Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY

MNRAS **524,** 32–42 (2023) https://doi.org/10.1093/mnras/stad1810
Advance Access publication 2023 June 16

Modelling force-free neutron star magnetospheres using physics-informed neural networks

Jorge F. Urbán,[1] Petros Stefanou,[1,2★] Clara Dehman [3,4] and José A. Pons[1]

[1] *Departament de Física Aplicada, Universitat d'Alacant, Ap. Correus 99, E-03080 Alacant, Spain*
[2] *Departament d'Astronomia i Astrofísica, Universitat de Valéncia, Dr Moliner 50, E-46100, Burjassot, Valéncia, Spain*
[3] *Institute of Space Sciences (ICE-CSIC), Campus UAB, Carrer de Can Magrans s/n, E-08193, Barcelona, Spain*
[4] *Institut d'Estudis Espacials de Catalunya (IEEC), Carrer Gran Capitá 2–4, E-08034 Barcelona, Spain*

Accepted 2023 June 12. Received 2023 May 31; in original form 2023 March 18

ABSTRACT

Using physics-informed neural networks (PINNs) to solve a specific boundary value problem is becoming more popular as an alternative to traditional methods. However, depending on the specific problem, they could be computationally expensive and potentially less accurate. The functionality of PINNs for real-world physical problems can significantly improve if they become more flexible and adaptable. To address this, our work explores the idea of training a PINN for general boundary conditions and source terms expressed through a limited number of coefficients, introduced as additional inputs in the network. Although this process increases the dimensionality and is computationally costly, using the trained network to evaluate new general solutions is much faster. Our results indicate that PINN solutions are relatively accurate, reliable, and well behaved. We applied this idea to the astrophysical scenario of the magnetic field evolution in the interior of a neutron star connected to a force-free magnetosphere. Solving this problem through a global simulation in the entire domain is expensive due to the elliptic solver's needs for the exterior solution. The computational cost with a PINN was more than an order of magnitude lower than the similar case solved with a finite difference scheme, arguably at the cost of accuracy. These results pave the way for the future extension to three-dimensional of this (or a similar) problem, where generalized boundary conditions are very costly to implement.

Key words: magnetic fields – methods: numerical – stars: magnetars – stars: neutron – neural networks – physics-informed neural networks.

1 INTRODUCTION

Deep learning (DL) is a subset of techniques comprehended in machine learning that is fundamentally based on multilayered neural networks (NNs). In recent years, DL has been widely used to perform a large variety of tasks. Examples include (among many others) computer vision (to perform image classification; Traore, Kamsu-Foguem & Tangara 2018), face recognition (Lawrence et al. 1997) or medical diagnosis (Kugunavar & Prabhakar 2021), speech recognition (Chan et al. 2015), and robotics to emulate human-like walking and running or mobile navigation in pedestrian environments (Hayat & Mall 2013).

Physics-informed neural network (PINN; Raissi, Perdikaris & Karniadakis 2019) is a DL approach used to numerically approximate the solution of non-linear partial differential equations (PDEs). The original idea was born more than 20 yr ago (Lagaris, Likas & Fotiadis 1997), but the lack of the necessary computational resources made it complicated to put it into practice. In recent years, we account with graph-based automatic differentiation, as well as different frameworks that support computations in CPUs and GPUs, such as Tensorflow or Pytorch, and a dramatic increase in computational power. These factors, combined with a blooming interest in machine-learning applications in science, have given birth to this promising new field. PINNs have been used, among many other applications, in fluid dynamics (Cai et al. 2021), nuclear reactor dynamics (Schiassi et al. 2022), radiative transfer (Mishra & Molinaro 2021; Chen et al. 2022), and black-hole spectroscopy (Luna et al. 2022). PINNs incorporate the underlying physical laws that govern a system (the PDEs) and then optimize the NN so that the residual of the PDE is minimal. Unlike more traditional DL approaches in other fields, PINNs do not require large amounts of data – or any data at all – for the training of the NN.

Compared to classical finite-differences/finite-elements methods, PINNs still fall short in terms of efficiency and precision. However, they present some advantages as flexible, multipurpose PDE solvers. For example, the PINN formulation allows us to solve problems in arbitrary, unstructured meshes without using high-resolution, memory-consuming grids. In addition, once a PINN is trained for a general problem, the calculation of a new solution is swift and consists only of the few operations needed during the forward pass through the network. This is a potential advantage in speed compared to classical methods.

In this work, we assess the applicability of a PINN solver in elliptic problems. In particular, we focus on the problem of modelling force-free (FF) magnetospheres of neutron stars (NS) in the non-rotating, axisymmetric limit. There is a wealth of work in the literature formulating this problem in terms of the GS equation (Glampedakis,

★ E-mail: petros.stefanou@uv.es

© 2023 The Author(s)
Published by Oxford University Press on behalf of Royal Astronomical Society

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

Lander & Andersson 2014; Pili, Bucciantini & Del Zanna 2015; Akgün et al. 2016; Akgün et al. 2017; Akgün et al. 2017; Kojima 2017; Akgün et al. 2018), which gives us the opportunity to make detailed comparisons and draw robust conclusions on the performance and generalizability of the PINN solver. In particular, we have compared our results with our previous work (Akgün et al. 2016), where a finite-difference scheme based on an iterative method solving a block-tridiagonal system was used.

The paper is organized as follows: in Section 2, we give a brief mathematical overview of the physics of NS magnetospheres. In Section 3, we describe in detail how the PINN solver is built. We present the solutions acquired by the PINN solver with error estimates in Section 4. In Section 5, we demonstrate PINN's capabilities through an astrophysical application. Section 6 is dedicated to discuss our main results and how to improve and generalize our approach to face more difficult problems in the near future.

2 MODELLING AXISYMMETRIC FF MAGNETOSPHERES

In the FF regime, and considering that the contribution of gravity, inertia, plasma pressure, and rotation in the dynamics of an NS magnetosphere is negligible compared to the magnetic force, the force-balance equation reduces to the simple form

$$(\nabla \times \boldsymbol{B}) \times \boldsymbol{B} = 0, \tag{1}$$

where $\boldsymbol{B}$ denotes the magnetic field. This regime is better suited for magnetar magnetospheres (Thompson & Duncan 1995, 1996), because magnetars are slow rotators and the absence of any rotationally induced electric fields is a very good approximation.

In axisymmetry, we can express the magnetic field in terms of a poloidal and a toroidal stream function $\mathcal{P}$ and $\mathcal{T}$. We will follow the notation and formalism as in Akgün et al. (2016). We refer the interested reader to that work for a more detailed description. In spherical coordinates (r, θ, ϕ) and in terms of the stream functions, the magnetic field reads

$$\boldsymbol{B} = \nabla\mathcal{P} \times \nabla\phi + \mathcal{T}\nabla\phi, \tag{2}$$

where $\nabla\phi = \frac{\boldsymbol{e}_\phi}{r\sin\theta}$ with $\boldsymbol{e}_\phi$ being the azimuthal unit vector. Substituting equations (2) into (1), the ϕ-component of the equation gives

$$\nabla\mathcal{P} \times \nabla\mathcal{T} = 0, \tag{3}$$

which simply states that $\mathcal{T} = \mathcal{T}(\mathcal{P})$ must be a function of $\mathcal{P}$ (or vice versa). The remaining components give us the so-called GS equation

$$\triangle_{\rm GS}\mathcal{P} + G(\mathcal{P}) = 0. \tag{4}$$

Here, $G(\mathcal{P}) = \mathcal{T}(\mathcal{P})\frac{d\mathcal{T}}{d\mathcal{P}}$ is the source term accounting for the presence of currents in the magnetosphere and $\triangle_{\rm GS}$ is the GS operator

$$\triangle_{\rm GS} \equiv r^2 \sin^2\theta\, \nabla \cdot \left(\frac{1}{r^2\sin^2\theta}\nabla\right). \tag{5}$$

For convenience, we will use compactified spherical coordinates (see Stefanou, Pons & Cerdá-Durán 2023) (q, μ, ϕ), where $q = \frac{1}{r}$ and $\mu = \cos\theta$ instead of the usual (r, θ, ϕ). In this set of coordinates, the GS operator reads

$$\triangle_{\rm GS} \equiv q^2\partial_q\left(q^2\partial_q\right) + \left(1-\mu^2\right)q^2\partial_{\mu\mu}. \tag{6}$$

