Physics-Informed Neural Networks for Multiphysics Coupling: Application to Conjugate Heat Transfer

Guillaume Coulaud, Régis Duvigneau

Physics-Informed Neural Networks for Multiphysics Coupling: Application to Conjugate Heat Transfer

Guillaume Coulaud, Régis Duvigneau

RESEARCH REPORT
N° 9520
October 2023
Project-Team Acumes

ISSN 0249-6399 ISRN INRIA/RR--9520--FR+ENG

Physics-Informed Neural Networks for Multiphysics Coupling: Application to Conjugate Heat Transfer

Guillaume Coulaud*, Régis Duvigneau*

Project-Team Acumes

Research Report n° 9520 — October 2023 — 49 pages

Abstract: Physics-Informed Neural Networks (PINNs) have emerged as a promising paradigm for modeling complex physical phenomena, offering the potential to handle diverse scenarios to simulate coupled systems. This is a supervised or unsupervised deep learning approach that aims at learning physical laws described by partial differential equations. This report presents an exploration of PINNs through three distinct test cases: heat transfer, and conjugate heat transfer, with forced and natural convection. The investigations reveal PINNs' proficiency in accommodating parameterized resolution, addressing piece-wise constant conditions, and enabling multiphysics coupling. Despite their versatility, challenges emerged, including difficulties in achieving high accuracy, error propagation near singularities, and limitations in scenarios with high Rayleigh values.

Key-words: Physics-Informed Neural Networks, Multiphysics Coupling, Conjugate Heat Transfer, Deep Learning, Partial differential equation.

* Université Côte d'Azur, Inria, CNRS, LJAD

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Réseaux de neurones informés par la physique pour le couplage multiphysique : application au transfert de chaleur conjugué

Résumé : Les réseaux neuronaux informés par la physique, également connus sous le nom de Physics-Informed Neural Networks (PINNs), se révèlent être une approche prometteuse pour la modélisation de phénomènes physiques complexes, permettant de mettre en oeuvre divers scenarios pour simuler les problèmes couplés. C'est une approche d'apprentissage profond, supervisé ou non, qui vise à apprendre des lois physiques décrites par des équations aux dérivées partielles. Ce rapport présente une exploration des potentialités des PINNs à travers trois cas distincts : le transfert de chaleur entre deux solides, le transfert de chaleur conjugué avec la convection forcée et naturelle. Les résultats de ces études mettent en évidence la capacité des PINNs à s'adapter à des problèmes paramétriques, à prendre en compte des conditions constantes par morceaux, et à permettre le couplage de plusieurs phénomènes physiques. Cependant, malgré leur polyvalence, des défis subsistent, notamment la difficulté à atteindre une précision élevée, la propagation d'erreurs près des singularités, et des limitations dans les scénarios impliquant des valeurs de Rayleigh élevées.

Mots-clés : réseaux de neurones informés par la physique, couplage multiphysique, transfert de chaleur conjugué, apprentissage profond, équation aux dérivées partielles.

Contents

1 Introduction

1.1 Motivation

Neural networks (NN), and especially deep neural networks (DNN), have proven to be highly effective tools for machine learning tasks - such as recognition problems applied to image, text, or speech recognition. With such advances, it is only natural that deep learning methods are explored in the context of scientific computing, leading to scientific machine learning [1]. These techniques are usually data-dependent, requiring large amounts of data to achieve good results. However, it is not always possible to obtain such data as it would involve conducting costly experiments. Moreover, data-driven methods do not take into account known knowledge such as physical laws. Such physics-based learning can benefit from mathematical models such as Partial Differential Equations (PDEs). In this context Physics-Informed Neural Networks (PINNs) [2, 3] are introduced.

1.2 What Are Physics-Informed Neural Networks?

Physics-informed neural networks are neural networks designed to solve PDEs. To approximate the PDE solution PINNs are trained to minimize a loss function encapsulating the initial and boundary conditions, as well as the PDE residual evaluated at given points, called residual or collocation points. The loss includes derivatives computed through automatic differentiation. This approach can be considered as an unsupervised strategy, as it does not require labeled data.

The method's advantage lies in its mesh-free aspect, as the network can provide the PDE solution at any given point on the domain. Another benefit is its simplicity in solving both forward and inverse problems, as the loss function must be specified. Additionally, the code used to solve the forward problem can be adapted to solve the inverse problem with minimal modifications. Furthermore, PINNs can also tackle parametric resolution with the exact same code.

However, PINNs suffer from two major limitations, the first being the accuracy of the solution. In fact, the absolute error typically falls between 10^{-3} and 10^{-5}. The second issue is the training cost as a lot of epochs, tens to hundreds of thousands of epochs are needed while using automatic differentiation on each of them. PINNs are relevant in some specific cases, e.g. where classical simulations are tedious or costly.

1.3 Purpose of this Study

This study aims to investigate multiphysics coupling, such as natural convection in a heated cavity, and to examine novel solutions utilizing PINNs. This study introduces the motivation for seeking solutions and provides an understanding of Physics-Informed Neural Networks (PINNs). In a world where complex physical phenomena often interact and influence each other, accurately modeling and accounting for these interactions is critical. This study aims to investigate the capabilities of PINNs in addressing multiphysics problems.

The study is organized as follows: Chapter 2 discusses PINNs in-depth, covering their construction, activation functions, and training methods. It highlights the optimizer's significance and various sampling methods. The chapter also expands on the use of PINNs to manage multiphysics coupling.

Chapter 3 introduces various test cases, which act as benchmarks to assess the PINN technique's efficiency in multiphysics problem-solving. These cases involve simulations of heat transfer, ranging from straightforward situations involving a single solid to more intricate scenarios with various materials and parameterized resolutions. Additionally, this study investigates the application

of Physics-Informed Neural Networks (PINNs) in resolving conjugate heat transfer problems, including both forced and natural convection scenarios.

1.4 Physics-informed Neural Networks

Physics-Informed Neural Networks are designed to solve differential equations expressed in the following general form,

$$\begin{cases} \mathcal{F}(u(z);\gamma) = f(z) & z \in \Omega \\ \mathcal{B}(u(z)) = g(z) & z \in \partial\Omega \end{cases}$$

where z is the space-time coordinate vector, $z := [x_1, ..., x_{d-1}; t]$, u is the unknown solution, γ the parameters related to the physics, f the function identifying the data of the problem, $\mathcal{F}$ is the non-linear differential operator, $\mathcal{B}$ is the operator indicating the initial or boundary conditions on the domain. Different types of boundaries, such as Dirichlet, Neumann, or Robin conditions can be considered. For forward problems, the goal is to find the function u for every specified parameter γ. Meanwhile for the inverse problems, γ must also be determined from the data

PINNs leverage the ability of neural networks' potential to approximate functions effectively. The theorem formulating this phenomenon is known as the universal approximation theorem [4]. It implies that neural networks can represent a wide variety of functions. However, there is no assurance for the possibility of constructing such a network, or that training will converge to the correct weights.

Therefore, the goal of PINNs is to construct a neural network with parameters (weights) θ to approximate $u(z)$. This NN is trained to predict u for the space-time coordinates given as input. From the output u it is possible to compute, with automatic differentiation, $\mathcal{F}(u(z))$ and $\mathcal{B}(u(z))$ to evaluate and to compute the mean square error (MSE) of the different residuals of the equations. The MSEs are summed to build the loss function to minimize. If observation data (also called labeled data) are available, they can also be introduced into the loss. Otherwise, it is strictly an unsupervised problem with no labeled data. The optimization problem can be written as follows,

$$\theta^* = \arg\min_{\theta} \{\omega_{\mathcal{F}}\mathcal{L}_{\mathcal{F}}(\theta) + \omega_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\theta) + \omega_d\mathcal{L}_{data}(\theta)\}$$

where $\omega_{\{\mathcal{F},\mathcal{B},d\}}$ are the weights associated with the different terms of the loss.

The research on PINNs covers various aspects. Some focus on the NN aspect with different architectures and activation functions, while others concentrate on optimization problems by introducing techniques to compute the loss weights. Other studies have emphasized the importance of sampling and the various strategies used.

Figure 1 taken from [5] summarizes the construction and the training of a PINN. They are made from different residual terms from differential equations. These residuals form the loss functions. The neural network inputs are transformed into a field u. The network is defined by its weights/parameters θ. The second aspect is the physics-informed network, which uses the provided equations to compute the derivative of the output field u. Finally, a feedback mechanism utilizing an optimizer is applied to minimize loss with all of the residuals, according to a specific learning rate, in order to obtain the NN parameters θ.

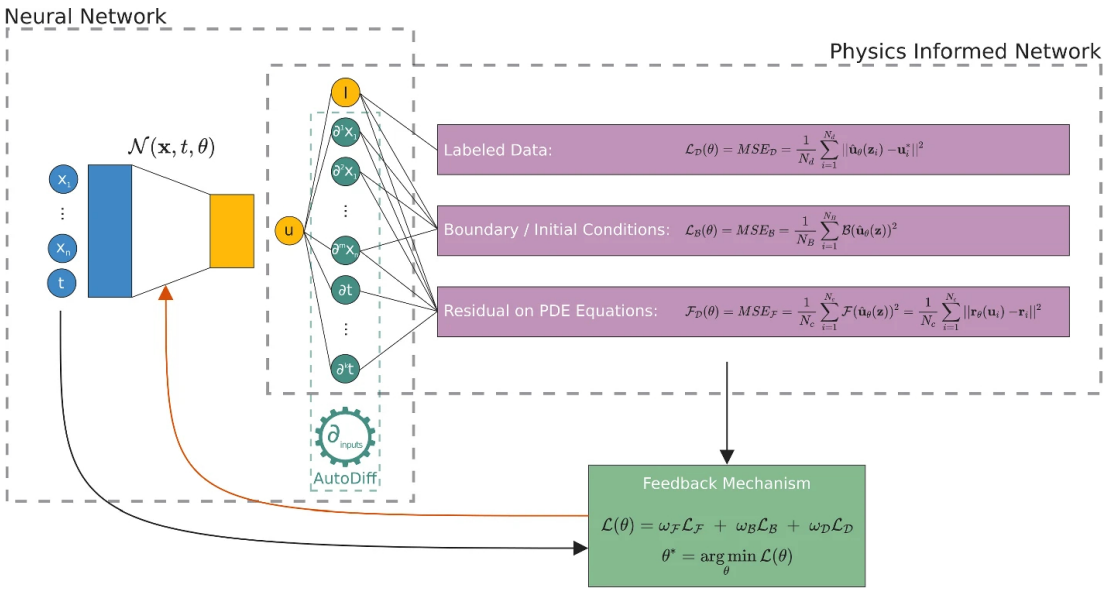


Figure 1: Physics-informed neural networks building blocks [5].

PINNs are by designed "mesh-free" approach as they do not rely on a grid or a mesh structure to discretize the computational domain. Contrary to traditional numerical methods, such as finite difference or finite element methods, the domain is divided into a grid or mesh of discrete points or elements. These methods require careful mesh generation and refinement, which can be complex and time-consuming, especially for irregular or complex geometries.

In contrast, PINNs operate directly with the continuous domain and circumvent the need for a pre-defined mesh. Instead, they learn the solution of the PDEs and the underlying physics from the residual points. Allowing to enforce the governing equations and boundary conditions in a way that respects the continuous nature of the problem. Several sampling strategies are possible to select these residual points.

By being mesh-free, PINNs simplify the process of setting up and solving complex physical problems. It also provides "on-demand" solution computation for given points.

Different architecture Since PINNs are neural networks there is a wide range of possible architectures [5].

The most commonly used is the fully connected NN or multilayer perceptron (FCNN, FNN, or MLP).

Some work has explored the use of convolutional NN (CNN), however, CNN is not a mesh-free approach because it needs a uniform grid to perform convolution. Nevertheless, CNN has the advantage of being a very well-studied model, with many available techniques like residual blocks, auto-encoder, and Fourier operator [6].

Recurrent NN (RNN) can be used to solve time-dependent tasks. It can be enhanced with long short-term memory (LSTM) to tackle challenges requiring long-term memories.

Multiple networks can also be used, allowing domain decomposition approach [7, 8]. Lu et al. proposed the DeepONet model [9, 10] using two networks, one to encode the input space and one to encode the domain of the output function.

Additionally, Wang et al. proposed an architecture [11] inspired by transformers, aimed at capturing multiplicative interactions between input dimensions, while benefiting from residual

connections.

Losses weight balancing As the loss is a combination of multiple terms, it can potentially impact the training capacity and result in gradient pathologies [11]. To address this issue, Wang et al. proposed a loss reweighting technique, the learning rate annealing [11] to alleviate this problem. Let $\mathcal{L}_r$ be the loss of the PDE residuals, and $\mathcal{L}_i$ be the other loss terms. λ_i is the weight associated with the i-th loss, and α is a momentum hyperparameter. We want to minimize

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta).$$

The learning rate annealing is defined as the following,

1. Compute $\hat{\lambda}_i$
$$\hat{\lambda}_i = \frac{\max_{\theta_n} \{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{|\overline{\nabla_\theta \mathcal{L}_i(\theta_n)}|}$$
with θ_n the parameters at the nth iteration, $|\cdot|$ the element-wise absolute value, $|\overline{\nabla_\theta \mathcal{L}_i(\theta_n)}|$ the mean of $|\nabla_\theta \mathcal{L}_i(\theta_n)|$

2. Update the weight λ_i using a relaxation
$$\lambda_i = (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i$$

3. Update the parameters via gradient descent
$$\theta_{n+1} = \theta_n - \eta\nabla_\theta \mathcal{L}_r(\theta) + \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta)$$

The generic recommended hyperparameter values are $\eta = 10^{-3}$ and $\alpha = 0.1$

Another approach called Inverse-Dirichlet Weighting [12] proposed to use the gradient variance information to compute $\hat{\lambda}_i$

$$\hat{\lambda}_i = \frac{\max_{\theta_n} \{\mathrm{Var}\left[\nabla_\theta \mathcal{L}_k(\theta_n)\right]\}}{\mathrm{Var}\left[\nabla_\theta \mathcal{L}_i(\theta_n)\right]}$$

Other techniques are possible, Wang et al. [13] introduce two methods, one based on the norm of the gradients, and another based on the neural tangent kernel [14] to balance the contributions of the different loss terms.

The optimizer choice Taylor et al. [15] investigated the use of different optimizers to solve the Burgers equation with a PINN.

Adam Adam is a first-order gradient-based optimization algorithm of stochastic loss functions, it is based on adaptive estimates of lower-order moments. Adam is an optimizer usually used in deep learning applications as it is well suited for problems with large data or neural networks. It is a robust and efficient algorithm able to train neural networks with millions of parameters. However, to exploit the algorithm's full potential some tuning is necessary. The two main components of the tuning are the choice of the learning rate and the momentum options. The learning rate can be constant, but it will require performing several runs to find a correct value. It can also be adaptive with the use of a scheduler to update the learning rate with a given strategy, the strategy choice and its parameters also require some tuning.

BFGS The BFGS algorithm is a second-order iterative gradient method that approximates the Hessian matrix of the loss function using gradient evaluations. As this method implies to store the Hessian matrix, the memory usage grows as the square of the number of parameters. This memory limitation makes the BFGS optimizer unsuitable for neural networks with a lot of parameters. Some hyperparameter tuning may be needed, especially for the line search criteria.

L-BFGS The L-BFGS algorithm is the extension of the BFGS which addresses the problem of memory usage. To estimate the Hessian, L-BFGS only stores a few vectors that represent the approximation implicitly. It can be extended to take into account simple box constraints with the L-BFGS-B algorithm.

Levenberg-Marquardt The Levenberg-Marquardt (LM) is a method used to solve non-linear least squares problems. It is a combination of both first-order and second-order methods. The algorithm interpolates between the Gauss–Newton algorithm (GNA) and the method of gradient descent. This method is both computationally and memory-intensive. Similarly to the BFGS algorithm, it is restricted to smaller neural networks. This algorithm does not require hyperparameter tuning.

