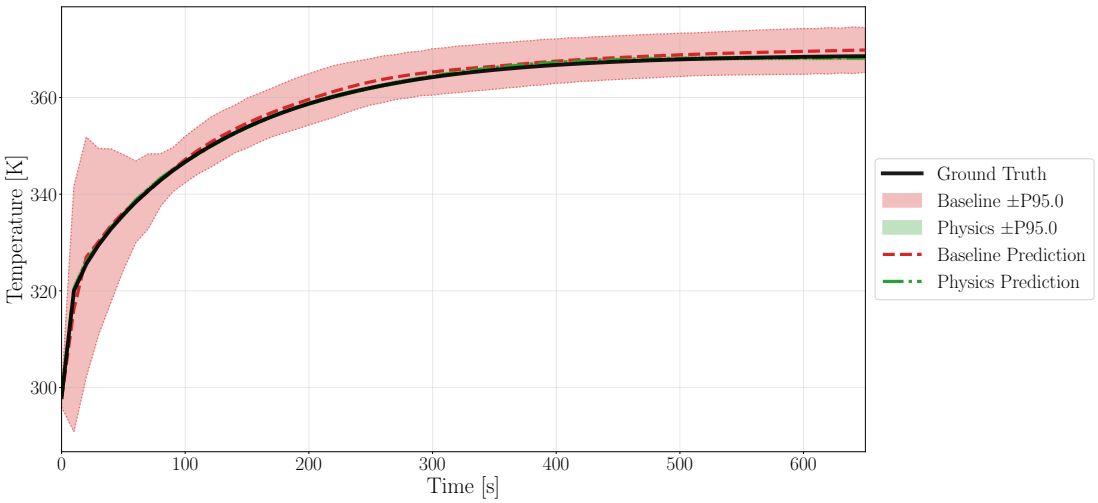


Figure 9.26: Temporal evolution of the prediction error at selected nodes, for $n_{\text{train}} = 500$.

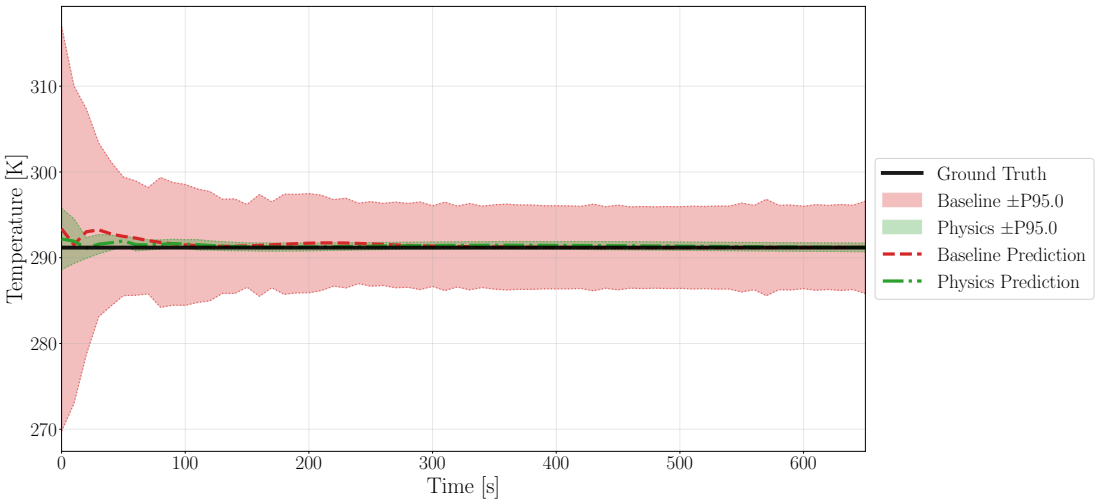
Finally, Figure 9.27 illustrates the predicted temperatures and ±95% confidence bands for the same three nodes. As in previous cases, the addition of physics-based regularization leads to significantly narrower uncertainty intervals, especially at the heater node (6,3), where thermal gradients are more pronounced.



(a) (6,6)



(b) (6,3)



(c) (0,0)

Figure 9.27: Temperature evolution and 95% confidence bands at selected nodes, for $n_{\text{train}} = 500$.

9.2.0.6 $n_{\text{train}} = 1000$

In the final experiment, the training dataset is extended to 1000 cases, entering a highly data-rich regime. The goal is to examine whether physics-informed models still offer measurable benefits when large amounts of training data are available.

As shown in Table 9.11, the top-ranked physics-informed model achieves a 67.1% reduction in MAE compared to the baseline. Interestingly, three of the four best configurations use a small physical weight $\lambda_{\text{phy}} = 10^{-5}$, suggesting that even a mild regularization can significantly enhance performance in this setting.

Table 9.11: Top 10 physics-informed models for $n_{\text{train}} = 1000$, ordered by overall MAE.

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-5}	10^{-3}	0.0869	3.8003	+67.1%
2	1	1	10^{-3}	0.0896	4.1618	+66.0%
3	1	10^{-4}	10^{-3}	0.0918	3.5979	+65.2%
4	1	10^{-2}	10^{-5}	0.0935	5.0900	+64.6%
5	1	10^{-3}	10^{-4}	0.0935	2.7901	+64.5%
6	1	10^{-2}	10^{-3}	0.0981	3.6680	+62.8%
7	1	10^{-1}	10^{-5}	0.0988	4.0350	+62.6%
8	1	10^{-5}	10^{-4}	0.1010	3.4616	+61.7%
9	1	10^{-5}	10^{-2}	0.1044	8.7046	+60.4%
10	1	10^{-3}	10^{-3}	0.1048	4.6641	+60.3%
Baseline	1	0	0	0.2638	12.2686	–

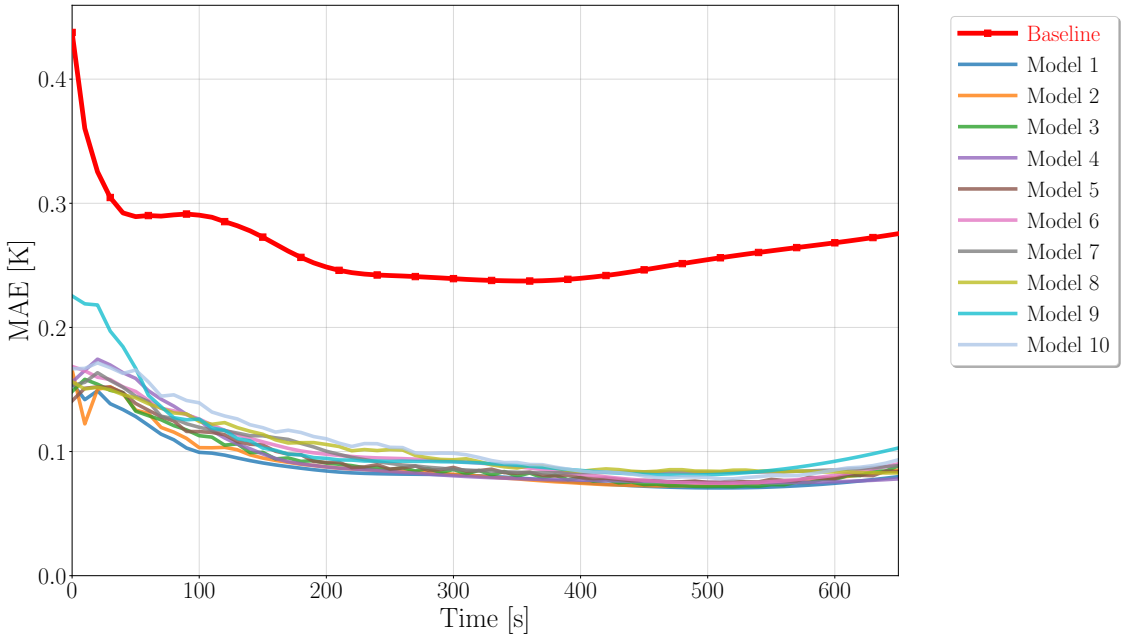
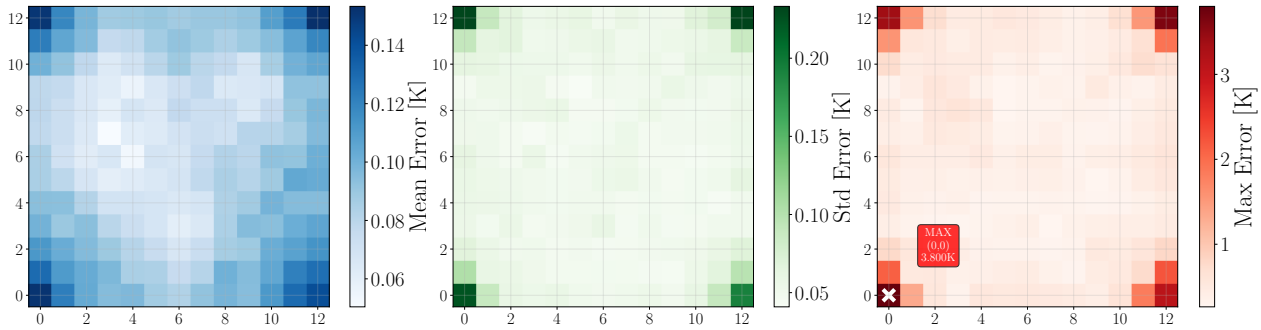


Figure 9.28: MAE vs. time for the top physics-informed models and the baseline, with $n_{\text{train}} = 1000$.

To better understand the spatial error distribution of the best model, Figure 9.29 displays the domain-wide mean, standard deviation, and maximum error across all 100 test cases. While the central region remains accurate, boundary nodes such as (0,0) show higher errors. Node (12,12) is the most variable, while (9,5) is the most uniform.



(a) Mean error (b) Standard deviation (c) Maximum error

Figure 9.29: Spatial distribution of error metrics across all time steps for the best model at $n_{\text{train}} = 1000$.

Temporal error evolution at the three representative nodes is shown in Figure 9.30, confirming the robustness of the physics-informed prediction throughout the simulation horizon.

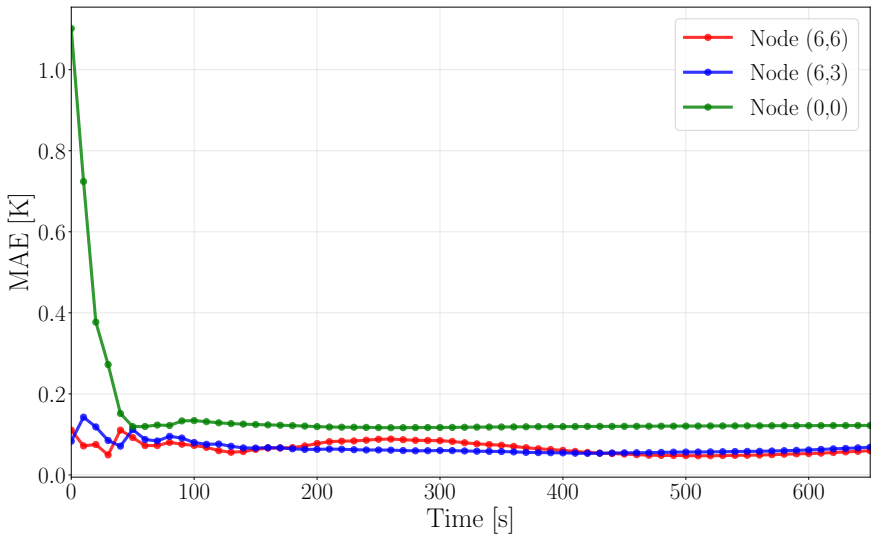
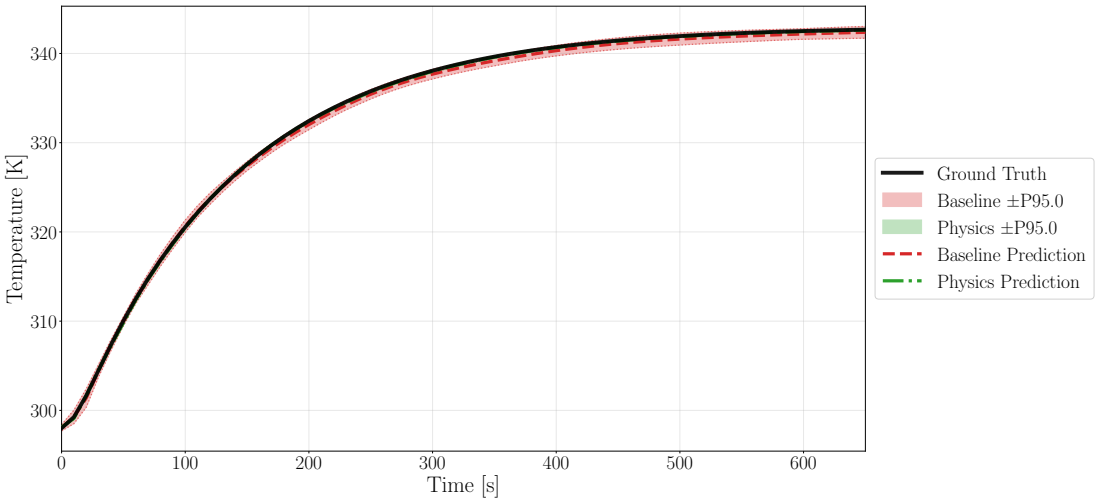
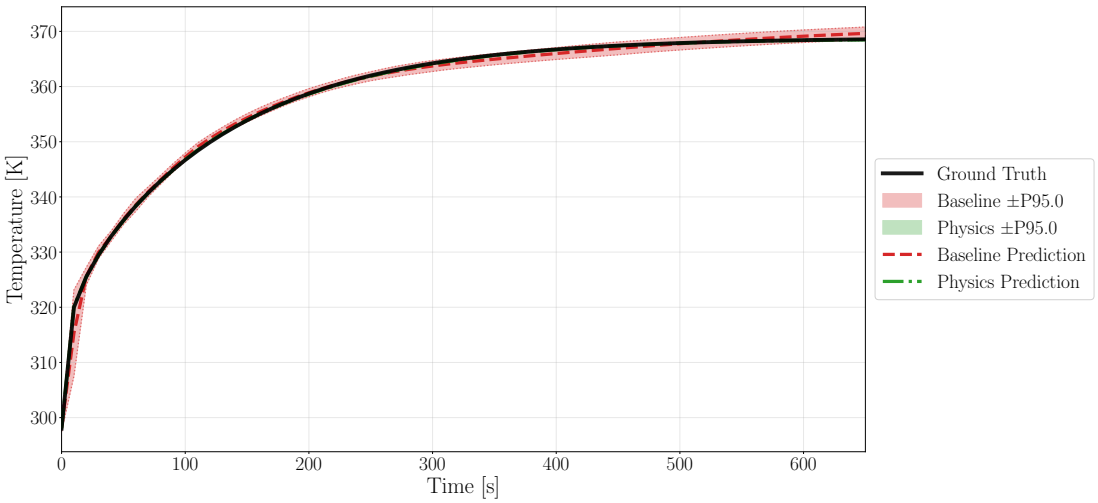


Figure 9.30: Temporal evolution of the prediction error at selected nodes, for $n_{\text{train}} = 1000$.

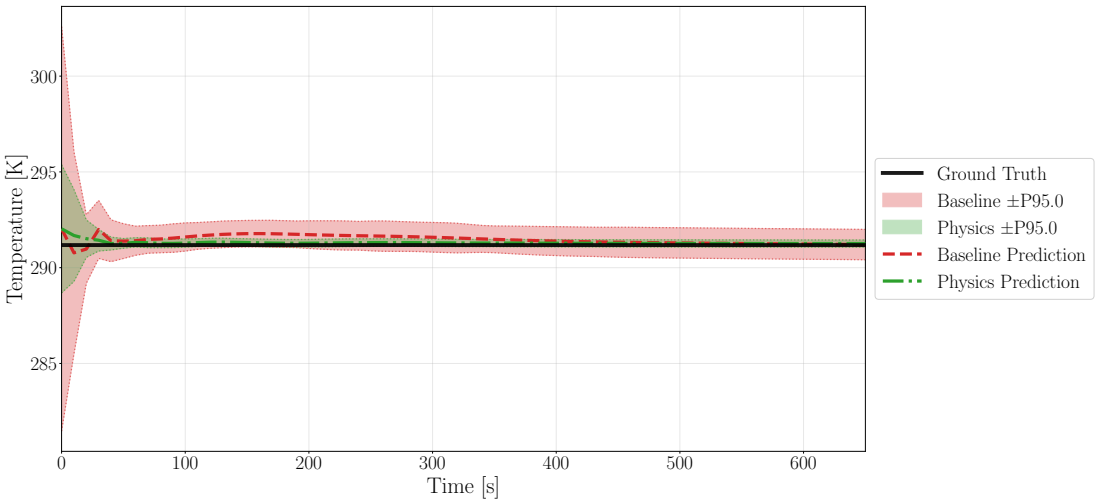
Finally, Figure 9.31 presents the predicted temperatures and ±95% confidence bands at the same nodes. Compared to the baseline, the physics-informed model exhibits significantly tighter uncertainty margins, with an improvement of 0.16–0.39 K depending on the region.



(a) (6,6)



(b) (6,3)



(c) (0,0)

Figure 9.31: Temperature evolution and 95% confidence bands at selected nodes, for $n_{\text{train}} = 1000$.

9.2.1 Impact of training size on performance and cost

After analyzing each training size individually, this section synthesizes the overall trends across experiments. The goal is to understand how the benefit of physics-informed training evolves with increasing data, and to quantify the associated training cost.

9.2.1.1 Improvement versus training size

Figure 9.32 and Figure 9.33 show the absolute and relative improvement in prediction error, respectively, obtained by the best PINN for each training set size. In all cases, the baseline corresponds to a model trained purely with the MSE loss.

The highest improvement is achieved for $n_{\text{train}} = 50$, with a reduction of almost 8 K in overall MAE (96.4%). This aligns with the goal of using PINNs to improve generalization under low-data regimes. As more training data becomes available, the improvement decreases, yet remains significant even for $n = 1000$, where a 67.1% error reduction is still observed.

This trend is confirmed by a negative Pearson-product moment correlation coefficient [78] of -0.849 between the number of training samples and the improvement percentage, suggesting diminishing returns of the physics term in data-rich settings.

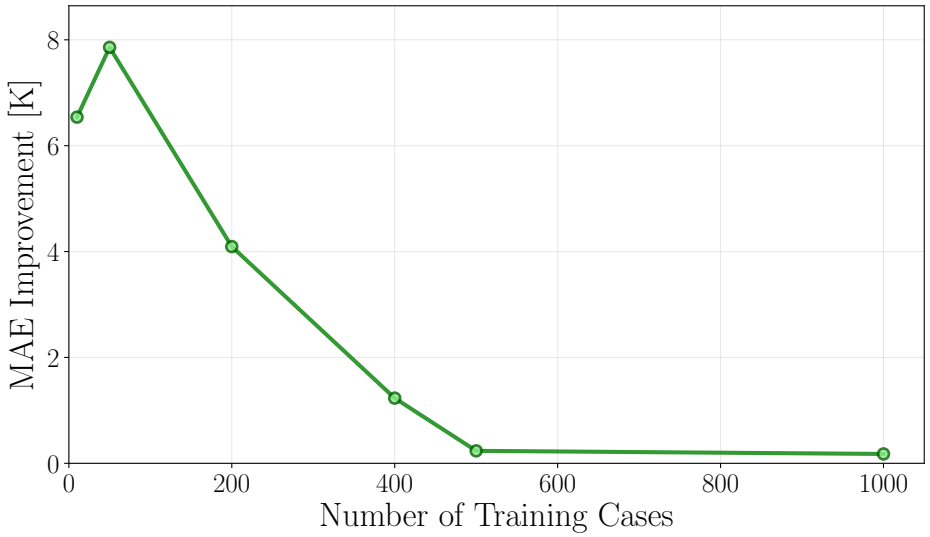


Figure 9.32: Absolute reduction in overall MAE achieved by the best PINN at each training size.

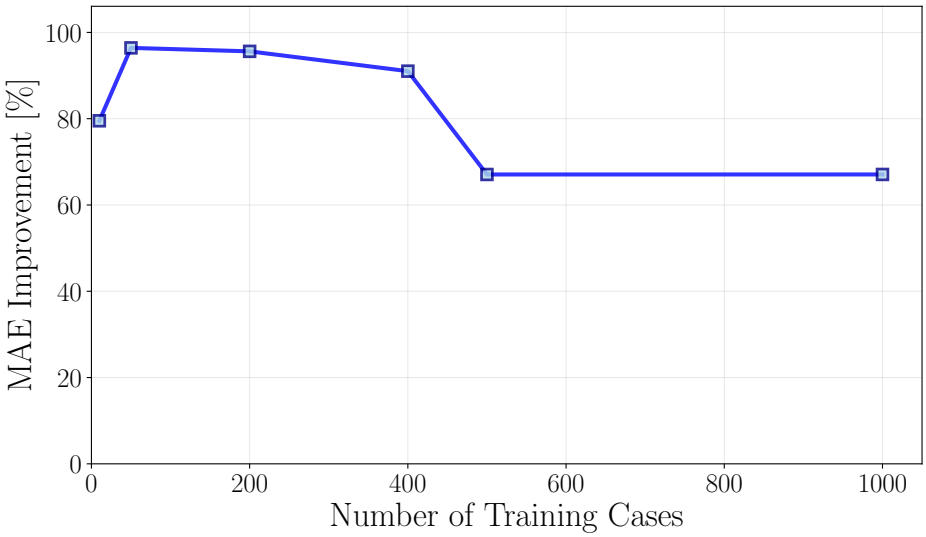


Figure 9.33: Relative improvement (%) in overall MAE obtained by the best PINN at each training size.