To solve equation (4), we must also provide BCs and the functional form of the source term, that is, $\mathcal{T}(\mathcal{P})$. The particular functional form is arbitrary and different choices are possible. In Akgün et al. (2016), they used

$$\mathcal{T}(\mathcal{P}) = s\left(\mathcal{P} - \mathcal{P}_c\right)^\sigma \Theta\left(\mathcal{P} - \mathcal{P}_c\right), \tag{7}$$

where Θ is the Heaviside function, and s, $\mathcal{P}_c$, and σ are parameters that control the relative strength of the toroidal and poloidal components, the region where the toroidal field is non-zero and the non-linearity of the model. However, this has the limitation that it assumes $\mathcal{P} > 0$. A possible generalization overcoming this constraint is

$$\mathcal{T}(\mathcal{P}) = s\left(|\mathcal{P}| - \mathcal{P}_c\right)^\sigma \Theta\left(|\mathcal{P}| - \mathcal{P}_c\right), \tag{8}$$

which allows for currents even for negative values of $\mathcal{P}$. We have explored different options but, for simplicity and the purpose of this paper, we will use a quadratic function for the astrophysical application in Section 5.2, defined as follows:

$$\mathcal{T}(\mathcal{P}) = s_1\mathcal{P} + s_2\mathcal{P}^2. \tag{9}$$

We impose BCs for $\mathcal{P}$ at the surface of the star ($q = 1$), at radial infinity ($q = 0$) and at the axis ($\mu = \pm 1$). Regularity and symmetry of the problem lead to

$$\mathcal{P}(q, \mu = \pm 1) = \mathcal{P}(q = 0, \mu) = 0.$$

In particular, one advantage of compactifying the radial coordinate (going from r to q) is to make it easier to impose BCs at radial infinity: rather than imposing a specific decay rate at large r, we can impose Dirichlet BCs at just one point ($q = 0$). This reduces unwanted numerical noise from the external boundary.

At the surface, we must provide the function $\mathcal{P}(\mu)$. Our implementation of BCs in an NN must be as general as possible but keep the number of parameters reasonably low. A reasonable and practical choice is to use some decomposition of the arbitrary function in terms of orthonormal polynomials. Considering the symmetry of our problem and that we are working with functions describing magnetic fields, the natural choice is to express $\mathcal{P}(q = 1)$ in terms of coefficients of a Legendre polynomial expansion. We use the following decomposition:

$$\mathcal{P}(q = 1, \mu) = \left(1 - \mu^2\right)\sum_{l=1}^{l_{\max}} \frac{b_l}{l} P_l'(\mu), \tag{10}$$

where l is the order of the multipole ($l = 1$ corresponds to a dipole), P_l are the Legendre polynomials (not to be confused with $\mathcal{P}$, the poloidal flux function), and the prime denotes differentiation with respect to μ. Thus, the BC at the surface is completely determined by prescribing the b_l coefficients[1].

3 METHODOLOGY

3.1 Neural networks

NNs are universal approximators of mathematical functions (Hornik, Stinchcombe & White 1989). They are the result of compositions of simple but non-linear transformations at different layers. The way that these layers are interconnected indicates the NN *architecture*. Each layer contains a number of *neurons* that transform the inputs received from the previous layer and then pass the result to the next layer. This transformation is done in two basic steps. A linear combination of the inputs received and an evaluation through a non-linear *activation function*. Mathematically, this process can be expressed for each layer as follows:

$$\boldsymbol{a}^j = g(\mathbf{W}^j\boldsymbol{a}^{j-1} + \boldsymbol{b}^j), \tag{11}$$

[1]The $1/l$ normalization factor and the coefficients b_l in the expansion have been chosen to match the b_l coefficients used in Dehman et al. (2023).

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

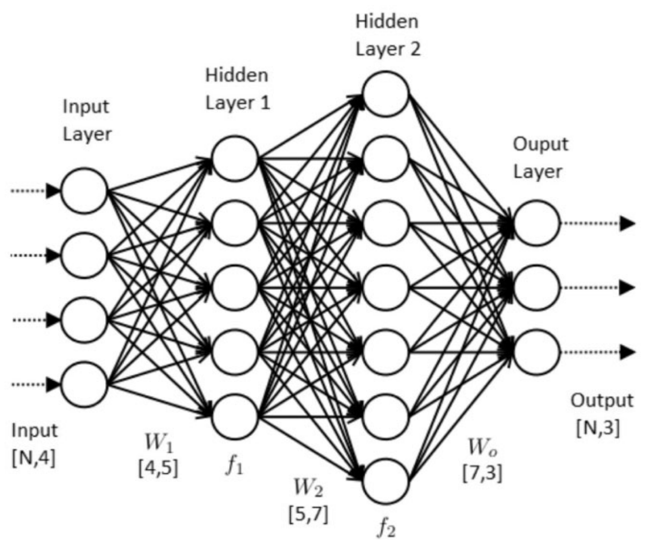

Figure 1. A schematic representation of a deep fully connected NN.

where $\boldsymbol{a}^j$ is a vector containing the values of the neurons in layer j, $\mathbf{W}^j$ and $\boldsymbol{b}^j$ denote the *weight* matrix and the *bias* vector of the layer, and g is the non-linear activation function. The most commonly used type of architecture is the fully connected neural network (FCNN). Each neuron is connected to all the neurons of the previous layer and connects to all the neurons of the next layer. The first layer of an FCNN is called the *input layer*, where user-provided information (the *inputs*) enter the network. The last layer is called the *output layer*. This is the final result (the *output*) of the network. Fig. 1 shows a schematic representation of an FCNN. The set Ω of the weights and biases of all the layers constitute the trainable parameters of an NN. They can be adjusted so that the relation between the input and the output approximates a desired function u. The process of adjusting the trainable parameters is called *training* and involves the following steps:

(i) The network is initialized with a random set of weights and biases.

(ii) A number of inputs $\boldsymbol{x}$ is fed to the network. The network propagates the inputs through its layers, transforms them according to equation (11) and produces some outputs $\tilde{u}$ (the *prediction* of the network), which depend on the inputs and on the trainable parameters, so that $\tilde{u} = \tilde{u}(\boldsymbol{x}; \Omega)$. This is called a *forward pass*.

(iii) The difference between prediction and desired function is calculated. This is done through a *loss function*, which is an average global measure of the deviation of $\tilde{u}(\boldsymbol{x}; \Omega)$ from $u(\boldsymbol{x})$.

(iv) The value of the loss function depends on all the trainable parameters. Therefore, one can adjust these parameters so that the loss is minimized and $\tilde{u}(\boldsymbol{x}; \Omega)$ approximates $u(\boldsymbol{x})$. This is done by applying small corrections to each weight and bias, dictated by their contribution to the value of the loss. This process is called *backpropagation*.

(v) The backpropagation is performed using *gradient descent* based techniques, i.e. by calculating the gradients of the loss function with respect to the trainable parameters and then updating the trainable parameters by applying corrections to them that are proportional to those gradients.

(vi) All these gradient calculations would be very costly if they were to be calculated numerically. However, in modern machine-learning frameworks, all operations during a forward pass are recorded in *graphs*. This allows gradients to be calculated very efficiently and to machine accuracy by inverting the recorded operations, a technique called *Automatic Differentiation* (AD) (see Baydin et al. (2017) for a detailed description of AD in machine learning).

(vii) These steps are repeated iteratively in order to minimize the value of the loss. When convergence is achieved, the network's prediction should be a good approximation to the desired function.

The above procedure can be implemented using Tensorflow (Abadi et al. 2016), a freely available software framework for machine learning. It provides us with the tools and resources needed to develop and implement various machine-learning algorithms and models. Particularly, AD is implemented by the application programming interface (API) *GradientTape*,[2] which automatically records all the operations made during the forward pass and generates the corresponding graph used for calculating the gradients. On the other hand, different gradient descent optimization algorithms are implemented in the API Keras.[3] In this work, we used the ADAM algorithm (Kingma & Ba 2014) for all the calculations.

A slightly different version of equation (11), where any hidden layer is only interconnected with its closest neighbours, are the so-called *residual blocks*. A residual block is formed if, before the evaluation of the activation function at a certain layer j, we add the output of the K_{th} previous layer (a *skip connection*). Mathematically, the output $\boldsymbol{a}^j$ of a residual block corresponding to the layer j could be written as follows

$$\boldsymbol{a}^j = g(\mathbf{W}^j \boldsymbol{a}^{j-1} + \boldsymbol{b}^j + \boldsymbol{a}^{j-K}). \quad (12)$$

Stacking many of these blocks together forms a *Residual Neural Network* (ResNet), introduced first by He et al. (2015). ResNets were introduced in DL to address the problem in which, by increasing the depth of the NN, the gradients used to update the trainable parameters tend to diminish during backpropagation, making it difficult for the network to learn effectively (see He et al. (2016) for a detailed explanation). As ResNets are not explored in the literature of PINNs and are straightforward to implement by extending an FCNN, we have additionally considered this architecture.