As the BFGS-based and LM algorithms are second-order methods, thus less stable than first-order methods but they exhibit greater efficiency under favorable conditions. Additionally, Krishnapriyan et al. (2021) [16] show that the loss landscapes are difficult to optimize for the PINN. In this context, solely using a second-order method that requires convergence in a convex region may not be suitable for training a PINN. For this reason, it is often necessary to perform multiple Adam iterations prior to utilizing these algorithms. He et al. [17] suggest that in cases with few training data and residual points, the L-BFGS-B performs better than the Adam algorithm, allowing a faster convergence rate and reduced computational cost. They also argue that the Adam optimizer necessitates much deeper models with significantly more hidden units containing up to approximately 26 times more parameters, to reach prediction quality comparable to that achieved by the LM optimizer.

1.5 A Range of Activation Functions

A well-known issue in neural networks is the choice of activation function, as it has a significant impact on convergence. One constraint on PINNs is that the output must be differentiable in order to compute the loss terms. This condition limits the choice of loss function which must be at least smooth enough. Thus commonly applied activation functions such as ReLU and leaky ReLU (LReLU) seem less suitable, with $\text{ReLU}(x) \coloneqq max(0, x) = x_+$ and $\text{LReLU} \coloneqq x_+ + \alpha x_-$ where $x_- = min(0, x)$. For the PINNs, the most commonly employed functions are hyperbolic tangent and logistic Sigmoid. But others can be used, such as the exponential linear unit (ELU) which is a smoother version of ReLU, $\text{ELU} \coloneqq x_+ + (\alpha \exp(x - 1))_-$.

Recent research [18, 5] has shown that the use of adaptive activation functions can improve the training convergence, these functions are parameterized functions for which one parameter is trainable. A first adaptive function is $\text{Swish}(x) \coloneqq x \cdot \text{Sigmoid}(\beta x)$ where β is a trainable parameter. Jagtap et al. [18] proposed another function that introduces a scalable parameter na, where n is a predefined scaling factor and a is a trainable parameter. The goal of this function is to increase the slope of activation to avoid the apparition of vanishing gradients and to increase the training speed. The parameter n acts as a slope of activation function. Moreover, this function can be defined layer-wise or neuron-wise:

Layer-Wise locally adaptive activation function (L-LAAF) let D be the number of layers and σ a nonlinear activation function. The new activation function on each layer is

$$\sigma(na^k x), \quad k = 1, \cdots, D-1$$

Neuron-Wise locally adaptive activation function (N-LAAF) let D be the number of layers, N_k the number of neurons on the layer k, σ a nonlinear activation function. The new activation function is

$$\sigma(na_i^k x), \quad k = 1, \cdots, D-1, \quad i = 1, \ldots, N_k$$

Both iterations of this activation function (L-LAAF, N-LAAF) can be associated with a regularization term, the slope recovery. This term affects the training by forcing the network to increase the value of the activation slope.

Figure 2 displays some activation functions previously mentioned, but also the impact of the parameter value for the Swish function and the LAAF where $\sigma = \tanh$.

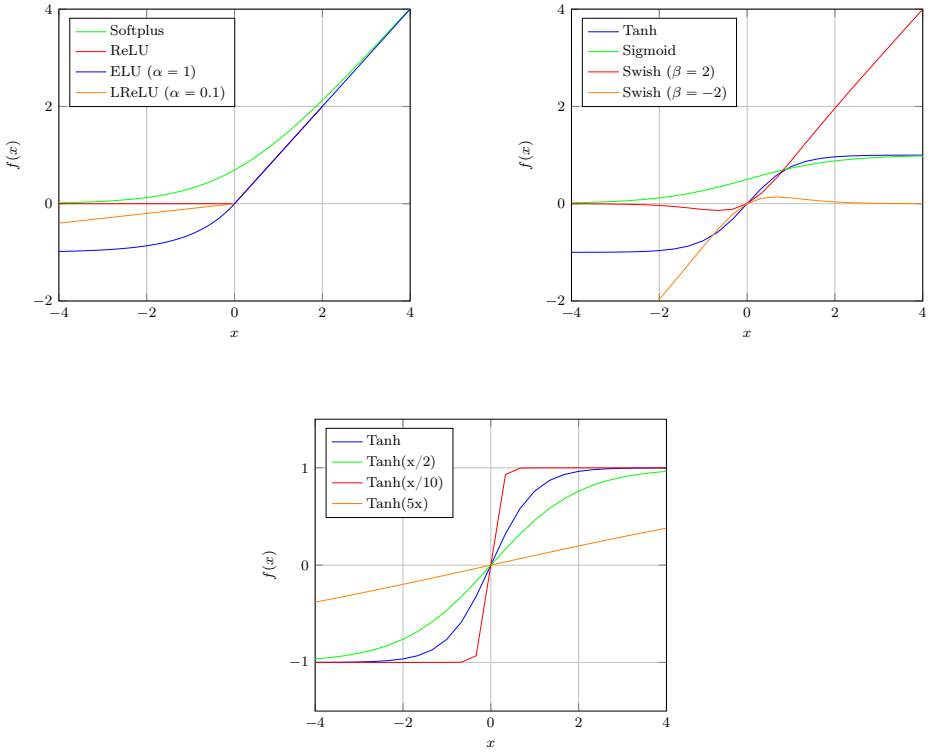


Figure 2: Some activation functions

1.6 The Importance of Sampling

Wu et al. [19] study the impact of different sampling strategies on both forward and inverse problems. The sampling methods can be split into two families fixed and adaptive sampling. On the proposed test cases the adaptive sampling almost always outperforms the fixed sampling. The different methods studied are described below.

1.6.1 Fixed Sampling Methods

Grid sampling

- **equispaced uniform grid**, the points are sampled along a uniform grid, so each of them is equispaced

Random sampling

- **uniformly random sampling**, the sampling of the points follows a continuous uniform distribution.

Latin hypercube sampling is a method for generating random samples of a parameter value. It is the generalization of the Latin square, where there is only one sample in each row and each column.

Quasi-random low-discrepancy sequences

- **Halton sequence**, it uses deterministic sequences to generate points in space. The Halton sequence is based on a method that uses coprime numbers.
- **Hammersley sequence**, it is the same as the Halton sequence, except in the first dimension where points are located equidistant from each other.
- **Sobol sequence**, it uses a sequence from a base of two to sample in a highly uniform manner.

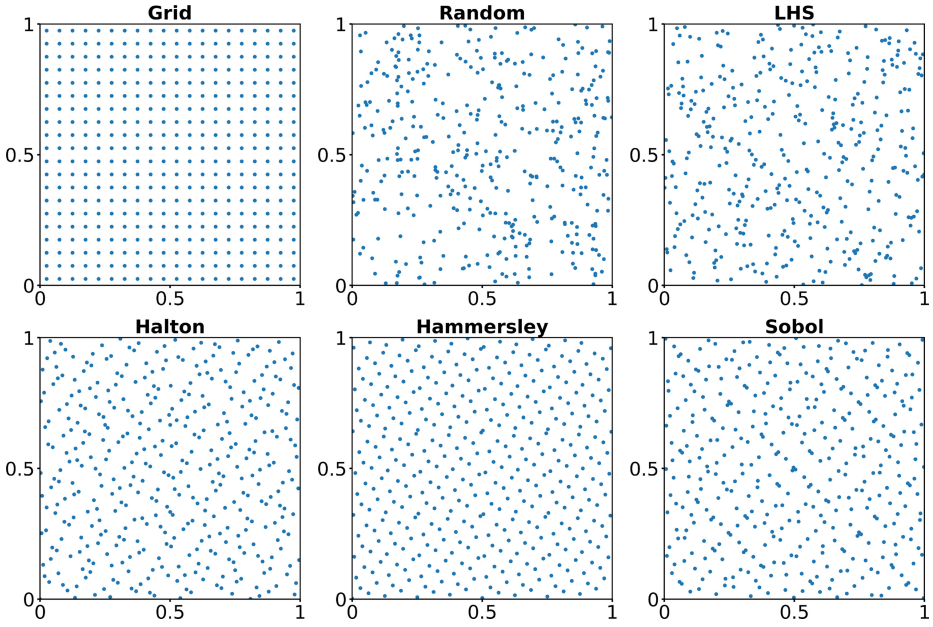


Figure 3: Examples of 400 points generated using different uniform sampling methods from [19]

While the performance appears to be problem-dependent, among the six uniform sampling methods with fixed residual points, the low-discrepancy sequences (Halton, Hammersley, and Sobol) generally outperform Random and LHS. Both latter, outperform the grid sampling in some specific test cases as the Allen–Cahn equation.

1.6.2 Resampling Methods

Uniform points with resampling Instead of using the same fixed residual points during the training, we could regenerate a new set of residual points after a given number of steps, N. Different sampling methods can be used than the ones presented so far.

Random Resampling (Random-R) Similar to the previous approach the residual points are randomly resampled every N iteration. The resampling period N is also an important hyperparameter for the accuracy of the residual.

Residual-based Adaptive Refinement with Greed (RAR-G) The goal is to improve the distribution of residual points during the PINN training by adding more points where the residual predicted by the PINN is large.

Algorithm 1: RAR-G

Sample the initial residual points $\mathcal{T}$ using a fixed sampling method;
Train the PINN for a certain number of iterations on $\mathcal{T}$;
repeat
  Sample a set of dense points $\mathcal{S}_0$ using a fixed sampling method;
  Compute the PDE residuals for the points in $\mathcal{S}_0$;
  $\mathcal{S} \leftarrow m$ points with the largest residual in $\mathcal{S}_0$;
  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{S}$;
  Train the PINN for a certain number of iterations on $\mathcal{T}$;
until *the total number of iterations OR the total number of residual points has not reached the limit*;

Residual-based Adaptive Distribution (RAD) In this method, the residual points are resampled according to a probability density function (PDF) $p(X)$ proportional to the PDE residual $p(x) \propto \varepsilon(X)$ i.e. $p(X) = \dfrac{\varepsilon(X)}{A} + c$, where A is a normalizing constant. For RAR-D we use,

$$p(X) \propto \frac{\varepsilon^k(X)}{\mathbb{E}[\varepsilon^k(X)]} + C$$

where $k \geqslant 0$ and $c \geqslant 0$ are two hyperparameters that control the profile of $p(X)$ thus the distribution of the sampled points. $\mathbb{E}[\varepsilon^k(X)]$ can be approximated by a numerical integration such as Monte Carlo integration.

Algorithm 2: RAD

Sample the initial residual points $\mathcal{T}$ using a fixed sampling method;
Train the PINN for a certain number of iterations;
repeat
    $\mathcal{T} \leftarrow$ a new sets of points randomly sampled according to the PDF;
    Train the PINN for a certain number of iterations;
until *the total number of iterations reaches the limit*;

When X is low-dimensional, we can do the sampling the following way: 1) Sample a set of dense points $\mathcal{S}_0$ using a fixed sampling method. 2) Compute p(x) for the points in $\mathcal{S}_0$. 3) Define a probability mass function $\tilde{p}(X) = \frac{p(X)}{A}$ with the normalizing constant $A = \sum_{x \in (S)_0} p(x)$. 4) Sample a subset of points from $\mathcal{S}_0$ according to $\tilde{p}(X)$.

For more complex cases we can use other methods: - Inverse transform sampling - Markov Chain Monte Carlo (MCMC) methods - Generative Adversarial Networks (GAN)

The combination of $k = 1$ and $c = 1$ is usually a good default choice.

Residual-based adaptive refinement with distribution (RAR-D) It is the hybrid method between RAR-G and RAR.

Algorithm 3: RAR-D

Sample the initial residual points $\mathcal{T}$ using a fixed sampling method;
Train the PINN for a certain number of iterations;
repeat
    $\mathcal{S} \leftarrow m$ points randomly sampled according the PDF;
    $\mathcal{S} \leftarrow \mathcal{T} \cup \mathcal{S}$;
    Train the PINN for a certain number of iterations;
until *the total number of iterations OR the total number of residual points reaches the limit*;

Figure 4 showcases the effect of the two parameters, k and c, on the location of the points with respect to the residual. The authors indicate that the combination of $k = 2$ and $c = 0$ is usually a good default choice.

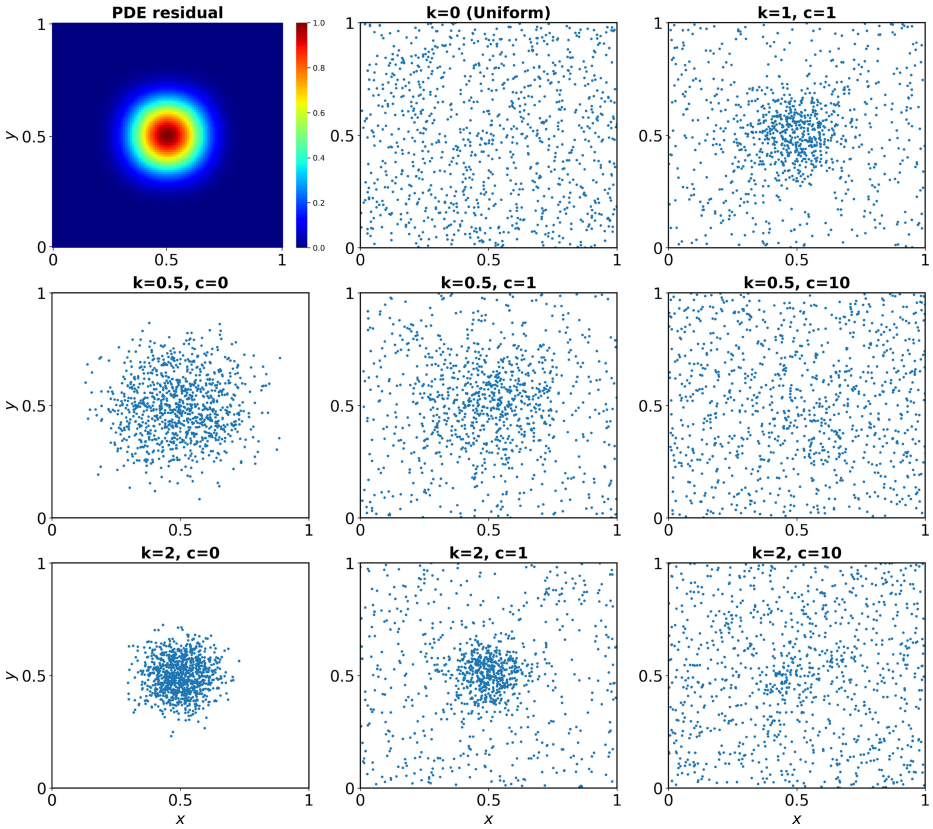


Figure 4: Examples of 1000 residual points sampled by RAD with different values of k and c for the PDE residual from [19]

Of the 10 sampling techniques tested, the RAD method consistently showed the best performance in solving both forward and inverse problems. When dealing with complex solutions in PDEs such as Burgers' and multiscale wave equations, the RAD and RAR-D methods were particularly successful, yielding significantly smaller errors. On the other hand, for PDEs with smooth solutions, such as the diffusion equation and the diffusion-reaction equation, certain uniform sampling approaches, such as Hammersley and Random-R, also produced acceptably small errors. Among these uniform sampling methods, Random-R typically showed better overall performance.

1.7 A Domain Decomposition Approach

The goal of the domain decomposition is to divide the whole domain into sub-domains to reduce computation times. On each sub-domain, the problem is solved locally with a PINN, called sub-PINN. The sub-domain can be sewn up using conservative quantities such as flux at the interface, this approach is called conservatives PINNs (cPINNs) [7] Since the sub-PINNs solve the PDE locally they can be tailored (architecture, hyperparameters, optimizer) to be the most efficient possible. The validity of the overall solution is guaranteed by the interface conditions determined by the problem. Flux continuity at the interface is enforced by an additional term in the loss function. Solution continuity can also be enforced. An important aspect of PINNs that cPINNs address is the computational efficiency by allowing more parallelism, only the computation of the

interface loss requires a communication. As each sub-domain provides additional information about the fluxes it allows to reduce the error propagation.

cPINNs can be generalized to extended PINNs (XPINNs) [8], which allows a space-time domain decomposition where the interface loss is not only related to a conservative quantity. The other quantities that can be enforced are the average solution and the residual continuity. Additionally, the flux continuity, or the C^k solution continuity can be enforced. There is no theoretical guarantee that this method will converge to a global minimum. However, it is worth noting that there are similarities between the training of sub-PINN with cooperation at the interface and Nash equilibrium. To address the performance component of this method, Shukla et al. [20] proposed an MPI + X algorithm where the sub-domains and their associated sub-PINNS are distributed across multiple GPUs. The only communication required is the information needed to compute the conditions on the shared interface.