9.2.1.2 Training time overhead

To quantify the computational cost of physics-informed training, we measured the training time of the best model for each training size and compared it to the baseline. As shown in Figure 9.34, the absolute training time increases with dataset size, and so does the difference between the PINN model and the baseline one. This difference—referred to as the physics-induced overhead—represents the additional time required to compute and backpropagate the physical loss terms at each step, on top of the standard MSE loss. This overhead becomes more noticeable as the dataset grows, emphasizing the need for powerful GPUs, such as the L4 described in Table 8.1, to keep training times manageable.

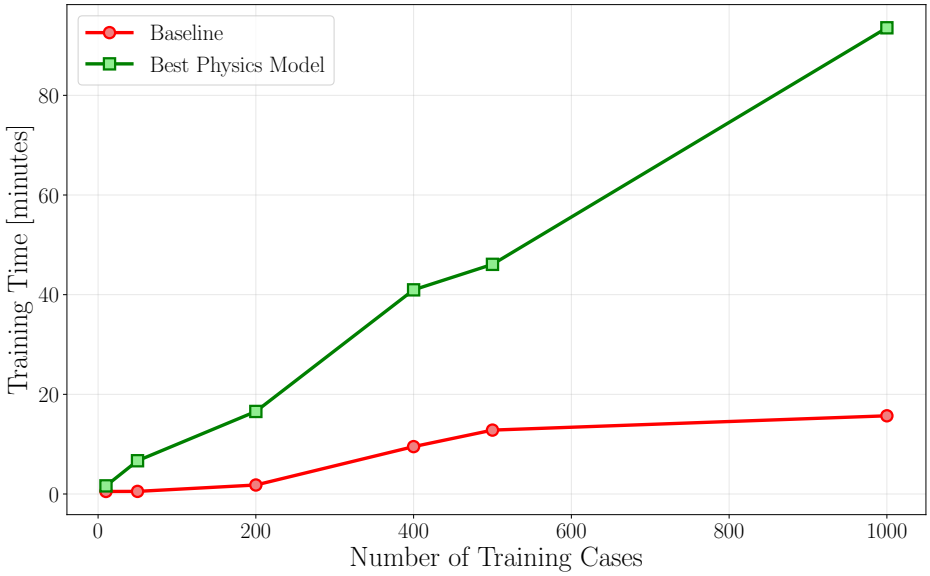


Figure 9.34: Total training time for the baseline and the best physics-informed model at each training size.

Figure 9.35 displays the relative overhead. In extremely low-data settings, the overhead is massive (e.g., 1176% for $n = 50$) due to the early stopping of baseline models that cannot extract that much information from reduced datasets. However, for medium and large datasets, the overhead stabilizes between 250% and 500%.

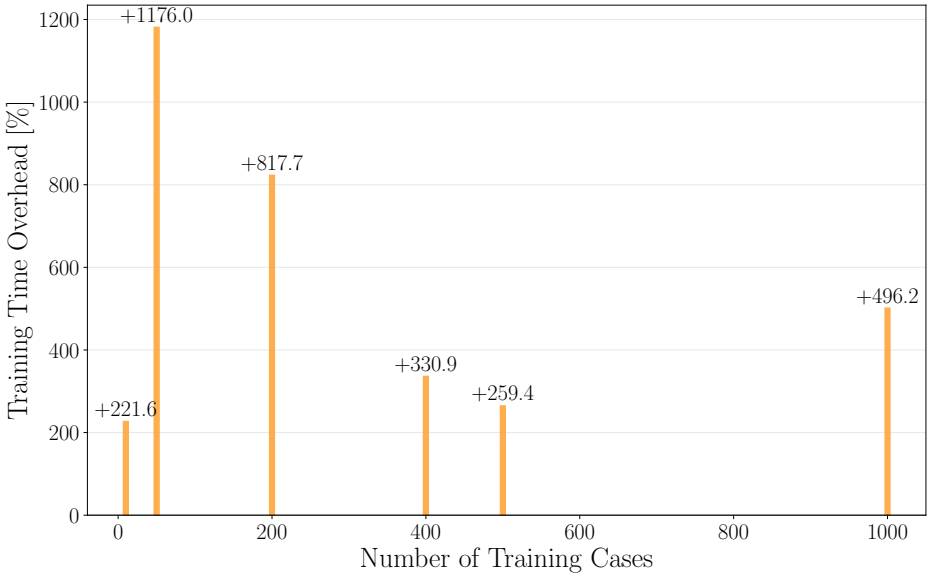


Figure 9.35: Relative overhead in training time of the best PINN compared to the baseline.

It is important to note that this overhead occurs only during training. At inference time, all models—both baseline and physics-informed—share the same architecture and number of parameters,

resulting in virtually identical execution times per sample. The marginal differences observed are due to runtime variability rather than model complexity.

Complete training time metrics for all models, including number of epochs and total duration, are provided in Appendix D.

9.2.1.3 Epoch-level efficiency

The number of epochs also plays a key role. Figure 9.36 shows that PINNs require longer epochs as training size increases, suggesting more complex gradient updates due to physics terms. Yet, the per-epoch time remains reasonable even for large datasets, meaning that, apart from the increase in per epoch time, the increase in training time is a direct consequence of learning more complex data relations.

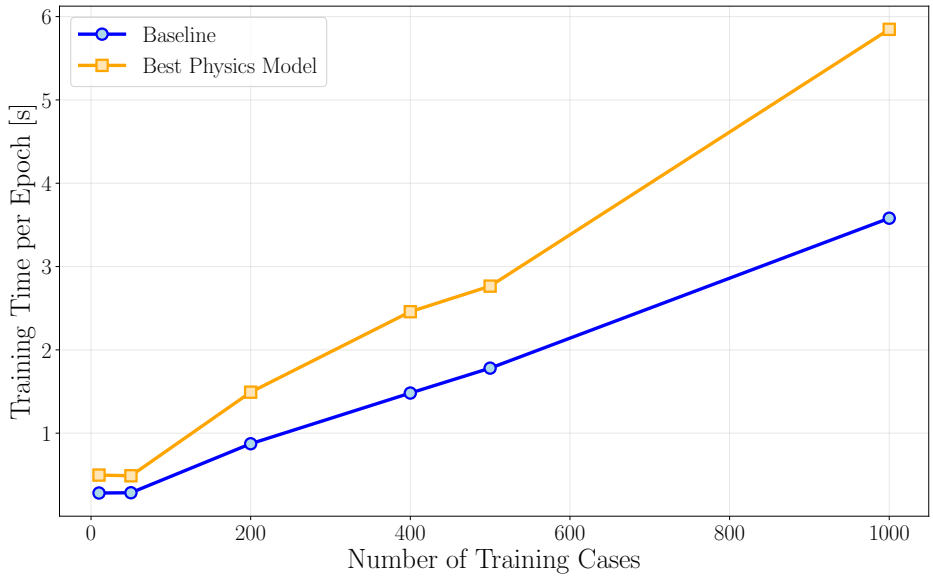


Figure 9.36: Mean training time per epoch for baseline and best physics-informed models.

9.2.1.4 Performance vs efficiency trade-off

Finally, Figure 9.37 compares the performance (MAE) of each best model against its training time.

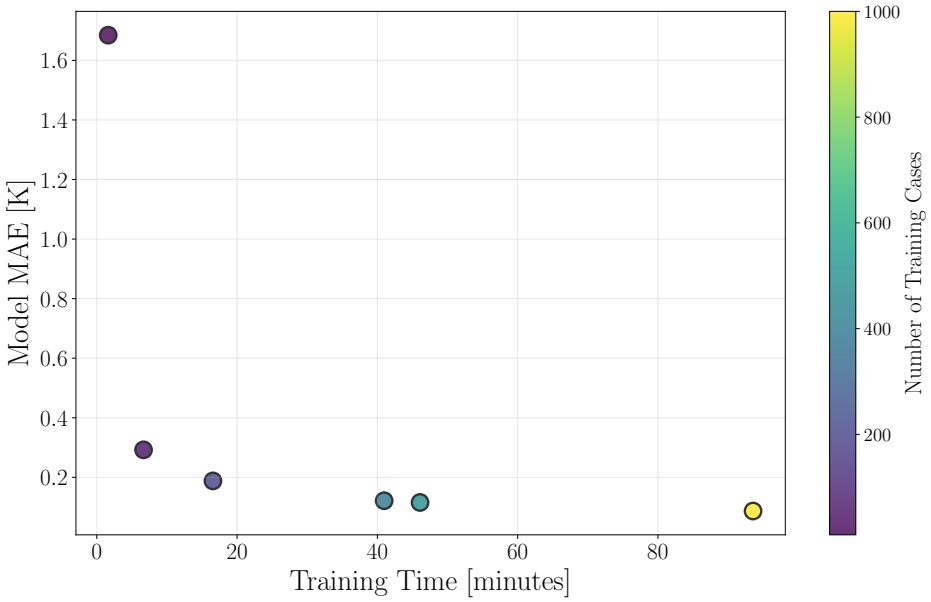


Figure 9.37: Trade-off between training time and final MAE for the best PINN at each training size.

At $n_{\text{train}} = 50$, the best physics-informed model achieves an overall MAE of 0.2923 K, compared to 8.1499 K for the MSE-only baseline. This corresponds to an improvement of approximately 27× in prediction accuracy. However, this gain comes at the cost of a training time increase from 0.52 to 6.67 minutes—roughly 13× longer.

This trade-off illustrates that physics-based regularization can yield substantial accuracy gains in low-data regimes, while maintaining training times within reasonable bounds.

9.2.1.5 Execution and training time

To assess the practical feasibility of the trained models, we now compare their execution time during inference with that of the numerical solver. This helps determine whether deep learning models could eventually replace traditional simulation pipelines.

Figure 9.38 shows the inference time (per case) of each model versus its training time. Although training duration varies significantly, ranging from less than a minute to over 90 minutes, the execution time remains relatively stable across all configurations. This is expected, as all models share the same number of parameters and perform the same number of operations during inference. Thus, inference time depends mostly on the model architecture and not on the amount of training data.

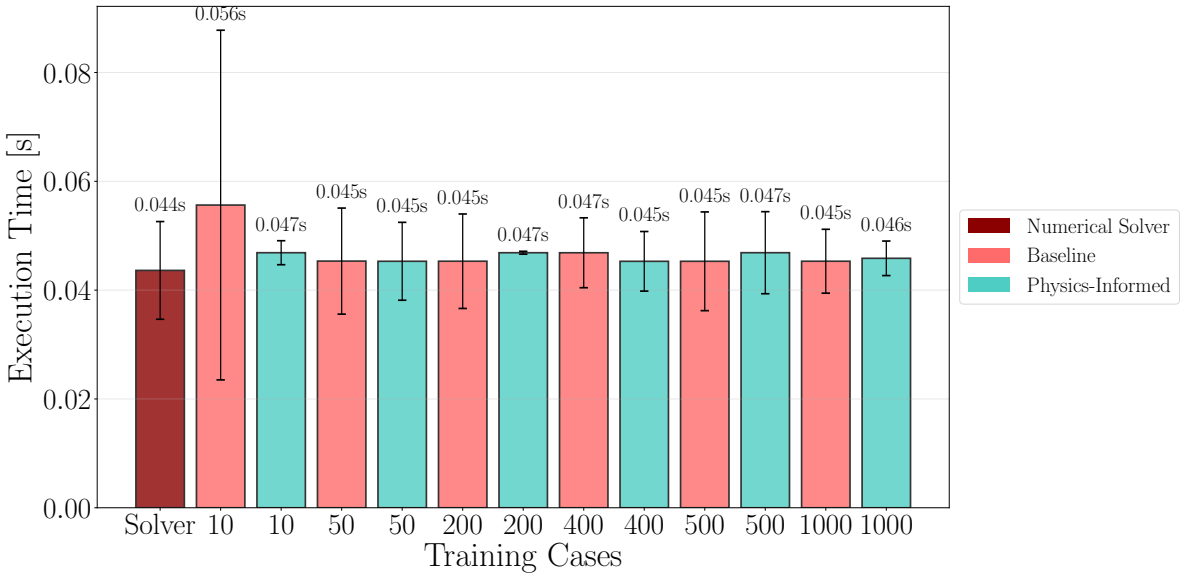


Figure 9.38: Execution time per test case for each model.

Figure 9.39 shows the speedup of each model relative to the numerical solver, with values greater than 1 indicating faster inference. Most models are slightly slower or comparable to the solver, which is expected given the simplicity and efficiency of the reference numerical method. In particular, the solver accounts for radiative heat transfer using a simplified formulation that avoids calculating radiative exchange factors (view factors) and similar terms —typically the most computationally intensive part of spacecraft thermal analysis [79]— making it relatively inexpensive to evaluate. Therefore, this work should be regarded as a preliminary exploration, with the expectation that in more complex simulations, where full radiative calculations are required, physics-informed neural networks may offer a clearer advantage in both speed and scalability.

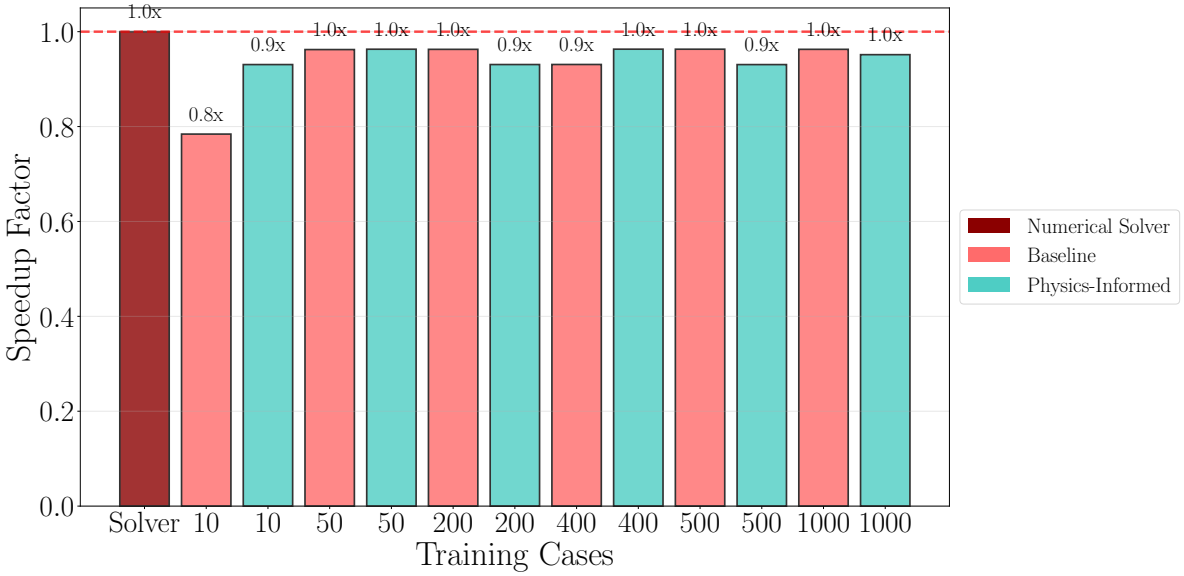


Figure 9.39: Speedup of each model compared to the solver.

Figure 9.40 shows the training duration of all models as a function of the number of training cases. The physics-informed models consistently require longer training times than the baseline models, primarily due to the additional computation of the physics-based loss terms, with the exception observed at $n_{\text{train}} = 1000$, where both models have similar durations. As expected, the training time generally increases with the size of the dataset, reflecting the added computational burden introduced by the physics-informed regularization.

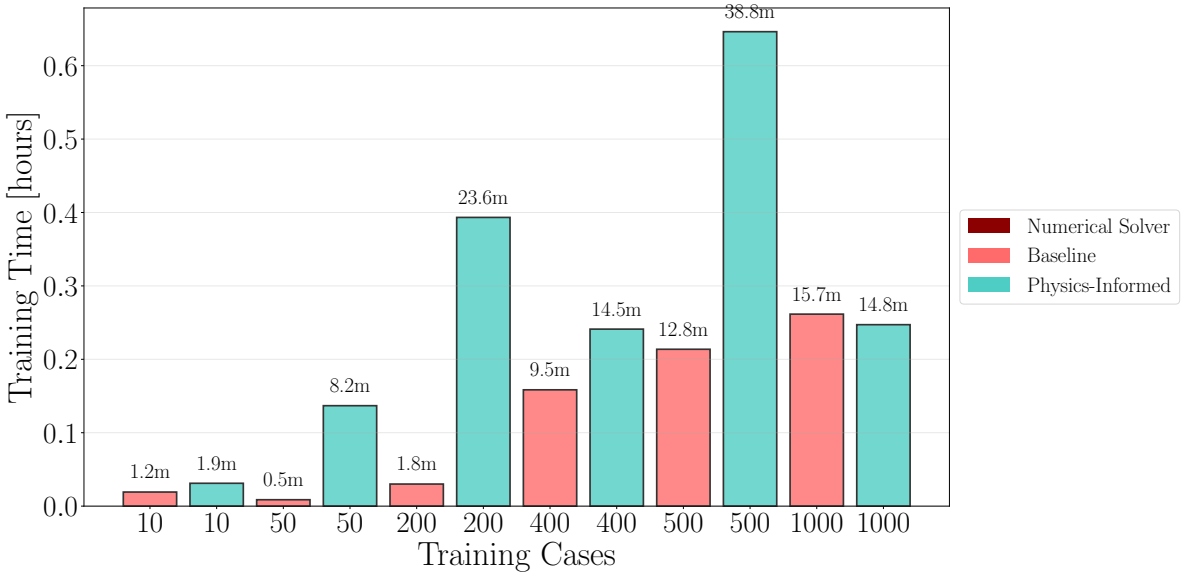


Figure 9.40: Training duration for all models.

9.3 Impact of variable initial temperature

To at least assess whether this assumption significantly affects the conclusions, we performed a comparative study of the baseline models trained with and without variable initial temperatures. Specifically, the models trained with variable T_{init} used a random constant temperature. The values of T_{init} were sampled for each node from a uniform distribution identical to that of the interface and environmental temperatures, in this case, $\mathrm{U}(270\,\mathrm{K}, 320\,\mathrm{K})$. The goal was to verify whether the error levels remain comparable, assuming the impact of other hyperparameters would remain qualitatively similar.

Table 9.12 summarizes the results across different training set sizes, comparing the MAE, standard deviation, and maximum error for constant and variable T_{init}. These results are shown for baseline (non-physics) models.

Table 9.12: Comparison of baseline models with constant and variable initial temperature at different training sizes.

n_{train}	Constant T_{init}			Variable T_{init}		
	MAE [K]	STD [K]	Max Error [K]	MAE [K]	STD [K]	Max Error [K]
10	10.217	8.075	65.632	9.421	7.419	65.974
50	9.368	7.335	54.569	5.658	4.333	60.191
200	5.727	4.702	58.900	7.226	5.941	58.027
400	5.506	3.670	49.846	3.803	3.392	43.036
500	3.365	2.121	40.831	2.563	2.158	37.552
1000	3.305	2.253	30.802	0.229	0.214	9.853

The results suggest that the inclusion of variable T_{init} does not degrade the model's predictive ability; in fact, in some regimes it improves the MAE, particularly at large n_{train}. The most notable improvement is observed at $n_{\text{train}} = 1000$, where the MAE drops from 3.305 K (constant) to 0.229 K (variable). For intermediate training sizes, the differences are more subtle, and in some cases, the constant T_{init} even performs slightly better, likely due to the simpler learning task in the homogeneous case.

This behavior suggests that introducing variability in the initial temperature distribution may help the model learn a more general representation of the underlying physics, improving its ability to generalize to unseen conditions. At low data regimes, the increased complexity may slightly harm performance, but as n_{train} grows, the benefits of learning from more diverse scenarios become evident, leading to superior predictive accuracy. Therefore, in practical applications where sufficient training data is available, it is advisable to include variability in T_{init} to enhance robustness and generalization.

Figure 9.41 complements these findings by visualizing the MAE versus training set size for both scenarios. The trend confirms that variable initial conditions are learnable and can be accommodated by the baseline architecture.

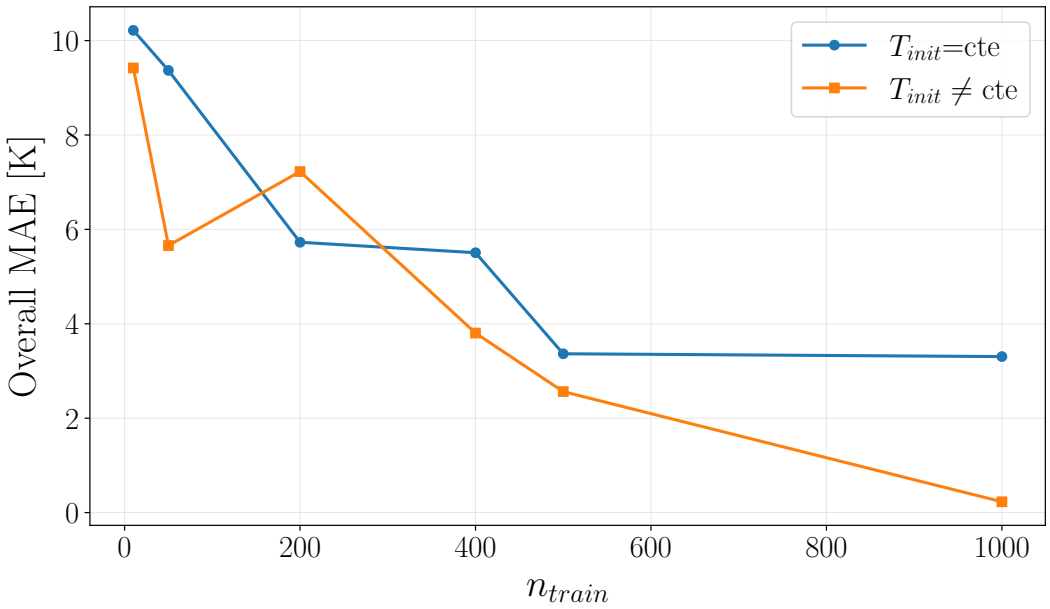


Figure 9.41: Comparison of overall MAE versus n_{train} for baseline models trained with constant and variable initial temperatures.