3.2 Physics-informed neural networks

The standard way of training an NN accounts with data that consist of values of the true solution in a given discrete set of points in the input domain. However, this approach demands a large number of training examples to build a reliable relation between the inputs and the outputs. In particular, for astrophysical systems, this method would rely on data obtained through a large set of observations. The key novelty in PINNs is the incorporation of information about the physical laws into the training process. This is accomplished by minimizing the residual of the PDEs that govern the system, instead of the difference between prediction and real data/exact solution. Using data in the loss function is optional and sometimes may facilitate the optimization, but in practise is not necessary.

Consider that the function u that we aim to obtain with the PINN satisfies the following boundary value problem

$$\mathcal{L}u(\boldsymbol{x}) - G(\boldsymbol{x}, u(\boldsymbol{x})) = 0, \quad (13)$$

$$u|_{\partial\mathcal{D}} = f_b(\boldsymbol{x})|_{\boldsymbol{x}\in\partial\mathcal{D}}, \quad (14)$$

[2]https://www.tensorflow.org/api_docs/python/tf/GradientTape
[3]https://keras.io/api/

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

where $\mathcal{L}$ is a general non-linear differential operator, G is a source term, and $\boldsymbol{x}$ is a vector of coordinates in some domain $\mathcal{D}$. The PDE is subject to some BCs f_b at the boundary $\partial\mathcal{D}$ of the domain. If $\tilde{u}$ is an approximation to an exact solution given by a PINN, then equation (13) will have a residual, that is, the right-hand side will not be exactly zero. The smaller this residual is, the closer $\tilde{u}$ be to an exact solution. Thus, the loss function is precisely a suitable norm of this residual, defined as

$$\mathcal{J} = ||\mathcal{L}\tilde{u}(\boldsymbol{x};\Omega) - G(\boldsymbol{x}, \tilde{u}(\boldsymbol{x};\Omega))||, \tag{15}$$

where the derivatives of the function $\tilde{u}$ with respect to the coordinates $\boldsymbol{x}$ are also computed with AD (with the Tensorflow *GradientTape* API).

The function $\tilde{u}$ should also satisfy the BCs (14). There are different approaches to implement them. The most commonly used is to add a term in the loss function consisting of a norm of the residual of equation (14), so that the residuals of both equations (13) and (14) are minimized simultaneously, as follows:

$$\mathcal{J} = c_1||\mathcal{L}\tilde{u}(\boldsymbol{x};\Omega) - G(\boldsymbol{x}, \tilde{u}(\boldsymbol{x};\Omega))|| + c_2||\tilde{u}(\boldsymbol{x};\Omega) - f_b(\boldsymbol{x})||_{x\in\partial\mathcal{D}}, \tag{16}$$

where c_1 and c_2 are coefficients that control the relative weight of the two terms. Then, the solution of the boundary value problem is directly the output $\mathcal{N}$ of the PINN

$$\tilde{u}(\boldsymbol{x};\Omega) = \mathcal{N}(\boldsymbol{x};\Omega). \tag{17}$$

We believe that this is not the optimal way to impose BCs. The individual terms in equation (16) can differ significantly, which means that minimizing $\mathcal{J}$ does not guarantee that both the PDE and the BCs are satisfied with the same accuracy. In order to surpass this problem, the coefficients c_1, c_2 can be adjusted so that the relative contribution of the two terms is of the same order. This can be done, for example, by arbitrarily choosing these coefficients and adjusting them through a trial-and-error process or by calculating the *neural tangent kernel* of the network (Wang, Yu & Perdikaris 2022). We find that, overall, the need for fine tuning additional hyperparameters in this method is an inconvenience.

Instead, we opt for an approach inspired by Lagaris et al. (1997) (see also a similar one, based on distance functions in Sukumar & Srivastava 2022). In that approach, the loss function is given only by equation (15), but the approximate solution of the boundary value problem is written as

$$\tilde{u}(\boldsymbol{x};\Omega) = f_b(\boldsymbol{x}) + h_b(\boldsymbol{x})\mathcal{N}(\boldsymbol{x};\Omega), \tag{18}$$

where f_b is a smooth and (at least) twice differentiable function that satisfies the BCs (see equation 14), h_b is a smooth and twice differentiable function that defines the boundary ($h_b = 0$ at $\partial\mathcal{D}$ and $h_b \neq 0$ in $\mathcal{D}$), and $\mathcal{N}$ is the output of the network. This redefinition ensures that $\tilde{u}$ matches exactly the BCs at $\partial\mathcal{D}$ regardless of the value of $\mathcal{N}$. In the rest of the domain, $\tilde{u}$ should satisfy the PDE. The PINN takes care of this by adapting $\mathcal{N}$ during training. We stress again that in this approach $\mathcal{N}$ by itself does not satisfy the PDE (in contrast to equation 17), but the combination of f_b, h_b, and $\mathcal{N}$ does. The choice of f_b and h_b is not unique: variations of this functions can lead to slight changes in the convergence rate or the final value of the loss function (see the examples in Section 4.2.1 for a more quantitative description).

In this work, we attempt to generalize the PINN approach to build a PDE solver valid for different and varied BCs (and possibly, source terms). We want our network to learn how to approximate any particular solution for a given operator $\mathcal{L}$. This means that the information about the BCs should be part of the *input* of the network (along with the coordinates) and not hardcoded in the loss function or in the parametrization (18). During training, the network needs to process a large number of points $\boldsymbol{x}$ and a large number of f_b functions so that it can generalize and provide solutions of (13) for any point in the domain and any BC. Of course, f_b could, in principle, be an arbitrary continuous function. For this reason, it should be intelligently encoded into the network's input to keep the number of parameters small and manageable.

In the following section, we present tests for our PINN solver for the GS equation as described in Section 2. We omit hereinafter to state explicitly the dependence of all the functions on the training variables Ω.

4 TEST AND MODELS

4.1 Current-free GS equation

As a first test, we consider the GS equation (equation 4) without current, that is $G(\mathcal{P}) = 0$. We set up the various elements of the PINN solver as follows. The function h_b describing the boundary is given by

$$h_b(q,\mu) = q(1-q)(1-\mu^2). \tag{19}$$

The function f_b that satisfies the BCs, where $h_b = 0$, is given by

$$f_b(q,\mu) = q^n\left(1-\mu^2\right)\sum_{l=1}^{l_{\max}}\frac{b_l}{l}P_l'(\mu). \tag{20}$$

Notice that equation (20) differs from equation (10) by a factor q^n with $n > 0$. We include this factor to enforce BCs both at the surface ($q = 1$) and infinity ($q = 0$). If $n = 1$, this parametrization is the same as that used in Lagaris et al. (1997) considering essential BCs in a rectangle. However, we prefer to leave n as a free parameter that is used to give more or less weight to the solution close to the star or away from the surface. We performed a detailed study of the influence of the hyperparameters of the model, including n, in the following section. We must remark that other parametrizations are possible and in principle can be tuned to improve the results of each specific problem.

The input layer of the NN consists of the coordinates of the point where the solution will be evaluated (q, μ) and the coefficients b_l determining the BC at $q = 1$. For the physical applications in this paper, we expect the dipole ($l = 1$) component of the magnetic field to be dominant. Therefore, we normalize all other multipole coefficients by dividing them by b_1 and we reduce the range of the possible values of $b_{l>1}$ between -1 and 1. With this choice, we can omit the b_1 coefficient because it is reabsorbed in the normalization factor of the magnetic field strength.

Hereafter, we limit ourselves to $l_{\max} = 7$, which suffices for our purposes and is a good compromise between generality of solutions and ease of training. Increasing the number of multipoles adds complexity and it would require larger networks to achieve the desired accuracy. Thus, one training point is defined as

$$(q, \mu, b_2, b_3, b_4, b_5, b_6, b_7),$$

In each forward pass providing $\mathcal{N}(q, \mu, \{b_l\})$, the solution of the differential equation at each training point i is given by

$$\mathcal{P}_i = \left(1-\mu^2\right)\left[\sum_{l=1}^{l_{\max}}\frac{b_l}{l}q^nP_l'(\mu) + q(1-q)\mathcal{N}(q,\mu,\{b_l\})\right]. \tag{21}$$

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

Notice that the output of the network $\mathcal{N}$ depends both on the coordinates and on the BCs. This is the crux of our approach, as we want the network to be able to generalize for any BC (expressed in terms of the b_l coefficients).