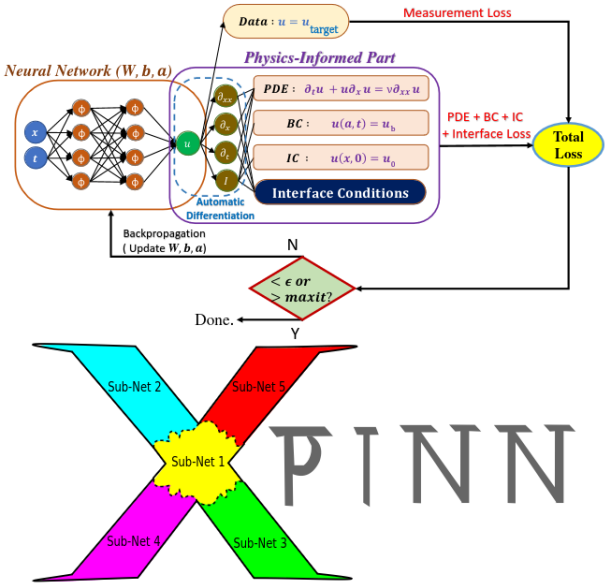


Figure 5: The top figure is the schematic of XPINN sub-net employed in a subdomain where neural network and physics-informed part for viscous Burgers equation are shown. The bottom figure shows the irregularly shaped subdomain divisions in 'X'-shaped domain, where sub-net is employed in each subdomain and they are stitched together using the interface conditions. In this case, the domain boundary is shown by black continuous line, whereas the interface is shown by black dash line [8].

1.8 Extension to Multiphysics Coupling

The domain decomposition approach can be applied for coupled problems [21, 22]. The only difference is that on sub-domains, different PDEs are solved. For instance for Conjugate Heat Transfer, on one sub-domain we solve the flow equations (velocity, pressure, temperature) meanwhile on the other sub-domain, we solve the heat equation (temperature). The interface conditions are enforced by the physics, in the case of heat transfer the conditions are the temperature and the cheat flux continuity.

Wang et al. [21] propose two different types of training, separate and joint. Suppose a given

domain is partitioned into $p = 1, \ldots, n$ sub-domains. Each sub-domain corresponds to a PINN characterized by weights denoted as θ_p.

Separate training The losses, which are composed of various conditions, including the interface, are computed for each sub-domain. Each sub-domain operates with its dedicated local optimizer to adjust the weights of its sub-PINN based on its local loss. Specifically, for each sub-domains i, the weights θ_i are updated using $\theta_i = \arg\min L_i$.

The training procedure involving 2 sub-domains with two neural networks is described in Algorithm 4. For each epoch the loss on the interface, $\mathcal{L}_I(\theta_1, \theta_2)$, is computed. It is the only step that requires communication. The losses on the first sub-domain (resp. the second) are computed for the residual $\mathcal{L}_{\Omega_1}(\theta_1)$ and the boundary conditions $\mathcal{L}_{\Gamma_1}(\theta_1)$ (resp. $\mathcal{L}_{\Omega_2}(\theta_2)$ and $\mathcal{L}_{\Gamma_2}(\theta_2)$), this can be done in parallel. Afterward the new parameters θ_1^* (resp. θ_2^*) are updated thanks to the optimizer applied on the total loss L_1 (resp. L_2). This optimization step can also be performed in parallel. The algorithm is also schematically represented in Figure 6.

The separate training allows using optimizer with well-suited hyperparameters to enhance the convergence speed. However, it does not allow the use of optimizers that use information based on the previous iteration such as BFGS, and L-BFGS, as the interface loss definition is constantly changing until convergence.

Algorithm 4: Parallel Separate Adam Training

Data: θ_1, θ_2 the parameters of the model 1 and 2
foreach *epoch* **do**
  Compute $\mathcal{L}_I(\theta_1, \theta_2)$
  Compute $\mathcal{L}_{\Omega_1}(\theta_1)$ and $\mathcal{L}_{\Gamma_1}(\theta_1)$
  Compute $\mathcal{L}_{\Omega_2}(\theta_2)$ and $\mathcal{L}_{\Gamma_2}(\theta_2)$
  $\theta_1^* \leftarrow \text{Optimizer}(L_1, \theta_1)$ where $L_1 = \mathcal{L}_{\Omega_1}(\theta_1) + \mathcal{L}_{\Gamma_1}(\theta_1) + \mathcal{L}_I(\theta_1, \theta_2)$
  $\theta_2^* \leftarrow \text{Optimizer}(L_2, \theta_2)$ where $L_2 = \mathcal{L}_{\Omega_2}(\theta_2) + \mathcal{L}_{\Gamma_2}(\theta_2) + \mathcal{L}_I(\theta_1, \theta_2)$
end

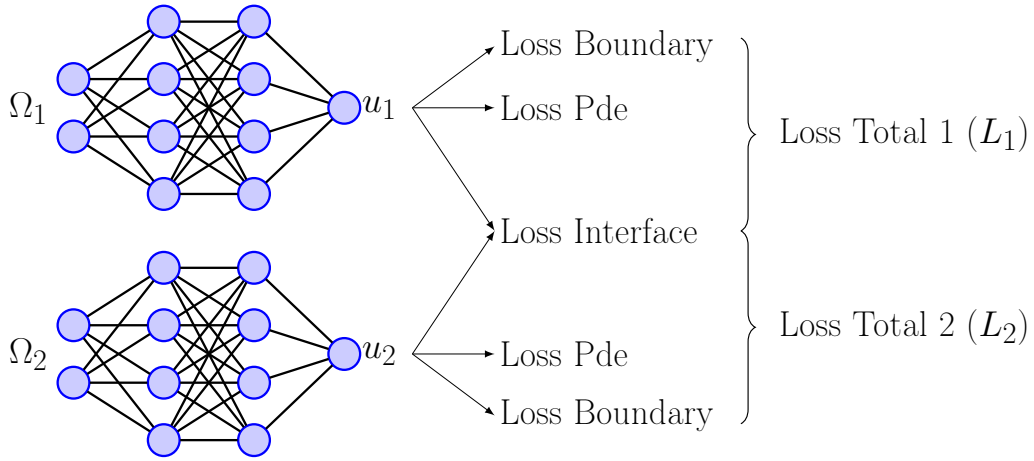


Figure 6: Loss calculation scheme for separate education

Joint training In the joint training scenario, local losses are computed for each sub-domain and then aggregated into a global loss. Consequently, all the weights are updated together by a single optimizer. Thus $\theta_{1,\dots,n} = \arg\min L$.

The training protocol for two neural networks is explained in Algorithm 5. It follows the same steps as the separate training to local losses to compute residual and boundary conditions. Then, these losses are summed together with the loss on the interface conditions to form a single total loss L. The algorithm is also schematically represented in Figure 7.

Algorithm 5: Adam Joint Training

Data: θ_1, θ_2 the parameters of the model 1 and 2, and $\theta_{1,2} = [\theta_1|\theta_2]$

foreach *epoch* **do**

  $L_1 \leftarrow \mathcal{L}_{\Omega_1}(\theta_1) + \mathcal{L}_{\Gamma_1}(\theta_1)$

  $L_2 \leftarrow \mathcal{L}_{\Omega_2}(\theta_2) + \mathcal{L}_{\Gamma_2}(\theta_2)$

  $L = L_1 + L_2 + \mathcal{L}_I(\theta_1, \theta_2)$

  $\theta^*_{1,2} \leftarrow \text{Optimizer}(L,\ \theta_{1,2})$

end

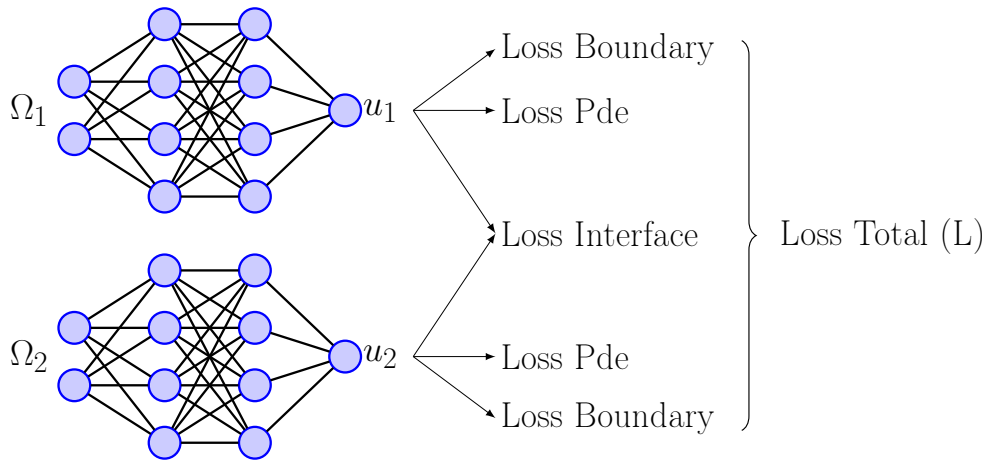


Figure 7: Loss calculation scheme for joint education

Contrary to the separate training, the joint training allows to use the BFGS optimizer. This allows to further improve the convergence of training at the cost of less parallelism and a higher computational cost. Indeed, this approach requires solving an optimization problem on more parameters, while the separate training solves a smaller optimization problem in parallel. So there's a trade-off between convergence quality and computational costs.

2 Test Cases

2.1 Heat Transfer Simulations

2.1.1 Basic Heat Transfer in a Solid

In this test case, we examine the heat transfer within a disk, Ω, containing a hole at the center. The two boundaries, Γ_1 and Γ_2, are heated to distinct temperatures. Figure 8 represents the

domain of computation where Ω refers to the interior of the domain, whereas the boundaries Γ_1 and Γ_2 consist of circles with respective radius R_{in} and R_{out}.

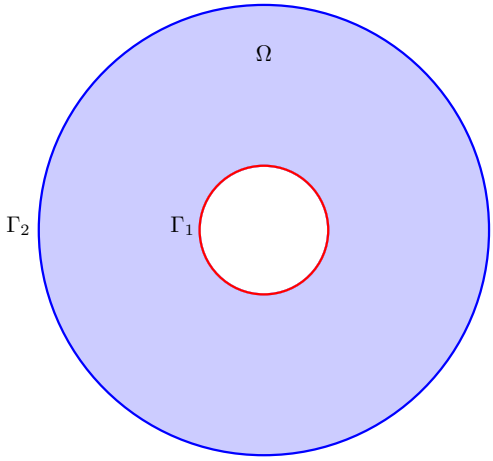


Figure 8: Schematic of the domain

The governing equations and the boundary conditions can be written as follows,

$$\begin{cases} \nabla \cdot (k \nabla T) = 0 \text{ on } \Omega & (1) \\ T = T_1 \text{ on } \Gamma_1 & (2) \\ T = T_2 \text{ on } \Gamma_2 & (3) \end{cases}$$

where k is the thermal conductivity of the material. The governing equation can be reformulated in polar coordinates and is written,

$$\frac{1}{r}\frac{\partial}{\partial r}\left(rk\frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \theta}\left(k\frac{\partial T}{\partial \theta}\right) = 0. \tag{4}$$

Solving this problem with a PINN is straightforward. Given the previous equations the losses of the different domains as the MSE of the residuals,

$$\mathcal{L}_\Omega = \frac{1}{|\Omega|}\sum_{\mathrm{x}\in\Omega} \left(\nabla \cdot (k\nabla T)(\mathrm{x})\right)^2$$
$$\mathcal{L}_{\Gamma_i} = \frac{1}{|\Gamma_i|}\sum_{\mathrm{x}\in\Gamma_i} \left(T(\mathrm{x}) - T_i\right)^2, \ i = 1, 2.$$

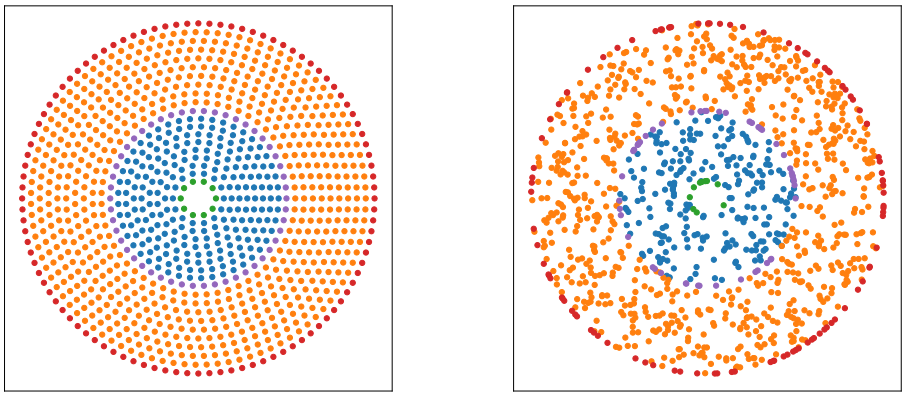
The total loss to minimize is $L = \mathcal{L}_\Omega + \mathcal{L}_{\Gamma_1} + \mathcal{L}_{\Gamma_2}$ with parameters given in Table 1.

Parameters	R_{in}	R_{out}	T_1	T_2
Values	1	10	10	5

Table 1: Parameters of the problem

For this test case, we consider two types of sampling, a uniform grid in polar coordinates and a random uniform sampling. Figure 9 illustrates the different samplings, for the grid sampling all

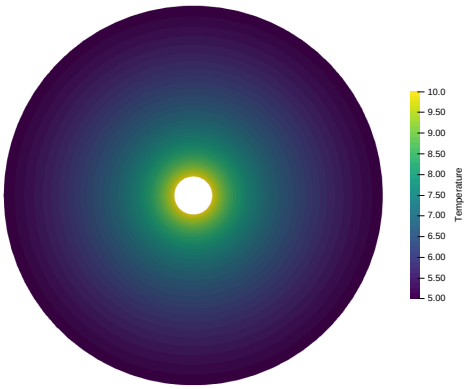
the lines have the same point density. While for the random sampling, the different domains have the same point density.



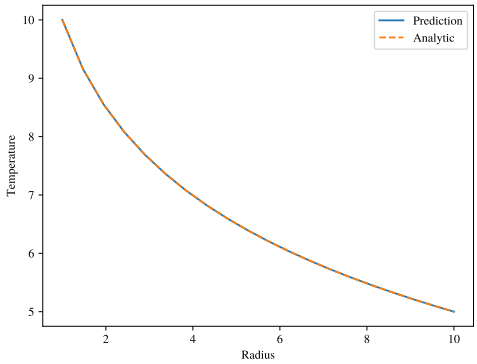
(a) Grid sampling (b) Random sampling

Figure 9: Three simple graphs

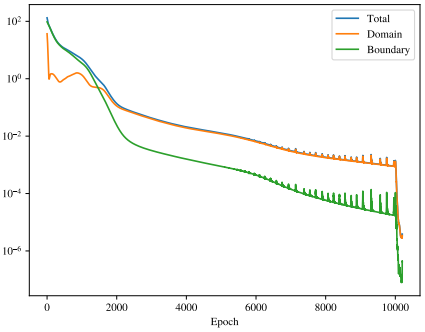
To solve this problem, we use a network with 2 layers each containing 10 units which is 481 trainable parameters. The activation function is tanh. We use the polar formulation with a grid sampling, the input of the neural network are the polar coordinates r, θ and the output is the temperature T. The training is performed with at first 10,000 epochs using the Adam optimizer with the default parameters, followed by 200 epochs of BFGS. The mean absolute relative error is 5.0×10^{-5} and its standard deviation is 2.4×10^{-5} where the exact solution is computed according to the analytical solution detailed in Appendix A.1. Figure 10 displays the solution both Cartesian and polar coordinates. Figure 10c displays the advantages of incorporating BFGS steps following the Adam iterations, as a clear drop can be seen. Moreover, it illustrates the challenge of learning the governing equation compared to the boundary conditions.