These findings provide confidence that the conclusions drawn in earlier sections remain valid even when generalizing to more realistic, non-uniform initial conditions.

10

Conclusions and future work

10.1 Summary of the work

This thesis has demonstrated that a PINN can accurately predict the transient thermal response of a 13×13 PCB, achieving a MAE of just 0.2923 K in the best configuration ($n_{\text{train}} = 50$). By incorporating physics-based constraints into the loss function, the model improves its generalization ability dramatically when data is scarce. For instance, with only 50 training samples, the best PINN reduced the baseline error from 8.15 K to 0.29 K, a reduction of approximately 96.4%. Even when increasing the training set to 1000 samples, the PINN maintained a 67.1% error reduction compared to a purely data-driven baseline.

In addition to accuracy improvements, the study analyzed the relationship between training set size, sequence length, and training cost. Notably, the improvement brought by the physics term diminishes as the amount of data increases, as indicated by a strong negative correlation (-0.849) between the number of samples and improvement percentage. Furthermore, the experiments showed that moderate sequence lengths (achieved with a time step of 10 s) offered the best balance between prediction accuracy and computational efficiency. In particular, models trained with $\Delta t = 10$ s outperformed both finer and coarser resolutions in most cases, minimizing errors while keeping inference sequences manageable.

The training cost of physics-informed models was also quantified. Compared to the baseline, the best PINN at $n = 50$ required about 6.67 minutes to train, versus 0.52 minutes for the baseline—a $13\times$ overhead. This overhead stems from the additional computation and backpropagation of the physical constraints during training, and it remains significant even as data size grows. At inference time, however, the models—both baseline and physics-informed—show comparable execution times,

as the overhead is exclusive to the training phase.

Additional experiments were conducted to assess the impact of using variable initial temperature distributions, as all previous experiments assumed a constant $T_{\text{init}} = 298$ K. These tests showed that incorporating variability in T_{init} does not degrade the model's predictive ability; in fact, in high-data regimes ($n_{\text{train}} = 1000$) the MAE improved from 3.305 K (constant) to 0.229 K (variable). This suggests that introducing diversity in the initial conditions helps the model learn a more generalizable representation of the physics, particularly when sufficient data is available.

10.2 Future work and directions

While the presented framework achieves substantial improvements over purely data-driven approaches —reducing the baseline MAE from over 8 K to as low as 0.29 K with only 50 training cases— several avenues for extending and strengthening it have been identified. Many of these directions address limitations observed during this work, aiming to enhance generality, flexibility, and applicability in real-world aerospace scenarios.

The following future directions are proposed to address these limitations and advance the methodology:

- **Non-uniform initial conditions:** Additional experiments indicated that incorporating variable T_{init} improves generalization and accuracy in high-data regimes. Future models should explicitly include non-uniform initial conditions during training to enhance realism and robustness.
- **Variable heater placement:** Replace the static heater masks with dynamic, user-defined inputs. This would let the model adapt to different board layouts without retraining —although it may require a larger dataset to generalize effectively.
- **Time-step agnostic architectures:** Incorporate Δt as an input feature, or explore continuous-time models (e.g., *neural ODEs* [80], *Fourier Neural Operators* [81], or *diffusion-based models* [82]) to support arbitrary temporal resolutions without retraining.
- **Improved data efficiency and reusability:** Use *transfer learning* to fine-tune pretrained networks for new geometries or boundary conditions, reducing the amount of additional data required. Alternatively, train the model using *thermographic measurements* or infrared camera footage to avoid relying solely on synthetic simulations.
- **Full-system modeling:** Extend the approach to entire spacecraft by combining *3D CNNs* and *ConvLSTM* layers to capture volumetric spatial–temporal dependencies, as demonstrated in medical imaging [83].

- **Graph Neural Networks (GNN):** Explore GNNs for representing irregular spacecraft geometries more flexibly and physically consistently. This is currently being investigated in the Bachelor's Thesis of Ernesto Enfedaque Medina [84].
- **Onboard deployment:** Prune and quantize the models to meet hardware constraints for integration into onboard systems, enabling real-time thermal monitoring and adaptive control during missions.

These directions aim to overcome the current limitations —fixed geometry, uniform initial conditions, rigid heater placement, and lack of transferability— while further improving the model's accuracy, efficiency, and deployability. Together, they point toward a general, robust, and scalable framework for transient thermal modeling of aerospace systems.

References

[1] R. M. d. M. dos Santos, W. R. Telles, A. J. da Silva Neto, et al., Hydrological modeling using artificial neural networks for flood event forecasting. case study: Pomba river in santo antônio de pádua-rj, Ciência e Natura 46 (esp. 1) (2024) e87221–e87221.

[2] B. Said, L. Mazouz, T. T. NAAS, Ö. Yildirim, R. D. Mohammedi, Broken magnets fault detection in pmsm using a convolutional neural network and svm, ITEGAM-JETIA 10 (48) (2024) 55–62.

[3] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation (2015). arXiv:1505.04597.

[4] D. Yamashita, H. Tanaka, T. Ikami, H. Nagai, Transient Surrogate Model using Recurrent Neural Networks for Spacecraft Thermal Analysis, in: 53rd International Conference on Environmental Systems, St. Paul, Minnesota, USA, 2024, iCES-2024-252.

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

[6] PCB Power Market, Importance of pcb thickness, accessed: 2025-06-10 (2023).
URL https://www.pcbpower.us/blog/importance-of-pcb-thickness

[7] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-c. Woo, Convolutional lstm network: A machine learning approach for precipitation nowcasting, Advances in neural information processing systems 28 (2015).

[8] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[9] G. Zhang, Neural networks for classification: a survey, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 30 (4) (2000) 451–462. doi:10.1109/5326.897072.

[10] H. Tanaka, H. Nagai, Data-driven thermal state estimation for in-orbit systems via physics-informed machine learning, Acta Astronautica 212 (2023) 316–328. doi:https://doi.org/10.1016/j.actaastro.2023.07.039.

[11] Diederik P. Kingma and Jimmy Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980.

[12] M. Claesen, B. D. Moor, Hyperparameter Search in Machine Learning (2015). arXiv:1502.02127.

[13] X. Ying, An Overview of Overfitting and its Solutions, Journal of Physics: Conference Series 1168 (2) (2019) 022022. doi:10.1088/1742-6596/1168/2/022022.

[14] 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 (56) (2014) 1929–1958.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors (2012). arXiv:1207.0580.

[16] K. You, M. Long, J. Wang, M. I. Jordan, How Does Learning Rate Decay Help Modern Neural Networks? (2019). arXiv:1908.01878.

[17] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, How Does Batch Normalization Help Optimization?, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 31, Curran Associates, Inc., 2018.

[18] C. C. S. Balne, S. Bhaduri, T. Roy, V. Jain, A. Chadha, Parameter Efficient Fine Tuning: A Comprehensive Analysis Across Applications (2024). arXiv:2404.13506.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation 1 (4) (1989) 541–551. doi:10.1162/neco.1989.1.4.541.

[20] F. Bieder, R. Sandkühler, P. C. Cattin, Comparison of Methods Generalizing Max- and Average-Pooling (2021). arXiv:2103.01746.

[21] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012).

[22] 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).

[23] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, nature 323 (6088) (1986) 533–536.

[24] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[25] F. A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with LSTM, Neural computation 12 (10) (2000) 2451–2471.

[26] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561 (2017).

[27] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561 (2017).

[28] R. Laubscher, Simulation of multi-species flow and heat transfer using physics-informed neural networks, Physics of Fluids 33 (8) (2021).

[29] Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020) 112789.

[30] S. Cai, Z. Wang, F. Fuest, Y. J. Jeon, C. Gray, G. E. Karniadakis, Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks, Journal of Fluid Mechanics 915 (2021) A102.

[31] G. S. Misyris, A. Venzke, S. Chatzivasileiadis, Physics-informed neural networks for power systems, in: 2020 IEEE power & energy society general meeting (PESGM), IEEE, 2020, pp. 1–5.

[32] H. Tanaka, H. Nagai, Thermal surrogate model for spacecraft systems using physics-informed machine learning with POD data reduction, International Journal of Heat and Mass Transfer 213 (2023) 124336. doi:https://doi.org/10.1016/j.ijheatmasstransfer.2023.124336.

[33] European Cooperation for Space Standardization (ECSS), Space Engineering. Thermal Control General Requirements (ECSS-E-ST-31C), Tech. rep., ESA Requirements and Standards Division, ESTEC, Noordwijk, The Netherlands, publicación de noviembre de 2008 (Nov. 2008).

[34] J. P. Holman, Transferencia de calor, 8th Edition, McGraw-Hill, Madrid [etc.], 1998.

[35] D. Fixsen, The temperature of the cosmic microwave background, The Astrophysical Journal 707 (2) (2009) 916.

[36] European Cooperation for Space Standardization (ECSS), Space Engineering. Space Environment (ECSS-E-ST-10-04C), Tech. rep., ESA Requirements and Standards Division, ESTEC, Noordwijk, The Netherlands, publicación de noviembre de 2008 (Nov. 2008).

[37] International Organization for Standardization, ISO 21348 (2007) Space environment (natural and artificial). Process for determining solar irradiances, Standard, International Organization for Standardization, publicación de mayo de 2007 (May 2007).

[38] B. Anderson, C. Justus, G. Batts, Guidelines for the selection of near-earth thermal environment parameters for spacecraft design, Tech. rep. (2001).

[39] D. R. E. Ewim, M. O. Okwu, E. J. Onyiriuka, A. S. Abiodun, S. M. Abolarin, A. Kaood, A quick review of the applications of artificial neural networks (ANN) in the modelling of thermal systems (2021).

[40] B. Souayeh, S. Bhattacharyya, N. Hdhiri, M. Waqas Alam, Heat and fluid flow analysis and ann-based prediction of a novel spring corrugated tape, Sustainability 13 (6) (2021) 3023.

[41] A. K. Yadav, S. S. Chandel, Solar radiation prediction using Artificial Neural Network techniques: A review, Renewable and sustainable energy reviews 33 (2014) 772–781.

[42] S. A. Kalogirou, Artificial intelligence for the modeling and control of combustion processes: a review, Progress in energy and combustion science 29 (6) (2003) 515–566.

[43] S. Y. Heng, Y. Asako, T. Suwa, K. Nagasaka, Transient thermal prediction methodology for parabolic trough solar collector tube using artificial neural network, Renewable energy 131 (2019) 168–179.

[44] K. Sikka, R. Lall, T. Sinha, Artificial neural networks for package thermal analysis, in: 2021 20th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (iTherm), IEEE, 2021, pp. 103–113.

[45] J.-A. Martinez-Heras, A. Donati, Artificial neural networks in support of spacecraft thermal behaviour modelling, in: 2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720), Vol. 2, IEEE, 2004, pp. 1269–1274 Vol.2. doi:10.1109/AERO.2004.1367724.

[46] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, et al., PyTorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), ACM, 2024, pyTorch Team. doi:10.1145/3620665.3640366.

[47] J. D. Reis Junior, A. M. Ambrosio, F. L. de Sousa, Towards Spacecraft Real-Time Thermal Simulation with Artificial Neural Networks, in: 23rd ABCM International Congress of Mechanical Engineering, Rio de Janeiro, Brazil, 2015, cOB-2015-0316. doi:10.20906/CPS/COB-2015-0316.

[48] J. D. Reis Junior, A. M. Ambrosio, F. L. de Sousa, Real-Time CubeSat Thermal Simulation using Artificial Neural Networks, in: CCIS 2016 - Conference on Computational Intelligence and Systems, 2016.

[49] J. D. Reis Junior, A. M. Ambrosio, F. L. de Sousa, D. F. Silva, Spacecraft real-time thermal simulation using artificial neural networks, Journal of the Brazilian Society of Mechanical Sciences and Engineering 43 (2021) 198. doi:10.1007/s40430-021-02908-7.

[50] A. Chatterjee, An Introduction to the Proper Orthogonal Decomposition, Current Science 78 (7) (2000) 808–817.

[51] A. Cozad, N. V. Sahinidis, D. C. Miller, Learning surrogate models for simulation-based optimization, AIChE Journal 60 (6) (2014) 2211–2227.

[52] A. C. de Pina, A. A. de Pina, C. H. Albrecht, B. S. L. P. de Lima, B. P. Jacob, ANN-based surrogate models for the analysis of mooring lines and risers, Applied Ocean Research 41 (2013) 76–86.

[53] G. Sun, S. Wang, A review of the artificial neural network surrogate modeling in aerodynamic design, Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 233 (16) (2019) 5863–5872.

[54] A. Dadras Eslamlou, S. Huang, Artificial-neural-network-based surrogate models for structural health monitoring of civil structures: A literature review, Buildings 12 (12) (2022) 2067.

[55] T. Braconnier, M. Ferrier, J.-C. Jouhaud, M. Montagnac, P. Sagaut, Towards an adaptive POD/SVD surrogate model for aeronautic design, Computers & Fluids 40 (1) (2011) 195–209.

[56] E. Iuliano, D. Quagliarella, Aerodynamic shape optimization via non-intrusive POD-based surrogate modelling, in: 2013 IEEE congress on evolutionary computation, IEEE, 2013, pp. 1467–1474.

[57] B. M. de Gooijer, J. Havinga, H. J. Geijselaers, A. H. van den Boogaard, Evaluation of POD based surrogate models of fields resulting from nonlinear FEM simulations, Advanced Modeling and Simulation in Engineering Sciences 8 (1) (2021) 25.

[58] A. Dey, N. Shafiei, R. Khandekhar, W. Eberle, R. Li, Lumped parameter thermal network modelling of power transformers, in: 2021 20th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (iTherm), IEEE, 2021, pp. 172–178.

[59] A. Kačenka, A.-C. Pop, I. Vintiloiu, D. Fodorean, Lumped parameter thermal modeling of permanent magnet synchronous motor, in: 2019 Electric Vehicles International Conference (EV), IEEE, 2019, pp. 1–6.

[60] H. S. Abdelkhalek, H. Medhat, I. Ziedan, M. Amal, Simulation and Prediction for a Satellite Temperature Sensors Based on Artificial Neural Network, Journal of Aerospace Technology and Management 11 (2019) e3719. doi:10.5028/jatm.v11.1055.

[61] M. A. C. Perpinan, The Modelling of the thermal subsystem in spacecraft real time simulators, in: Proceedings of the 3rd Workshop on Simulators for European Space Programmes, ESA/ESTEC, 1994, pp. 69–78.

[62] D. González Barcena, ia_thermal:, `https://github.com/dgbarcena/ia_thermal`, accessed: 2025-06-16 (2025).

[63] V. N. Ioannidis, A. G. Marques, G. B. Giannakis, A recurrent graph neural network for multi-relational data, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 8157–8161.

[64] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, B. Catanzaro, Image inpainting for irregular holes using partial convolutions, in: Proceedings of the European conference on computer vision (ECCV), 2018, pp. 85–100.

[65] E. Bisong, Google Colaboratory, Apress, Berkeley, CA, 2019, pp. 59–64. `doi:10.1007/978-1-4842-4470-8_7`.

[66] F. Schmidt, Generalization in generation: A closer look at exposure bias, arXiv preprint arXiv:1910.00292 (2019).

[67] S. Bengio, O. Vinyals, N. Jaitly, N. Shazeer, Scheduled sampling for sequence prediction with recurrent neural networks, Advances in neural information processing systems 28 (2015).

[68] 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.

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

[70] T. Domhan, J. T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves., in: IJCAI, Vol. 15, 2015, pp. 3460–8.

[71] S. Arora, S. Kumar, P. Kumar, An efficient attention-based lstm framework for blood glucose level prediction., Traitement du Signal 42 (2) (2025).

[72] S. Gao, Y. Huang, S. Zhang, J. Han, G. Wang, M. Zhang, Q. Lin, Short-term runoff prediction with gru and lstm networks without requiring time step optimization during sample generation, Journal of Hydrology 589 (2020) 125188.

[73] Y. Zhao, W. Zhang, X. Liu, Grid search with a weighted error function: Hyper-parameter optimization for financial time series forecasting, Applied Soft Computing 154 (2024) 111362.

[74] F. Heldmann, S. Berkhahn, M. Ehrhardt, K. Klamroth, Pinn training using biobjective optimization: The trade-off between data loss and residual loss, Journal of Computational Physics 488 (2023) 112211.

[75] H. Tarbiyati, B. Nemati Saray, Weight initialization algorithm for physics-informed neural networks using finite differences, Engineering with Computers 40 (3) (2024) 1603–1619.

[76] P. Sharma, L. Evans, M. Tindall, P. Nithiarasu, Hyperparameter selection for physics-informed neural networks (pinns)–application to discontinuous heat conduction problems, Numerical Heat Transfer, Part B: Fundamentals 85 (10) (2024) 1304–1318.

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

[78] P. Sedgwick, Pearson's correlation coefficient, Bmj 345 (2012).

[79] Spacecraft thermal control handbook, 2nd Edition, The Aerospace Press ; American Institute of Aeronautics and Astronautics, El Segundo (California) : Reston (Virginia), 2002.

[80] R. T. Chen, Y. Rubanova, J. Bettencourt, D. K. Duvenaud, Neural ordinary differential equations, Advances in neural information processing systems 31 (2018).

[81] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).

[82] K. Wang, D. Tang, B. Zeng, Y. Yin, Z. Xu, Y. Zhou, Z. Zang, T. Darrell, Z. Liu, Y. You, Neural network diffusion, arXiv preprint arXiv:2402.13144 (2024).

[83] A. Mustafa, R. Rastegar, G. AlRegib, Recognet: Recurrent context-guided network for 3d mri prostate segmentation, arXiv preprint arXiv:2506.19687 (2025).

[84] E. E. Medina, Use of graph neural networks for the development of surrogate thermal models of space systems, bachelor's Thesis, supervised by David González Bárcena (2025).

[85] K. Wang, H. Wu, G. Zhang, J. Fang, Y. Liang, Y. Wu, R. Zimmermann, Y. Wang, Modeling spatio-temporal dynamical systems with neural discrete learning and levels-of-experts, IEEE Transactions on Knowledge and Data Engineering 36 (8) (2024) 4050–4062.

[86] R. Wu, Y. Liang, L. Lin, Z. Zhang, Spatiotemporal multivariate weather prediction network based on cnn-transformer, Sensors (Basel, Switzerland) 24 (23) (2024) 7837.

[87] L. Huang, F. Mao, K. Zhang, Z. Li, Spatial-temporal convolutional transformer network for multivariate time series forecasting, Sensors 22 (3) (2022) 841.

[88] D. Song, X. Su, W. Li, Z. Sun, T. Ren, W. Liu, A.-A. Liu, Spatial-temporal transformer network for multi-year enso prediction, Frontiers in Marine Science 10 (2023) 1143499.

A

Node indexing scheme

The thermal solver used in this work operates on a square 13×13 mesh representing the PCB surface. Each node in the mesh is assigned a unique identifier following a row-wise flattened indexing scheme. The indexing starts at the bottom-left corner (node 0) and increases from left to right, row by row, moving upwards.

The figure below shows the full node indexing used in the solver:

156	157	158	159	160	161	162	163	164	165	166	167	168
143	144	145	146	147	148	149	150	151	152	153	154	155
130	131	132	133	134	135	136	137	138	139	140	141	142
117	118	119	120	121	122	123	124	125	126	127	128	129
104	105	106	107	108	109	110	111	112	113	114	115	116
91	92	93	94	95	96	97	98	99	100	101	102	103
78	79	80	81	82	83	84	85	86	87	88	89	90
65	66	67	68	69	70	71	72	73	74	75	76	77
52	53	54	55	56	57	58	59	60	61	62	63	64
39	40	41	42	43	44	45	46	47	48	49	50	51
26	27	28	29	30	31	32	33	34	35	36	37	38
13	14	15	16	17	18	19	20	21	22	23	24	25
0	1	2	3	4	5	6	7	8	9	10	11	12

This indexing is used throughout the solver to define the position of heaters, interface nodes, and fixed boundary conditions.

B

Alternative neural network architectures

This appendix summarizes several alternative neural network architectures that were explored during the early stages of this project. These models include a fully connected MLP, a spatio-temporal regressor with explicit time conditioning, and a convolutional encoder combined with a Transformer-based decoder.

While each of these architectures offered distinct advantages in terms of structure or flexibility, none achieved the combination of accuracy, stability, and autoregressive capability required for long-term thermal prediction. Notably, all of them assume a uniform initial state and cannot be recursively fed with their own outputs, making them unsuitable for real-time inference or deployment in onboard applications.

For completeness, the following sections document the configuration, training results, and limitations of these preliminary models. Their inclusion provides valuable insight into the iterative design process that led to the selection of the ConvLSTM as the final architecture.

The hyperparameters presented in this section have not necessarily been validated as optimal configurations. In many cases, they correspond to early experiments that were later abandoned in favor of more promising architectures. Consequently, the results shown here should be interpreted primarily as preliminary evaluations.

Moreover, all experiments in this stage were limited to a short rollout duration of 50 seconds[1]. This restriction was introduced to ensure manageable training times during the exploratory phase,

[1]Rollout refers to the forward prediction of the system's state over a sequence of future time steps.

allowing faster iteration and comparison across multiple architectural candidates.