The weights and biases of the network are optimized by minimizing the loss function averaged over a large sample of training points $i_{\max}$

$$\mathcal{J} = \frac{1}{i_{\max}} \sum_{i=1}^{i_{\max}} [\triangle_{\rm GS} \mathcal{P}_i]^2. \tag{22}$$

We consider training sets of size $i_{\max} = 10^4$. At first sight, this number might look large. However, we must stress that it is required by the high dimensionality of our problem. Our parameter space consists of two coordinates plus six coefficients to describe the BC (fixing $b_1 = 1$). Covering this eight-dimensional parameter space with only three points in each dimension would need $3^8 = 6561$ points, which makes evident the crucial difference between training to solve a PDE with fixed BCs or training with arbitrary BCs (formally, an infinite number of additional parameters). The training set is changed periodically every few thousand epochs in order to feed the network with as many points as possible.

Fig. 2(a) shows the evolution of the loss function (22) with the number of training epochs. For this particular model, we have chosen an FCNN architecture with 4 hidden layers and 80 neurons at each layer with $n = 5$ in f_b (see the next section for details on the choice of these hyperparameters). The activation function g (see equation 11) chosen for all the hidden layers is the tanh function. The periodic spikes correspond to renewals of the training set. They can be understood as a measure of the ability of a model to generalize to new, unseen points. Fig. 2(b) shows an example of the final result once the PINN has been trained. The black solid lines show the *exact* analytical solution which is uniquely determined by the b_l coefficients

$$\mathcal{P}_{\rm ex}(q,\mu) = \left(1-\mu^2\right) \sum_{l=1}^{l_{\max}} \frac{b_l}{l} q^l P_l'(\mu), \tag{23}$$

while the yellow dashed lines show the solution acquired by the PINN ($\mathcal{P}$). They are indistinguishable at the figure scale. The colourmap indicates the relative difference between $\mathcal{P}$ and $\mathcal{P}_{\rm ex}$, at most $\sim$0.5 per cent for this particular model.

The magnetic field components can be computed from $\mathcal{P}$ via AD. From equation (2), we have

$$B_r = -q^2 \frac{\partial \mathcal{P}}{\partial \mu}, \tag{24}$$

$$B_\theta = \frac{q^3}{\sqrt{1-\mu^2}} \frac{\partial \mathcal{P}}{\partial q}. \tag{25}$$

In finite difference schemes, one usually losses accuracy when taking numerical derivatives. To explore the performance of the PINN in this respect, we have computed different relative error norms of different orders (p) for $\mathcal{P}$, B_r, B_θ and for the magnetic field modulus $B = \sqrt{B_r^2 + B_\theta^2}$. Results are summarized in Table 1. These p-norms for a given order p are calculated for every variable as in Eivazi et al. (2022)

$$E_u = \frac{\|\tilde{u}(\boldsymbol{x}) - u(\boldsymbol{x})\|_p}{\|u(\boldsymbol{x})\|_p} \times 100, \tag{26}$$

Interestingly, the errors are of the same order of magnitude for the function $\mathcal{P}$ and its derivatives. This can be attributed to the fact that we train the PINN with a second-order PDE (the loss function involves second-order derivatives) and this includes additional information on the derivatives. Furthermore, using automatic differentiation is also an advantage over finite difference schemes, where accuracy depends on the resolution.

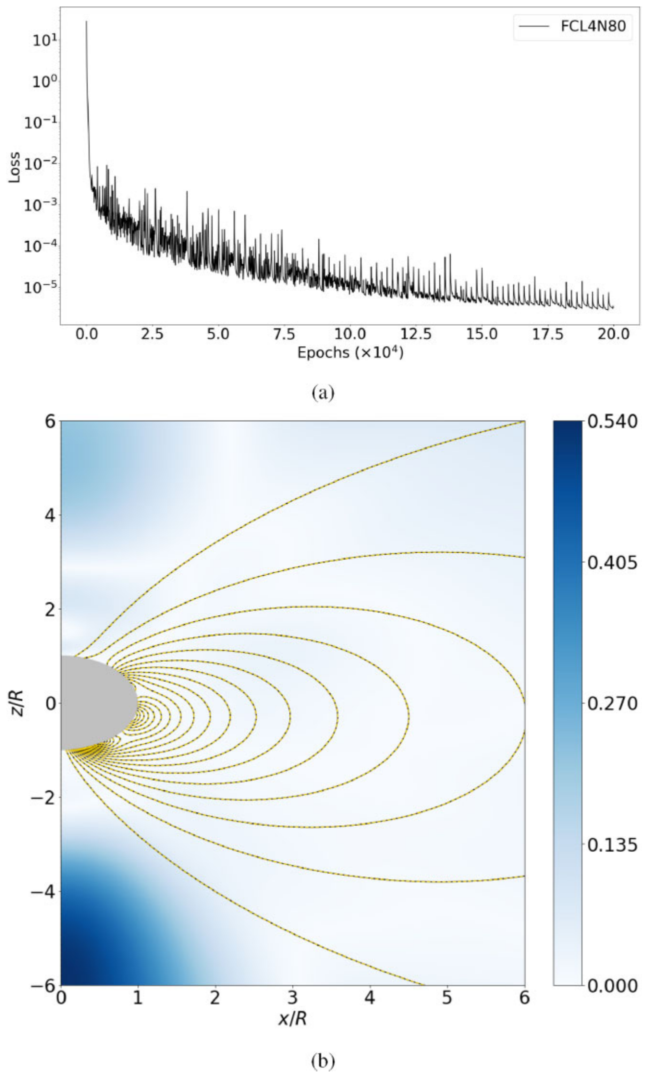

Figure 2. (a) Evolution of the loss function with the training epochs. The periodic spikes correspond to renewals of the training set. (b) Colourmap of the relative error (percentage) between $\mathcal{P}$ and $\mathcal{P}_{\rm ex}$. Yellow dashed lines correspond to contours of $\mathcal{P}$ while black solid lines correspond to $\mathcal{P}_{\rm ex}$. The multipole coefficients in f_b for this particular example are $b_1 = 1, b_{l\geq 2} = (-1)^{l+1}0.6$.

Table 1. Relative error norms (percentage) between PINN and the exact solution. The numbers shown are averages of the respective norms of 100 new sets, consisting of 10^4 random points each. For each set, we compute the norms using equation (26).

	$E_{\mathcal{P}}$	E_{B_r}	E_{B_θ}	E_B
L_1 norm	0.017	0.017	0.025	0.015
L_2 norm	0.019	0.023	0.045	0.023

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

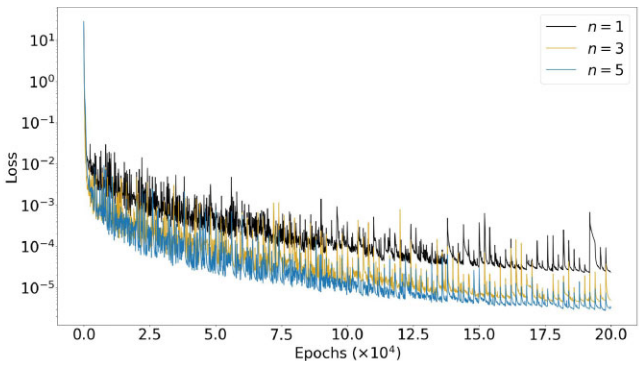

Figure 3. Evolution of the loss function with the training epochs for different values of the exponent n in f_b. The rest of the hyperparameters are as in Section 4.1.

Table 2. Relative error L_2 *norms* for different values of n. The results indicate a higher accuracy of the overall solution as n increases.

	$p=2$			
n	$E_{\mathcal{P}}$	E_{B_r}	E_{B_θ}	E_B
1	0.057	0.100	0.131	0.084
3	0.030	0.071	0.117	0.064
5	0.019	0.023	0.045	0.023

4.2 Influence of the PINN hyperparameters

We have performed a detailed study to measure the influence of various hyperparameters of our model. In particular, we have considered the following:

(i) Changes of the parametrization of the boundary (q^n power).
(ii) Number of neurons at each layer.
(iii) Number of hidden layers.
(iv) Resnet versus FC architectures.

We changed one hyperparameter at a time while keeping the rest fixed to the reference values of the previous section. The results of this study are presented separately in the following subsections.

4.2.1 Changes of the parametrization of the BCs

We begin by considering different values of the exponent in the q^n term in the boundary function (20). Fig. 3 shows the evolution of the loss with the training epochs for three different values of the exponent n, namely 1 (corresponding to the Lagaris parametrization), 3 and 5. As n increases, the impact of the surface BC becomes less important. In general, increasing n improves the convergence of the model and leads to more accurate solutions. Table 2 shows the relative error L_2 norms for the four quantities that we use to evaluate our results ($\mathcal{P}$, B_r, B_θ, B). All of them decrease with increasing n.