(a) Solution in Cartesian coordinates



(b) Average with respect to to θ of the solution in polar coordinates



(c) Convergence of the three losses with 10000 epochs of Adam followed by 200 epochs of BFGS

Figure 10: Representation of the solution in both Cartesian (a) and polar (b) coordinates, as well as the loss evolution

2.1.2 Extension to Two Different Materials

To further study the case of heat transfer, the previous case is extended to a domain with two sub-domains, Ω_1 and Ω_2. The boundaries remain, but there is now an interface I at radius R_{in} between the two domains. Each sub-domain has a different thermal conductivity, k_1 and k_2. The new computational domain is described in Figure 11.

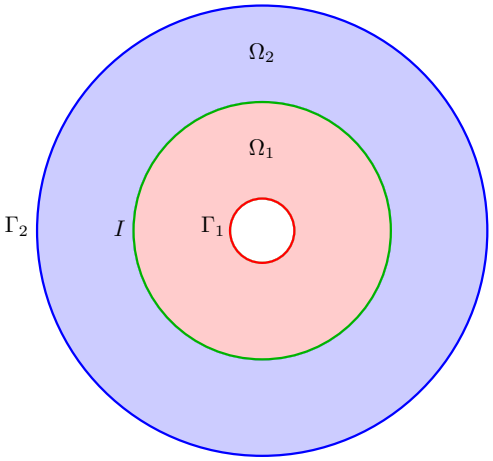


Figure 11: Schematic of the domain

The problem is now defined as follows,

$$\begin{cases} \nabla \cdot (k_1 \nabla T) = 0 \text{ on } \Omega_1 & (5) \\ \nabla \cdot (k_2 \nabla T) = 0 \text{ on } \Omega_2 & (6) \\ T = T_1 \text{ on } \Gamma_1 & (7) \\ T = T_2 \text{ on } \Gamma_2 & (8) \end{cases}$$

with the following conditions on the interface

$$\begin{cases} T_1^I = T_2^I \\ \phi_1^I = \phi_2^I \end{cases} \tag{9}$$

where T_1^I (resp. T_2^I) is the temperature predicted by the model 1 (resp. model 2) and $\phi_1^I = k_1 \dfrac{\partial T}{\partial n}$ the heat flux computed on the interface. The parameters of the problem are given in Table 2.

Parameters	R_{in}	R_{int}	R_{out}	T_1	T_2	k_1	k_2
Values	1	5	10	10	5	10	5

Table 2: Parameters of the problem

The two losses to minimize for the separate training are now,
$L_1 = \mathcal{L}_{\Omega_1} + \mathcal{L}_{\Gamma_1} + \mathcal{L}_I$ and $L_2 = \mathcal{L}_{\Omega_2} + \mathcal{L}_{\Gamma_2} + \mathcal{L}_I$,
where $\mathcal{L}_I = \dfrac{1}{|I|} \sum_{\mathrm{x} \in I} \left(T_1(\mathrm{x}) - T_2(\mathrm{x})\right)^2 + \dfrac{1}{|I|} \sum_{\mathrm{x} \in I} \left(\phi_1(\mathrm{x}) - \phi_2(\mathrm{x})\right)^2$

To solve this problem, we consider two networks with 3 layers each containing 20 units which gives 921 trainable parameters. The activation function is tanh. We use the Cartesian formulation with a grid sampling, the inputs of the neural network are the coordinates x, y, and the output is the temperature T.

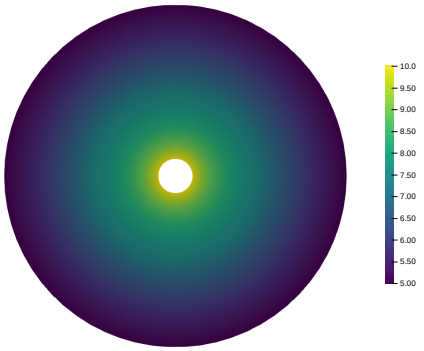
The domain is split into two sub-domains, $\mathcal{D}_1 = \Gamma_1 \cup \Omega_1 \cup I$ and $\mathcal{D}_2 = I \cup \Omega_2 \cup \Gamma_2$, such that $k = k_i$ on $\mathcal{D}_i$, $i = 1, 2$. The first network (resp. the second) will be trained on $\mathcal{D}_1$ (resp. $\mathcal{D}_2$) and its output must verify the governing equation, Equation 5 (resp. 6), and the boundary condition,

Equation 7, and similarly for the second neural network. Both NNs must verify the interface conditions described in Equation 9.

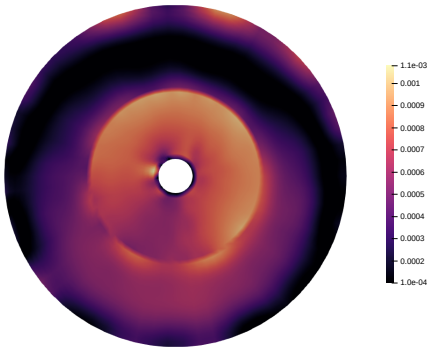
The training is performed using the separate training method described in Algorithm 4 with 50,000 epochs of Adam with the default parameters for both neural networks.

We compute the analytical solution as detailed in Appendix A.2 to evaluate the error with respect to the predictions. The mean absolute relative error is 3.5×10^{-4} and its standard deviation is 3.8×10^{-4}, the exact solution is computed according to the analytical solution demonstrated in Appendix A.2.

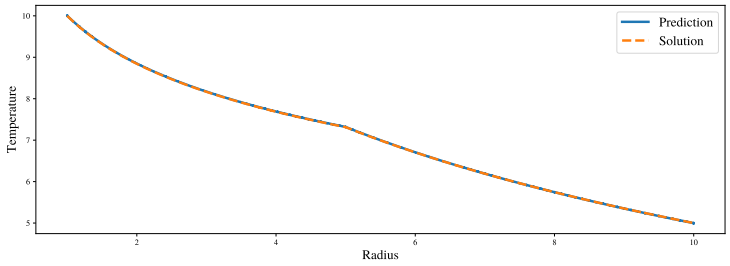
Figure 12 shows the prediction and the absolute relative error in Cartesian coordinates as well as the comparison between the solution and the prediction average along the radius. We can notice that the error is higher near the interface. It is expected as it is only implicitly enforced through the interface conditions contrary to the Dirichlet conditions.



(a) Prediction of the neural networks



(b) Absolute relative error



(c) Average solution in polar coordinates

Figure 12: Solution

It is also possible to use the polar coordinates, however using the same network as the Cartesian coordinates leads to noisy solutions. This issue is overcome by using a smaller NN with two layers instead of three layers.

2.1.3 Parameterized Resolution

One of the promising aspects of PINNs is to solve parametric PDEs. It is the ability to solve a PDE where a parameter can take several values. To this end, the PINN needs knowledge of the parameter value, which can be incorporated into the inputs to inform the model.

For instance with different values of thermal conductivity on a domain $K = (k^1, \dots, k^n)$. There are two ways to create the new data, the first one is to compute the tensor product between the space domain $\Omega = \Omega_1 \cup \Omega_2$ and the vector of parameter K,

$$\Omega \otimes K = \begin{bmatrix} \Omega & k_2^1 \\ \vdots & \vdots \\ \Omega & k_2^n \end{bmatrix}.$$

However, this method requires linearly increasing the size of the space domain in relation to the size of K. Additionally, utilizing the same space points repeatedly may not aid in resolving the PDE, as it offers unnecessary information. The second approach for procuring the training data is to append a dimension for the parameter to Ω, and subsequently sample the hypercube. The primary distinction is that with the first method, we would be sampling the entire space domain for every discrete training parameter value. While for the second method, we perform a sparse sampling in the three-dimensional domain, we utilize the first approach to explore the parameterized resolution.

Parameter of the PDE We experiment with the resolution with different values of k_2 on Ω_2. The training data for the model 1 and 2 will be $\Omega_1 \otimes K$ and $\Omega_2 \otimes K$. Information of k_2 must be provided to the model 1 as it will impact the interface values.

The training is done for 15 values of k_2 equally spaced between 5 and 30. The validation metrics are computed using 100 values of k_2 equally spaced between 5 and 30. The NN used has 3 layers with 20 units each with tanh as activation function, the training data is the output of the tensor product between a grid in Cartesian coordinates and the parameter vector. The training is performed using 50,000 epochs of Adam.

For the validation parameters the absolute relative error mean is 6.3×10^{-4} and its standard deviation is 6.2×10^{-4}. Figure 14 shows the prediction and the solution for different k_2 value taken in the training range.

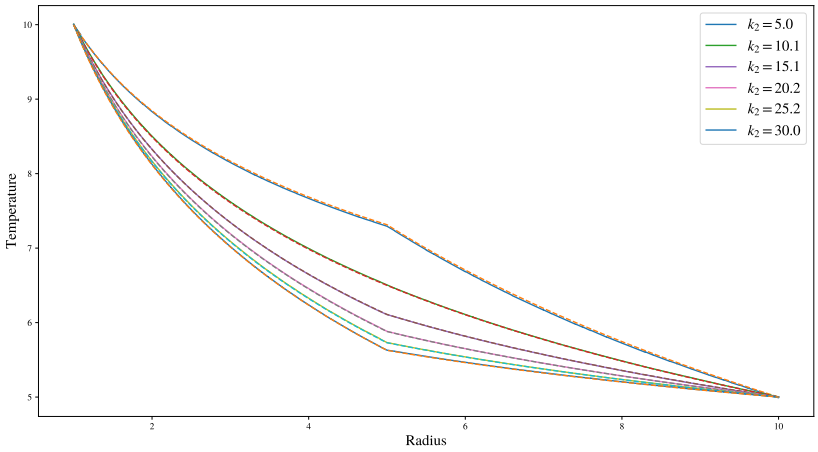


Figure 13: Prediction (——) and solution (- - -) for different values of k_2.

Parameter of the Geometry We now experiment with the resolution with different radius values for the interface. The training data for the model 1 and 2 will be $\Omega_1 \otimes R$ and $\Omega_2 \otimes R$, where $R = (r_i^1, \ldots, r_i^n)$ the vector of the interface radius. The training is done for 4 values of k_2 equally spaced between 3 and 7. The validation metrics are computed using 32 values of k_2 equally spaced between 3 and 7. The NN used has 2 layers with 20 units each with tanh as activation function. The training data is the output of the tensor product between a grid in polar coordinates and the parameter vector. The training is performed using 50,000 epochs of Adam.

For the validation parameters the absolute relative error mean is 1.4×10^{-3} and its standard deviation is 7.8×10^{-4}. Figure 14 shows the prediction and the solution for different radius values taken in the training range.

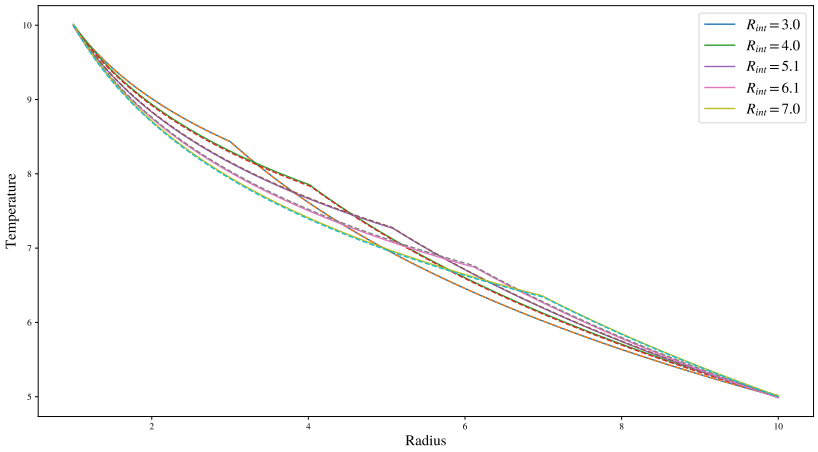


Figure 14: Prediction (—) and solution (- - -) for different values of R_{int}.

Discussion We have shown that a PINN is capable of solving parametric PDEs, at the expense of a linear increase in the size of the spatial domain. For out-of-distribution parameter values, the PINN's performance is poor, increasingly degrading as we move further away from the range of training values. This shows that despite being able to solve the PDE for certain values of a parameter, the PINN is unable to extrapolate to values outside the training interval. Because this resolution increases the size of the data, it can be scaled with a high degree of parallelism, for example, by distributing the data across multiple GPUs.

2.2 Conjugate Heat Transfer

The main objective of the following test cases is to use PINNs to achieve coupling of multiphysics phenomena, specifically addressing the complex heat transfer between a fluid and a solid. All cases in this section are considered through steady-state equations.

2.2.1 Forced convection Scenarios

The forced convection scenario presented in this subsubsection serves as a preliminary exploration of multiphysics coupling in the context of conjugate heat transfer. The primary focus of this

study is to investigate the effects of one-way coupling between fluid velocity and temperature, specifically in the context of heat transfer from a solid domain to a fluid domain.

Methodology The methodology involves a computational setup, shown in 15, with a fluid domain (a pipe) $\Omega_f = [-1, 1] \times [-0.25, 0.25]$ and a solid domain (a rectangle) $\Omega_s = [0, 1] \times [-0.75, -0.25]$ placed adjacent to each other. The goal is to model the heat transfer process from the solid to the fluid outlet. The fluid domain is governed by the two-dimensional Navier-Stokes equations for fluid flow and the energy conservation equation for heat transfer.

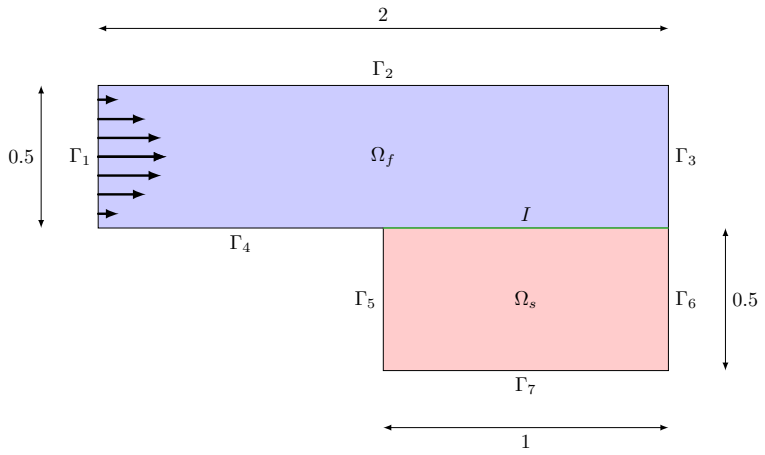


Figure 15: Representation of the domain

Fluid equations The fluid subsubsection is governed by the two-dimensional Navier-Stokes and the energy conservation equations, which can be written as follows,

$$\begin{cases} \dfrac{\partial u}{\partial x} + \dfrac{\partial v}{\partial y} = 0 & (10) \\ u\dfrac{\partial u}{\partial x} + v\dfrac{\partial u}{\partial y} + \dfrac{1}{\rho}\dfrac{\partial p}{\partial x} - \nu(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}) = 0 & (11) \\ u\dfrac{\partial v}{\partial x} + v\dfrac{\partial v}{\partial y} + \dfrac{1}{\rho}\dfrac{\partial p}{\partial y} - \nu(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}) = 0 & (12) \\ C_p(u\dfrac{\partial T}{\partial x} + v\dfrac{\partial T}{\partial y}) - \dfrac{k_f}{\rho}(\dfrac{\partial^2 T}{\partial x^2} + \dfrac{\partial^2 T}{\partial y^2}) = 0 & (13) \end{cases}$$

where u and v are the velocities along the x and y axes, ρ is the fluid density, p is the pressure, ν is the kinematic viscosity, C_p is the heat capacity and k_f is the thermal conductivity.