The configuration shared across all models discussed in this section is summarized in Table B.1.

Table B.1: Common hyperparameters used across all architectures

Hyperparameter	**Value**
Number of training samples	1000
Number of training epochs	1000
Number of time steps per sample	51
Batch size	16
Learning rate	10^{-2}
Learning rate decay factor	0.1
Learning rate decay patience	10 epochs
Early stopping patience	50 epochs

Each model was trained to predict the thermal behavior of 1000 independent cases, each consisting of a 50-second simulation divided into 1-second intervals. The predicted output is therefore a sequence of 51 thermal maps per case. Model performance was evaluated in terms of the average MAE and its standard deviation over the test set, providing a quantitative measure of both accuracy and robustness.

To ensure reproducibility across runs and fair comparison between models, a fixed random seed was used for data shuffling, initialization, and training procedures.

B.1 Feedforward MLP with explicit physical inputs

B.1.1 Architecture overview

First, an MLP was used. Although it may not be the most sophisticated solution, it is an excellent interpolator when the whole state space of the solution can be calculated, as it is our case. Its effectiveness has been proved both in steady [10, 32] and transient [47–49] spacecraft thermal simulation.

This architecture addresses the thermal prediction task using a fully connected feedforward neural network (MLP), in which the system state is predicted at a given time step from a set of physical scalar inputs. Unlike models based on sequences or spatial encoders, this approach treats the entire input as a flat vector, assuming no spatial or temporal structure beyond what is explicitly provided.

The model receives a 10-dimensional input vector composed of the following elements:

- The ambient temperature (T_{env})
- The temperatures at the four interface nodes ($T_{\text{interface},i}$)

- The power applied at each of the four heaters ($Q_{\text{heater},i}$)
- A normalized time value (t)

These features represent the thermal conditions of the system as well as the time at which the temperature distribution is to be predicted. This formulation enables direct regression of the system's state at any desired time step, bypassing the need for sequential inference.

The network architecture consists of a series of fully connected layers with nonlinear activation functions, progressively transforming the input into a high-dimensional representation from which the final temperature map is decoded. The output of the network is a 169-dimensional vector corresponding to a 13×13 spatial grid representing the temperature at each node.

This design enables simple and efficient prediction of thermal fields under arbitrary boundary conditions and at arbitrary time points, as long as they lie within the distribution seen during training, but it is restricted to the same homogeneous initial conditions. The model acts as a powerful interpolator across the space of physical conditions and time, leveraging the dense sampling of the solution space to estimate outputs with high accuracy despite its structural simplicity.

B.1.2 Hyperparameter configuration

The ANN was implemented as a feedforward MLP with three hidden layers, each containing 256 units. This configuration was selected to balance expressiveness and computational efficiency, while remaining suitable for dense interpolation over the physical parameter space.

The model received a 10-dimensional input vector and produced a 169-dimensional output representing the 13×13 temperature map. All layers used the ReLU activation function, and the model was trained using the Adam optimizer with a learning rate of 10^{-3}.

To improve generalization and prevent overfitting, a weight decay factor of 10^{-4} was applied during optimization. Input and output data were normalized based on the training distribution, ensuring stable convergence throughout training.

B.1.3 Results

The feedforward MLP was evaluated using standard error metrics over the predicted temperature maps, with a particular focus on temporal evolution and consistency.

The overall average error across all predictions was:

- **MAE**: 2.977 K
- **Standard deviation of error**: 2.327 K

The error remains relatively constant across the 50-second rollout, with only slight fluctuations between time steps. For instance, the MAE at the initial time step was 3.213 K, and decreased modestly to 2.545 K at the final time step. The highest average error was observed at $t = 23$ s, where the MAE reached 3.486 K. Additionally, the maximum pointwise error across the entire test set was 24.742 K, indicating that while the model is generally stable, it can occasionally produce large localized deviations.

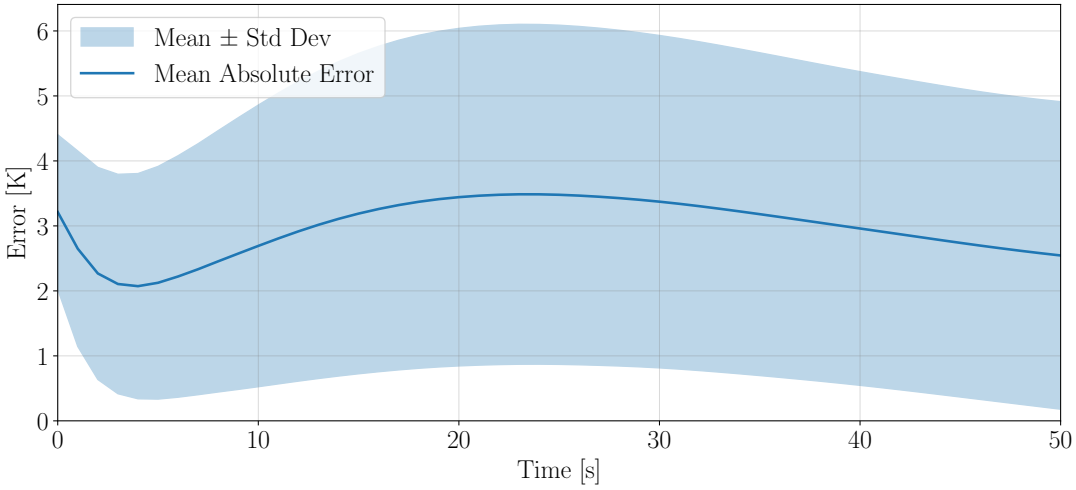


Figure B.1: MAE with standard deviation bands over time for the MLP model.

Figure B.1 shows the evolution of the MAE across the 51 time steps. The shaded region corresponds to one standard deviation above and below the mean. The error remains mostly stable throughout the rollout, with a slight downward trend towards the end. The narrow spread in standard deviation suggests that the model's predictions are consistently close to the mean error, with moderate variability.

This temporal consistency is a direct consequence of the architecture itself. Unlike sequential models, the MLP does not propagate information through time. Each prediction is computed independently based on the current input conditions, including time as just another feature. As a result, the model behaves as a multidimensional interpolator: it estimates the system's state by mapping input conditions directly to outputs, regardless of whether the target time step is early or late in the sequence. This design allows it to avoid the compounding errors typical of autoregressive or recurrent architectures.

Figure B.2 complements the previous figure by displaying the full range between the minimum and maximum prediction errors at each time step. This representation highlights the presence of outlier cases, particularly in the first half of the simulation.

Overall, the MLP demonstrates reasonable accuracy and robustness considering its simplicity. Its low MAE and relatively stable standard deviation confirm its utility as a baseline architecture

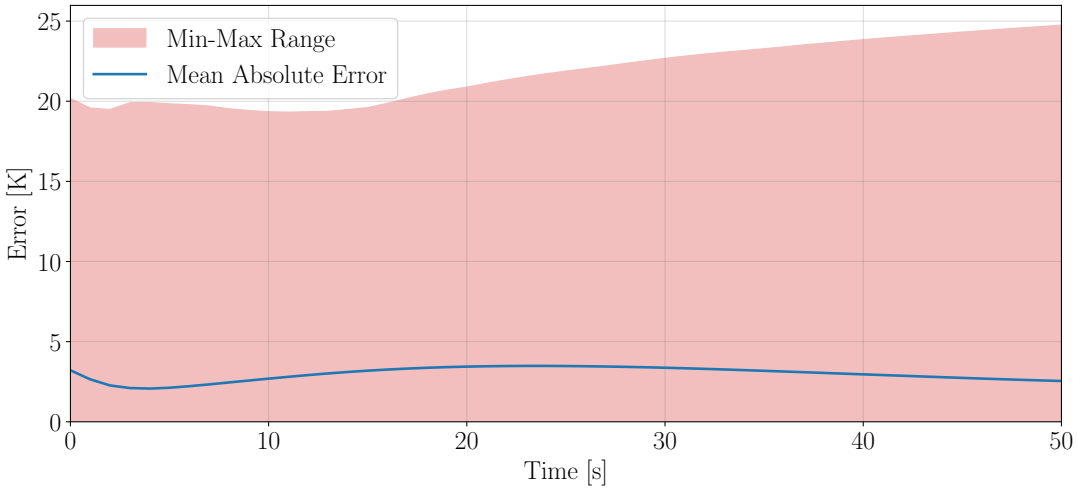


Figure B.2: MAE with min-max error range bands over time for the MLP model.

for fast thermal prediction under well-sampled physical conditions. However, regarding its objective, has several limitations:

- No possibility of changing the initial conditions.
- Not allowing dynamic changes during rollout, hence discarding its deployment to real-time applications.

B.2 Spatio-Temporal Regressor with explicit time conditioning

B.2.1 Architecture overview

This architecture formulates the thermal prediction problem as a regression task conditioned on time, in which each predicted state depends explicitly on both the initial spatial configuration and the target time step. Unlike sequence-based models, which process the input as a time series, this model treats time as an independent variable and allows for non-sequential or continuous inference, following an approach similar to that proposed by Wang et al. [85].

The input consists of two components: the initial thermal state of the system, encoded as a multi-channel spatial map, and a sequence of normalized time values, choosing 0 as the first time step and 1 as the last, dividing the interval into equal parts for each time step. The spatial map includes physical variables such as the temperature at interface nodes, the applied heating power, and the ambient boundary temperature. These inputs are processed through a convolutional encoder that extracts local spatial features and compresses the result into a latent representation.

Each time value is independently embedded through a small feedforward network, producing a temporal feature vector. The temporal and spatial representations are then concatenated, forming a combined latent sequence where each element corresponds to a specific time step and is conditioned on the shared initial state.

This combined sequence is processed by a Transformer-based decoder, which captures interactions across the temporal dimension and refines the temporal predictions. The output is a sequence of predicted thermal maps, one per time step, reconstructed from the latent space through a lightweight projection network.

This modeling approach enables flexible and interpretable simulation of the system's evolution over time. By explicitly conditioning each prediction on time, the network can handle irregular time sampling and enables inference at arbitrary time points within the trained range, without relying on autoregressive dependencies.

B.2.2 Hyperparameter configuration

The ANN was configured with an embedding dimension of 128, which determines the size of the latent representations produced by both the spatial and temporal encoding blocks. This dimension represents a balance between expressiveness and computational efficiency, and it serves as the foundation for all internal transformations in the model.

The temporal decoder, based on a Transformer architecture, was implemented with a single layer, which was empirically sufficient for capturing the relatively smooth thermal evolution of the system under study. Each layer uses multi-head attention with 4 parallel attention heads, allowing the model to learn from different subspaces of the latent representation simultaneously.

The feedforward subnetwork within each Transformer block was configured to have a dimensionality of 640, corresponding to a scaling factor of 5 with respect to the embedding size. This expansion allows the attention mechanism to operate with a higher degree of nonlinearity and representational capacity.

This configuration reflects a deliberate design choice: to keep the architecture lightweight and stable while still benefiting from the flexibility and generalization capacity offered by Transformer-based temporal modeling.

B.2.3 Results

The spatio-temporal regressor was evaluated using standard error metrics over the predicted temperature maps. The analysis focused on the temporal behavior of the error and its variability throughout the rollout.

The overall prediction error was:

- **MAE**: 4.869 K
- **Standard deviation of error**: 7.176 K

Unlike the MLP, this model exhibits a more pronounced dependence on the temporal position within the rollout. The MAE increases gradually as time progresses, starting at 3.717 K at the initial step and reaching 5.576 K at the final step. The maximum average error was observed at the last time step ($t = 50$ s), and the most extreme pointwise error recorded across all test cases reached 68.619 K — significantly higher than in other architectures.

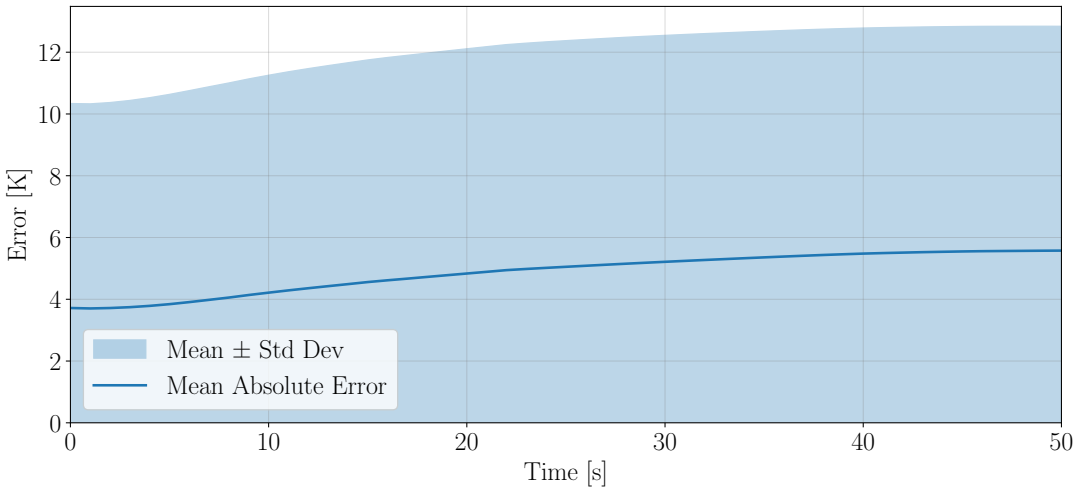


Figure B.3: MAE with standard deviation bands over time for the regressor model.

Figure B.3 illustrates the evolution of the MAE over time. The error clearly increases as the simulation progresses, suggesting that the model accumulates inaccuracies over long rollouts. The standard deviation bands also widen, reflecting increasing uncertainty in the predictions.

This degradation is a natural consequence of the model architecture. Although the regressor does not propagate the system state recursively, it does condition the prediction of each time step on a shared spatial encoding and a time embedding. The ability to generalize well to later stages in the rollout depends on how well the training data represent those time positions. In contrast to the MLP, which interpolates over time directly, the regressor treats time more structurally and is therefore more sensitive to gaps or imbalances in temporal coverage.

Figure B.4 shows the minimum and maximum errors per time step. The widening range confirms the presence of significant outlier cases at later stages, where the model struggles to maintain accuracy.

Overall, the spatio-temporal regressor shows poor accuracy in the early stages of the rollout and suffers from a progressive degradation of performance as time advances. Its behavior highlights the importance of temporal conditioning and the challenges of generalizing over long time horizons

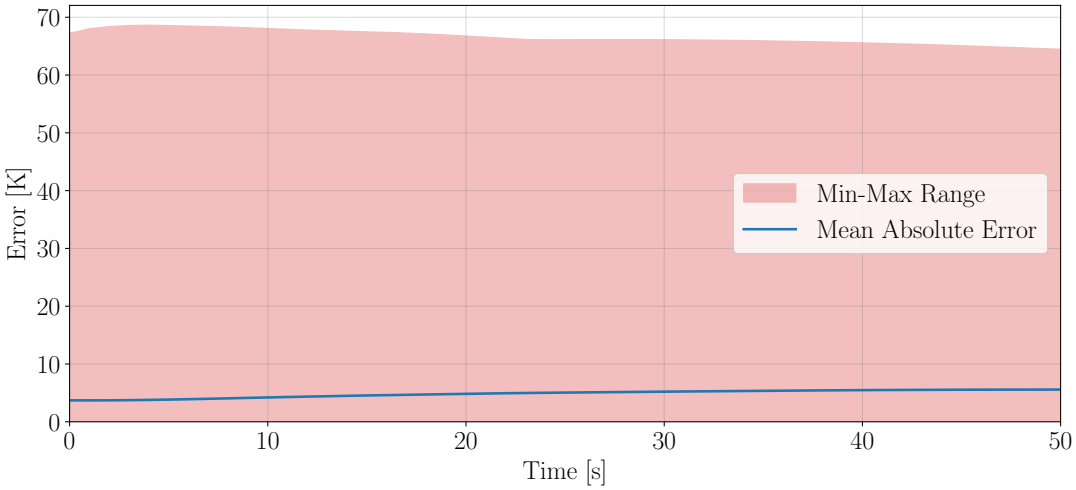


Figure B.4: MAE with min-max error range bands over time for the regressor model.

without explicit memory or recurrence. Furthermore, it suffers from the same limitations as the MLP model commented earlier.

B.3 CNN Encoder and Conditional Spatio-Temporal Decoder

B.3.1 Architecture overview

This architecture combines a spatial encoder based on CNNs with a temporal decoder implemented using a Transformer, inspired by similar work in [86–88]. The main objective is to predict the temporal evolution of the thermal state of the system, given an initial configuration and the relevant boundary and heating conditions.

The spatial encoder processes the input maps at each time step, extracting local features that represent the temperature distribution and its relation to surrounding elements. Each input map contains multiple physical variables encoded as separate channels, including the interface temperatures, the power applied by the heaters, and the ambient temperature. These components are crucial to accurately characterize the thermal behavior of the system over time.

To incorporate the notion of time explicitly, a normalized temporal value can be added as an additional channel. This allows the encoder to differentiate between otherwise similar spatial configurations occurring at different moments in the sequence. Once all time steps are encoded, their latent representations are enriched with positional encodings and passed to a Transformer-based decoder, which models the temporal dependencies across the sequence.

The Transformer captures how the spatial thermal patterns evolve under the influence of the

physical conditions, leveraging its attention mechanism to learn long-range interactions and temporal correlations. Finally, a lightweight MLP transforms the decoder outputs into estimated thermal maps for each time step.

By decoupling spatial and temporal modeling, this architecture enables effective learning of localized thermal dynamics as well as their evolution over time, providing a structured and interpretable framework for transient thermal simulation.

B.3.2 Hyperparameter configuration

The architecture was configured using a latent embedding dimension of 512. This value determines the size of the feature vectors produced by the CNN encoder and processed by the Transformer decoder. A larger embedding dimension allows the network to capture more detailed representations of the spatial thermal state at each time step, though at the cost of increased computational complexity.

The Transformer decoder was composed of 4 stacked layers, each responsible for refining the temporal representation through self-attention and feedforward operations. This depth enables the model to learn both short-term and long-term temporal dependencies in the thermal sequence.

The number of attention heads was set to 8, which allows the model to attend to different aspects of the temporal dynamics in parallel. Multi-head attention improves the model's capacity to represent diverse temporal relationships and helps stabilize training.

Finally, temporal information was explicitly incorporated by enabling the temporal channel option. This setting appends a normalized time variable as an additional input channel to each spatial map, facilitating the model's ability to distinguish between thermally similar configurations occurring at different moments in time. This approach is especially useful in systems where thermal patterns evolve gradually and depend on accumulated effects from previous states.

B.3.3 Results

The CNN encoder combined with a conditional Transformer decoder was evaluated using standard error metrics applied to the predicted temperature maps. Particular attention was given to the evolution of the prediction error across time and its variability.

The overall prediction error was:

- **MAE**: 3.977 K
- **Standard deviation of error**: 5.286 K

The model's performance decreases gradually over time. The MAE at the initial time step was 2.650 K, increasing to 5.003 K at the final time step. The highest average error occurred at $t = 48$ s,

with a value of 5.086 K. The most extreme pointwise error across all predictions was 50.829 K, which, although high, remains lower than that observed in the regressor model.

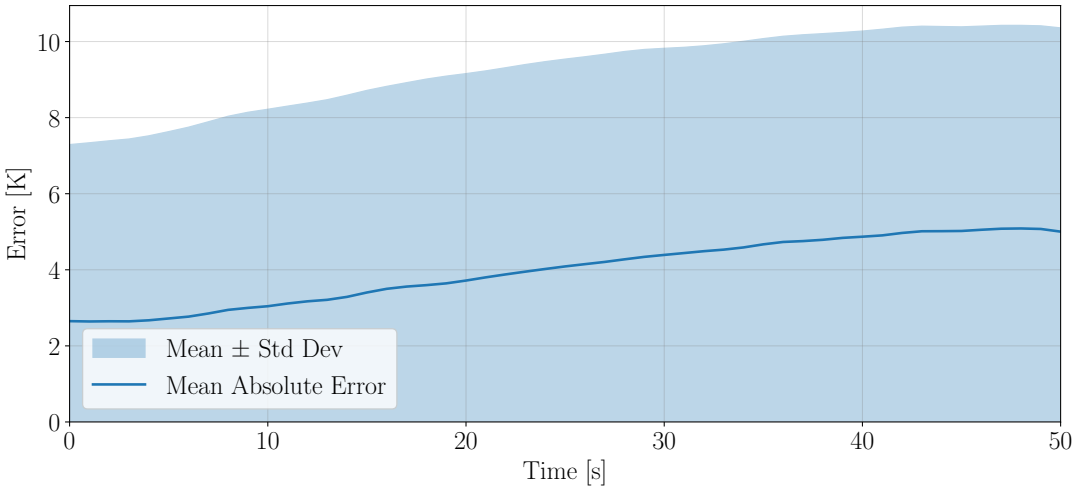


Figure B.5: MAE with standard deviation bands over time for the CNN encoder model.

Figure B.5 shows the evolution of the MAE over time. The error follows a clear upward trend, suggesting that predictions made at later stages of the simulation are more prone to inaccuracies. The standard deviation also increases as the rollout progresses, indicating growing variability in the predicted temperatures.

This behavior is typical of models that process temporal information sequentially or in blocks, such as Transformer-based decoders. While the spatial encoder ensures consistent representation of the initial state, the temporal decoder must infer the dynamics across time steps. In the absence of explicit memory or recurrence, errors can gradually accumulate as the decoder extrapolates to later stages.

Figure B.6 highlights the full range of prediction errors at each time step. As expected, the error range widens significantly after the midpoint of the rollout, reinforcing the notion that this architecture struggles to maintain precision in long-term predictions.

