Neural Computing and Applications (2024) 36:22677–22696
https://doi.org/10.1007/s00521-024-10178-2

ORIGINAL ARTICLE

On the generalization of PINNs outside the training domain and the hyperparameters influencing it

Andrea Bonfanti[1,2,3] · Roberto Santana[2] · Marco Ellero[3,4,5] · Babak Gholami[1]

Received: 24 August 2023 / Accepted: 1 July 2024 / Published online: 25 September 2024

Abstract
Generalization is a key property of machine learning models to perform accurately on unseen data. Conversely, in the field of scientific machine learning (SciML), generalization entails not only predictive accuracy but also the capacity of the model to encapsulate underlying physical principles. In this paper, we delve into the concept of generalization for Physics-informed neural networks (PINNs) by investigating the consistency of the predictions of a PINN outside of its training domain. Through the lenses of a novel metric and statistical analysis, we study the scenarios in which a PINN can provide consistent predictions outside the region considered for training and hereinafter assess whether the algorithmic setup of the model can influence its potential for generalizing. Our results highlight why overparametrization is not a crucial component in SciML while encouraging overfitting on the training data. Despite being counterintuitive, the outcome of our analysis serves as a guideline for training PINNs for engineering applications.

Keywords Scientific machine learning · Physics-informed neural networks · Differential equations · Generalization

1 Introduction

The field of scientific machine learning [1], encapsulates Machine Learning models which are tailored for scientific applications. Within classical engineering applications, models from SciML have shown practical applicability for forward and inverse problems on partial differential equations (PDEs) [2], constrained shape and topology optimization, and surrogate modeling [3, 4]. PINNs represent a pillar model in the field of SciML: They were first introduced in the 90 s [5, 6] as an alternative method to solve ordinary and partial differential equations, based on the structure and approximation capabilities of Neural Networks. More recently, Raissi et al. introduced an analogous architecture with the aforementioned name in [7], to investigate its range of applicability to forward and inverse problems for PDEs. The analysis provided shows that PINNs are capable of solving classical monodimensional and bidimensional PDEs with little training effort and with satisfying precision. However, the complexity of the training routine rapidly increases if the underlying equations become more complex or the domain of the solution is large and/or multidimensional. Despite their drawbacks, PINNs have already seen their applications for a large variety of fields ranging from biology [8, 9], meteorology [10], system safety [11] and heat transfer in porous media [12]. They are ubiquitous in recent research literature in general, due to the endless potential fields of applications.

✉ Andrea Bonfanti
abonfanti001@ikasle.ehu.eus

Roberto Santana
roberto.santana@ehu.eus

Marco Ellero
mellero@bcamath.org

Babak Gholami
babak.gholami@bmw.de

[1] Functional Development and Data Management, BMW AG, Munich, Germany

[2] Intelligent Systems Group, University of the Basque Country, San Sebastian-Donostia, Spain

[3] CFD Group, BCAM - Basque Center for Applied Mathematics, Bilbao, Spain

[4] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain

[5] Zienkiewicz Centre for Computational Engineering (ZCCE), Swansea University, Swansea, UK

We refer the reader to [13] for a more detailed overview of the PINN model.

The generalization error of a PINN outside of its training domain is a relevant aspect that has not been explored systematically in the literature. Indeed, a consistent generalization at the boundaries of the training domain can be a crucial advantage in multi-body problems [14], shape optimization [15], or applications governed by PDEs solved in domains that are approximated as low dimensional, which is the case in applications as longitudinal domains for nanofluids [16, 17] and thermal variation in longitudinal porous fin [18, 19].

Simple intuitions about the behavior of PINNs outside their training domain are given in publications that exploit decomposition techniques, as for the case of *hp*-VPINNs [20] and Finite-Basis-PINNs [21]. However, the prediction of each PINN outside the training area is only showcased to the reader for visualization purposes. Nevertheless, it is important to be mindful of the behavior of PINN prediction outside the training area, especially when the decomposition relies on soft decompositions [22] or learnable gating mechanisms [23].

In general, a PINN is trained to learn the solution of a PDE in a pre-specified domain, hence, it is not expected to fit the solution of the PDE outside such domain. Indeed, to the best of our knowledge, the error bounds applicable outside the training domain are found in the work of Mishra et al. [24] which focuses on deriving error estimates of trained PINN architectures, based on the value of their loss function and the number of sample points included in the training domain. The only bound applicable to the outside of the training domain grows exponentially with respect to the distance from the training boundaries. Intuitively, this suggests that the error of a PINN cannot be guaranteed to be small when evaluated outside of the training area. However, due to the continuity of the output of neural networks, the solution learned by a PINN can maintain considerable precision in the vicinity of the training area.