Equation 10 is the continuity equation, it represents the conservation of mass in a fluid. Equation 11 and Equation 12 are the x and y momentum equations representing the conservation of momentum in the two directions. It describes the change in velocity due to advection, pressure gradient, and viscous forces. Equation 13 is the energy conservation equation. It represents the change in temperature due to advection and conduction in both directions.

In this case, the energy equation and the Navier-Stokes equations are decoupled. More specifically, it is a one-way coupling in the sense that the velocities are involved in the energy conservation equation, while the temperature does not appear in the Navier-Stokes equations. In

other words, flow velocity has an impact on temperature, but not vice versa. The solution of the flow is the Poiseuille flow.

On the pipe boundaries, no-slip conditions are imposed

$$\begin{cases} u = 0 & \text{on } \Gamma_2 \cup \Gamma_4 \cup I, \quad (14) \\ v = 0 & \text{on } \Gamma_2 \cup \Gamma_4 \cup I. \quad (15) \end{cases}$$

For the inlet, Γ_1, a parabolic inflow is given through a Dirichlet condition to enforce a Poiseuille flow with incoming temperature T_c.

$$\begin{cases} u = 1 - \left(\dfrac{y}{y_{max}}\right)^2 & \text{on } \Gamma_1 \quad (16) \\ v = 0 & \text{on } \Gamma_1 \quad (17) \\ T = T_c & \text{on } \Gamma_1 \quad (18) \end{cases}$$

According to flow conservation, we impose that the outflow on the outlet, Γ_3, must be equal to the inflow,

$$\int_{\Gamma_1} u \, dy = \int_{\Gamma_3} u \, dy. \quad (19)$$

Finally, the Neumann condition for the temperature flux is considered,

$$\frac{\partial T}{\partial n} = 0 \quad \text{on } \Gamma_2 \cup \Gamma_3 \cup \Gamma_4. \quad (20)$$

Solid equations In the solid domain, we only solve the heat conduction equation as in the previous test case,

$$k_s\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = 0 \quad (21)$$

with the following boundary conditions,

$$\begin{cases} \dfrac{\partial T}{\partial n} = 0 & \text{on } \Gamma_5 \cup \Gamma_6 \quad (22) \\ T = T_h & \text{on } \Gamma_7 \quad (23) \end{cases}$$

Fluid-solid coupling The coupling is done by enforcing the temperature and the heat flux continuity on the interface.

$$\begin{cases} T_f^I = T_s^I & (24) \\ \phi_f^I = \phi_s^I & (25) \end{cases}$$

where T_f^I (resp. T_s^I) is the temperature predicted by the fluid model (resp. solid model) and $\phi_f^I = k_f \dfrac{\partial T}{\partial n}$ is the heat flux computed on the interface.

The parameters of the problem are $T_c = 0.2$, $T_h = 1.0$, $\nu = 0.01$, $C_p = 1.0$, $\rho = 1.0$, $k_f = 0.025$, and $k_s = 0.1$.

The reference temperature is calculated using the `FreeFem` program to solve the heat transfer between two domains while considering the exact velocity values. The reference temperature is illustrated in Figure 16. Since there is a singularity at $(0, -0.25)$, the elements sharing a point with it produce inaccurate values.

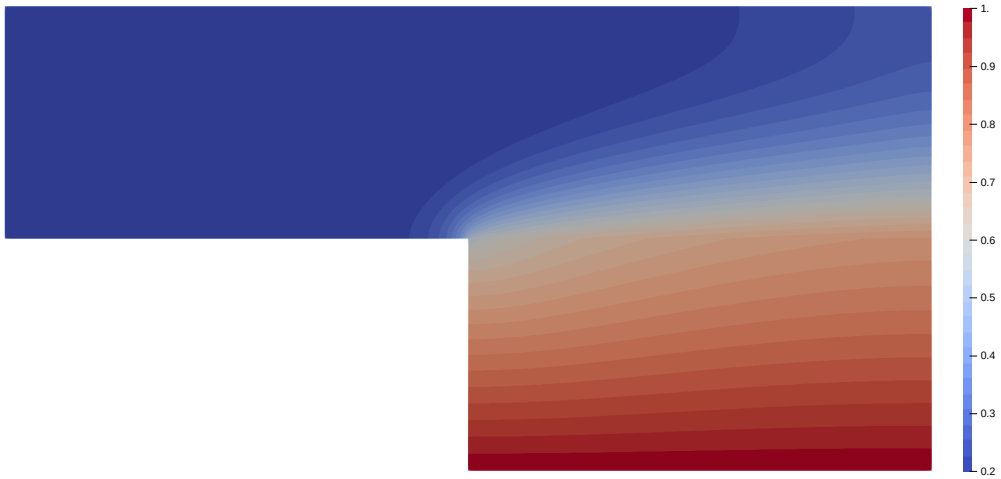


Figure 16: Temperature reference computed with `FreeFem`

Training Similarly to the previous test case we use two sub-PINNs, the fluid PINN and the solid PINN. The fluid PINN has three layers with 50 neurons units each and the solid PINN has three layers with 20 neurons each. Both use tanh as activation function. Both PINNs take as inputs points of coordinates x, y but the fluid PINN predicts four quantities u, v, p, T while the solid PINN only predicts the temperature T. They are shown schematically in Figure 17. The training is performed with data from a 80×20 uniform grid.

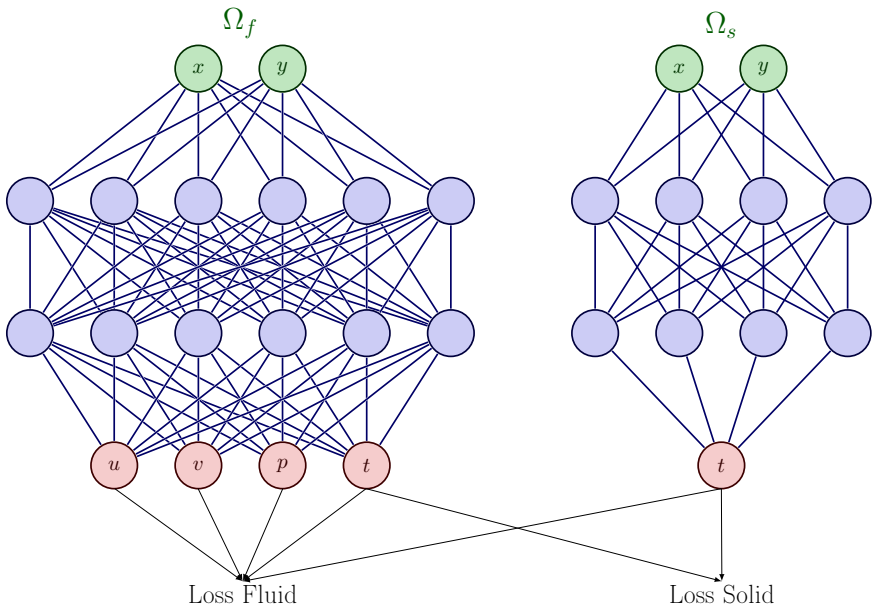


Figure 17: Schematic of the two neural networks and the communications to compute the losses.

Separate training The first experiment is made by training the two PINNs using the separate method, Algorithm 4. We consider the Adam optimizer with the default learning rate, 0.001, and perform 50,000 iterations. Figure 18 shows the temperature predicted on both domains.

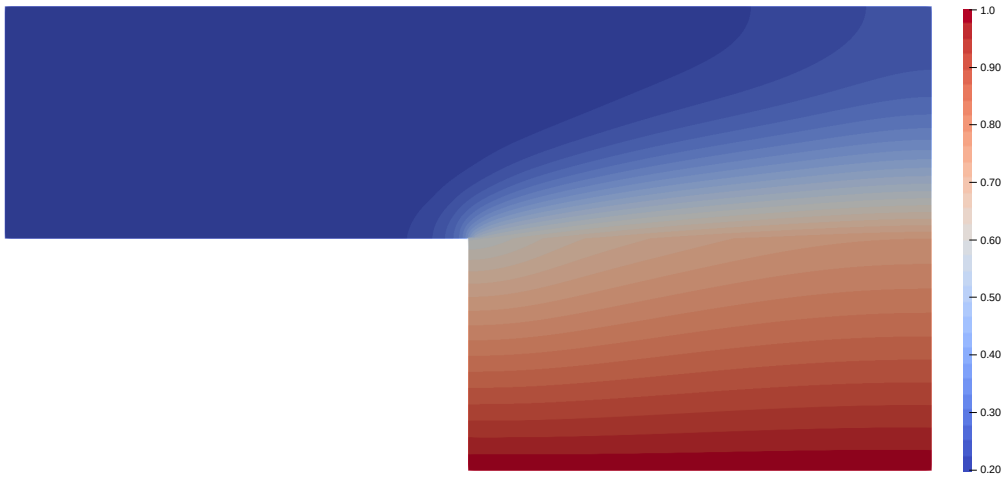
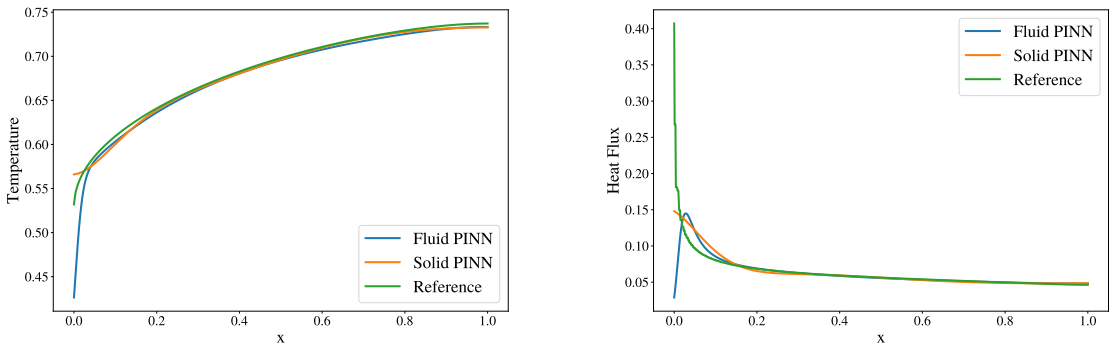


Figure 18: Temperature prediction using the separate training method.

Additionally Figure 19 shows the temperature predicted and the heat flux computed along the

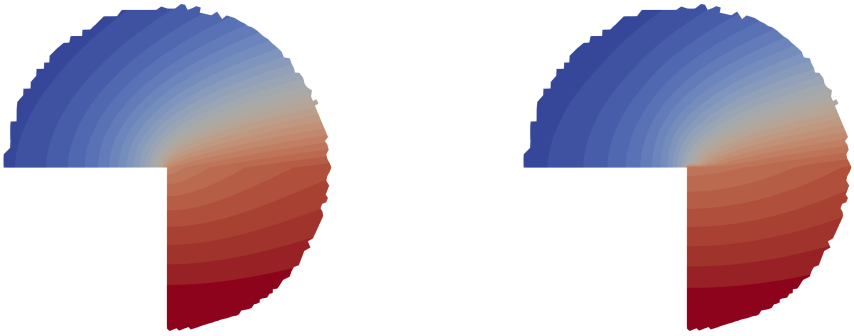
interface, we compare them with the reference. We observe that the fluid sub-PINN's prediction quality is poor near the singularity. It appears that the sub-PINN faces difficulty in transitioning from the Neumann condition of imposing a null heat flux to a non-null flux. Furthermore, it can be observed that the temperature remains nearly constant for the solid sub-PINN as it approaches the boundary, specifically closer to the null Neumann condition. From this observation, the primary challenge in this test case is accurately forecasting the interface values due to their contamination by boundary conditions. Figure 20 presents a close-up of the prediction and reference near the singularity, it enlightens that the PINNs are unable to capture the non-linearity of the temperature and the flux in this region. Finally, Table 3 displays the mean squared error (MSE) for the temperature predictions compared to the reference, as well as the MSE for the velocities u and v using the analytical solution. It also presents the MSE for the various residuals, which are the quantities targeted by the training process. These metrics are computed on a 100×100 uniform grid, it has 6.25 more points than the training grid. We observe that the average error is approximately 10^{-5}, which is adequate to provide a convincing prediction. However, it is not sufficient for the method to be robust. The worst metric is the one that corresponds to the loss of the governing equation (PDE all), which stands at 2.1×10^{-3}. The computation of the loss of the four equations shows that the aforementioned loss is dominated by the loss of the energy conservation equation (PDE Heat), which is at 8.2×10^{-3}. This explains the modest performance and emphasizes the difficulty of coupling the temperature.



(a) Temperature comparison

(b) Heat flux comparison

Figure 19: Comparison, along the interface, of the temperature (a) and heat flux (b) predicted by the two models and the FreeFem reference



(a) Reference (b) Prediction

Figure 20: Close-up comparison of the temperature predicted and the reference near the singularity for a radius of 0.125

(a) **Fluid** related metrics

Posterior error	
Temperature	1.5×10^{-5}
Velocity (u)	3.6×10^{-5}
Velocity (v)	2.5×10^{-5}
Residual error	
Slip	1.2×10^{-5}
Inlet	9.8×10^{-6}
Outlet	1.2×10^{-7}
Neumann	5.6×10^{-5}
PDE (All)	2.1×10^{-3}
PDE (Heat)	8.2×10^{-3}
PDE (continuity)	6.5×10^{-5}
PDE (momentum x)	3.0×10^{-4}
PDE (momentum y)	3.6×10^{-5}

(b) **Solid** related metrics

Posterior error	
Temperature	5.5×10^{-6}
Residual error	
Dirichlet	2.9×10^{-7}
Neumann	2.2×10^{-5}
PDE	1.0×10^{-5}

Table 3: MSE for the physical quantity, and the PDE and conditions residuals for the separate training

Joint training The second experiment is made by training the PINNs using the joint method described in Algorithm 5. We use the Adam optimizer, with the default learning rate, and perform 50,000 iterations followed by 2,000 iterations of BFGS. In this framework using the default Adam

optimizer, both separate and joint training produce similar loss evolution and metrics. However, since joint training permits the use of the BFGS optimizer, it offers the opportunity to benefit from using an alternative optimizer. Considering this aspect, joint training can be seen as a refinement of separate training. Figure 22 shows that the PINNs suffer from the same phenomenon on the interface. However, the use of the BFGS optimizer allows to further optimize the losses, thus allowing a slightly lower error. However, the loss of the energy conservation equation is not improved, still dominating the optimization problem. As displayed in Table 4, the use of the BFGS optimizer allows to greatly improve loss of the boundary conditions. Finally, in Figure 23, we display the absolute relative error of the velocities, u, v, and the temperature, T. We can see that for the temperature the error is high near the singularity, which is expected given the previous observations. However, there is also error located near the outlet, which can explain error observed for u and v at the outlet.