In summary, the CNN encoder with conditional Transformer decoder offers better temporal consistency than the regressor, but remains susceptible to late-stage degradation due to the increasing complexity of the temporal extrapolation task. However, once again, this architecture suffers from the same limitations as the two previously mentioned.

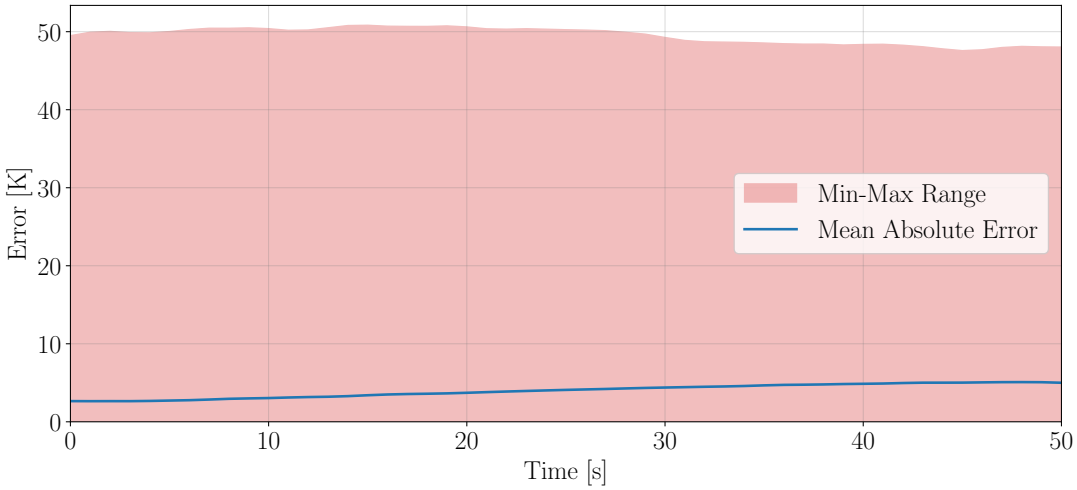


Figure B.6: MAE with min-max error range bands over time for the CNN encoder model.

B.4 Convolutional LSTM

B.4.1 Architecture overview

The Convolutional LSTM is a specialized recurrent neural network architecture designed to model spatio-temporal data. Originally introduced by Shi et al. [7] for precipitation nowcasting, this architecture replaces the fully connected operations in traditional LSTMs with convolutional layers, making it particularly effective for tasks involving sequences of images or spatial maps.

In this work, we implement a variant of the original ConvLSTM architecture based on the publicly available repository by ndrplz[2], adapting it to predict the thermal evolution of a printed circuit board (PCB). The network receives a sequence of 13×13 input maps, each with 6 channels corresponding to physically meaningful variables: interface temperatures, heating power inputs, ambient boundary temperature, temperature at the previous time step, and one binary mask for the location of interfaces and a similar one for the heaters.

At each time step, the input is concatenated with the hidden state from the previous time step and processed through convolutional gates to update the memory and hidden states. This design allows the network to simultaneously capture spatial structures and temporal dependencies, making it a natural fit for modeling heat propagation over time. This allows to overcome limitations presented for the other models and, apart from the better results, is the reason it is chosen.

[2] `https://github.com/ndrplz/ConvLSTM_pytorch`

B.4.2 Hyperparameter configuration

The network was configured with two ConvLSTM layers, each with 64 hidden channels and a kernel size of 3×3. These settings allow the model to extract localized spatio-temporal patterns while maintaining a manageable number of parameters. The kernel size was chosen to balance resolution and computational efficiency.

Scheduled sampling was applied to improve training stability and reduce compounding errors in the autoregressive rollout. The initial probability of using ground truth values was set to 1.0 and gradually decayed to 0.0 over 200 epochs, allowing the model to progressively rely on its own predictions.

The final prediction at each time step was obtained by applying a 1×1 convolutional layer to the hidden state of the last ConvLSTM layer, projecting the result into a single output channel representing temperature.

Optimization was carried out using the Adam optimizer with a learning rate of 10^{-2}, a decay factor of 0.1, and patience of 10 epochs. Early stopping was triggered after 50 epochs without improvement on the validation set.

This configuration reflects a compromise between model expressiveness and training stability, ensuring the ability to model non-linear spatio-temporal interactions while avoiding overfitting during long autoregressive sequences.

B.4.3 Results

The ConvLSTM model achieved the highest accuracy among all architectures tested, with significantly lower prediction errors. The evaluation was based on standard metrics applied to 1000 transient simulations of 50 seconds, sampled at 1-second intervals.

The overall performance metrics were:

- **MAE**: 0.114 K
- **Standard deviation of error**: 0.165 K

The model maintained low error values throughout the simulation. The MAE at the initial time step was 0.103 K, increasing slightly to 0.170 K by the end of the rollout. The highest average error occurred at $t = 50$ s, matching the final prediction step. Although the model exhibited a few large outliers, with a maximum pointwise error of 11.091 K, these cases were infrequent and did not significantly affect the overall performance.

Figure B.7 displays the evolution of the MAE over the 51 time steps. The error curve is remarkably flat, suggesting consistent predictive accuracy. The narrow standard deviation bands further confirm the model's reliability and low variance in its predictions.

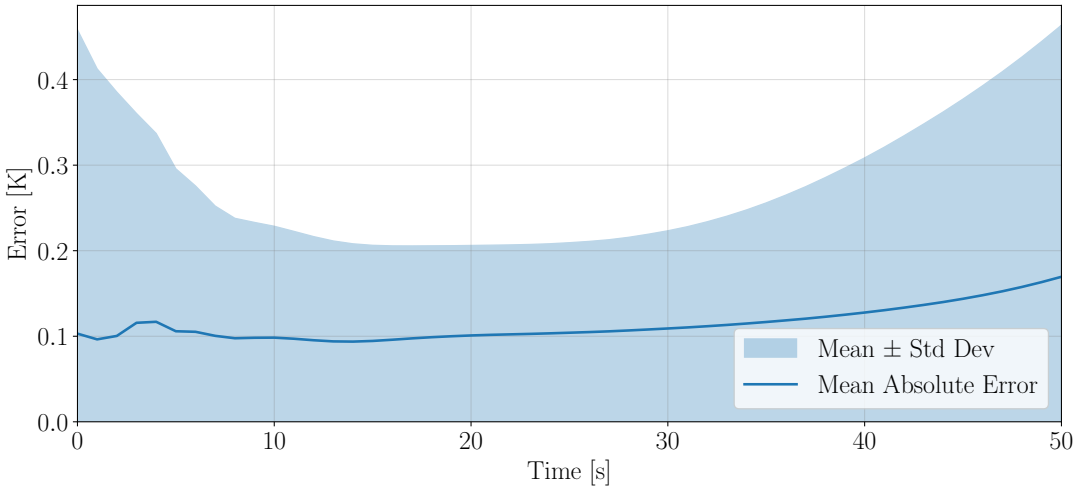


Figure B.7: MAE with standard deviation bands over time for the ConvLSTM model.

Figure B.8 complements the previous analysis by illustrating the complete range of errors. Although some isolated outliers appear, they are not representative of the overall error distribution, which remains tightly bounded throughout the rollout.

Overall, the ConvLSTM offers a powerful and reliable architecture for transient thermal prediction. Its ability to maintain low and stable prediction error, especially over long autoregressive sequences, confirms its robustness and effectiveness in learning spatiotemporal dynamics.

B.5 Comparison of Architectures

This section compares the performance of all tested models in predicting the transient thermal evolution of the system. Figure B.9 shows the temporal progression of the MAE for each architecture, along with standard deviation bands that reflect the variability across test samples.

A summary of the main error metrics is presented in Table B.2, which includes the overall average MAE, the maximum time-averaged MAE across all rollouts, and the maximum pointwise error observed in any prediction.

Table B.2: Quantitative comparison of prediction errors for all tested architectures

Model	**Overall MAE [K]**	**Max MAE @t [s]**	**Max Pointwise Error [K]**
MLP	2.977	3.486 @ 23	24.742
Regressor	4.869	5.576 @ 50	68.619
Encoder + Transformer	3.977	5.086 @ 48	50.829
ConvLSTM	0.114	0.170 @ 50	11.091

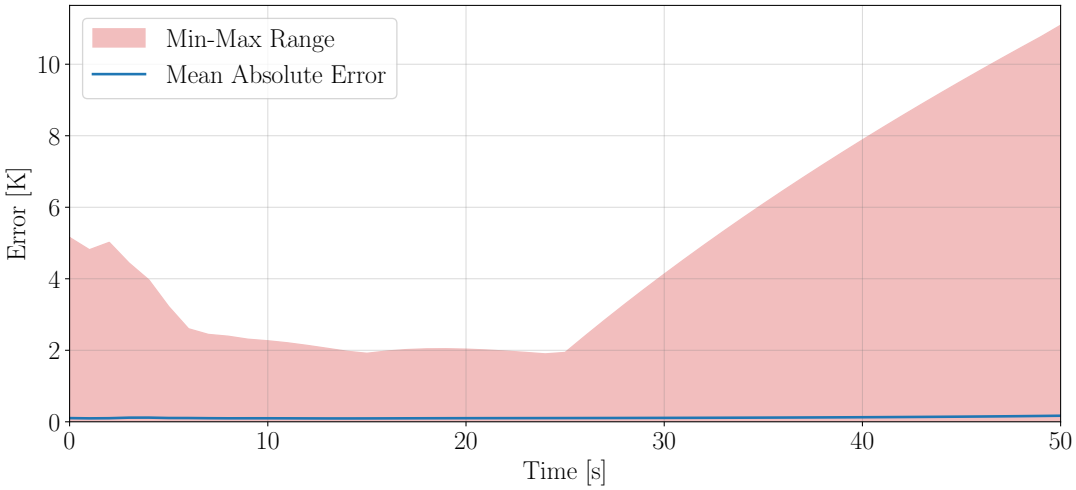


Figure B.8: MAE with min-max error range bands over time for the ConvLSTM model.

These results reveal that only the ConvLSTM model consistently achieves average prediction errors below the 5 K threshold, which was established as a desirable upper bound for acceptable performance. All other architectures, despite capturing some aspects of the thermal dynamics, exhibit significant deviations — particularly at later time steps or in specific regions of the domain.

Nonetheless, even the ConvLSTM is not yet fully satisfactory. While its average performance is excellent, a small number of predictions still exceed the desired accuracy threshold, indicating the need for additional physical constraints and improved generalization.

The next chapter will explore potential enhancements based on physics-informed approaches, in particular the integration of convolutional recurrent networks with explicit physical constraints using PINN-ConvLSTM architectures.

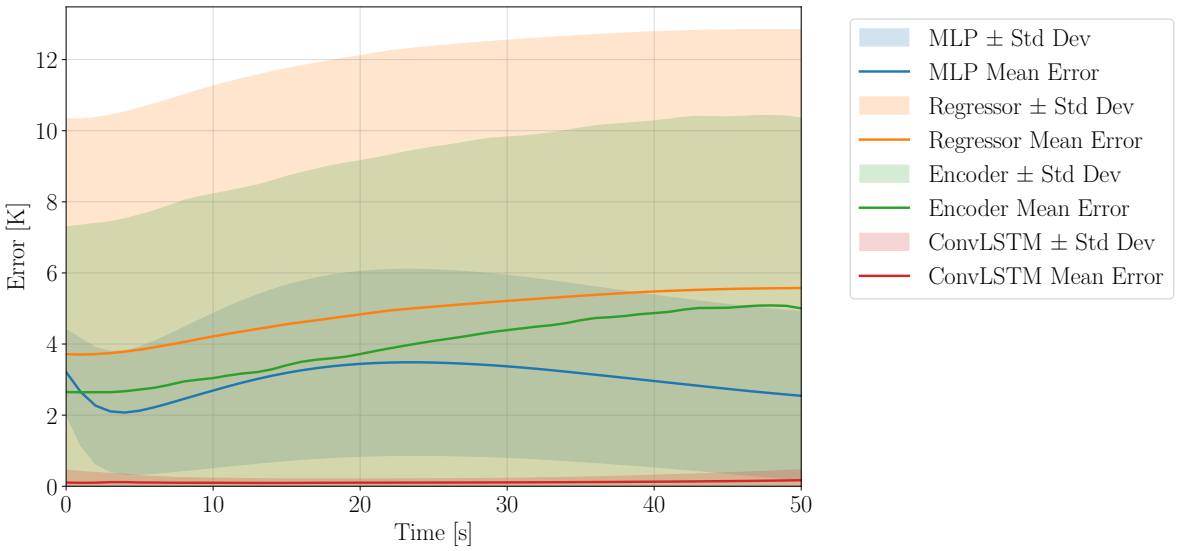


Figure B.9: Temporal evolution of the MAE for each architecture.

C

Complete PINN-parameter grid search

The following tables are complementary to those shown in section 9.2. They include the complete ranking of physics-informed models tested during the hyperparameter sweep, ordered by overall MAE.

Table C.1: All physics-informed models for $n_{\text{train}} = 10$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-6}	10^{-3}	1.6846	17.5397	+79.5%
2	1	1	10^{-2}	1.6854	17.4132	+79.5%
3	1	10^{-2}	10^{-3}	1.7480	15.5926	+78.7%
4	1	10^{-3}	10^{-2}	1.7963	19.6237	+78.2%
5	1	10^{-4}	10^{-2}	1.8199	22.5164	+77.9%
6	1	10^{-2}	10^{-4}	1.9471	23.8640	+76.3%
7	1	10^{-4}	10^{-4}	1.9961	20.9336	+75.7%
8	1	10^{-1}	10^{-2}	2.0524	20.0407	+75.0%
9	1	10^{-7}	0	2.0570	30.3938	+75.0%
10	1	10^{-5}	10^{-4}	2.2317	25.1944	+72.9%
11	1	10^{-3}	10^{-5}	2.2350	23.7347	+72.8%
12	1	10^{-4}	0	2.2964	22.4060	+72.1%
13	1	0	10^{-2}	2.3514	20.6837	+71.4%
14	1	10^{-7}	10^{-3}	2.3588	22.0460	+71.3%
15	1	10^{-4}	10^{-3}	2.3624	21.1764	+71.3%
16	1	1	10^{-3}	2.3994	21.7120	+70.8%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
17	1	10^{-3}	0	2.4618	25.2176	+70.1%
18	1	10^{-1}	10^{-3}	2.5058	34.9044	+69.5%
19	1	10^{-5}	0	2.5186	34.8688	+69.4%
20	1	10^{-4}	10^{-5}	2.5289	30.6791	+69.3%
21	1	1	10^{-5}	2.5315	28.3469	+69.2%
22	1	0	10^{-5}	2.5505	26.3147	+69.0%
23	1	10^{-3}	10^{-3}	2.6266	28.6726	+68.1%
24	1	10^{-7}	10^{-5}	2.7283	32.0832	+66.8%
25	1	10^{-6}	10^{-2}	2.7526	33.0568	+66.5%
26	1	10^{2}	10^{-4}	2.7557	33.3808	+66.5%
27	1	10^{-7}	10^{-4}	2.7968	41.9434	+66.0%
28	1	10^{2}	0	2.8116	46.6443	+65.8%
29	1	10^{-1}	10^{-5}	2.8229	29.4543	+65.7%
30	1	1	10^{-1}	3.0941	36.0800	+62.4%
31	1	10^{2}	10^{-5}	3.2040	37.9548	+61.0%
32	1	10^{3}	0	3.2692	38.7034	+60.2%
33	1	10^{2}	10^{-3}	3.3233	40.7574	+59.6%
34	1	10^{-2}	10^{-5}	3.3479	36.7432	+59.3%
35	1	10^{-7}	10^{-2}	3.6236	36.8296	+55.9%
36	1	10^{3}	10^{-1}	3.8120	38.4497	+53.6%
37	1	10^{-3}	10^{-4}	3.8418	52.7417	+53.3%
38	1	10^{-6}	10^{-5}	3.8912	52.9656	+52.7%
39	1	10^{3}	10^{-4}	4.1843	45.2309	+49.1%
40	1	10^{2}	10^{-1}	4.3060	45.4763	+47.6%
41	1	10^{-1}	10^{-4}	4.3235	42.8518	+47.4%
42	1	10^{-3}	10^{-1}	4.3707	40.1044	+46.9%
43	1	10^{-2}	10^{-2}	4.5802	46.0717	+44.3%
44	1	10^{2}	10^{-2}	4.6042	70.6763	+44.0%
45	1	10^{3}	10^{-2}	4.9844	43.0492	+39.4%
46	1	10^{-7}	10^{-1}	5.0779	37.6293	+38.3%
47	1	10^{-1}	0	5.3036	53.4569	+35.5%
48	1	0	10^{-3}	5.5601	55.1912	+32.4%
49	1	10^{-6}	0	5.8454	47.2546	+28.9%
50	1	10^{2}	1	5.8663	50.6231	+28.7%
51	1	0	10^{-4}	6.0352	51.1436	+26.6%
52	1	10^{-1}	10^{-1}	6.0422	38.9204	+26.5%
53	1	10^{-5}	10^{-3}	6.1291	57.4184	+25.5%
54	1	10^{-6}	10^{-1}	6.1716	46.3093	+25.0%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
55	0	1	10^{-5}	6.5981	58.2585	+19.8%
56	1	10^{-5}	10^{-5}	6.6781	51.0752	+18.8%
57	1	10^{-6}	10^{-4}	6.6880	52.4445	+18.7%
58	1	10^{-2}	0	7.3102	61.3133	+11.1%
59	1	1	10^{-4}	7.4866	54.2670	+9.0%
60	0	1	10^{-3}	7.9583	84.7718	+3.2%
Baseline	1	0	0	8.2242	55.4656	–
62	1	10^{-4}	1	8.4756	56.0128	–3.1%
63	1	10^{3}	10^{-3}	8.9259	61.5470	–8.5%
64	1	10^{3}	1	9.0164	60.4627	–9.6%
65	0	1	10^{-4}	9.2117	57.0940	–12.0%
66	1	1	0	9.6694	62.0895	–17.6%
67	1	10^{3}	10^{2}	9.9285	59.9710	–20.7%
68	1	10^{-5}	1	10.0118	60.8702	–21.7%
69	1	10^{-3}	1	10.4682	61.4697	–27.3%
70	0	0	0	10.5377	60.9797	–28.1%
71	1	10^{3}	10^{-5}	10.5472	61.2162	–28.2%
72	1	10^{-6}	1	10.6478	63.1034	–29.5%
73	0	1	0	11.0258	58.9380	–34.1%
74	1	10^{-2}	10^{-1}	11.1466	64.1675	–35.5%
75	1	1	1	11.2429	65.0094	–36.7%
76	1	10^{-5}	10^{-2}	11.3014	65.1033	–37.4%
77	1	10^{2}	10^{2}	11.5229	65.9814	–40.1%
78	1	10^{3}	10^{3}	11.6265	58.5611	–41.4%
79	1	10^{-4}	10^{3}	12.0978	68.8337	–47.1%
80	1	10^{-3}	10^{2}	12.2496	67.4180	–48.9%
81	1	10^{2}	10^{3}	12.6305	66.3493	–53.6%
82	1	10^{-5}	10^{3}	12.6482	72.0730	–53.8%
83	1	10^{-3}	10^{3}	12.7026	65.2977	–54.4%
84	1	10^{-2}	10^{3}	12.7674	66.5214	–55.2%
85	1	0	10^{3}	12.8366	67.4456	–56.1%
86	1	10^{2}	10^{1}	12.9108	71.4701	–57.0%
87	1	10^{-6}	10^{3}	12.9175	66.3967	–57.1%
88	1	10^{2}	10^{3}	13.0217	66.6696	–58.3%
89	1	10^{-7}	10^{3}	13.0355	68.1693	–58.5%
90	1	10^{3}	10^{1}	13.1354	66.9970	–59.7%
91	1	10^{-7}	10^{2}	13.2637	69.6104	–61.2%
92	1	10^{-6}	10^{2}	13.2824	69.2572	–61.4%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
93	1	10^{3}	10^{2}	13.4768	73.3231	–63.7%
94	1	10^{-7}	10^{1}	13.4965	70.1161	–63.9%
95	1	10^{-6}	10^{1}	13.5631	72.1459	–64.7%
96	1	10^{-5}	10^{2}	13.6661	76.1573	–65.9%
97	1	10^{-5}	10^{1}	13.9267	72.9042	–69.0%
98	1	10^{-5}	10^{3}	14.1655	74.5105	–71.9%
99	1	10^{-4}	10^{3}	14.2330	75.8911	–72.7%
100	1	10^{-3}	10^{-2}	15.0001	76.9706	–81.4%
101	1	10^{-4}	10^{2}	15.1174	79.2387	–82.8%
102	1	10^{-4}	10^{1}	15.3932	81.7274	–86.0%
103	1	10^{-2}	10^{2}	15.7636	84.3895	–90.3%
104	1	10^{-2}	10^{1}	15.7926	85.0405	–90.7%
105	1	10^{-3}	10^{1}	16.0643	87.0712	–94.0%
106	1	10^{-3}	10^{2}	16.4703	90.3591	–98.8%
107	1	10^{-2}	1	18.3103	90.7664	–120.5%
108	1	1	10^{2}	24.8419	91.5644	–202.1%