4.2.2 Number of neurons per hidden layer

Next, we explore the effect of the number of neurons per hidden layer N. Fig. 4 shows the evolution of the loss with the training epochs for $N = 20$, 40, 80. The number of neurons has a considerable impact on the convergence of each model. This is expected, because models with smaller N do not have enough free parameters to account for the complexity and variability of the solutions. Our results show that the loss reaches values that are smaller by a factor of 33 when doubling

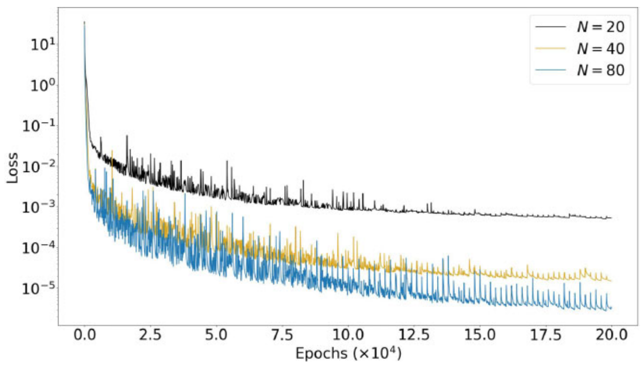

Figure 4. Evolution of the loss function with the training epochs for different values of the number of neurons per hidden layer N. The rest of the hyperparameters are as in Section 4.1.

Table 3. Relative error L_2 *norms* for different values of N. The results indicate a higher accuracy of the solution as N increases.

	$p=2$			
N	$E_{\mathcal{P}}$	E_{B_r}	E_{B_θ}	E_B
20	0.316	0.188	0.285	0.171
40	0.039	0.033	0.052	0.031
80	0.019	0.023	0.045	0.023

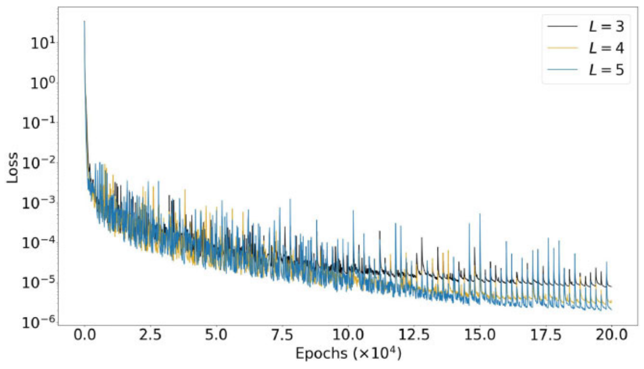

Figure 5. Evolution of the loss function with the training epochs for different values of the number of hidden layers L. The rest of the hyperparameters are as in Section 4.1.

N from 20 to 40 and by a factor of 5 when doubling from 40 to 80. This is reflected, as well, in Table 3, where all quantities show a significant improvement in accuracy as N increases.

4.2.3 Number of hidden layers

Following the same line of arguments, one could expect that increasing the number of hidden layers L would also lead to improved convergence and higher accuracy. However, our results show that adding more layers has a marginal impact on convergence and accuracy, or it can even lead to worse results for large networks. In other words, deeper networks are more prone to overfitting. Evidence of overfitting can be seen in Fig. 5 for $L = 5$. The spikes that correspond to renewals of the set of training points are much higher than expected, even at the later stages of training. This is, indeed, reflected in Table 4, where the model with $L = 5$ performs worse in terms of accuracy than the model with $L = 4$ because

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

Table 4. Relative error L_2 norms for different values of L. The results indicate that adding more layers does not improve the accuracy significantly and can lead to overfitting.

		$p = 2$		
L	$E_{\mathcal{P}}$	E_{B_r}	E_{B_θ}	E_B
3	0.030	0.029	0.050	0.028
4	0.019	0.023	0.045	0.023
5	0.014	0.025	0.064	0.027

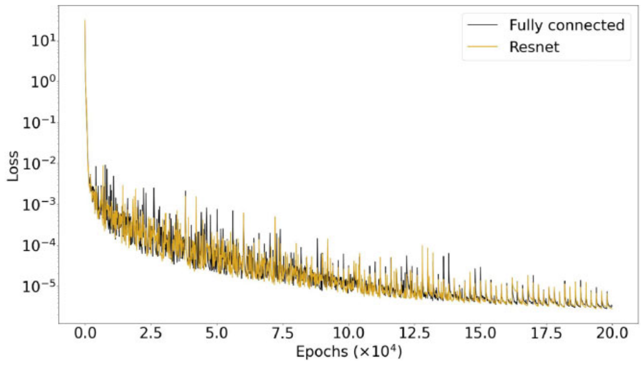

Figure 6. Evolution of the loss function with the training epochs for different NN architectures. The rest of the hyperparameters are as in Section 4.1.

Table 5. Relative error L_2 *norms* for different NN architectures. The results indicate that there is no appreciable difference for the two architectures considered.

		$p = 2$		
Model	$E_{\mathcal{P}}$	E_{B_r}	E_{B_θ}	E_B
FCL4N80	0.019	0.023	0.045	0.023
ResL4N80	0.018	0.019	0.034	0.018

it is overfitted to the training set and fails to generalize to unseen points. Considering, in addition, that training deeper networks is computationally expensive, we conclude that increasing too much the number of layers is not beneficial in terms of accuracy or convergence.

4.2.4 *Resnet versus fully connected*

Lastly, we consider two different types of NN architectures, FC architecture and ResNet architecture. Results are summarized in Fig. 6 and Table 5. No appreciable differences can be detected between the two models in convergence or overall accuracy.

4.3 FF GS equation

Once we have assessed the performance of our approach in the vacuum case by comparing our results with the analytical solutions, we now turn to the general case ($G(\mathcal{P}) \neq 0$).

The configuration of the PINN solver for this case is similar to the one described in the current-free case. f_b, h_b, n, $l_{\max}$, $i_{\max}$, N, L, architecture, number of epochs and optimizer are the same as in Section 4.1. There are two main differences: (a) the loss function and (b) the input. The loss function now includes the non-zero source term $G(\mathcal{P})$ which accounts for the presence of currents and is given

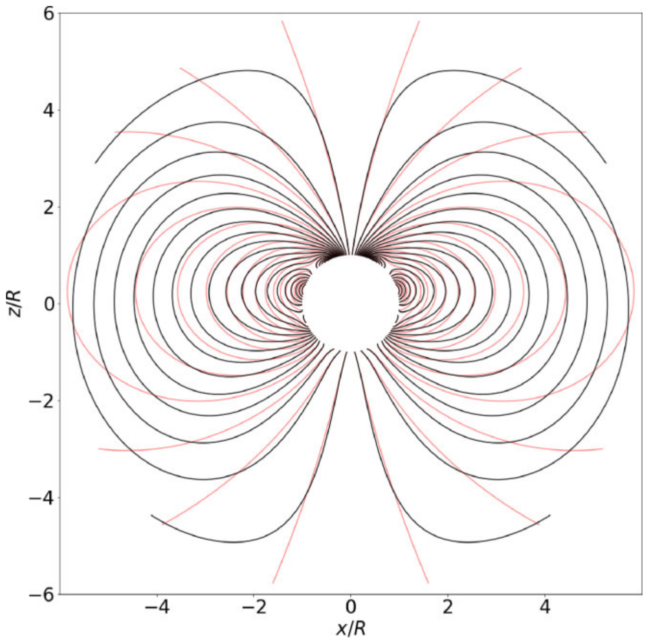

Figure 7. Field lines for the current-free (red) and FF (black) cases. The multipole coefficients at the surface are $b_{l>1} = 0.5$ for both. For the FF case, the coefficients in expression (9) for $\mathcal{T}(\mathcal{P})$ are $s_1 = 0.2$, $s = 0.4$.

by

$$\mathcal{J} = \frac{1}{i_{\max}} \sum_{i=1}^{i_{\max}} [\triangle_{\rm GS}\mathcal{P}_i - G(\mathcal{P}_i)]^2. \quad (27)$$

The input must include information about the functional form of $G(\mathcal{P})$ or equivalently $\mathcal{T}(\mathcal{P})$. In Section 2 we modelled $\mathcal{T}$ to be a quadratic function of $\mathcal{P}$ using two parameters, s_1 and s_2 (see equation (9)). Therefore, the input of the PINN must be extended to include these parameters. The PINN is trained to provide solutions for any value of s_1 and s_2 in the same sense that it is trained to provide solutions for any value of the multipole coefficients defining the BC. The input for the general FF case is

$$(q, \mu, b_2, b_3, b_4, b_5, b_6, b_7, s_1, s_2).$$

Fig. 7 shows, for reference, a comparison between a current-free and an FF magnetic field, where we observe notorious difference in the structure of the field lines.

We note that there can be regions in the input parameter space where mathematical solutions do not exist (see Akgün et al. 2018; Mahlmann et al. 2019 for a detailed discussion). This is reflected in the evolution of the loss function, which fluctuates around values of order unity, meaning that there are no solutions that can minimize the residual of the PDE. Nevertheless, the PINN will return approximate solutions. It is up to the user to carefully evaluate the validity and accuracy of the results. A sufficiently low final value of the loss function is a first filter in this regard.