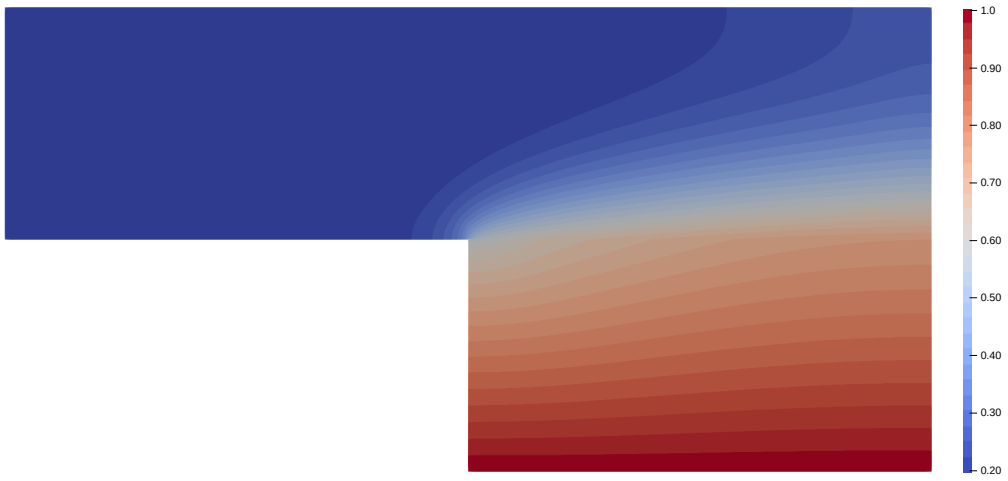
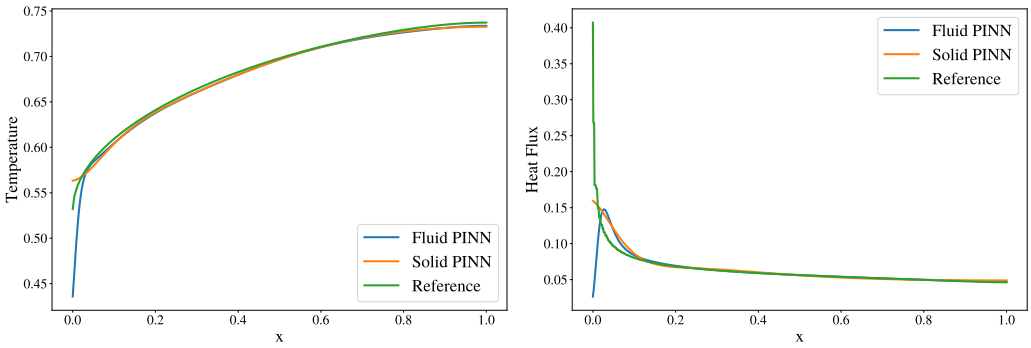


Figure 21: Temperature prediction using the separate training method.



(a) Temperature comparison (b) Heat flux comparison

Figure 22: Comparison, along the interface, of the temperature (a) and heat flux (b) predicted by the two models and the FreeFem reference

(a) **Fluid** related metrics

Posterior error	
Temperature	6.6×10^{-6}
Velocity (u)	1.0×10^{-5}
Velocity (v)	7.3×10^{-6}
Residual error	
Slip	3.4×10^{-7}
Inlet	3.1×10^{-8}
Outlet	2.9×10^{-12}
Neumann	1.3×10^{-5}
PDE (All)	1.9×10^{-3}
PDE (Heat)	7.9×10^{-3}
PDE (continuity)	7.1×10^{-6}
PDE (momentum x)	1.2×10^{-5}
PDE (momentum y)	7.0×10^{-6}

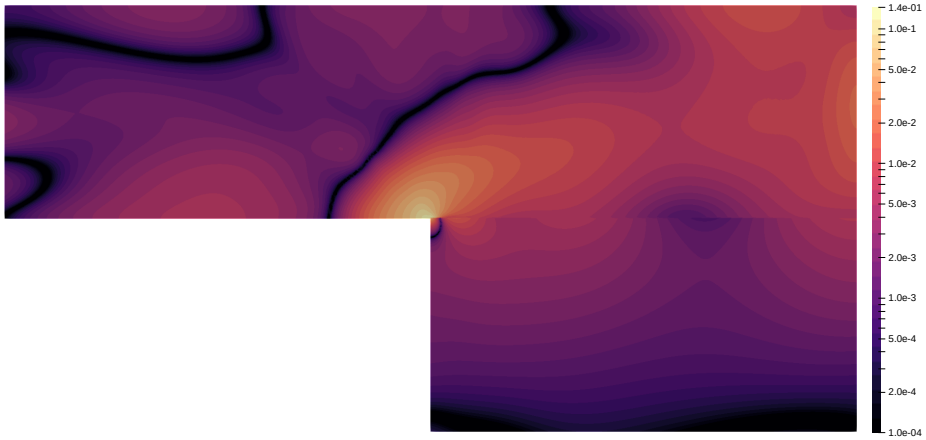
(b) **Solid** related metrics

Posterior error	
Temperature	2.2×10^{-6}
Residual error	
Dirichlet	5.1×10^{-8}
Neumann	1.4×10^{-5}
PDE	1.1×10^{-5}

Table 4: MSE for the physical quantity, and the PDE and conditions residuals for the joint training

(a) Absolute relative error u

(b) Absolute error v



(c) Absolute relative error T

Figure 23: Error metrics of the prediction u, v, T

Split training We propose an alternative training method that involves dividing the fluid sub-PINN into two sub-PINNs. One sub-PINN predicts temperature while another predicts velocities and pressure. This approach aims to capitalize on two properties. By splitting the PINN, we also split the loss function, reducing its complexity. Since the losses involving temperature dominate the loss function, we hope it will be simpler to separately minimize the momentum and energy equations without polluting each other. This occurs because one gradient dominates the others. Additionally, as previously mentioned, NNs are universal approximators capable of approximating a function. However, the fluid sub-PINN must represent a function that gives u, v, p, T for all x and y. If one calculates the analytical solution of the equation for u, v, p, T, it might be seen that the four functions are very different. Hence, the task of learning a single function to represent four various equations may be too difficult. The splitting is schematically represented in Figure 24.

As shown in Figure 26, the performance at the interface is still poor, but this time the heat flux in the fluid is better since we no longer observe the overshoot phenomenon once we leave the neighborhood of the singularity. However, it is the heat flux in the solid that fails to capture the non-linearity. For the temperature, both the fluid and the solid show the same behavior near the boundary, but they are shifted with respect to the reference. As for the metrics in Table 5, they

are slightly worse for the temperature, but the MSE for the velocities u, v are greatly improved. This is showcased in Figure 27.

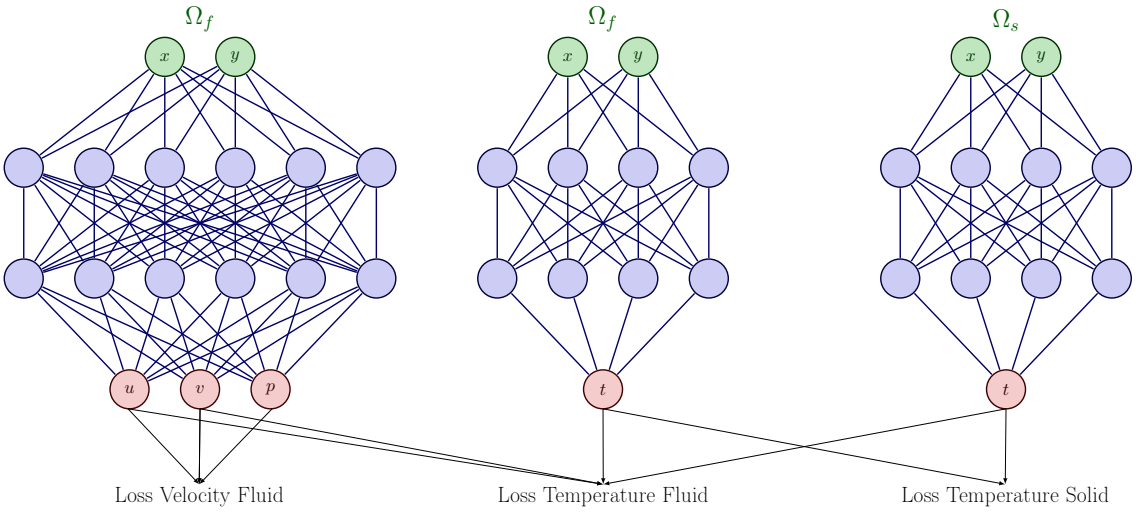


Figure 24: Schematic of the three neural networks and the communications to compute the losses.

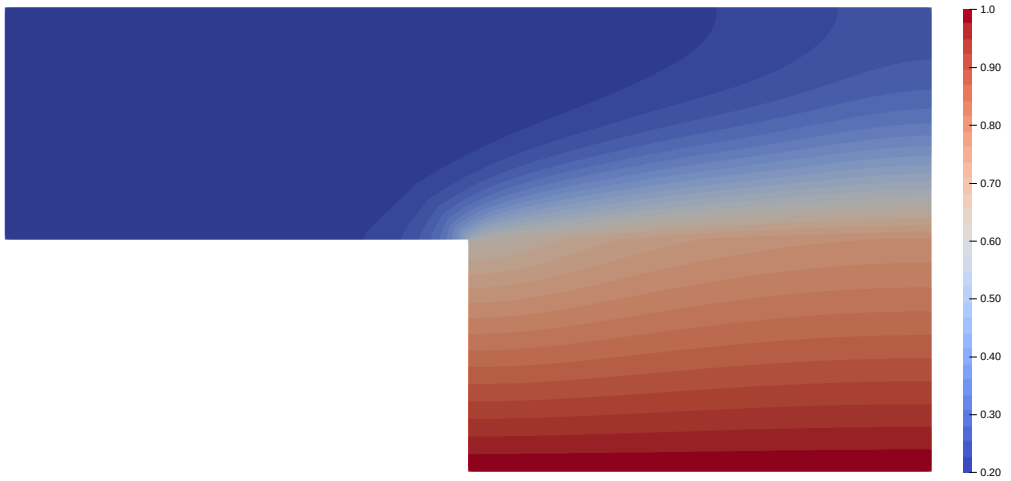
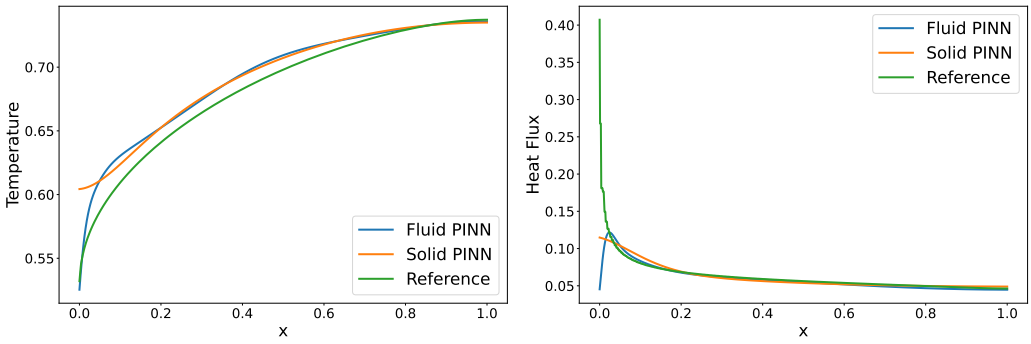


Figure 25: Temperature prediction using the separate training method



(a) Temperature comparison (b) Heat flux comparison

Figure 26: Comparison, along the interface, of the temperature (a) and heat flux (b) predicted by the two models and the FreeFem reference

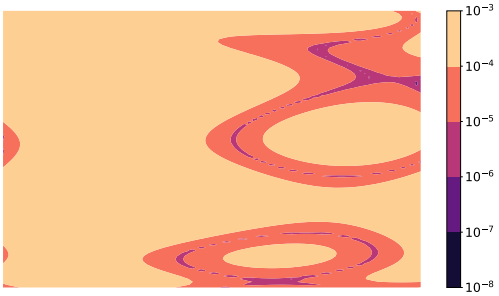
(a) **Fluid** related metrics

Posterior error	
Temperature	6.9×10^{-5}
Velocity (u)	8.3×10^{-8}
Velocity (v)	2.5×10^{-8}
Residual error	
Slip	2.7×10^{-8}
Inlet (Velocity)	4.2×10^{-8}
Inlet (Temperature)	5.5×10^{-8}
Outlet	6.0×10^{-10}
Neumann	1.3×10^{-4}
PDE (All)	3.8×10^{-3}
PDE (Heat)	1.5×10^{-2}
PDE (Continuity)	4.3×10^{-7}
PDE (Momentum x)	1.2×10^{-5}
PDE (Momentum y)	8.0×10^{-7}

(b) **Solid** related metrics

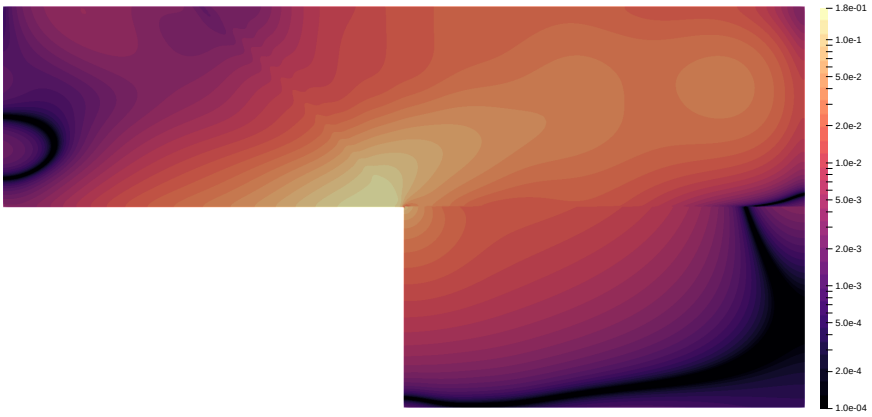
Posterior error	
Temperature	4.7×10^{-5}
Residual error	
Dirichlet	1.0×10^{-7}
Neumann	9.5×10^{-6}
PDE	3.5×10^{-6}

Table 5: MSE for the physical quantity, and the PDE and conditions residuals for the split training



(a) Absolute relative error u

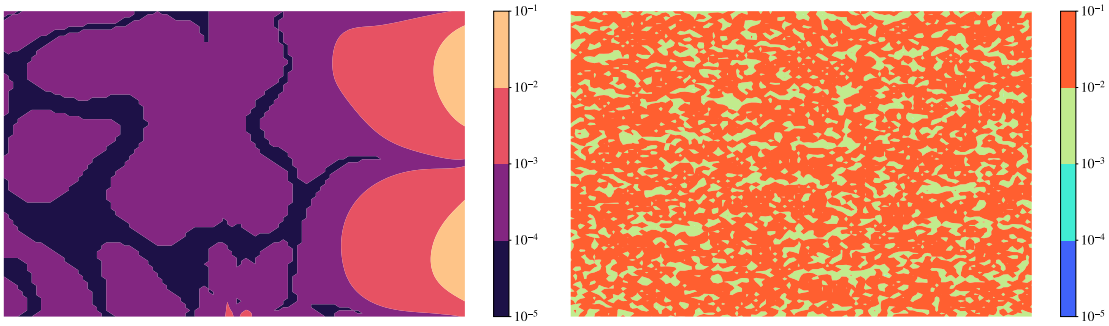
(b) Absolute error v



(c) Absolute relative error T

Figure 27: Absolute relative error T

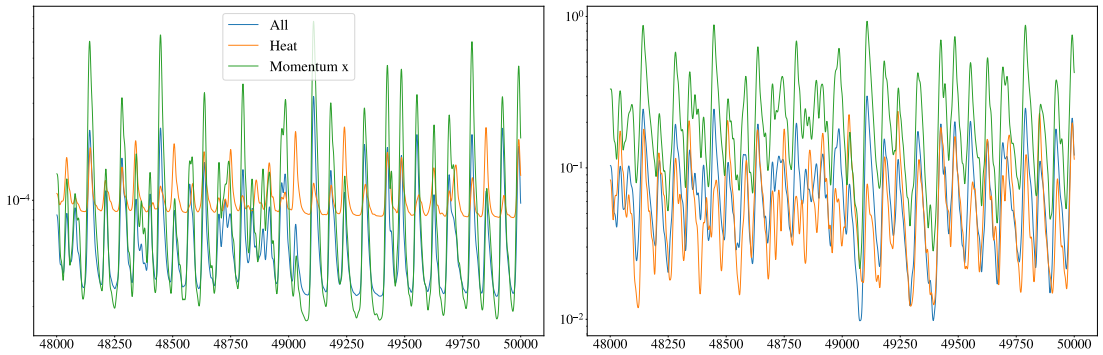
Complementary analysis Figure 28 shows the comparison of the absolute relative error for u and the momentum x residual on the fluid domain using the separate training. This shows that the error is not necessarily related to the residuals. As the residual appears to be minimized and looks like noise while the error is structured and concentrated near the outlet.