For a generic machine learning algorithm, generalization is a key property: it indicates how well a trained model learns the input–output mapping for the underlying distribution of the considered dataset. Indeed, in recent years, the concept of generalization for neural networks has gained increasing interest in the research community due to its atypical behavior in the overparametrized regime given by the double descent curve [25–27]. However, generalization has a particular connotation for a PINN since the model aims to mimic the behavior of a function that satisfies some deterministic properties and does not include any noise component. Hence, applying classical probabilistic bounds does not provide exact information on the accuracy of a PINN, which is a crucial component for applicability to some engineering use cases. Therefore, the concept of generalization of a PINN has to be carefully investigated through methods that minimize the effect of the stochasticity intrinsic to the training of machine learning models. Since it is not possible to provide analytical nor statistical bounds for errors in predictions far from the training domain of a PINN, we develop a novel metric and adopt it to study whether the algorithmic setup of a neural network can impact its behavior outside of the training domain.

In line with the approach of structural analysis followed in this paper, we can find similar works such as [28, 29], which provide an analysis of the performance of PINNs based on several hyperparameters of the architecture, or [30], which studies the effect of different sampling strategies when choosing the training data for PINNs. Another similar project is given by Shama and Shankar [31], where a new methodology proposed to train PINNs is analyzed through the lenses of the depth and width of the neural network used. The analysis presented in this paper stands out from previous research in several ways. Firstly, we believe it is the first study to comprehensively examine the behavior of PINNs beyond their training domain, taking into account the model's hyperparameters. Secondly, we employ statistical methodologies and tests to validate and quantify the influence of these hyperparameters on the PINN's predictive capabilities outside its training domain. Lastly, we conduct a thorough investigation into how varying the aforementioned hyperparameters of the PINN architecture impacts its generalization ability. We further discuss the effects observed based on recent findings in the scientific literature.

Our research is structured as follows: In Sect. 2, we provide a formal definition of the PINN architecture and define our new generalization error metric after clarifying the concept of generalization for PINNs. Then, in Sect. 3, we introduce the test case adopted for the majority of the paper and perform a preliminary numerical and statistical analysis of the PINN architecture to explore its potential and limitations in terms of generalization. Section 4 is then devoted to illustrating the results obtained in terms of the previously defined metric. Hereinafter, we extend our analysis to two additional PDEs in Sect. 5. Finally, the results are discussed in detail in Sect. 6. The paper concludes with Sect. 7, which wraps up the main insights obtained through the experiments and provides recommendations for future works.

2 Physics-informed neural networks

The main concept behind the PINN architecture is to use a classical deep neural network to approximate the solution of a PDE. No a priori knowledge of the correct solution is necessary to train the model, except for its values at the boundaries of the domain and/or the initial condition of the PDE to make the problem well-posed. In the interior of the domain, the network is trained to minimize the residuals of the underlying PDEs, computed through automatic differentiation. By doing so, the network tries to mimic a function that simultaneously fits the solution for the boundary and/or initial condition and is mathematically coherent with the underlying PDEs.

We consider the problem of finding a function $u : \Omega_T \rightarrow \mathbb{R}^n$ which solves a system of potentially nonlinear partial differential equations, which we denote with $\mathcal{F}$, and which satisfies some initial and/or boundary conditions in some domain, which we will denote as $\Omega_T \subseteq \Omega \subset \mathbb{R}^d$. Finding the function u amounts to solving the system shown in Eq. (1)

$$\begin{cases} \mathcal{F}[u(x,t)] = 0 & \forall (x,t) \in \Omega_T \\ u(x,t) = u_b(x,t) & \forall x \in \partial\Omega_T \\ u(x,0) = u_0(x) & \forall x \in \Omega_T \end{cases} \tag{1}$$

with u_b and u_0 being respectively equal to the value of u on $\partial\Omega_T$ and $\Omega_T \cap \{(x,t)|t=0\}$. To solve the above system with a traditional PDE solver, the domain Ω_T has to be discretized through a mesh first, then the equation must be solved in discrete settings on the said mesh. Alternatively, u can be approximated with a fully connected neural network $u_\theta : \Omega \rightarrow \mathbb{R}^d$ with L layers, defined as:

$$u_\theta(x) := W_L \cdot \sigma(\cdots \sigma(W_1 \cdot \sigma(W_0 x + b_0) + b_1) \cdots) + b_L. \tag{2}$$

where $W_i \in \mathbb{R}^{h_i \times h_{i-1}}$ and $b_i \in \mathbb{R}^{h_i}$ denote the weights matrix and bias vector of the i-th hidden layer with dimension h_i. For the sake of compactness, θ represents the collection of all the trainable parameters of the network, i.e., $\theta = \{W_i, b_i\}_{i=1}^{L}$. The activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is a smooth coordinate-wise function, such as the hyperbolic tangent, or the sine function which are common choices for PINNs.