Table C.2: All physics-informed models for $n_{\text{train}} = 50$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-2}	10^{-2}	0.2923	8.2837	+96.4%
2	1	10^{-4}	10^{-3}	0.3524	7.3460	+95.7%
3	1	10^{3}	10^{-2}	0.4055	22.4204	+95.0%
4	1	10^{-1}	10^{-2}	0.4259	11.6117	+94.8%
5	1	10^{-5}	10^{-4}	0.4270	12.8592	+94.8%
6	1	10^{-5}	10^{-3}	0.4438	9.7711	+94.6%
7	1	10^{2}	10^{-2}	0.5307	15.4444	+93.5%
8	1	10^{-3}	10^{-4}	0.5377	19.2638	+93.4%
9	1	1	10^{-3}	0.5521	14.2679	+93.2%
10	1	10^{-1}	0	0.5693	16.8806	+93.0%
11	0	10^{2}	10^{-1}	0.6063	24.9060	+92.6%
12	1	1	10^{-4}	0.6407	18.6046	+92.1%
13	1	10^{2}	0	0.7370	22.9093	+91.0%
14	1	1	0	0.7384	23.8145	+90.9%
15	1	0	10^{-2}	0.7433	22.2852	+90.9%
16	1	10^{-1}	10^{-4}	0.7744	16.6177	+90.5%
17	1	0	10^{-3}	0.7982	16.1385	+90.2%
18	1	10^{3}	10^{-1}	0.8270	21.5761	+89.9%
19	1	10^{2}	10^{-3}	0.8411	21.1951	+89.7%
20	1	10^{-2}	10^{-1}	0.8486	15.9033	+89.6%
21	1	10^{-4}	0	0.8536	27.7775	+89.5%
22	1	0	10^{-1}	0.9506	19.5525	+88.3%
23	1	10^{-4}	10^{-2}	0.9873	24.0050	+87.9%
24	0	10^{2}	10^{-2}	1.0847	29.0549	+86.7%
25	1	0	10^{-5}	1.0882	20.7186	+86.6%
26	1	10^{-5}	10^{-5}	1.1109	21.3821	+86.4%
27	1	10^{-5}	0	1.2598	18.4530	+84.5%
28	1	10^{-3}	10^{-2}	1.3332	23.3102	+83.6%
29	1	10^{-3}	10^{-3}	1.3458	20.0396	+83.5%
30	0	10^{3}	10^{-4}	1.3991	27.1212	+82.8%
31	1	10^{-2}	0	1.4476	24.5300	+82.2%
32	1	10^{2}	10^{-1}	1.5295	23.4258	+81.2%
33	0	10^{3}	10^{-5}	1.5342	40.0429	+81.2%
34	1	10^{2}	10^{-4}	1.5540	33.1344	+80.9%
35	0	10^{2}	10^{-4}	1.5991	30.2235	+80.4%
36	1	10^{-1}	10^{-5}	1.5992	23.8613	+80.4%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
37	1	10^{-3}	10^{-1}	1.6091	21.2999	+80.3%
38	1	10^{3}	10^{-4}	1.6166	39.1603	+80.2%
39	0	10^{3}	10^{-3}	1.7675	30.5421	+78.3%
40	1	10^{3}	0	1.8110	43.8107	+77.8%
41	1	10^{-1}	10^{-1}	1.9557	30.3415	+76.0%
42	0	10^{2}	0	1.9629	36.1352	+75.9%
43	1	10^{3}	10^{-3}	1.9711	34.7259	+75.8%
44	1	10^{-5}	10^{-1}	1.9884	30.0775	+75.6%
45	1	1	10^{-5}	2.0172	29.0933	+75.2%
46	1	10^{-3}	0	2.0214	33.5976	+75.2%
47	1	10^{3}	1	2.0751	35.2496	+74.5%
48	0	10^{3}	10^{-2}	2.2249	40.6979	+72.7%
49	0	10^{2}	10^{-5}	2.2618	31.1031	+72.2%
50	1	10^{-2}	10^{-3}	2.3137	35.1520	+71.6%
51	0	10^{-2}	0	2.3474	38.3301	+71.2%
52	1	0	10^{-4}	2.4328	28.1638	+70.1%
53	0	10^{-2}	10^{-5}	2.5251	36.9149	+69.0%
54	0	10^{-1}	0	2.6350	35.3484	+67.7%
55	0	10^{-3}	0	2.7284	38.7417	+66.5%
56	1	10^{2}	1	3.6044	35.0302	+55.8%
57	1	1	1	3.9519	41.6830	+51.5%
58	1	10^{-5}	10^{-2}	4.0091	38.6483	+50.8%
59	0	1	10^{-5}	4.0166	41.4202	+50.7%
60	0	10^{-4}	0	4.2910	44.5592	+47.3%
61	1	1	10^{-2}	4.3633	48.3356	+46.5%
62	0	1	10^{-4}	4.4197	40.3150	+45.8%
63	1	10^{-4}	10^{-1}	4.6272	37.9440	+43.2%
64	0	10^{-1}	10^{-4}	4.6442	47.5195	+43.0%
65	0	10^{3}	1	4.7420	43.9351	+41.8%
66	0	10^{3}	0	4.7472	40.4342	+41.8%
67	1	10^{-2}	10^{-4}	4.7783	47.1132	+41.4%
68	0	1	10^{-3}	4.9256	47.2499	+39.6%
69	1	10^{3}	2	5.0582	39.3604	+37.9%
70	1	10^{-2}	10^{-5}	5.5410	55.0272	+32.0%
71	1	10^{-4}	1	6.0845	44.3877	+25.3%
72	1	10^{-2}	1	6.3306	42.8449	+22.3%
73	0	10^{3}	2	6.5964	47.7087	+19.1%
74	1	10^{-4}	10^{-1}	6.7625	51.9305	+17.0%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
75	1	10^{-1}	10^{-2}	6.8451	57.2045	+16.0%
76	1	1	10^{-1}	7.0838	53.6540	+13.1%
77	1	0	1	7.1751	49.5603	+12.0%
78	1	10^{2}	10^{-5}	7.6213	58.9952	+6.5%
Baseline	1	0	0	8.1499	53.8969	–
80	1	10^{-3}	10^{-5}	8.6469	59.7790	-6.1%
81	1	10^{-4}	10^{-4}	8.6974	55.9859	-6.7%
82	0	1	10^{-2}	9.0434	54.1574	-11.0%
83	0	10^{-1}	10^{-2}	9.0749	59.7584	-11.3%
84	0	10^{-1}	10^{-5}	9.1694	64.1143	-12.5%
85	0	10^{2}	1	9.4699	59.6851	-16.2%
86	0	1	0	9.4821	58.8148	-16.3%
87	0	10^{2}	10^{-3}	9.5506	57.0667	-17.2%
88	1	10^{3}	10^{-5}	9.7038	59.1117	-19.1%
89	0	10^{3}	10^{-1}	9.8031	62.3434	-20.3%
90	0	10^{-2}	10^{-2}	10.1219	63.6604	-24.2%
91	1	10^{-5}	1	10.4506	61.8882	-28.2%
92	0	0	0	10.5111	60.4176	-29.0%
93	0	10^{-5}	0	10.5512	60.9988	-29.5%
94	1	10^{-4}	2	10.7174	64.5486	-31.5%
95	0	1	10^{-1}	10.9952	63.8639	-34.9%
96	0	10^{-3}	10^{-3}	11.0372	64.8903	-35.4%
97	1	10^{-3}	1	11.1217	65.2971	-36.5%
98	0	10^{-1}	10^{-1}	11.1591	65.4984	-36.9%
99	1	10^{2}	10^{2}	11.1808	64.7617	-37.2%
100	1	10^{-2}	10^{2}	11.1877	64.1328	-37.3%
101	0	10^{-3}	10^{-5}	11.7463	75.4874	-44.1%
102	0	10^{-4}	10^{-2}	11.9964	70.8299	-47.2%
103	0	0	10^{-5}	12.3471	70.0104	-51.5%
104	0	1	1	12.4010	66.7170	-52.2%
105	1	1	10^{2}	12.4652	66.1717	-52.9%
106	1	10^{-1}	10^{2}	12.6530	68.0308	-55.3%
107	1	10^{2}	10^{3}	12.7624	69.6201	-56.6%
108	0	10^{-5}	10^{3}	12.9528	70.5173	-58.9%
109	0	10^{-3}	10^{-2}	13.1836	69.6239	-61.8%
110	0	0	10^{3}	13.3175	67.5399	-63.4%
111	1	10^{-3}	10^{3}	13.3432	67.3344	-63.7%
112	0	10^{3}	10^{3}	13.4569	66.3807	-65.1%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
113	0	0	10^{-2}	13.4586	71.0417	-65.1%
114	0	10^{-4}	10^{-4}	13.5158	78.1216	-65.8%
115	1	0	10^{3}	13.5537	69.8541	-66.3%
116	0	10^{-1}	10^{-3}	13.5943	81.1129	-66.8%
117	1	10^{3}	10^{3}	13.6055	70.8212	-66.9%
118	1	10^{-2}	10^{3}	13.7373	68.7774	-68.6%
119	0	10^{2}	10^{3}	13.7497	70.1696	-68.7%
120	0	10^{-2}	10^{-3}	13.7943	67.9284	-69.3%
121	1	0	10^{2}	13.9180	67.3907	-70.8%
122	1	10^{-3}	10^{2}	13.9399	70.8701	-71.0%
123	0	0	1	14.0778	72.6468	-72.7%
124	0	0	10^{2}	14.1825	71.2784	-74.0%
125	0	10^{-3}	10^{3}	14.2180	69.3482	-74.5%
126	0	10^{-5}	1	14.2899	71.8813	-75.3%
127	0	10^{-2}	10^{3}	14.8686	71.3404	-82.4%
128	0	10^{-5}	10^{-5}	14.9170	72.9240	-83.0%
129	1	10^{-5}	10^{3}	14.9461	80.1060	-83.4%
130	0	10^{-5}	10^{-1}	14.9739	73.6157	-83.7%
131	1	10^{-5}	10^{2}	15.0662	70.8438	-84.9%
132	0	10^{-4}	10^{2}	15.1134	72.3911	-85.4%
133	1	10^{-1}	1	15.1390	84.5784	-85.8%
134	1	10^{-1}	10^{3}	15.1419	73.9164	-85.8%
135	0	10^{-1}	10^{3}	15.4917	70.2590	-90.1%
136	0	10^{-2}	10^{2}	15.5892	72.1039	-91.3%
137	0	10^{-3}	10^{2}	15.7701	74.7251	-93.5%
138	0	10^{-5}	10^{-3}	15.7791	71.0333	-93.6%
139	1	10^{-4}	10^{3}	15.8567	74.1169	-94.6%
140	0	10^{-3}	10^{-4}	15.8847	74.5739	-94.9%
141	1	1	10^{3}	16.0163	72.9654	-96.5%
142	0	10^{-4}	1	16.0579	84.7122	-97.0%
143	0	10^{-5}	10^{2}	16.1656	73.7574	-98.4%
144	0	10^{-1}	1	16.2221	76.4624	-99.0%
145	0	1	10^{2}	16.8293	76.2934	-106.5%
146	0	10^{-2}	10^{-1}	16.8638	81.4859	-106.9%
147	0	0	10^{-1}	17.8414	72.6705	-118.9%
148	0	10^{-4}	10^{-5}	18.1346	92.9403	-122.5%
149	0	10^{-1}	10^{2}	18.4846	77.8501	-126.8%
150	0	1	10^{3}	18.6646	70.9060	-129.0%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
151	0	10^{-4}	10^{-1}	19.0394	78.3851	-133.6%
152	0	0	10^{-3}	19.0462	76.4506	-133.7%
153	0	10^{-2}	10^{-4}	19.0980	77.3689	-134.3%
154	0	10^{-3}	1	19.1190	77.6388	-134.6%
155	0	10^{-5}	10^{-4}	19.5171	79.5535	-139.5%
156	0	10^{-5}	10^{-2}	19.8636	75.5079	-143.7%
157	0	10^{-4}	10^{-3}	20.0225	87.5700	-145.7%
158	0	10^{-2}	1	20.2689	86.3755	-148.7%
159	0	10^{2}	10^{2}	20.4040	94.3109	-150.4%
160	0	0	10^{-4}	20.8306	79.3133	-155.6%
161	0	10^{-4}	10^{3}	21.4452	79.9833	-163.1%
162	0	10^{-3}	10^{-1}	23.2011	75.0230	-184.7%

Table C.3: All physics-informed models for $n_{\text{train}} = 200$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-6}	10^{-3}	0.1880	4.9379	+95.6%
2	1	10^{-5}	10^{-3}	0.1897	7.2890	+95.6%
3	1	10^{-2}	10^{-3}	0.1979	6.1811	+95.4%
4	1	10^{-6}	10^{-2}	0.2053	7.3129	+95.2%
5	1	10^{-3}	10^{-2}	0.2060	8.3923	+95.2%
6	1	10^{-6}	10^{-4}	0.2145	6.2305	+95.0%
7	1	1	0	0.2238	12.3674	+94.8%
8	1	10^{-2}	10^{-4}	0.2246	6.1144	+94.8%
9	1	10^{-4}	10^{-2}	0.2258	5.3254	+94.7%
10	1	10^{-5}	0	0.2278	6.7252	+94.7%
11	1	10^{-5}	10^{-2}	0.2283	9.8255	+94.7%
12	1	10^{-7}	10^{-2}	0.2344	8.5228	+94.5%
13	1	1	10^{-3}	0.2395	5.6524	+94.4%
14	1	1	10^{-2}	0.2451	9.2895	+94.3%
15	1	10^{-4}	10^{-3}	0.2485	6.7053	+94.2%
16	1	10^{-1}	0	0.2560	9.3678	+94.0%
17	1	0	10^{-5}	0.2562	8.5247	+94.0%
18	1	0	10^{-2}	0.2644	5.5913	+93.8%
19	1	10^{-2}	0	0.2662	16.1375	+93.8%
20	1	10^{3}	10^{-5}	0.2807	20.0231	+93.4%
21	1	10^{2}	10^{-5}	0.2874	12.6512	+93.3%
22	1	10^{2}	10^{-2}	0.2889	8.3262	+93.3%
23	1	10^{2}	10^{-3}	0.2903	6.7545	+93.2%
24	1	10^{-2}	10^{-2}	0.2971	9.4814	+93.1%
25	1	10^{2}	0	0.2978	14.5403	+93.0%
26	1	1	10^{-4}	0.3020	11.8283	+92.9%
27	1	10^{-1}	10^{-3}	0.3285	9.3539	+92.3%
28	1	10^{-3}	10^{-5}	0.3502	7.9107	+91.8%
29	1	10^{-4}	10^{-5}	0.3834	9.2164	+91.0%
30	1	10^{-1}	10^{-2}	0.3850	6.1287	+91.0%
31	1	10^{-4}	10^{-4}	0.3866	9.5936	+91.0%
32	1	1	10^{-5}	0.3876	12.2494	+90.9%
33	1	10^{3}	10^{-2}	0.3899	25.7663	+90.9%
34	1	10^{-7}	10^{-1}	0.4039	11.4058	+90.6%
35	1	10^{-5}	10^{-5}	0.4057	13.1778	+90.5%
36	1	0	10^{-1}	0.4183	12.2098	+90.2%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
37	1	10^{-1}	10^{-1}	0.4187	10.8296	+90.2%
38	1	10^{3}	10^{-3}	0.4188	19.8586	+90.2%
39	1	10^{3}	0	0.4217	26.1864	+90.2%
40	1	0	10^{-3}	0.4235	5.0122	+90.1%
41	1	10^{2}	10^{-4}	0.4239	8.5507	+90.1%
42	1	10^{-3}	10^{-4}	0.4498	6.8358	+89.5%
43	1	10^{-5}	10^{-1}	0.4708	13.1837	+89.0%
44	1	10^{-3}	0	0.4850	13.2860	+88.7%
45	1	0	10^{-4}	0.5006	6.5991	+88.3%
46	1	10^{-7}	10^{-5}	0.5134	7.4913	+88.0%
47	1	10^{-3}	10^{-1}	0.5218	15.6264	+87.8%
48	1	10^{3}	10^{-4}	0.5366	28.6932	+87.5%
49	1	10^{-1}	10^{-5}	0.5448	13.7383	+87.3%
50	1	10^{-7}	10^{-4}	0.5492	12.5791	+87.2%
51	1	10^{-5}	10^{-4}	0.5624	10.8801	+86.9%
52	1	10^{-6}	10^{-1}	0.5820	18.8124	+86.4%
53	1	10^{-4}	0	0.5928	9.9971	+86.2%
54	1	10^{-2}	10^{-5}	0.6237	11.5647	+85.4%
55	1	10^{-7}	10^{-3}	0.6398	11.6956	+85.1%
56	1	10^{2}	10^{-1}	0.6571	16.5859	+84.7%
57	1	10^{3}	10^{-1}	0.6929	27.0691	+83.8%
58	1	10^{-6}	0	0.7011	13.9610	+83.6%
59	1	10^{-7}	0	0.8536	10.5801	+80.1%
60	1	10^{2}	1	0.8631	19.5472	+79.8%
61	1	10^{-3}	10^{-3}	0.9262	9.7358	+78.4%
62	1	10^{-6}	10^{-5}	0.9443	19.0014	+77.9%
63	1	1	10^{-1}	1.0115	22.0156	+76.4%
64	0	10^{-3}	0	1.0644	32.5893	+75.1%
65	1	10^{3}	1	1.2557	19.4321	+70.7%
66	1	10^{-4}	10^{-1}	1.4823	11.2698	+65.4%
67	1	10^{-2}	10^{-1}	1.5692	24.2415	+63.4%
68	1	10^{-1}	10^{-4}	1.6127	28.3150	+62.3%
69	1	10^{-4}	1	1.9369	35.5452	+54.8%
70	1	10^{-6}	1	2.2814	22.9942	+46.7%
71	1	10^{-5}	1	2.3586	36.4032	+44.9%
72	1	10^{-2}	1	2.4687	34.0167	+42.3%
73	1	10^{-1}	1	2.5579	34.7904	+40.3%
74	1	1	1	2.7973	30.0027	+34.7%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
75	1	10^{-4}	1	2.8731	41.3262	+32.9%
76	1	10^{-7}	1	2.9008	39.9518	+32.3%
Baseline	1	0	0	4.2818	57.7394	–
78	1	0	1	4.3982	46.5561	-2.7%
79	0	10^{-5}	0	6.4328	50.2639	-50.2%
80	1	10^{3}	10^{3}	7.5724	50.0736	-76.8%
81	0	10^{-4}	0	7.9484	55.1229	-85.6%
82	1	10^{3}	10^{2}	9.5659	59.7211	-123.4%
83	0	0	0	10.4513	59.6852	-144.1%
84	1	10^{-2}	10^{2}	10.9897	64.4353	-156.7%
85	0	10^{-5}	10^{-5}	11.0128	63.7180	-157.2%
86	1	10^{-3}	10^{3}	11.2711	66.2299	-163.2%
87	1	10^{-4}	10^{3}	11.2870	64.5821	-163.6%
88	1	1	10^{2}	11.3807	59.7778	-165.8%
89	1	10^{-6}	10^{2}	11.5020	65.0079	-168.6%
90	1	10^{-1}	10^{2}	11.8013	63.6904	-175.6%
91	0	10^{-5}	10^{3}	11.9506	67.4896	-179.1%
92	0	10^{-4}	1	12.0727	66.3893	-182.0%
93	0	0	10^{-3}	12.1442	64.6521	-183.6%
94	0	10^{-4}	10^{-5}	12.3525	90.6746	-188.5%
95	0	10^{-5}	10^{-4}	12.4497	62.2252	-190.8%
96	1	10^{-7}	10^{2}	12.5021	68.9606	-192.0%
97	0	10^{-4}	10^{-3}	12.7283	69.2595	-197.3%
98	1	10^{2}	10^{2}	12.7835	67.4800	-198.6%
99	0	0	10^{-2}	12.8394	67.9992	-199.9%
100	0	10^{-5}	1	12.8597	69.3105	-200.3%
101	1	10^{-3}	10^{2}	12.9499	67.5348	-202.4%
102	0	10^{-4}	10^{3}	13.0345	72.8045	-204.4%
103	0	10^{-5}	10^{-1}	13.0561	72.6797	-204.9%
104	0	10^{-4}	10^{-2}	13.3620	69.1100	-212.1%
105	0	10^{-5}	10^{-3}	13.3730	69.5735	-212.3%
106	1	1	10^{3}	13.3743	63.8975	-212.3%
107	1	10^{-1}	10^{3}	13.4643	67.1069	-214.5%
108	0	10^{-4}	10^{-1}	13.5387	68.6308	-216.2%
109	0	10^{-5}	10^{2}	14.0523	69.1578	-228.2%
110	1	10^{2}	10^{3}	14.1079	69.2152	-229.5%
111	0	10^{-5}	10^{-2}	14.3783	71.0439	-235.8%
112	1	10^{-5}	10^{2}	14.5660	68.3393	-240.2%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
113	0	0	10^{-1}	14.6812	70.1874	-242.9%
114	1	10^{-6}	10^{3}	14.8588	70.0669	-247.0%
115	0	10^{-4}	10^{2}	15.0999	72.9992	-252.6%
116	1	0	10^{3}	15.4485	72.7985	-260.8%
117	0	10^{-4}	10^{-4}	15.5324	72.7882	-262.7%
118	1	0	10^{2}	16.5075	74.7811	-285.5%
119	0	0	10^{-5}	17.0405	71.3426	-298.0%
120	1	10^{-5}	10^{3}	17.4753	78.0845	-308.1%
121	0	0	10^{2}	17.6487	72.6657	-312.2%
122	1	10^{-2}	10^{3}	18.3653	76.5586	-328.9%
123	1	10^{-7}	10^{3}	18.3930	75.3590	-329.6%
124	1	10^{-4}	10^{2}	19.4391	78.2607	-354.0%
125	0	0	10^{3}	20.0916	79.0819	-369.2%
126	0	0	1	20.7882	85.4284	-385.5%
127	0	0	10^{-4}	22.3064	80.8184	-421.0%