(a) Absolute relative error u (b) Map of the Momentum x residual

Figure 28: Comparison of the absolute relative error for u and the momentum x residual on the fluid domain using the separate training

Figure 29 illustrates a comparison of the loss and gradient norms for the three PDE residuals (momentum x, heat, and all) over the final 2,000 iterations of separate training. Both the loss and gradient norms are filtered using a Gaussian filter with $\sigma = 4$ and presented on a semi-logarithmic scale. The figure depicts a non-trivial relationship between the gradients and the losses. Meaning that the residual heat loss is higher than the residual momentum x residual, whereas the gradient norm of the momentum x residual is smaller than the gradient norm of the residual heat. This suggests that balancing the losses using only the statistics of the loss value may not be appropriate. Furthermore, the different terms within the PDE residual do not contribute equally, making it necessary to balance each term of the PDE residual instead of computing a single weight for the PDE residual.



(a) Losses (b) Gradients norm

Figure 29: Comparison of the loss and gradients for the three PDE residuals, momentum x, heat and all, on the last 2,000 of the separate training

Discussion Three different training approaches are explored and compared: separate training, joint training, and split training. In separate training, both fluid and solid domains are trained independently using their respective physics equations and boundary conditions. In joint training, both domains are trained together as a single coupled system. In split training, the fluid sub-PINN is divided into two sub-PINNs, one predicting temperature and the other predicting velocities and pressure. Discussion and analysis of the results reveal several key insights and challenges:

Interface Phenomena All training methods face challenges in accurately predicting temperature and heat flux at the interface between the fluid and solid domains. This is particularly evident near singularities or abrupt changes, leading to difficulties in transitioning between Neumann and coupling conditions.

Temperature Singularity The neural networks face a notable difficulty in accurately capturing the non-linearity of temperature and heat flux near the singularity point. This challenge can be attributed to the characteristics of the neural network architecture and its activation function, which in this case is the hyperbolic tangent. While neural networks are powerful tools for approximating a wide range of functions, including complex and nonlinear ones, they have limitations when dealing with singularities and discontinuities.
The activation function used in neural networks, such as tanh, is a continuous and differentiable function across its entire domain. This property allows the neural network to smoothly approximate functions that are continuous and differentiable. However, it also implies that the neural network struggles to accurately capture functions that exhibit sharp changes, abrupt transitions, or singularities. In the case of temperature and heat flux near the singularity point, where there is a rapid change in behavior, the neural network's smooth approximation might not adequately represent the actual behavior of the temperature field.

Dominance of Energy Equation The energy conservation equation (PDE Heat) plays a dominant role in affecting the accuracy of the overall solution, leading to relatively higher errors in its prediction compared to other equations. The split training aims to address this issue by dissociating the energy conservation equation and the Navier-Stokes equations but does not succeed. This failure may either come from the singularity described previously or from the neural network capacity. For the latter using bigger neural networks or adaptive activation functions may help. Another possibility is the spectral bias [23, 24] which is the fact that NNs are biased toward linear solution making it hard to learn the local non-linearity. To overcome this problem, Wang et al. introduce Fourier embedding [24, 13].
Instead of splitting the training, another way to tackle this phenomenon is to use the loss balancing method, introduced in subsection 1.4, to adapt the contributions of the various residuals in the optimization process

Joint Training Optimization Joint training, aided by the BFGS optimizer, shows slight improvements in losses and metrics compared to separate training. This suggests that using a more advanced optimization technique can refine the training process.

Split Training Splitting the fluid sub-PINN into two separate networks for temperature and velocity/pressure shows promising results, especially in improving the accuracy of velocity predictions. This approach reduces the complexity of the loss function and allows for separate optimization of different physics equations.

Overall, this study underscores the challenges and complexities involved in accurately coupling fluid dynamics and heat transfer phenomena using neural networks. The results from different training methods provide insights into potential strategies for improving predictions and optimizing neural network architectures for multiphysics simulations. Further research is needed to refine the training techniques, address spectral bias, and develop robust strategies for handling challenging regions in the computational domain.

For a similar conjugate heat transfer problem, NVIDIA's Simnet simulation framework uses Fourier features to approach the spectral distortion problem [25]. In the Modulus library, for example, the models are trained using the "hFTB" method [26, 27], which consists in replacing the coupling conditions with Dirichlet conditions in one domain and Robin conditions in another domain, and then iterating to make these conditions converge.

2.2.2 Natural convection Scenarios

The objective of this test case is to experiment with the coupling with another phenomenon, natural convection. Natural convection is the phenomenon where warmer fluid rises and cooler fluid sinks in a self-driven, buoyancy-induced circulation. Hence in this test case, the temperature and velocities are fully coupled inside the fluid. Let's take the case of a heated cavity separated by a wall introduced in [28] to have a baseline.

The domain, displayed in Figure 30, consists of two cavities separated by a wall. The wall is at distance $X_p = 0.5$ with a length of $T_p = 0.1$. On all borders, no-slip condition and null Neumann for the Temperature are imposed. The two vertical borders correspond to Dirichlet conditions with the hot and cold temperatures on the right and left borders. Finally, at the two interfaces continuity of the temperature and the heat flux is enforced.

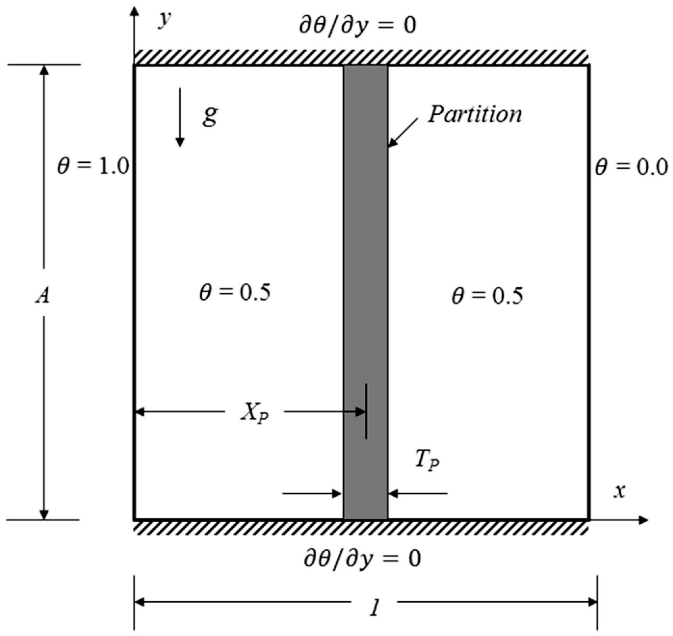


Figure 30: Schematic of the physical system under consideration, the computational domain and the initial and boundary conditions from [28]

Similarly to the previous test case, the fluid is governed by the Navier-Stokes but with a dimensionless form with the Boussinesq approximation for buoyancy to place ourselves in the same case of the baseline. To perform the nondimensionalization, the coordinates X and Y are divided by the length L. The velocities u and v are divided by a reference velocity U_{ref}. The pressure is scaled with respect to U_{ref} and the density ρ_f and the temperature is scaled accordingly to the hot and cold temperatures.

Thus $x = \dfrac{X}{L}$, $y = \dfrac{Y}{L}$, $u = \dfrac{U}{U_{ref}}$, $v = \dfrac{V}{U_{ref}}$, $p = \dfrac{P}{\rho_f U_{ref}^2}$, $\theta = \dfrac{T - T_c}{T_h - T_c}$, $\theta_{ref} = \dfrac{T_h + T_c}{2}$

$$\begin{cases} \dfrac{\partial u}{\partial x} + \dfrac{\partial v}{\partial y} = 0 & (26) \\ u\dfrac{\partial u}{\partial x} + v\dfrac{\partial u}{\partial y} + \dfrac{\partial p}{\partial x} - \sqrt{\dfrac{Pr}{Ra}}\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right) = 0 & (27) \\ u\dfrac{\partial v}{\partial x} + v\dfrac{\partial u}{\partial y} + \dfrac{\partial p}{\partial y} - \sqrt{\dfrac{Pr}{Ra}}\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right) + \theta - \theta_{ref} = 0 & (28) \\ u\dfrac{\partial \theta}{\partial x} + v\dfrac{\partial \theta}{\partial y} - \sqrt{\dfrac{1}{Pr\,Ra}}\left(\dfrac{\partial^2 \theta}{\partial x^2} + \dfrac{\partial^2 \theta}{\partial y^2}\right) = 0 & (29) \end{cases}$$

where $U_{ref} = \dfrac{\alpha_f}{H}\sqrt{Pr\,Ra}$ with $Pr = \dfrac{\nu_f}{\alpha_f}$ is the Prandtl number, and $Ra = \dfrac{g\beta(T_h - T_c)H^3}{\nu_f \alpha_f}$ is the Rayleigh number. The fluid is assumed to be air, thus $Pr = 0.71$. The complexity of the problem is determined by the Rayleigh number, the higher it is the more difficult the resolution will be, with higher variations in solution.

On the solid, we want to solve,

$$\frac{k_r}{\sqrt{Pr\,Ra}}\left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2}\right) = 0, \tag{30}$$

where $k_r = \dfrac{k_s}{k_f}$ is the thermal conductivity ratio. In the following test case $k_r = 1$.

To solve this problem we use 3 NNs, one per fluid domain and one for the solid. The fluid sub-PINNs have 3 layers with 50 units each, and the solid sub-PINN has 3 layers with 20 units each. The training is performed using the joint training with 50,000 epochs of Adam followed by 1,000 epochs of BFGS.

In this test case, PINNs often converge towards the trivial solution of the residuals equations which is a constant temperature in both domains and zero velocities in the fluid part. This solution appears more frequently for high Rayleigh values. To avoid this incorrect solution, we use the Curriculum Regularization introduced by Krishnapriyan et al [16], it consists in training the PINNs on lower Rayleigh values, which are easier for the PINNs to learn, and then gradually move to training the PINN on higher values. The methodology we chose is to train the PINNs for $Ra = 10^5$, then to use the trained PINNs to re-train them for $Ra = 10^6$, and then $Ra = 10^7$.

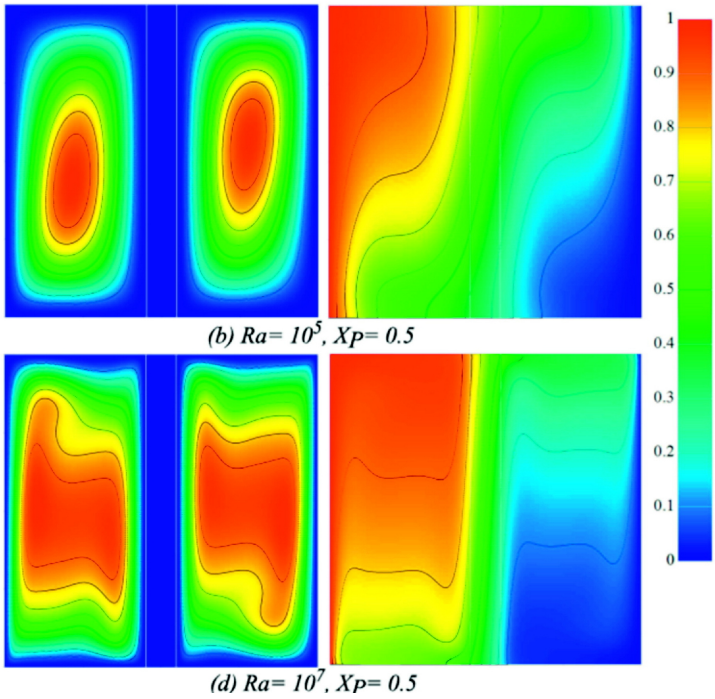
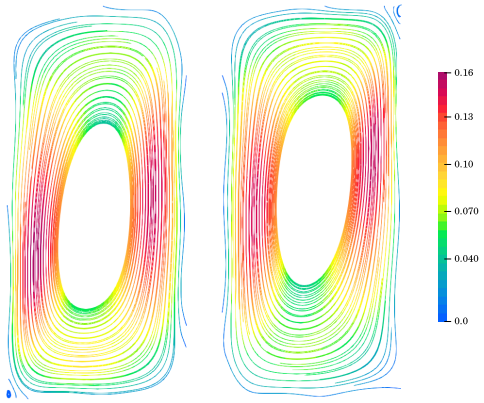
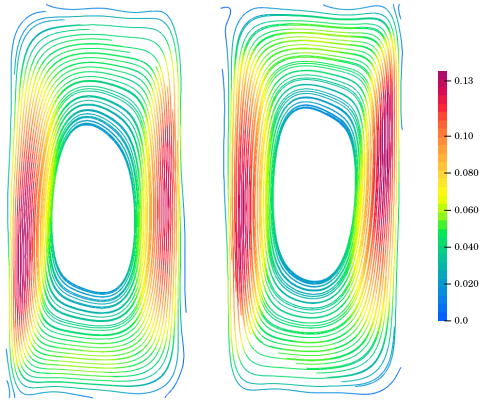


Figure 31: Streamlines (the left in each sub-figure) and temperature contours (the right in each sub-figure) at the steady-state stage obtained numerically for Rayleigh values at $10^5, 10^7$ from [28]