Therefore, the problem of solving the PDE becomes equivalent to finding the values of θ which make u_θ approximate u with a prescribed accuracy in Ω_T. Finding those values requires the usage of a set of N "collocation points" $\{(x_i, t_i)\}_{i=1}^{N}$. Collocation points are typically randomly sampled points that belong to Ω_T and indicate where the residuals of the equations $\mathcal{F}$ are penalized, and a set of N_b points in $\partial\Omega_T$ that represent the boundary and initial conditions satisfied by u.

The typical loss function used for a Physics-Informed neural network is expressed in Eq. (3) and represents the discretized version of Eq. (1).

$$\mathcal{L}\big(\theta, \{(x_i,t_i)\}_{i=1}^{N}, \{(x_j,t_j)\}_{j=1}^{N_b}\big) = \sum_{i=1}^{N} ||\mathcal{F}[u_\theta(x_i,t_i)]||^2 + \sum_{j=1}^{N_b} ||u_\theta(x_j,t_j) - u(x_j,t_j)||^2 \tag{3}$$

It is clear from the definition of $\mathcal{L}$ that its minimizer is the correct solution of the PDE, assuming that the problem is well-posed. However, learning the analytically exact solution with a neural network is practically cumbersome and theoretically infeasible due to limited machine precision. The solution u_θ obtained after training only aims to minimize its loss function in the points sampled before the training, included in some domain $\Omega_T \subseteq \Omega$. Once the training is successful, the model can accurately predict the value of u in Ω_T. This is possible thanks to the continuity of the target function and the known generalization power of neural networks.

2.1 What is generalization for a PINN and how to measure it?

The generalization of neural networks is what allows PINNs to learn the target PDE solution everywhere in Ω_T through some collocation points. However, once the model is trained, evaluating the accuracy of a PINN in some point $x \in \Omega_T$ does not indicate the generalization potential of the model, since the points used during training are typically dense in Ω_T. Therefore, a more suitable way to evaluate the generalization of a PINN can be given by studying the behavior of its output outside of the convex hull defined by the training points. Indeed, u_θ is trained until it accurately approximates u inside Ω_T but there is no guarantee that the provided approximation is also valuable in $\Omega \backslash \Omega_T$, where u evolves through its analytical extension.

To study the generalization potential of PINNs, we define and use the value G_l^ϵ, which we will refer to as "Generalization Level" (G_l). Consider the set of parameters of a PINN to be $\{\theta_i\}_{i=1}^{N_\theta} \in \Theta \subseteq \mathbb{R}^{N_\theta}$, and the remaining hyperparameters of its training algorithm to be collected in an additional vector variable $H \in \mathcal{H}_{\text{PINN}}$. Consider also u_θ to identify a PINN initialized and trained according to the hyperparameters H and with trainable parameters in Θ. We define the GL of the hyperparameters of a PINN as in Definition 4.

$$G_l^\epsilon(\Theta, H) = \min_{u_\theta}\left\{\max_{\Omega_G \subseteq \Omega \setminus \Omega_T}\left\{\frac{l(\Omega_G)}{l(\Omega_T)} \quad \text{s.t.:} ||u(x) - u_\theta(x)|| \leq \epsilon \quad \forall x \in \Omega_G\right\}\right\} \tag{4}$$

Here, the function l represents the area—length for one-dimensional cases—of the respective domain, while ϵ is a suitable threshold defined for the underlying problem. Therefore, G_l takes as input the space of the parameters Θ and the hyperparameters used for training a PINN and outputs the relative size of the biggest area outside the training domain in which the error remains bounded by ϵ for all the networks. This metric is equivalent to the length of the x segment outside of the training domain in which the prediction is accurate without noticeable variability. The value of G_l depends on the structure and dimensionality of Θ and on the values of $H \in \mathcal{H}_{\text{PINN}}$, which in this paper are considered as $H = (N_{\text{CP}}, A)$: the number of collocation points N_{CP}, and the size of the training domain A.

The chosen metric is affected as little as possible by the stochasticity intrinsic to the training of a neural network. Indeed, we expect high G_l values to indicate a more suitable structural choice for a PINN.