Table C.4: All physics-informed models for $n_{\text{train}} = 400$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	1	0	0.1215	4.2932	+91.0%
2	1	10^{-2}	10^{-3}	0.1258	3.6793	+90.7%
3	1	0	10^{-3}	0.1283	3.6538	+90.5%
4	1	10^{-7}	0	0.1350	4.3961	+90.0%
5	1	10^{-5}	0	0.1357	4.0503	+90.0%
6	1	10^{-4}	10^{-3}	0.1366	3.2666	+89.9%
7	1	10^{-7}	10^{-4}	0.1428	4.7796	+89.4%
8	1	10^{-6}	10^{-3}	0.1476	3.1357	+89.1%
9	1	10^{-6}	0	0.1544	5.7311	+88.6%
10	1	10^{-1}	10^{-4}	0.1584	5.6473	+88.3%
11	1	1	10^{-3}	0.1645	5.4755	+87.8%
12	1	0	10^{-4}	0.1707	3.6626	+87.4%
13	1	10^{-1}	0	0.1738	5.0821	+87.2%
14	1	10^{-1}	10^{-2}	0.1799	9.5598	+86.7%
15	1	10^{-7}	10^{-2}	0.1822	7.2082	+86.5%
16	1	10^{-2}	0	0.1839	5.4094	+86.4%
17	1	10^{-7}	10^{-3}	0.1853	3.6474	+86.3%
18	1	10^{-2}	10^{-4}	0.1854	4.4844	+86.3%
19	1	1	10^{-2}	0.1890	6.4731	+86.0%
20	1	10^{-5}	10^{-2}	0.1945	7.4581	+85.6%
21	1	10^{-2}	10^{-2}	0.2073	7.4340	+84.7%
22	1	10^{-3}	0	0.2140	6.5280	+84.2%
23	1	10^{-3}	10^{-2}	0.2151	8.1753	+84.1%
24	1	10^{-6}	10^{-4}	0.2307	6.4583	+82.9%
25	1	10^{-5}	10^{-3}	0.2486	5.5427	+81.6%
26	1	10^{-4}	10^{-4}	0.2590	4.5119	+80.8%
27	1	10^{-5}	10^{-4}	0.2736	7.3135	+79.8%
28	1	0	10^{-2}	0.2741	7.6244	+79.7%
29	1	10^{-4}	0	0.2884	7.7583	+78.7%
30	1	10^{-4}	10^{-2}	0.2954	5.3815	+78.2%
31	1	10^{-6}	10^{-2}	0.4672	8.1854	+65.5%
32	1	10^{-3}	10^{-4}	0.5156	10.4219	+61.9%
33	1	10^{-1}	10^{-3}	0.5286	8.0597	+60.9%
34	1	1	10^{-4}	0.7740	10.1858	+42.8%
35	1	10^{-3}	10^{-3}	0.9579	19.9636	+29.2%
Baseline	1	0	0	1.3527	30.4930	–

Table C.5: All physics-informed models for $n_{\text{train}} = 500$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-6}	10^{-3}	0.1156	4.0574	+67.1%
2	1	0	10^{-3}	0.1169	5.1126	+66.7%
3	1	1	0	0.1209	4.3138	+65.6%
4	1	10^{-1}	10^{-4}	0.1222	3.5352	+65.2%
5	1	10^{-2}	0	0.1232	6.4490	+64.9%
6	1	10^{-4}	0	0.1300	3.0866	+63.0%
7	1	10^{-1}	10^{-3}	0.1308	3.3396	+62.7%
8	1	10^{-7}	10^{-4}	0.1315	4.0293	+62.5%
9	1	10^{-7}	0	0.1319	5.2489	+62.4%
10	1	10^{-3}	10^{-4}	0.1340	5.4654	+61.8%
11	1	10^{-4}	10^{-2}	0.1373	7.9906	+60.9%
12	1	10^{-5}	0	0.1378	3.9852	+60.8%
13	1	10^{-5}	10^{-3}	0.1383	3.0006	+60.6%
14	1	0	10^{-2}	0.1393	8.0773	+60.3%
15	1	10^{-6}	10^{-4}	0.1415	5.3219	+59.7%
16	1	10^{-4}	10^{-4}	0.1425	3.7318	+59.4%
17	1	10^{-1}	10^{-2}	0.1463	8.6627	+58.3%
18	1	10^{-5}	10^{-2}	0.1480	9.3591	+57.9%
19	1	10^{-3}	10^{-3}	0.1493	5.6059	+57.5%
20	1	10	0	0.1545	3.9174	+56.0%
21	1	10	10^{-4}	0.1552	4.7207	+55.8%
22	1	10^{-4}	10^{-3}	0.1581	3.2180	+55.0%
23	1	10^{-6}	0	0.1594	3.9249	+54.6%
24	1	10^{-1}	0	0.1596	4.8676	+54.6%
25	1	10^{-2}	10^{-2}	0.1642	8.6054	+53.2%
26	1	10	10^{-2}	0.1680	7.8052	+52.1%
27	1	100	0	0.1791	9.3169	+49.0%
28	1	10^{-7}	10^{-2}	0.1804	9.9299	+48.6%
29	1	10^{-6}	10^{-2}	0.1816	6.0132	+48.3%
30	1	10^{-3}	0	0.1900	4.2121	+45.9%
31	1	10^{-2}	10^{-4}	0.1953	4.2472	+44.4%
32	1	1	10^{-3}	0.2046	3.1189	+41.7%
33	1	10	10^{-3}	0.2084	4.5069	+40.6%
34	1	100	10^{-2}	0.2140	7.5345	+39.1%
35	1	100	10^{-3}	0.2199	9.3814	+37.4%
36	1	0	10^{-4}	0.2325	4.8106	+33.8%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
37	1	10^{-2}	10^{-3}	0.2419	4.8655	+31.1%
38	1	1	10^{-4}	0.2607	5.8866	+25.8%
39	1	10^{-7}	10^{-3}	0.2678	4.9911	+23.7%
40	1	1	10^{-2}	0.3282	5.0024	+6.5%
41	1	10^{-5}	10^{-4}	0.3367	5.8716	+4.1%
Baseline	1	0	0	0.3512	17.1021	–
43	1	10^{-3}	10^{-2}	0.3535	8.2020	-0.7%
44	1	100	10^{-4}	1.9373	19.5220	-451.7%
45	1	1	1	13.9966	78.3488	-3885.9%

Table C.6: All physics-informed models for $n_{\text{train}} = 1000$

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	**Overall MAE [K]**	**Max Error [K]**	**Improvement**
1	1	10^{-5}	10^{-3}	0.0869	3.8003	+67.1%
2	1	1	10^{-3}	0.0896	4.1618	+66.0%
3	1	10^{-4}	10^{-3}	0.0918	3.5979	+65.2%
4	1	10^{-2}	10^{-5}	0.0935	5.0900	+64.6%
5	1	10^{-3}	10^{-4}	0.0935	2.7901	+64.5%
6	1	10^{-2}	10^{-3}	0.0981	3.6680	+62.8%
7	1	10^{-1}	10^{-5}	0.0988	4.0350	+62.6%
8	1	10^{-5}	10^{-4}	0.1010	3.4616	+61.7%
9	1	10^{-5}	10^{-2}	0.1044	8.7046	+60.4%
10	1	10^{-3}	10^{-3}	0.1048	4.6641	+60.3%
11	1	10^{-4}	10^{-2}	0.1069	7.2182	+59.5%
12	1	10^{-3}	0	0.1082	3.6938	+59.0%
13	1	10^{-2}	10^{-4}	0.1102	1.9897	+58.2%
14	1	0	10^{-3}	0.1102	5.1931	+58.2%
15	1	10^{-1}	10^{-3}	0.1142	3.0933	+56.7%
16	1	10^{-4}	0	0.1147	5.0342	+56.5%
17	1	10^{-4}	10^{-4}	0.1153	4.9268	+56.3%
18	1	10^{-2}	0	0.1207	3.7972	+54.2%
19	1	0	10^{-2}	0.1230	6.0745	+53.4%
20	1	10^{-5}	0	0.1339	5.6721	+49.2%
21	1	10^{-1}	0	0.1362	3.9753	+48.4%
22	1	10^{-4}	10^{-5}	0.1364	4.2007	+48.3%
23	1	10^{-2}	10^{-2}	0.1396	3.1560	+47.1%
24	1	10^{-3}	10^{-5}	0.1434	4.9075	+45.7%
25	1	1	10^{-4}	0.1522	5.4514	+42.3%
26	1	10^{-2}	10^{-1}	0.1580	10.2137	+40.1%
27	1	10^{-3}	10^{-1}	0.1644	12.2685	+37.7%
28	1	10^{-5}	10^{-1}	0.1649	10.6891	+37.5%
29	1	10^{-1}	10^{-4}	0.1753	3.1532	+33.6%
30	1	0	10^{-5}	0.2048	5.1812	+22.4%
31	1	10^{-1}	10^{-2}	0.2167	4.0932	+17.9%
32	1	0	10^{-4}	0.2238	5.0434	+15.2%
33	1	10^{-3}	10^{-2}	0.2292	3.6736	+13.1%
Baseline	1	0	0	0.2638	12.2686	–
35	1	10^{-4}	10^{-1}	0.3084	9.8254	-16.9%
36	1	10^{-1}	10^{-1}	0.3109	5.3568	-17.9%

Rank	λ_{MSE}	λ_{phy}	λ_{bnd}	Overall MAE [K]	Max Error [K]	Improvement
37	1	0	10^{-1}	0.3120	5.2569	-18.3%
38	1	1	10^{-1}	0.3952	11.1448	-49.8%
39	1	1	10^{-2}	0.4017	5.1294	-52.3%
40	1	10^{-3}	1	0.5739	21.7873	-117.5%
41	1	10^{-2}	1	0.5850	17.4250	-121.7%
42	1	10^{-1}	1	0.5865	12.7533	-122.3%
43	1	10^{-4}	1	0.6563	10.9492	-148.8%
44	1	0	1	0.7087	14.0325	-168.6%
45	1	10^{-5}	1	1.5317	29.2703	-480.6%
46	1	10^{-5}	10^{-5}	4.8818	47.5795	-1750.4%
47	1	1	10^{2}	11.0108	62.8765	-4073.6%
48	1	0	10^{3}	12.2117	67.5715	-4528.8%
49	1	1	10^{3}	12.5641	63.7546	-4662.4%
50	1	0	10^{2}	13.2980	81.8228	-4940.6%

D

Complete PINN training time metrics

This appendix provides detailed training time metrics for all physics-informed models evaluated in this work. Each entry includes the loss configuration, the total training time, the number of epochs, and the mean time per epoch. These results help contextualize the training cost introduced by each physics configuration and complement the global overhead analyses presented in subsection 9.2.1.

All experiments were run with the same hardware and training configuration: 650-second sequences, $\Delta t = 10$ s, batch size 64. The baseline always corresponds to the MSE-only loss.

Table D.1: Training time metrics for all tested models with $n_{\text{train}} = 10$.

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	0	0	0.51	108	0.005
1	0	10^{0}	0.75	100	0.007
1	0	10^{-4}	1.18	157	0.008
1	0	10^{-3}	0.94	124	0.008
1	0	10^{-2}	1.01	133	0.008
1	0	10^{-5}	0.89	118	0.008
1	10^{0}	0	0.80	102	0.008
1	10^{0}	10^{-4}	0.95	100	0.010
1	10^{0}	10^{-3}	0.53	55	0.010
1	10^{0}	10^{0}	1.07	132	0.008
1	10^{0}	10^{-5}	0.51	64	0.008
1	10^{0}	0	1.87	225	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{0}	10^{-4}	2.15	259	0.008
1	10^{0}	10^{-3}	1.85	223	0.008
1	10^{0}	10^{-2}	2.78	334	0.008
1	10^{0}	10^{-1}	1.24	150	0.008
1	10^{0}	10^{0}	1.81	219	0.008
1	10^{0}	10^{2}	1.70	205	0.008
1	10^{0}	10^{3}	3.08	370	0.008
1	10^{0}	10^{-5}	2.25	271	0.008
1	10^{0}	0	2.33	281	0.008
1	10^{0}	10^{-4}	1.82	219	0.008
1	10^{0}	10^{-3}	1.72	208	0.008
1	10^{0}	10^{-3}	1.38	181	0.008
1	10^{0}	10^{-2}	1.79	235	0.008
1	10^{0}	10^{-2}	2.66	319	0.008
1	10^{0}	10^{-1}	0.96	128	0.008
1	10^{0}	10^{-1}	1.47	178	0.008
1	10^{0}	10^{0}	1.10	145	0.008
1	10^{0}	10^{0}	1.32	160	0.008
1	10^{0}	10^{2}	1.11	147	0.008
1	10^{0}	10^{2}	2.09	250	0.008
1	10^{0}	10^{3}	1.67	220	0.008
1	10^{0}	10^{3}	2.71	326	0.008
1	10^{0}	10^{-5}	1.72	222	0.008
1	10^{-2}	0	2.02	266	0.008
1	10^{-2}	0	0.98	118	0.008
1	10^{-2}	10^{-4}	1.19	145	0.008
1	10^{-2}	10^{-4}	4.16	545	0.008
1	10^{-2}	10^{-3}	1.87	226	0.008
1	10^{-2}	10^{-2}	1.62	196	0.008
1	10^{-2}	10^{-2}	1.63	216	0.008
1	10^{-2}	10^{-1}	1.06	128	0.008
1	10^{-2}	10^{-1}	0.99	130	0.008
1	10^{-2}	10^{0}	1.16	152	0.008
1	10^{-2}	10^{0}	1.30	157	0.008
1	10^{-2}	10^{2}	1.67	202	0.008
1	10^{-2}	10^{2}	2.79	365	0.008
1	10^{-2}	10^{3}	1.24	150	0.008
1	10^{-2}	10^{3}	1.42	186	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-2}	10^{-5}	1.73	209	0.008
1	10^{-2}	10^{-5}	2.79	368	0.008
1	0	0	0.77	101	0.008
1	0	10^{-4}	1.04	126	0.008
1	0	10^{-3}	1.16	141	0.008
1	0	10^{-2}	3.89	466	0.008
1	0	10^{-1}	0.97	118	0.008
1	0	10^{0}	1.14	138	0.008
1	0	10^{2}	2.97	357	0.008
1	0	10^{3}	1.66	200	0.008
1	0	10^{-5}	2.72	328	0.008
1	10^{-1}	0	1.46	177	0.008
1	10^{-1}	10^{-4}	0.96	116	0.008
1	10^{-1}	10^{-4}	1.37	180	0.008
1	10^{-1}	10^{-3}	2.18	286	0.008
1	10^{-1}	10^{-3}	1.83	220	0.008
1	10^{-1}	10^{-2}	1.49	196	0.008
1	10^{-1}	10^{-2}	1.78	216	0.008
1	10^{-1}	10^{-1}	1.40	170	0.008
1	10^{-1}	10^{-1}	1.92	253	0.008
1	10^{-1}	10^{0}	1.30	158	0.008
1	10^{-1}	10^{0}	1.22	159	0.008
1	10^{-1}	10^{2}	2.11	276	0.008
1	10^{-1}	10^{2}	1.41	171	0.008
1	10^{-1}	10^{3}	1.32	158	0.008
1	10^{-1}	10^{-5}	1.99	239	0.008
1	10^{-1}	10^{-5}	2.14	281	0.008
1	10^{0}	0	0.89	118	0.008
1	10^{0}	10^{-4}	0.90	120	0.008
1	10^{0}	10^{-3}	2.34	309	0.008
1	10^{0}	10^{-2}	2.40	317	0.008
1	10^{0}	10^{-1}	4.63	603	0.008
1	10^{0}	10^{0}	1.37	182	0.008
1	10^{0}	10^{2}	1.09	145	0.008
1	10^{0}	10^{3}	1.64	216	0.008
1	10^{0}	10^{-5}	3.19	418	0.008
1	10^{1}	10^{-1}	0.22	26	0.008
1	10^{2}	0	2.68	353	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{2}	10^{-4}	1.69	224	0.008
1	10^{2}	10^{-3}	1.95	257	0.008
1	10^{2}	10^{-2}	2.03	267	0.008
1	10^{2}	10^{-1}	1.62	214	0.008
1	10^{2}	10^{0}	1.36	180	0.008
1	10^{2}	10^{2}	2.73	358	0.008
1	10^{2}	10^{3}	1.16	154	0.008
1	10^{3}	0	1.90	250	0.008
1	10^{3}	10^{-4}	1.72	227	0.008
1	10^{3}	10^{-3}	0.86	115	0.008
1	10^{3}	10^{-2}	1.14	151	0.008
1	10^{3}	10^{-1}	1.46	193	0.008
1	10^{3}	10^{0}	1.01	134	0.008
1	10^{3}	10^{2}	2.29	302	0.008
1	10^{3}	10^{3}	2.40	318	0.008
1	10^{3}	10^{-5}	0.75	100	0.008
1	10^{-5}	0	7.80	922	0.008
1	10^{-5}	10^{-4}	1.47	178	0.008
1	10^{-5}	10^{-3}	2.54	156	0.016
1	10^{-5}	10^{-2}	0.89	108	0.008
1	10^{-5}	10^{-1}	0.93	113	0.008
1	10^{-5}	10^{0}	1.44	174	0.008
1	10^{-5}	10^{2}	2.99	358	0.008
1	10^{-5}	10^{3}	2.50	301	0.008
1	10^{-5}	10^{-5}	1.07	129	0.008
1	10^{-6}	0	1.25	151	0.008
1	10^{-6}	10^{-4}	0.99	119	0.008
1	10^{-6}	10^{-3}	1.63	197	0.008
1	10^{-6}	10^{-2}	1.51	183	0.008
1	10^{-6}	10^{-1}	1.27	153	0.008
1	10^{-6}	10^{0}	1.59	192	0.008
1	10^{-6}	10^{2}	4.21	505	0.008
1	10^{-6}	10^{3}	2.08	251	0.008
1	10^{-6}	10^{-5}	1.72	208	0.008
1	10^{-7}	0	5.45	652	0.008
1	10^{-7}	10^{-4}	4.23	506	0.008
1	10^{-7}	10^{-3}	1.83	221	0.008
1	10^{-7}	10^{-2}	1.41	170	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-7}	10^{-1}	1.41	170	0.008
1	10^{-7}	10^{0}	1.27	154	0.008
1	10^{-7}	10^{2}	4.64	552	0.008
1	10^{-7}	10^{3}	2.21	267	0.008
1	10^{-7}	10^{-5}	1.84	222	0.008

Table D.2: Training time metrics for all tested models with $n_{\text{train}} = 50$.