The lack of analytical solutions in the general case makes it difficult to estimate errors, which is a fundamental part of any scientific analysis. Despite the abundant literature on PINNs as PDE solvers in recent years, a systematic and consistent way for measuring errors and deciding on the quality of the provided approximate solutions is still lacking. Here, we adopt the following approach:

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

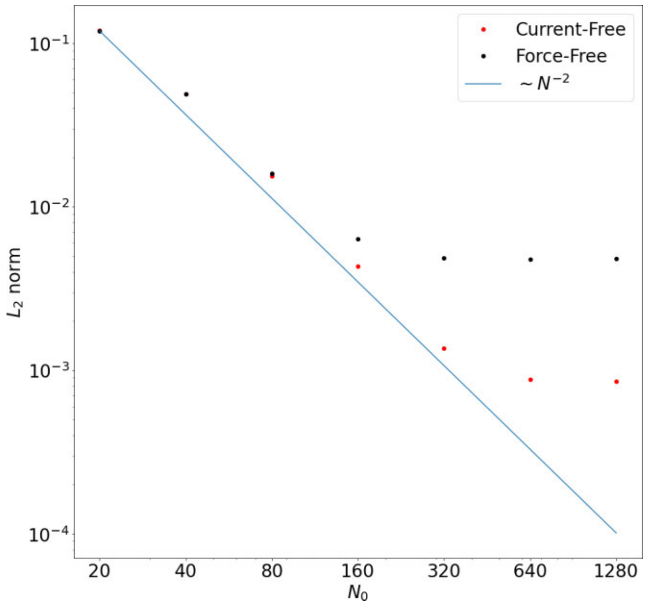

Figure 8. The L_2-norm of the discretized GS equation for the current-free (red) and FF (black) cases as a function of the resolution N_0.

(i) After our NN is trained, we create a regular grid (q_i, μ_j), where $i, j = 0, \ldots, N_0$, and evaluate the PINN solution $\mathcal{P}$ at all the points with a forward pass.

(ii) Then, we discretize equation (4) using a second-order finite difference scheme, which gives us a different evaluation of the residual $\epsilon_{\rm FD}$ using the same function values evaluated in the previous step.

(iii) If $\mathcal{P}$ was the 'exact' solution of the PDE, the second-order residual $\epsilon_{\rm FD}$ would decrease with increasing resolution as $\sim N_0^{-2}$, where N_0 is the number of grid points (assuming both dimensions have the same resolution). In reality, $\mathcal{P}$ is only an approximate solution with an intrinsic error $\epsilon_{\rm NN}$ inherited from the quality and accuracy of the PINN. Therefore, $\epsilon_{\rm FD}$ will follow this power law only up to the point where the PINN approximation error $\epsilon_{\rm NN}$ starts to dominate the discretization error $\epsilon_{\rm FD}$.

Fig. 8 illustrates this behaviour. We plot the L_2-norm of the discretized GS equation for both the current-free ($G(\mathcal{P}) = 0$) and FF cases ($G(\mathcal{P}) \neq 0$) as a function of the number of grid points N_0. In both cases the L_2-norm drops as $\sim N_0^{-2}$ until it reaches a plateau which signalizes that $\epsilon_{\rm NN} > \epsilon_{\rm FD}$. We expect that, at worst, $\epsilon_{\rm NN}$ will be of the order of the square root of the loss function, because in equations (22) and (27) $\mathcal{J}$ is precisely L_2^2. In other words, when calculating $\epsilon_{\rm NN}$ for a particular example, with fixed multipole coefficients and source terms, we expect the error to be of the order $\sqrt{\mathcal{J}}$, within a factor of a few. We note that we obtain errors of the same order of magnitude for both cases, with a factor of ~5 less for the vacuum. We expect a slightly higher error when we introduce the current term $G(\mathcal{P})$, because we increase the dimensionality of the problem, and we also introduce non-linear terms into the differential equation.

5 APPLICATION TO THE MAGNETOTHERMAL EVOLUTION OF NEUTRON STARS

Our astrophysical scenario of interest is the long-term evolution of magnetic fields in NSs. The evolution of the system is governed by two coupled equations: the heat diffusion equation and the induction equation (see the review by Pons & Viganò 2019 for more details). They must be complemented with a detailed specification of the local microphysics (neutrino emissivity, heat capacity, thermal and electrical conductivity) and the structure of the star, usually assumed as fixed throughout the NS's life. Our NS background model is a 1.4M$_\odot$ NS built with the Sly4[4] equation of state (Douchin & Haensel 2001). We use the two-dimensional (2D) magnetothermal code (latest version in Viganò et al. 2021) developed by our group suitably modified to implement the external BCs using the PINN (trained as described in the previous section) to assess its performance and potential. In particular, this implementation allows us to quickly switch and compare between vacuum BCs and the barely explored FF BCs. To our knowledge, only the work by Akgün et al. (2018) has presented results from simulations that included the effect of a magnetosphere threaded by currents. They had to implement a costly elliptic solver as a BC which slowed down the code considerably. The PINN implementation should, in principle, be much easier to change, efficient, and generalizable.

5.1 Current-free magnetospheric BCs

We begin by considering a crustal-confined magnetic field topology and vacuum BCs (no electrical currents circulating in the envelope and across the surface). We enforce BCs via multipole expansion of the radial magnetic field at the surface as described in Pons, Miralles & Geppert (2009) and Pons & Viganò (2019).

The coefficients of the multipole expansion can be computed from the radial component of the magnetic field at the surface of the star as follows:

$$b_l = \frac{2l+1}{2(l+1)} \int_0^\pi B_r(R,\theta) P_l(\cos\theta) \sin\theta \mathrm{d}\theta. \tag{28}$$

During the evolution, we calculate at each time-step the b_l coefficients using equation (28). In the classical approach, we reconstruct the values of B_r and B_θ in the external ghost cells, explicitly

$$B_r = \sum_{l=1}^{l_{\max}} b_l\,(l+1)\,P_l\,(\cos\theta)\left(\frac{R}{r}\right)^{l+2}, \tag{29}$$

$$B_\theta = -\sin\theta \sum_{l=1}^{l_{\max}} b_l\,P_l'\,(\cos\theta)\left(\frac{R}{r}\right)^{l+2}, \tag{30}$$

where R is the radius of the NS. For conciseness, we refer to this procedure by the nomenclature OLD.

In the new PINNs approach, we use the b_l coefficients obtained from the Legendre decomposition as inputs to the PINN. The latter returns values of the poloidal flux function $\mathcal{P}$, or any required component of the magnetic field by taking derivatives (equations 24 and 25). Obviously, in this case (vacuum BCs), the PINN approach does not represent any advantage because we already know how to build the analytical solution. However, we want to ensure that the results of the simulations do not show any undesirable effects before moving on to a more complex case.

To compare the different employed techniques described above, we run axisymmetric crustal-confined magnetic field simulations using the 2D magnetothermal code (Viganò et al. 2021) with a grid of 99 angular points (from pole to pole) and 200 radial points. The initial field has a poloidal component of 10^{14} G (value at the pole and consists of a sum of a dipole ($b_1 = 1$), a quadrupole ($b_2 = 0.6$) and an octupole ($b_3 = 0.3$). The initial toroidal quadrupolar component

[4]https://compose.obspm.fr/

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

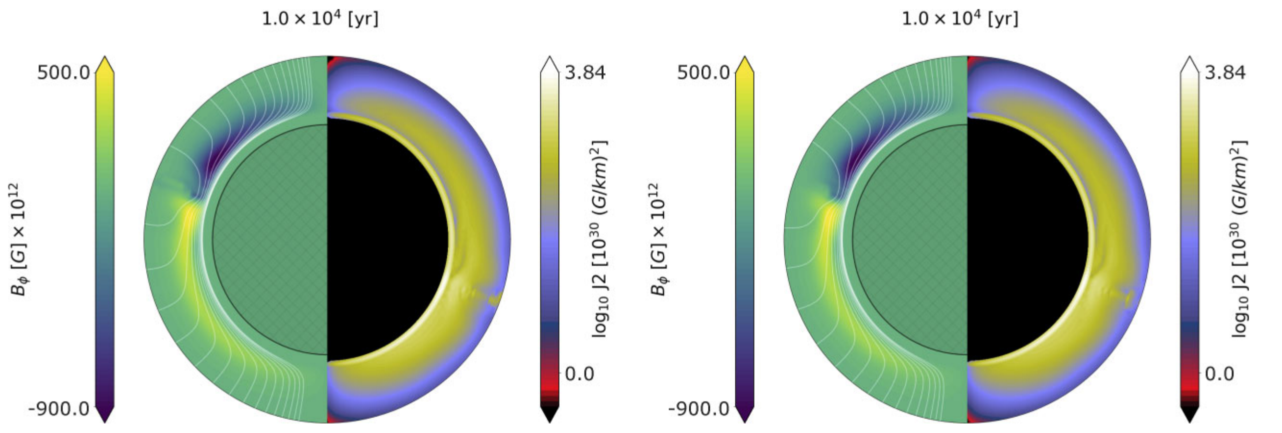

Figure 9. A snapshot of the magnetic field evolution and the electric current at 10 kyr, obtained using OLD (left-hand panel) and PINN (right-hand panel). In the left hemisphere, we show the meridional projection of the magnetic field lines (white lines) and the toroidal field (colours). In the right hemishere, we display the square of the modulus of the electric current, i.e. $|J|^2$ (note the log scale). The crust has been enlarged by a factor of 8 for visualization purposes.