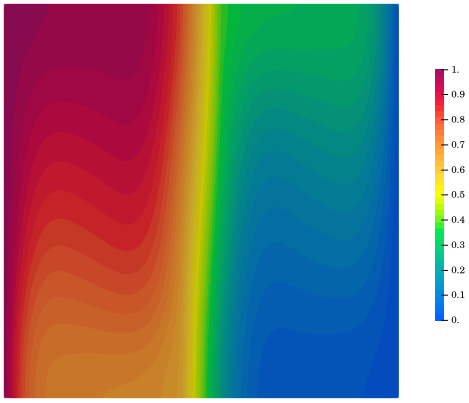


(a) Streamlines at $Ra = 10^5$

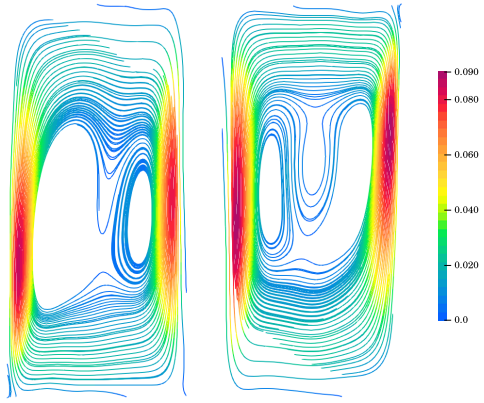
(b) Temperature at $Ra = 10^5$



(c) Streamlines at $Ra = 10^6$



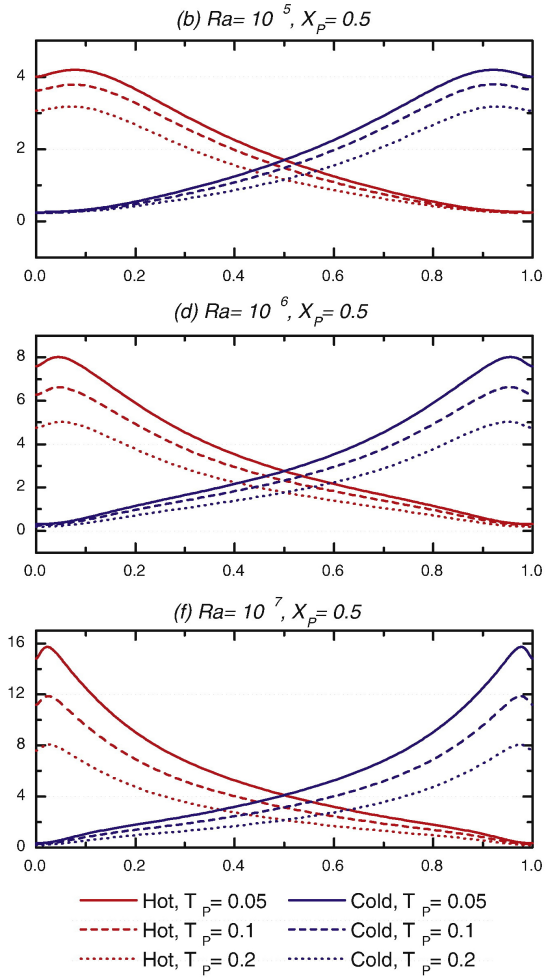
(d) Temperature at $Ra = 10^6$



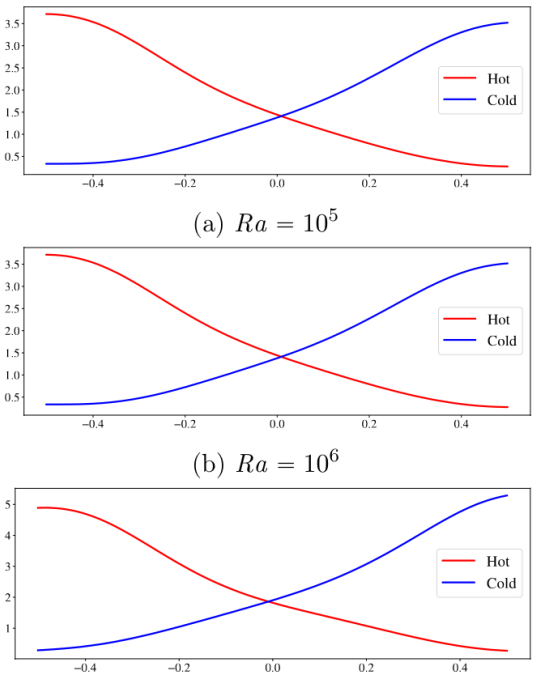
(e) Streamlines at $Ra = 10^7$

(f) Temperature at $Ra = 10^7$

Figure 32: Residuals



(a) References



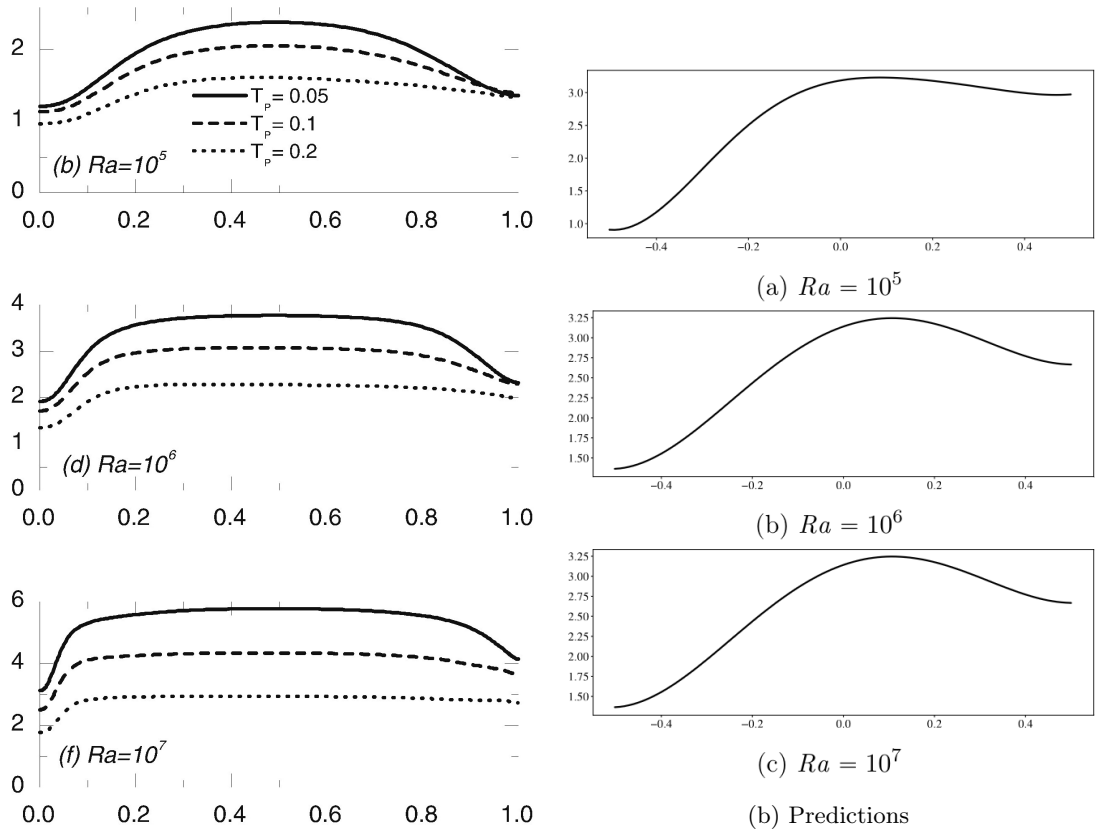
(a) $Ra = 10^5$

(b) $Ra = 10^6$

(c) $Ra = 10^7$

(b) Predictions

Figure 33: The vertical profiles of the local Nusselt number on the hot and cold sidewalls for different Rayleigh values at $10^5, 10^6, 10^7$



(a) References

(a) $Ra = 10^5$

(b) $Ra = 10^6$

(c) $Ra = 10^7$

(b) Predictions

Figure 34: The vertical profiles of the local Nusselt number on the left side of partition for different Rayleigh values at $10^5, 10^6, 10^7$

In conclusion, the natural convection scenarios within the heated cavity yielded noteworthy results and posed challenges for PINNs. The PINNs obtained satisfactory results for Rayleigh values up to 10^6. They captured the streamlines and temperature contours quite well, as illustrated in Figure 32.

For a Rayleigh value of 10^5, Figure 33 illustrates that the local Nusselt number's vertical profiles reveal a good agreement between the PINN predictions and the reference values. This suggests that the PINNs accurately captured the heat transfer characteristics for such scenarios. However, for a Rayleigh value of 10^6, the PINN's maximum Nusselt value fails to reach the reference value. However, Figure 34 shows that on the interface the flux are not correctly modeled.

For a Rayleigh value of 10^7, PINNs were unable to learn meaningful solutions, despite using Curriculum regularization. In an attempt to improve their performance, Chebyshev sampling was used to refine the grid near the border of the fluid domain. However, this approach ultimately resulted in the PINNs converging towards the trivial solution.

In summary, it appears that PINNs have potential in capturing natural convection phenomena for moderate Rayleigh values (10^5 and 10^6); however, their performance declines for more challenging scenarios (10^7). While the Curriculum regularization technique improved performance, further investigation is needed to overcome the limitations in capturing highly turbulent and complex fluid dynamics. For high Rayleigh values, a smoother grid with refinement near the

borders is necessary to capture the local phenomena. However, accurately approximating these phenomena may require larger neural networks or Fourier features.

3 Conclusion

The exploration of PINNs in different test cases has revealed both strengths and limitations in addressing complex physical phenomena. Three test cases were considered in this study, namely, heat transfer in a solid, conjugate heat transfer scenarios with forced convection in a laminar flow, and natural convection in a heated cavity. The first test case displayed the potential of PINNs for parameterized resolution, effectively handling piece-wise constants through a domain decomposition approach. The preceding two test cases demonstrate the potential of multiphysics coupling. Nevertheless, there were several challenges encountered, such as complications in achieving low errors, propagating nearby singularities, and effectively learning scenarios with high Rayleigh values.

Addressing the limitations of PINNs requires a multifaceted approach. To achieve high-resolution results, it may be beneficial to leverage larger neural networks, refine and smooth sampling strategies. In addition, the capabilities of PINNs could be improved by using more complex architectures or Fourier embedding. However, such efforts require effective parallelism and multi-GPU training to guarantee computational feasibility. Additional convergence improvements can be achieved through advanced training methods such as loss balancing, adaptive sampling, or a better-fitting optimizer.

References

[1] Nathan Baker, Frank Alexander, Timo Bremer, Aric Hagberg, Yannis Kevrekidis, Habib Najm, Manish Parashar, Abani Patra, James Sethian, Stefan Wild, Karen Willcox, and Steven Lee. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. Technical report, USDOE Office of Science (SC), Washington, D.C. (United States), February 2019.

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045.

[3] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, June 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00314-5.

[4] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8.

[5] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. *Journal of Scientific Computing*, 92(3): 88, July 2022. ISSN 1573-7691. doi: 10.1007/s10915-022-01939-z.

[6] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, May 2021.

[7] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, June 2020. ISSN 00457825. doi: 10.1016/j.cma.2020.113028.

[8] Ameya D. Jagtap & George Em Karniadakis. Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations. *Communications in Computational Physics*, 28(5):2002–2041, June 2020. ISSN 1815-2406, 1991-7120. doi: 10.4208/cicp.OA-2020-0164.

[9] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5.

[10] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40): eabi8605, September 2021. doi: 10.1126/sciadv.abi8605.

[11] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, January 2021. ISSN 1064-8275. doi: 10.1137/20M1318043.

[12] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications, March 2023.

[13] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An Expert's Guide to Training Physics-informed Neural Networks, August 2023.

[14] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, January 2022. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110768.

[15] John Taylor, Wenyi Wang, Biswajit Bala, and Tomasz Bednarz. Optimizing the optimizer for data driven deep neural networks and physics informed neural networks, May 2022.

[16] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 26548–26560. Curran Associates, Inc., 2021.

[17] QiZhi He, David Barajas-Solano, Guzel Tartakovsky, and Alexandre M. Tartakovsky. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *Advances in Water Resources*, 141:103610, July 2020. ISSN 0309-1708. doi: 10.1016/j.advwatres.2020.103610.

[18] Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2239):20200334, July 2020. doi: 10.1098/rspa.2020.0334.

[19] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, January 2023. ISSN 0045-7825. doi: 10.1016/j.cma.2022.115671.

[20] Khemraj Shukla, Ameya D. Jagtap, and George Em Karniadakis. Parallel physics-informed neural networks via domain decomposition. *Journal of Computational Physics*, 447:110683, December 2021. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110683.

[21] TongSheng Wang, ZhiHeng Wang, Zhu Huang, and Guang Xi. Multi-domain physics-informed neural network for solving heat conduction and conjugate natural convection with discontinuity of temperature gradient on interface. *Science China Technological Sciences*, 65 (10):2442–2461, October 2022. ISSN 1869-1900. doi: 10.1007/s11431-022-2118-9.

[22] Wen Zhang and Jian Li. CPINNs: A coupled physics-informed neural networks for the closed-loop geothermal system. *Computers & Mathematics with Applications*, 132:161–179, February 2023. ISSN 08981221. doi: 10.1016/j.camwa.2023.01.002.

[23] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the Spectral Bias of Neural Networks, May 2019.

[24] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, October 2021. ISSN 0045-7825. doi: 10.1016/j.cma.2021.113938.

[25] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Max Rietmann, Jose del Aguila Ferrandis, Wonmin Byeon, Zhiwei Fang, and Sanjay Choudhry. NVIDIA SimNet{TM}: An AI-accelerated multi-physics simulation framework, December 2020.

[26] F. Duchaine, A. Corpron, L. Pons, V. Moureau, F. Nicoud, and T. Poinsot. Development and assessment of a coupled strategy for conjugate heat transfer with Large Eddy Simulation: Application to a cooled turbine blade. *International Journal of Heat and Fluid Flow*, 30(6): 1129–1141, December 2009. ISSN 0142727X. doi: 10.1016/j.ijheatfluidflow.2009.07.004.

[27] Sebastian Scholl, Bart Janssens, and Tom Verstraete. Stability of static conjugate heat transfer coupling approaches using Robin interface conditions. *Computers & Fluids*, 172: 209–225, August 2018. ISSN 00457930. doi: 10.1016/j.compfluid.2018.06.016.

[28] Mehdi Khatamifar, Wenxian Lin, S.W. Armfield, David Holmes, and M.P. Kirkpatrick. Conjugate natural convection heat transfer in a partitioned differentially-heated square cavity. *International Communications in Heat and Mass Transfer*, 81:92–103, February 2017. ISSN 07351933. doi: 10.1016/j.icheatmasstransfer.2016.12.003.

A Analytical solutions of the test cases

A.1 Heat transfer: single domain

Let's look at the analytical solution in the case $\dfrac{\partial T}{\partial \theta} = 0$ and $r \neq 0$.

We have,

$$\frac{1}{r}\frac{\partial}{\partial r}\left(rk\frac{\partial T}{\partial r}\right) = 0$$

Thus,

$$\frac{\partial}{\partial r}\left(rk\frac{\partial T}{\partial r}\right) = 0$$

Integrating again with respect to r gives:

$$rk\frac{\partial T}{\partial r} = C_1$$

where C_1 is a constant of integration.Thus,

$$\frac{\partial T}{\partial r} = \frac{C_1}{k}\frac{1}{r}$$

Integrating once more with respect to r gives:

$$T(r) = \frac{C_1}{k}\ln r + C_2,$$

where C_2 is another constant of integration.

We can now use the boundary conditions $T(r_1) = T_1$ and $T(r_2) = T_2$ to determine the values of C_1 and C_2. We know that

$$T(r_1) = \frac{C_1}{k}\ln r_1 + C_2 = T_1,$$

$$T(r_2) = \frac{C_1}{k}\ln r_2 + C_2 = T_2.$$

Solving $T_1 - T_2$ with $k = 1$ yields,

$$C_1 = \frac{(T_1 - T_2)}{\ln\left(\frac{r_2}{r_1}\right)},$$

$$C_2 = T_1 - \frac{T_1 - T_2}{\ln\left(\frac{r_2}{r_1}\right)}\ln r_1.$$

Substituting these values back into the solution, we obtain:

$$T(r) = T_1 + \frac{T_1 - T_2}{\ln\left(\frac{r_2}{r_1}\right)}\ln\left(\frac{r}{r_1}\right)$$

A.2 Heat transfer: two domains

By using the previous analytical solution we know that

$$T(r) = \frac{c_1}{k_1} \ln r + d_1 \text{ on } \Omega_1$$
$$T(r) = \frac{c_2}{k_2} \ln r + d_2 \text{ on } \Omega_2$$

The boundary conditions yield

$$T_1(r_1) = \frac{c_1}{k_1} \ln r + d_1 = T_1 \text{ on } \Omega_1$$
$$T_2(r_2) = \frac{c_2}{k_2} \ln r + d_2 = T_2 \text{ on } \Omega_2$$

Thus,

$$d_1 = T_1 - \frac{c_1}{k_1} \ln \left(\frac{r}{r_1} \right)$$
$$d_2 = T_2 - \frac{c_2}{k_2} \ln \left(\frac{r}{r_2} \right)$$

Finally, we obtain

$$T_1(r) = T_1 + \frac{c_1}{k_1} \ln \left(\frac{r}{r_1} \right) \text{ on } \Omega_1$$
$$T_2(r) = T_2 + \frac{c_2}{k_2} \ln \left(\frac{r}{r_2} \right) \text{ on } \Omega_2$$

The continuity of the solution and the flux at the interface yields, $\forall s \in \Gamma$

$$k_1 \frac{\partial T_1}{\partial r} = k_2 \frac{\partial T_2}{\partial r} \tag{31}$$
$$T_1(s) = T_2(s) \tag{32}$$

By integrating Equation 31 we obtain

$$k_1 \frac{c_1}{k_1} \frac{1}{s} = k_2 \frac{c_2}{k_2} \frac{1}{s}$$

Thus, $c_1 = c_2$

Equation 32 yields,

$$T_1 + \frac{c_1}{k_1} \ln \left(\frac{s}{r_1} \right) = T_2 + \frac{c_2}{k_2} \ln \left(\frac{s}{r_2} \right)$$

Thus,

$$T_1 - T_2 = c_1 \left[\frac{1}{k_1} \ln \left(\frac{s}{r_1} \right) - \frac{1}{k_2} \ln \left(\frac{s}{r_2} \right) \right]$$

Finally, we have

$$c_1 = (T_1 - T_2) \cdot \left[\frac{1}{k_1} \ln \left(\frac{s}{r_1} \right) - \frac{1}{k_2} \ln \left(\frac{s}{r_2} \right) \right]^{-1} \tag{33}$$

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399