2.2 Parameters influencing the potential in generalization

It is reasonable to ask whether this extrapolation potential is affected by some of the hyperparameters of the neural network or the algorithmic setup of the PINN training. Among the possibly influential parameters, we consider the following ones:

- *Network complexity* The number of neurons and the number of layers used for the neural networks used to mimic the solution of the chosen PDE.
- *Activation function* The activation function used for the neural architecture.
- *Collocation points* The number of reference locations in the PDE domain considered to compute the loss function of the PINN during training.
- *Domain size* The size of the domain Ω_T where the PINN is trained to solve the PDE.
- *Training method* The algorithm used to optimize the parameters of the network. This represents an interesting scenario as it can influence the learning based on the aforementioned hyperparameters.

There are also other components of the PINN architecture that can influence the behavior of the solution outside the training domain. The complexity of the target function is a clear example that is not treated in this paper as it represents an external component that is not known a priori when training a PINN. Moreover, the kind of machine learning architecture used—adding skip connections, dropout, or a residual block—can also affect the quality of the generalization obtained. We restrict our study to the base case of multilayered perceptrons for simplicity and because the underlying problem chosen as a base case can be easily handled by such models.

3 Problem description and explorative analysis

For a preliminary and intuitive analysis, we choose a one-dimensional Poisson equation with a source term in the domain $[-\pi, \pi]$, presented in Eq. (5), as an academic example to showcase and analyze the output of a trained PINN. The presented equations are solved through PINNs in [32], such a choice for the equation relies on the richness of frequencies included in its solution. Moreover, training PINN architectures is also extremely efficient in one dimension, which facilitates the analysis provided in this paper.

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} = f \quad x \in [-\pi, \pi] \\ u(-\pi) = u(\pi) = 0 \end{cases}. \tag{5}$$

With $f : \mathbb{R} \longrightarrow \mathbb{R}$, the source term, being defined as:

$$f(x) = \sum_{k=1}^{5} 2k \sin(2kx) \tag{6}$$

The correct solution of this equation can be easily derived analytically by integrating twice the source term in $[-\pi, \pi]$ and setting the integration constants to zero to match the boundary conditions. The function resulting from this operation and solution to Eq. (5) is given by:

$$u(x) = \sum_{k=1}^{5} \frac{1}{2k} \sin(2kx) \tag{7}$$

To train the PINN architectures, we sample 100 collocation points in the domain $[-\pi, \pi]$ with the Latin hypercube sampling strategy and use a two-layer neural network architecture with 50 neurons per layer and hyperbolic tangent as activation function. To train the model, we restrict it to 10^4 iterations of the Adam optimizer [33], followed by the L-BFGS optimizer [34] to fine-tune the results, limited to 10^4 iterations and a tolerance of 10^{-8}. The training process takes a few minutes to converge to a solution and outputs predictions that are visually indistinguishable from the correct solution, which can be seen in Fig. 1.

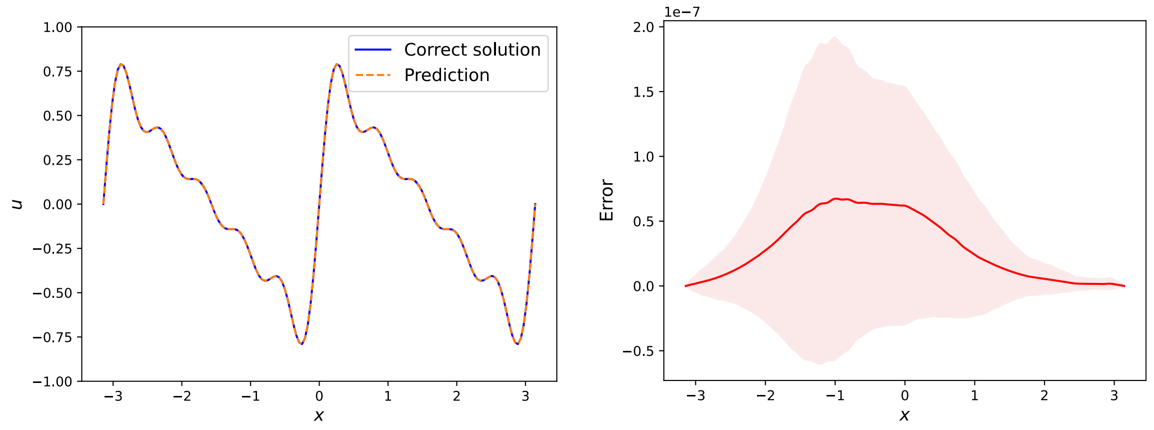

Fig. 1 Correct versus predicted solutions (left) and relative L^2 error (right) for 100 randomly initialized and trained PINN architectures. Variance is also included in both figures but it is too small to be visible in the leftmost figure

In the same way, we used automatic differentiation to compute the equation residuals, we can also use it to access the derivatives of u_θ and compare them with the true ones. Due to the PDE that we are solving, it is already known that the prediction of the second derivative of u will be accurate. Therefore, we compute and showcase the comparison of derivatives up to the fourth order along with the related relative L^2 error made in prediction. The choice of using the relative error rather than the classical squared error used in the previous section is due to the difference in magnitude of the values of u and its first four derivatives, which go from unitary order to 10^3. The results are shown in Fig. 2. Again, it is barely possible to distinguish the correct values from the ones obtained through the PINN architectures.