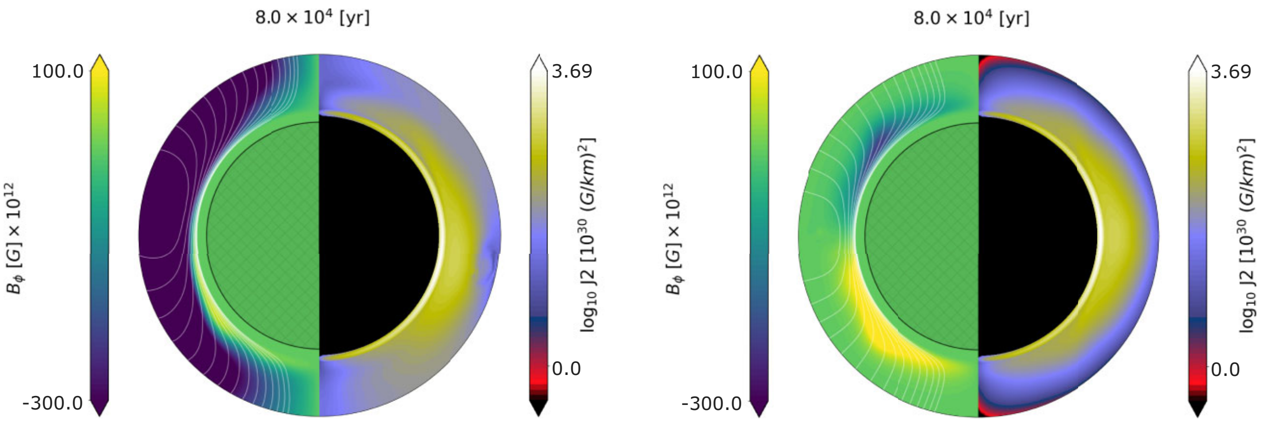

Figure 10. Same as Fig. 9. A snapshot of the magnetic field evolution and the electric current at 80 kyr. Left-hand panel: FF BCs. Right-hand panel: vacuum BCs.

has also a maximum initial value of 10^{14} G. The maximum number of multipoles is fixed to $l_{\max} = 7$ for PINN and to $l_{\max} = 50$ for OLD. The objective behind the use of different $l_{\max}$ is to assess the impact of truncating the multipole number when using the PINN.

The results of the comparison at $t = 10$ kyr are displayed in Fig. 9. On the left (right), we show the magnetic field profiles obtained with OLD (PINN) BCs. The overall evolution of the magnetic field and the electric current is very similar. Slight differences appear due to the multipolar truncation in the PINN case. We must note that, if the same maximum number of multipoles is set for both systems, we obtain almost identical results.

5.2 FF magnetospheric BCs

To couple the internal field evolution with an FF magnetosphere PINN solver, we must extend the vacuum case (Section 5.1) with additional steps. In our magnetothermal evolution code, we impose the external BCs by providing the values of the magnetic field components in two radial ghost cells for every angular cell. In the vacuum case, once the multipolar decomposition of the radial field over the surface is known, the solution in the ghost cells can be built analytically. However, in the general case, one must solve an elliptic equation in a different grid that must be extended far from the surface to properly capture to asymptotic behaviour at long distances. This process is very costly because it must be repeated tens of thousands of time-steps as the interior field evolves. In this situation, having trained the PINN, allows us to use it as a fast tool to provide required values of the solution in the ghost cells. We proceed as follows: First, at each evolution time-step, we must know the toroidal function $\mathcal{T}(\mathcal{P})$. For simplicity, in this application we use a quadratic function. At each time step we fit the values obtained from the internal evolution one cell below the surface. The fit provides the coefficients of the quadratic interpolation s_1 and s_2 defined in equation (9). Next, as described in Section 4.3, s_1 and s_2 are provided as additional input parameters to the forward pass. The PINN returns the poloidal flux function $\mathcal{P}$ and the components of the magnetic field needed at the ghost cells of the magnetothermal evolution code. With this information, the internal evolution can proceed to the next time-step.

We assume an initial FF magnetic field with a poloidal component of 3×10^{14} G at the polar surface and a maximum toroidal field of 3×10^{14} G. To understand the impact of the different BCs, we consider in one case FF BCs (the left-hand panel of Fig. 10) and in the other case vacuum BCs (the right-hand panel of Fig. 10). The results of the comparison are illustrated by two snapshots at $t = 80$ kyr of the evolution with the same initial model. We note that the initial FF magnetic field allows current sheets to thread the star's surface. A distinct magnetic field evolution is clearly observed if we apply one type of BCs or the other. The FF BCs (left) result in a stronger

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

toroidal dipole close to the surface and slightly displaced towards the north. The stronger toroidal component compresses the poloidal field lines closer to the poles. In contrast, for vacuum BCs, the poloidal field lines retain certain symmetry with respect to the equator, and the dominant toroidal component is now quadrupolar and concentrated at the crust/core interface, as shown in the right-hand panel of Fig. 10. The distribution of the electric current in the stellar crust is also different. Enforced by the vacuum BCs, current tends to vanish around the poles and close to the surface. This is similar to what was observed in Fig. 9 although the initial field topology is different. Instead, with FF BCs, currents near the surface are not forced to vanish. In the left-hand panel, the slightly more yellowish region in the Northern hemisphere and mid-latitudes indicates that significant current flows into the magnetosphere. This difference in current configurations would have important implications in the observed temperature distribution, as discussed in Akgün et al. (2018). We will address, in future works, a more detailed exploration of the effect of BCs since our purpose here is to illustrate with a few examples the potential of our approach.

6 CONCLUSIONS

Using PINNs to obtain a solution of a particular boundary value problem is, up to date, far more computationally expensive and arguably less accurate than using classical methods. The drawbacks are related to the training process, which involves the minimization of a high-dimensional loss function. Once a PINN is trained for a given boundary value problem, its utility is limited because it would be necessary to re-train to generate new solutions with different BCs.

The functionality of PINNs to real physical problems would become significantly better if their flexibility and adaptability can be increased. In this work, we explore this idea by training our PINN for general BCs and source terms, expressed through appropriate coefficients (a limited number of them) that enter as additional inputs in the network. Of course, this makes the training process computationally more expensive, but the evaluation of new generic solutions is very fast. In our study, the coverage of the parameter space is not exhaustive because we have used very limited computational resources (a personal computer), and our purpose is to show that this proof-of-concept works and can already be applied to some physical problems, even by non-experts in computer science. If necessary, it is straightforward to adapt our implementation for the specific needs of other applications (e.g. adding more multipoles in the boundary if we need to capture smaller scales, or extending the parametrization of the BCs). We are aware that it is possible to drastically improve the efficiency of our computations by employing GPU clusters or enriching our algorithms with advanced machine-learning techniques and that will be required, for example, in the extension to three-dimesion (3D) of this work.

We have also explored various configurations for our network through a basic hyperparameter space study. We conclude that the most impactful element is the number of neurons per layer. On the contrary, making the network deeper by adding more layers is not beneficial, beyond a reasonable minimum. Furthermore, the way that the solution is parametrized to always satisfy the BCs is important, indicating that the human orientation in some choices (as opposed to a zero-like approach) is still critical for physics problems. We also found that ResNet architectures do not offer any advantage for the kind of applications that we are dealing with. We emphasize that, from a theoretical point of view, for any continuous function there is always an NN that is able to approximate it to arbitrary precision (see the *Universal Approximation Theorem*; Cybenko 1989; Leshno et al. 1993; Pinkus 1999). However, the theorem does not specify the properties said network (e.g. its size, architecture, or the activation functions) nor if it is feasible to build and train it with current computational capabilities, but merely state that this NN exists. For this reason, in general, there is not guarantee that this function can be represented by another NN, and therefore during the training process the loss function will stagnate to a finite value. Moreover, the presence of non-linear activation functions in conjunction with the compositional structure of NNs renders the associated optimization problem non-convex. Therefore, the gradient descent based algorithm used during the training process may get stuck in a saddle point or in a local minimum, instead of moving towards the global one. All these issues affect unavoidably the convergence of the NNs. In classical numerical approaches, the dependence of the convergence on the implementation properties (order of the scheme, resolution, etc.) is known and well studied. On the other hand, in the DL field there is sparse mathematical evidence to instruct us how to build and train a network to achieve a more accurate result, although it is currently subject to active research (e.g. De Ryck, Lanthaler & Mishra 2021; De Ryck & Mishra 2022; De Ryck, Jagtap & Mishra 2022; Mishra & Molinaro 2022).