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
0	0	0	0.52	110	0.005
0	10^{-4}	0	2.77	347	0.008
0	10^{-4}	10^{-4}	2.02	256	0.008
0	10^{-4}	10^{-3}	2.49	312	0.008
0	10^{-4}	10^{-2}	1.98	250	0.008
0	10^{-4}	10^{-1}	1.32	167	0.008
0	10^{-4}	1	0.90	115	0.008
0	10^{-4}	10^{2}	3.70	465	0.008
0	10^{-4}	10^{3}	3.98	499	0.008
0	10^{-4}	10^{-5}	1.24	158	0.008
0	10^{-3}	0	2.27	287	0.008
0	10^{-3}	10^{-4}	1.14	145	0.008
0	10^{-3}	10^{-3}	2.34	295	0.008
0	10^{-3}	10^{-2}	1.68	212	0.008
0	10^{-3}	10^{-1}	1.73	216	0.008
0	10^{-3}	1	3.55	428	0.008
0	10^{-3}	10^{2}	4.52	535	0.008
0	10^{-3}	10^{3}	5.61	660	0.009
0	10^{-3}	10^{-5}	1.09	138	0.008
0	10^{-2}	0	2.79	330	0.008
0	10^{-2}	10^{-4}	0.94	113	0.008
0	10^{-2}	10^{-3}	1.30	155	0.008
0	10^{-2}	10^{-2}	1.29	154	0.008
0	10^{-2}	10^{-1}	1.73	206	0.008
0	10^{-2}	1	1.73	207	0.008
0	10^{-2}	10^{2}	4.23	503	0.008
0	10^{-2}	10^{3}	5.22	617	0.008
0	10^{-2}	10^{-5}	3.58	422	0.008
0	0	0	0.79	100	0.008
0	0	10^{-4}	2.20	276	0.008
0	0	10^{-3}	1.61	202	0.008
0	0	10^{-2}	2.17	273	0.008
0	0	10^{-1}	2.22	280	0.008
0	0	1	2.52	319	0.008
0	0	10^{2}	4.71	590	0.008
0	0	10^{3}	4.29	536	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
0	0	10^{-5}	1.95	246	0.008
0	10^{-1}	0	2.23	265	0.008
0	10^{-1}	10^{-4}	3.50	412	0.008
0	10^{-1}	10^{-3}	1.60	189	0.008
0	10^{-1}	10^{-2}	1.98	234	0.008
0	10^{-1}	10^{-1}	5.62	657	0.009
0	10^{-1}	1	1.56	187	0.008
0	10^{-1}	10^{2}	4.55	538	0.008
0	10^{-1}	10^{3}	8.40	985	0.009
0	10^{-1}	10^{-5}	0.92	110	0.008
0	1	0	0.91	109	0.008
0	1	10^{-4}	1.33	159	0.008
0	1	10^{-3}	1.24	148	0.008
0	1	10^{-2}	1.35	161	0.008
0	1	10^{-1}	1.96	233	0.008
0	1	1	4.15	493	0.008
0	1	10^{2}	3.76	445	0.008
0	1	10^{3}	4.45	524	0.008
0	1	10^{-5}	1.40	167	0.008
0	10^{2}	0	2.89	342	0.008
0	10^{2}	10^{-4}	4.18	493	0.008
0	10^{2}	10^{-3}	0.98	118	0.008
0	10^{2}	10^{-2}	3.74	442	0.008
0	10^{2}	10^{-1}	6.18	726	0.009
0	10^{2}	1	2.60	309	0.008
0	10^{2}	10^{2}	5.08	600	0.008
0	10^{2}	10^{3}	3.97	471	0.008
0	10^{2}	10^{-5}	1.93	230	0.008
0	10^{3}	0	1.22	145	0.008
0	10^{3}	10^{-4}	7.66	904	0.008
0	10^{3}	10^{-3}	4.38	519	0.008
0	10^{3}	10^{-2}	8.46	1000	0.008
0	10^{3}	10^{-1}	0.96	115	0.008
0	10^{3}	1	2.73	324	0.008
0	10^{3}	10^{2}	4.32	512	0.008
0	10^{3}	10^{3}	8.51	1000	0.009
0	10^{3}	10^{-5}	8.55	1000	0.009
0	10^{-5}	0	0.79	100	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
0	10^{-5}	10^{-4}	1.12	142	0.008
0	10^{-5}	10^{-3}	1.39	175	0.008
0	10^{-5}	10^{-2}	8.14	1000	0.008
0	10^{-5}	10^{-1}	1.37	172	0.008
0	10^{-5}	1	2.34	295	0.008
0	10^{-5}	10^{2}	4.05	508	0.008
0	10^{-5}	10^{3}	4.29	537	0.008
0	10^{-5}	10^{-5}	1.39	175	0.008
1	10^{-4}	0	8.11	1000	0.008
1	10^{-4}	10^{-4}	0.87	110	0.008
1	10^{-4}	10^{-3}	8.11	1000	0.008
1	10^{-4}	10^{-2}	2.76	345	0.008
1	10^{-4}	10^{-1}	1.77	222	0.008
1	10^{-4}	1	1.69	212	0.008
1	10^{-4}	10^{2}	4.65	575	0.008
1	10^{-4}	10^{3}	4.40	549	0.008
1	10^{-4}	10^{-5}	1.10	139	0.008
1	10^{-3}	0	2.64	329	0.008
1	10^{-3}	10^{-4}	5.23	647	0.008
1	10^{-3}	10^{-3}	2.07	260	0.008
1	10^{-3}	10^{-2}	8.12	1000	0.008
1	10^{-3}	10^{-1}	3.18	396	0.008
1	10^{-3}	1	1.30	163	0.008
1	10^{-3}	10^{2}	4.29	534	0.008
1	10^{-3}	10^{3}	2.97	372	0.008
1	10^{-3}	10^{-5}	0.92	116	0.008
1	10^{-2}	0	1.93	241	0.008
1	10^{-2}	10^{-4}	1.06	133	0.008
1	10^{-2}	10^{-3}	2.37	295	0.008
1	10^{-2}	10^{-2}	6.67	819	0.008
1	10^{-2}	10^{-1}	8.11	1000	0.008
1	10^{-2}	1	2.03	255	0.008
1	10^{-2}	10^{2}	4.38	548	0.008
1	10^{-2}	10^{3}	3.64	457	0.008
1	10^{-2}	10^{-5}	8.20	1000	0.008
1	0	0	0.90	111	0.008
1	0	10^{-4}	5.65	694	0.008
1	0	10^{-3}	3.76	466	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	0	10^{-2}	8.17	1000	0.008
1	0	10^{-1}	7.81	958	0.008
1	0	1	2.65	328	0.008
1	0	10^{2}	8.09	997	0.008
1	0	10^{3}	4.12	513	0.008
1	0	10^{-5}	3.89	481	0.008
1	10^{-1}	0	6.86	850	0.008
1	10^{-1}	10^{-4}	2.54	320	0.008
1	10^{-1}	10^{-3}	0.99	126	0.008
1	10^{-1}	10^{-2}	8.14	1000	0.008
1	10^{-1}	10^{-1}	3.29	413	0.008
1	10^{-1}	1	0.95	121	0.008
1	10^{-1}	10^{2}	3.96	495	0.008
1	10^{-1}	10^{3}	3.92	489	0.008
1	10^{-1}	10^{-5}	8.13	1000	0.008
1	1	0	4.77	594	0.008
1	1	10^{-4}	4.62	576	0.008
1	1	10^{-3}	8.12	1000	0.008
1	1	10^{-2}	1.49	187	0.008
1	1	10^{-1}	1.98	249	0.008
1	1	1	4.35	541	0.008
1	1	10^{2}	4.45	554	0.008
1	1	10^{3}	8.12	1000	0.008
1	1	10^{-5}	2.34	294	0.008
1	10^{2}	0	5.18	646	0.008
1	10^{2}	10^{-4}	2.66	336	0.008
1	10^{2}	10^{-3}	4.92	616	0.008
1	10^{2}	10^{-2}	4.54	565	0.008
1	10^{2}	10^{-1}	3.13	390	0.008
1	10^{2}	1	2.23	280	0.008
1	10^{2}	10^{2}	3.17	399	0.008
1	10^{2}	10^{3}	8.07	1000	0.008
1	10^{2}	10^{-5}	0.90	114	0.008
1	10^{3}	0	7.98	1000	0.008
1	10^{3}	10^{-4}	8.00	1000	0.008
1	10^{3}	10^{-3}	8.05	1000	0.008
1	10^{3}	10^{-2}	8.09	1000	0.008
1	10^{3}	10^{-1}	8.14	1000	0.008

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{3}	1	5.65	704	0.008
1	10^{3}	10^{2}	6.83	852	0.008
1	10^{3}	10^{3}	8.07	1000	0.008
1	10^{3}	10^{-5}	8.00	1000	0.008
1	10^{-5}	0	1.69	212	0.008
1	10^{-5}	10^{-4}	8.21	1000	0.008
1	10^{-5}	10^{-3}	5.99	737	0.008
1	10^{-5}	10^{-2}	1.56	195	0.008
1	10^{-5}	10^{-1}	8.02	980	0.008
1	10^{-5}	1	1.91	238	0.008
1	10^{-5}	10^{2}	3.50	435	0.008
1	10^{-5}	10^{3}	4.05	505	0.008
1	10^{-5}	10^{-5}	5.21	643	0.008

Table D.3: Training time metrics for all tested models with $n_{\text{train}} = 200$.

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	0	0	1.80	124	0.015
0	10^{-4}	0	2.61	113	0.023
0	10^{-4}	10^{-4}	6.10	264	0.023
0	10^{-4}	10^{-3}	6.42	278	0.023
0	10^{-4}	10^{-2}	4.27	185	0.023
0	10^{-4}	10^{-1}	5.55	240	0.023
0	10^{-4}	1	7.16	309	0.023
0	10^{-4}	10^{2}	4.53	196	0.023
0	10^{-4}	10^{3}	4.99	216	0.023
0	10^{-4}	10^{-5}	2.33	101	0.023
0	10^{-3}	0	10.22	442	0.023
0	0	0	2.30	100	0.023
0	0	10^{-4}	5.26	227	0.023
0	0	10^{-3}	5.17	223	0.023
0	0	10^{-2}	5.12	221	0.023
0	0	10^{-1}	5.03	217	0.023
0	0	1	5.11	221	0.023
0	0	10^{2}	6.00	259	0.023
0	0	10^{3}	6.44	278	0.023
0	0	10^{-5}	9.72	420	0.023
0	10^{-5}	0	7.00	302	0.023
0	10^{-5}	10^{-4}	5.55	240	0.023
0	10^{-5}	10^{-3}	4.30	185	0.023
0	10^{-5}	10^{-2}	5.89	254	0.023
0	10^{-5}	10^{-1}	6.07	262	0.023
0	10^{-5}	1	4.65	201	0.023
0	10^{-5}	10^{2}	5.56	240	0.023
0	10^{-5}	10^{3}	4.71	204	0.023
0	10^{-5}	10^{-5}	6.24	270	0.023
1	10^{-4}	0	11.99	512	0.023
1	10^{-4}	10^{-4}	11.89	508	0.023
1	10^{-4}	10^{-3}	19.52	831	0.023
1	10^{-4}	10^{-2}	10.77	461	0.023
1	10^{-4}	10^{-1}	8.75	374	0.023
1	10^{-4}	1	7.80	333	0.023
1	10^{-4}	10^{2}	6.11	261	0.023

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-4}	10^{3}	4.06	174	0.023
1	10^{-4}	10^{-5}	23.51	1000	0.024
1	10^{-3}	0	10.45	447	0.023
1	10^{-3}	10^{-4}	23.43	1000	0.023
1	10^{-3}	10^{-3}	7.83	335	0.023
1	10^{-3}	10^{-2}	12.66	540	0.023
1	10^{-3}	10^{-1}	6.83	292	0.023
1	10^{-3}	1	10.03	429	0.023
1	10^{-3}	10^{2}	4.97	213	0.023
1	10^{-3}	10^{3}	4.69	201	0.023
1	10^{-3}	10^{-5}	17.41	743	0.023
1	10^{-2}	0	23.44	1000	0.023
1	10^{-2}	10^{-4}	19.01	810	0.023
1	10^{-2}	10^{-3}	12.21	521	0.023
1	10^{-2}	10^{-2}	8.70	372	0.023
1	10^{-2}	10^{-1}	10.41	445	0.023
1	10^{-2}	1	10.06	431	0.023
1	10^{-2}	10^{2}	4.47	192	0.023
1	10^{-2}	10^{3}	5.76	247	0.023
1	10^{-2}	10^{-5}	10.01	427	0.023
1	0	0	4.35	185	0.023
1	0	10^{-4}	11.35	483	0.024
1	0	10^{-3}	11.94	508	0.023
1	0	10^{-2}	23.60	1000	0.024
1	0	10^{-1}	7.98	340	0.023
1	0	1	10.85	460	0.024
1	0	10^{2}	6.00	256	0.023
1	0	10^{3}	5.46	233	0.023
1	0	10^{-5}	13.97	592	0.024
1	10^{-1}	0	23.48	1000	0.023
1	10^{-1}	10^{-4}	11.07	473	0.023
1	10^{-1}	10^{-3}	23.47	1000	0.023
1	10^{-1}	10^{-2}	13.34	569	0.023
1	10^{-1}	10^{-1}	6.90	295	0.023
1	10^{-1}	1	6.68	286	0.023
1	10^{-1}	10^{2}	6.34	272	0.023
1	10^{-1}	10^{3}	4.76	204	0.023
1	10^{-1}	10^{-5}	12.87	550	0.023

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	1	0	11.03	471	0.023
1	1	10^{-4}	9.18	393	0.023
1	1	10^{-3}	12.06	515	0.023
1	1	10^{-2}	7.15	306	0.023
1	1	10^{-1}	10.48	448	0.023
1	1	1	6.61	283	0.023
1	1	10^{2}	4.74	203	0.023
1	1	10^{3}	6.74	288	0.023
1	1	10^{-5}	17.30	739	0.023
1	10^{2}	0	11.51	492	0.023
1	10^{2}	10^{-4}	12.20	521	0.023
1	10^{2}	10^{-3}	11.68	499	0.023
1	10^{2}	10^{-2}	8.90	381	0.023
1	10^{2}	10^{-1}	10.38	444	0.023
1	10^{2}	1	6.59	282	0.023
1	10^{2}	10^{2}	7.72	330	0.023
1	10^{2}	10^{3}	4.79	205	0.023
1	10^{2}	10^{-5}	13.84	590	0.023
1	10^{3}	0	14.02	599	0.023
1	10^{3}	10^{-4}	17.68	754	0.023
1	10^{3}	10^{-3}	14.39	614	0.023
1	10^{3}	10^{-2}	12.14	519	0.023
1	10^{3}	10^{-1}	2.26	97	0.023
1	10^{3}	1	0.47	18	0.026
1	10^{3}	1	14.82	640	0.023
1	10^{3}	10^{2}	6.06	262	0.023
1	10^{3}	10^{3}	6.69	290	0.023
1	10^{3}	10^{-5}	13.82	590	0.023
1	10^{-5}	0	23.62	1000	0.024
1	10^{-5}	10^{-4}	20.19	860	0.023
1	10^{-5}	10^{-3}	23.52	1000	0.024
1	10^{-5}	10^{-2}	10.80	460	0.023
1	10^{-5}	10^{-1}	8.73	373	0.023
1	10^{-5}	1	13.87	592	0.023
1	10^{-5}	10^{2}	6.57	281	0.023
1	10^{-5}	10^{3}	5.06	216	0.023
1	10^{-5}	10^{-5}	10.72	456	0.024
1	10^{-6}	0	15.37	619	0.025

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-6}	10^{-4}	24.88	1000	0.025
1	10^{-6}	10^{-3}	16.56	666	0.025
1	10^{-6}	10^{-2}	12.99	522	0.025
1	10^{-6}	10^{-1}	11.51	463	0.025
1	10^{-6}	1	7.24	292	0.025
1	10^{-6}	10^{2}	5.60	226	0.025
1	10^{-6}	10^{3}	6.32	255	0.025
1	10^{-6}	10^{-5}	9.50	383	0.025
1	10^{-7}	0	9.90	397	0.025
1	10^{-7}	10^{-4}	12.48	501	0.025
1	10^{-7}	10^{-3}	13.17	529	0.025
1	10^{-7}	10^{-2}	9.34	375	0.025
1	10^{-7}	10^{-1}	7.56	305	0.025
1	10^{-7}	1	13.49	542	0.025
1	10^{-7}	10^{2}	5.38	217	0.025
1	10^{-7}	10^{3}	5.60	226	0.025
1	10^{-7}	10^{-5}	11.14	446	0.025

Table D.4: Training time metrics for all tested models with $n_{\text{train}} = 400$.

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	0	0	9.51	385	0.025
1	10^{-4}	0	33.39	817	0.041
1	10^{-4}	10^{-4}	36.02	882	0.041
1	10^{-4}	10^{-3}	31.88	777	0.041
1	10^{-4}	10^{-2}	31.79	776	0.041
1	10^{-3}	0	40.91	1000	0.041
1	10^{-3}	10^{-4}	21.48	526	0.041
1	10^{-3}	10^{-3}	6.84	168	0.041
1	10^{-3}	10^{-2}	16.95	415	0.041
1	10^{-2}	0	40.95	1000	0.041
1	10^{-2}	10^{-4}	40.92	1000	0.041
1	10^{-2}	10^{-3}	40.96	1000	0.041
1	10^{-2}	10^{-2}	12.29	300	0.041
1	0	0	42.41	1000	0.042
1	0	10^{-4}	42.06	1000	0.042
1	0	10^{-3}	42.24	1000	0.042
1	0	10^{-2}	22.88	543	0.042
1	10^{-1}	0	40.96	1000	0.041
1	10^{-1}	10^{-4}	40.93	1000	0.041
1	10^{-1}	10^{-3}	7.51	184	0.041
1	10^{-1}	10^{-2}	14.47	354	0.041
1	1	0	40.97	1000	0.041
1	1	10^{-4}	14.19	346	0.041
1	1	10^{-3}	41.24	1000	0.041
1	1	10^{-2}	14.82	361	0.041
1	10^{-5}	0	42.29	1000	0.042
1	10^{-5}	10^{-4}	42.18	1000	0.042
1	10^{-5}	10^{-3}	7.21	176	0.041
1	10^{-5}	10^{-2}	12.51	306	0.041
1	10^{-6}	0	28.68	677	0.042
1	10^{-6}	10^{-4}	42.41	1000	0.042
1	10^{-6}	10^{-3}	42.32	1000	0.042
1	10^{-6}	10^{-2}	22.96	544	0.042
1	10^{-7}	0	42.18	1000	0.042
1	10^{-7}	10^{-4}	42.37	1000	0.042
1	10^{-7}	10^{-3}	22.78	538	0.042

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-7}	10^{-2}	14.34	339	0.042

Table D.5: Training time metrics for all tested models with $n_{\text{train}} = 500$.

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	0	0	12.82	432	0.030
1	10^{-4}	0	46.06	1000	0.046
1	10^{-4}	10^{-4}	44.05	957	0.046
1	10^{-4}	10^{-3}	27.83	605	0.046
1	10^{-4}	10^{-2}	16.82	365	0.046
1	10^{-3}	0	38.78	842	0.046
1	10^{-3}	10^{-4}	46.10	1000	0.046
1	10^{-3}	10^{-3}	46.08	1000	0.046
1	10^{-3}	10^{-2}	20.31	441	0.046
1	10^{-2}	0	46.07	1000	0.046
1	10^{-2}	10^{-4}	45.99	1000	0.046
1	10^{-2}	10^{-3}	41.02	892	0.046
1	10^{-2}	10^{-2}	16.63	362	0.046
1	0	0	32.25	700	0.046
1	0	10^{-4}	23.72	515	0.046
1	0	10^{-3}	37.08	805	0.046
1	0	10^{-2}	13.89	301	0.046
1	10^{-1}	0	46.01	1000	0.046
1	10^{-1}	10^{-4}	46.04	1000	0.046
1	10^{-1}	10^{-3}	46.00	1000	0.046
1	10^{-1}	10^{-2}	20.39	444	0.046
1	1	0	46.02	1000	0.046
1	1	10^{-4}	33.39	727	0.046
1	1	10^{-3}	31.12	677	0.046
1	1	10^{-2}	27.30	581	0.047
1	1	1	0.19	4	0.048
1	10	0	23.39	498	0.047
1	10	10^{-4}	27.85	593	0.047
1	10	10^{-3}	47.34	1000	0.047
1	10	10^{-2}	15.69	330	0.048
1	100	0	16.74	352	0.048
1	100	10^{-4}	14.83	312	0.048
1	100	10^{-3}	18.24	384	0.048
1	100	10^{-2}	14.56	305	0.048
1	10^{-5}	0	46.15	1000	0.046
1	10^{-5}	10^{-4}	28.53	620	0.046

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-5}	10^{-3}	32.64	709	0.046
1	10^{-5}	10^{-2}	18.36	398	0.046
1	10^{-6}	0	46.14	1000	0.046
1	10^{-6}	10^{-4}	22.25	483	0.046
1	10^{-6}	10^{-3}	46.08	1000	0.046
1	10^{-6}	10^{-2}	11.72	255	0.046
1	10^{-7}	0	46.14	1000	0.046
1	10^{-7}	10^{-4}	46.13	1000	0.046
1	10^{-7}	10^{-3}	7.90	172	0.046
1	10^{-7}	10^{-2}	17.21	374	0.046

Table D.6: Training time metrics for all tested models with $n_{\text{train}} = 1000$.

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	0	0	15.69	263	0.060
1	10^{-4}	0	97.56	1000	0.098
1	10^{-4}	10^{-5}	93.86	1000	0.094
1	10^{-4}	10^{-4}	91.17	1000	0.091
1	10^{-4}	10^{-3}	91.28	1000	0.091
1	10^{-4}	10^{-2}	45.61	500	0.091
1	10^{-4}	10^{-1}	13.72	151	0.091
1	10^{-4}	1	52.10	534	0.098
1	10^{-3}	0	31.27	336	0.093
1	10^{-3}	10^{-5}	91.23	1000	0.091
1	10^{-3}	10^{-4}	91.15	1000	0.091
1	10^{-3}	10^{-3}	91.13	1000	0.091
1	10^{-3}	10^{-2}	73.31	805	0.091
1	10^{-3}	10^{-1}	26.20	288	0.091
1	10^{-3}	1	54.25	574	0.095
1	10^{-2}	0	93.00	1000	0.093
1	10^{-2}	10^{-5}	91.16	1000	0.091
1	10^{-2}	10^{-4}	91.01	1000	0.091
1	10^{-2}	10^{-3}	81.03	889	0.091
1	10^{-2}	10^{-2}	58.28	639	0.091
1	10^{-2}	10^{-1}	29.10	319	0.091
1	10^{-2}	1	27.30	289	0.094
1	0	0	14.83	152	0.098
1	0	10^{-5}	17.71	181	0.098
1	0	10^{-4}	97.79	1000	0.098
1	0	10^{-3}	97.56	1000	0.098
1	0	10^{-2}	70.95	728	0.097
1	0	10^{-1}	39.94	410	0.097
1	0	1	27.66	283	0.098
1	0	10^{2}	9.98	102	0.098
1	0	10^{3}	21.36	218	0.098
1	10^{-1}	0	93.09	1000	0.093
1	10^{-1}	10^{-5}	91.45	1000	0.091
1	10^{-1}	10^{-4}	91.35	1000	0.091
1	10^{-1}	10^{-3}	91.38	1000	0.091
1	10^{-1}	10^{-2}	42.97	472	0.091

λ_{MSE}	λ_{phy}	λ_{bnd}	**Total [min]**	**Epochs**	**Time/epoch [min]**
1	10^{-1}	10^{-1}	51.12	561	0.091
1	10^{-1}	1	53.89	578	0.093
1	1	10^{-5}	14.45	149	0.097
1	1	10^{-3}	78.72	813	0.097
1	1	10^{-2}	26.79	274	0.098
1	1	10^{-1}	85.44	873	0.098
1	1	10^{2}	20.84	213	0.098
1	1	10^{3}	20.31	208	0.098
1	10^{-5}	0	28.53	304	0.094
1	10^{-5}	10^{-4}	97.47	1000	0.097
1	10^{-5}	10^{-3}	93.55	960	0.097
1	10^{-5}	10^{-2}	42.98	440	0.098
1	10^{-5}	10^{-1}	27.71	284	0.098
1	10^{-5}	1	33.30	342	0.097
1	10^{-5}	10^{-5}	97.75	1000	0.098