Nevertheless, we cannot ignore that NNs have proven to be very good function approximators in very distinct areas, even though perfect convergence is currently unlikely to be achieved for the practical reasons mentioned above. For this particular work, our results show that the PINN solutions are relatively accurate, reliable, and well behaved. For the current-free GS equation, where comparisons with an analytical solution can be made, we found relative differences of typically less than 1 per cent. For the force–free GS equation, we propose a method for estimating the error through the use of a finite difference discretization scheme. Our analysis shows that solutions are accurate up to a point that we associate to the intrinsic approximation error of the PINN. This approach is straightforward to implement and self-consistent and could be set as the standard procedure for estimating errors of PINN-based PDE solvers in general. Even if PINNs fall short in terms of precision compared to classical PDE solvers, they can still be used in conjunction with them in various cases. For example, many iterative solvers rely on good initial guesses to converge. A PINN can provide such an initial guess to be subsequently refined by a classical method.

The most interesting case where PINNs overcome the capabilities of classical methods are physical systems with two or more domains that are governed by vastly different physical conditions and time-scales. An example of such a case is the magnetothermal evolution in the interior of an NS that is connected to an FF magnetosphere. Solving this problem through a global simulation in the entire domain is very costly due to the needs of the elliptic solver for the exterior solution. On the contrary, PINNs provide a very effective way of imposing BCs at the interface of the two domains. Once a PINN is trained, accurate enough magnetospheric solutions in a few points (ghost cells of the interior evolution code) can be swiftly computed at each time-step without adding an excessive amount of computational cost. We have shown that for the well-tested case of vacuum BC, the PINN is accurate enough to satisfactorily reproduce the reference results obtained by the exact spectral decomposition approach. Nevertheless, it is about two times slower. As proof of concept, to demonstrate the potentiality of our approach, we also presented results for the less explored FF magnetospheric BCs. In this case, the computational cost was more than an order of magnitude smaller than the similar problem solved with classical methods. Indeed, we obtain solutions that allow currents that thread the NS's surface and flow into the magnetosphere, giving rise to a new family of internal field

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

configurations. We reserve a more detailed analysis of the physical properties of these solutions in 2D, and the more physically relevant extension to 3D, for future works.

ACKNOWLEDGEMENTS

We acknowledge the support through the grant PID2021-127495NB-I00 funded by MCIN/AEI/10.13039/501100011033 and by the European Union, and the Astrophysics and High Energy Physics programme of the Generalitat Valenciana ASFAE/2022/026 funded by MCIN and the European Union NextGenerationEU (PRTR-C17.I1). JFU is supported by the predoctoral fellowship UAFPU21-103 funded by the University of Alicante. CD is supported by the ERC Consolidator Grant 'MAGNESIA' no. 817661 (PI: N. Rea) and has the partial support of NORDITA. This work has been carried out within the framework of the doctoral program in Physics of the Universitat Autònoma de Barcelona and it is partially supported by the programme Unidad de Excelencia María de Maeztu CEX2020-001058-M.

DATA AVAILABILITY

All data produced in this work will be shared on reasonable request to the corresponding author.

REFERENCES

Abadi M. et al., 2016 OSDI, USENIX Association, p. 265, https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi
Akgün T., Cerdá-Durán P., Miralles J. A., Pons J. A., 2017, MNRAS, 472, 3914
Akgün T., Miralles J. A., Pons J. A., Cerdá-Durán P., 2016, MNRAS, 462, 1894
Akgün T., Cerdá-Durán P., Miralles J. A., Pons J. A., 2017, MNRAS, 474, 625
Akgün T., Cerdá-Durán P., Miralles J. A., Pons J. A., 2018, MNRAS, 481, 5331
Baydin A. G., Pearlmutter B. A., Radul A. A., Siskind J. M., 2017, J. Mach. Learn. Res., 18, 5595
Cai S., Mao Z., Wang Z., Yin M., Karniadakis G. E., 2021, Acta Mechanica Sinica, 37, 1727
Chan W., Jaitly N., Le Q. V., Vinyals O., 2015, preprint (arXiv:1508.01211)
Chen X., Jeffery D. J., Zhong M., McClenny L., Braga-Neto U., Wang L., 2022, preprint (arXiv:2211.05219)
Cybenko G., 1989, Math. Control Signals Syst., 2, 303
De Ryck T., Mishra S., 2022, Adv. Comput. Math., 48, 79
De Ryck T., Lanthaler S., Mishra S., 2021, Neural Netw., 143, 732
De Ryck T., Jagtap A. D., Mishra S., 2022, preprint (arXiv:2203.09346)
Dehman C., Viganò D., Pons J. A., Rea N., 2023, MNRAS, 518, 1222
Douchin F., Haensel P., 2001, A&A, 380, 151
Eivazi H., Tahani M., Schlatter P., Vinuesa R., 2022, Phys. Fluids, 34, 075117
Glampedakis K., Lander S. K., Andersson N., 2014, MNRAS, 437, 2
Hayat S., Mall R. N., 2013, IJERT, 2
He K., Zhang X., Ren S., Sun J., 2015, preprint (arXiv:1512.03385)
He K., Zhang X., Ren S., Sun J., 2016, preprint (arXiv:1603.05027)
Hornik K., Stinchcombe M., White H., 1989, Neural Netw., 2, 359
Kingma D. P., Ba J., 2014, preprint (arXiv:1412.6980)
Kojima Y., 2017, MNRAS, 468, 2011
Kugunavar S., Prabhakar C. J., 2021, Visual Comput. Ind. Biomed. Art, 4,12
Lagaris I. E., Likas A., Fotiadis D. I., 1997, preprint (physics/9705023)
Lawrence S., Giles C., Tsoi A. C., Back A., 1997, IEEE Trans. Neural Netw., 8, 98
Leshno M., Lin V. Y., Pinkus A., Schocken S., 1993, Neural Netw., 6, 861
Luna R., Calderón Bustillo J., José Seoane Martínez J., Torres-Forné A., Font J. A., 2023, Phys. Rev. D, 107, 064025
Mahlmann J. F., Akgün T., Pons J. A., Aloy M. A., Cerdá-Durán P., 2019, MNRAS, 490, 4858
Mishra S., Molinaro R., 2021, J. Quant. Spectrosc. Radiative Trans., 270, 107705
Mishra S., Molinaro R., 2022, IMA J. Num. Anal., 43, 1
Pili A. G., Bucciantini N., Del Zanna L., 2015, MNRAS, 447, 2821
Pinkus A., 1999, Acta Numer., 8, 143
Pons J. A., Viganò D., 2019, Living Rev. Comput. Astrophys., 5, 3
Pons J. A., Miralles J. A., Geppert U., 2009, A&A, 496, 207
Raissi M., Perdikaris P., Karniadakis G. E., 2019, J. Comput. Phys., 378, 686
Schiassi E., De Florio M., Ganapol B. D., Picca P., Furfaro R., 2022, Ann. Nucl. Energy, 167, 108833
Stefanou P., Pons J. A., Cerdá-Durán P., 2023, MNRAS, 518, 6390
Sukumar N., Srivastava A., 2022, Comput. Methods in Appl. Mech. Eng., 389, 114333
Thompson C., Duncan R. C., 1995, MNRAS, 275, 255
Thompson C., Duncan R. C., 1996, ApJ, 473, 322
Traore B. B., Kamsu-Foguem B., Tangara F., 2018, Ecological Inform., 48, 257
Viganò D., Garcia-Garcia A., Pons J. A., Dehman C., Graber V., 2021, Comput. Phys. Commun., 265, 108001
Wang S., Yu X., Perdikaris P., 2022, J. Comput. Phys., 449, 110768

This paper has been typeset from a TEX/LATEX file prepared by the author.

Downloaded from https://academic.oup.com/mnras/article/524/1/32/7199782 by Universidad de Alicante user on 13 July 2023

© 2023 The Author(s)
Published by Oxford University Press on behalf of Royal Astronomical Society