3.1 Generalization beyond the training domain

We now proceed to investigate the concept of generalization for a PINN architecture expressed in Sect. 2. We first split the training domain of Eq. (5) into three subdomains, namely $\Omega_1 = [-\pi, -\frac{\pi}{3}]$, $\Omega_2 = [-\frac{\pi}{3}, \frac{\pi}{3}]$, and $\Omega_3 = [\frac{\pi}{3}, \pi]$ and perform training of 100 randomly initialized PINN architectures with identical structures, to grasp the effect of

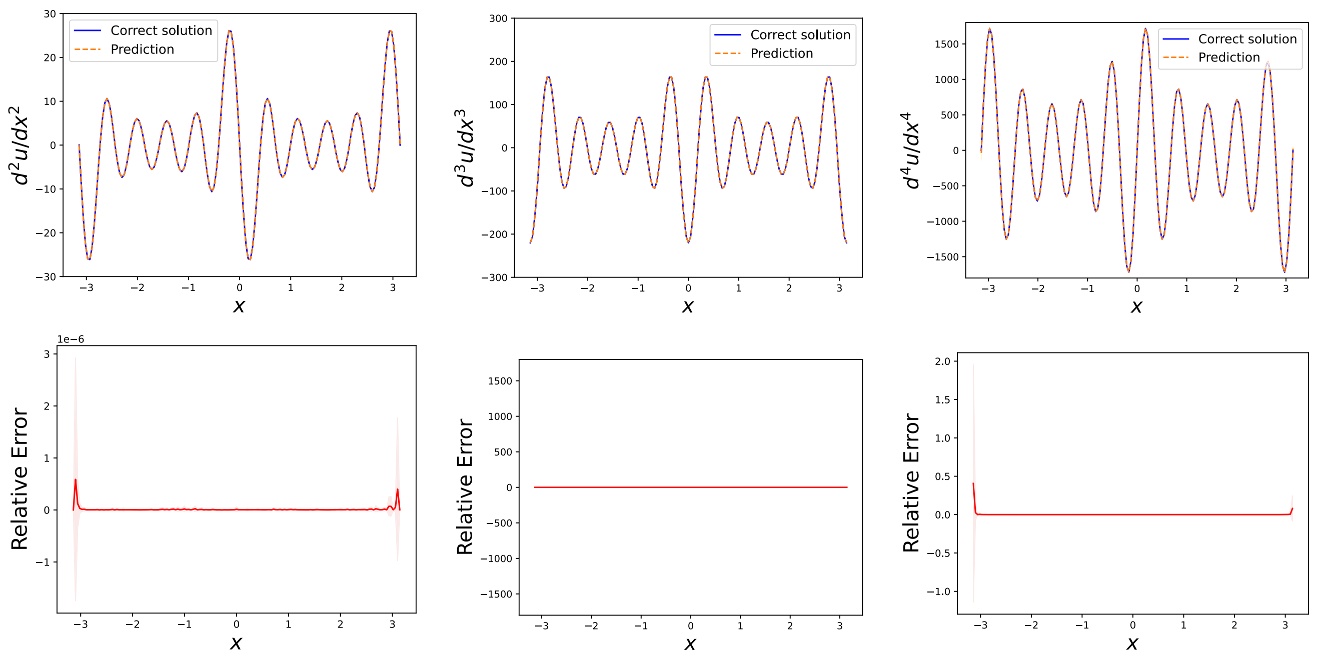

Fig. 2 True versus predicted values for the second, third, and fourth derivatives of the target solution for five randomly initialized and trained PINN architectures (top row) along with their respective relative L^2 errors in prediction (bottom row). The variance of the prediction is also included but almost indistinguishable

stochasticity in the training procedure. From an analytical perspective, the exact solutions u^j solving Eq. (5) in the respective subdomains Ω_j are all represented by the same function, due to the uniqueness of the solution. This would mean that each u^j_θ in each Ω_j should theoretically behave the same in the whole Ω, which is not the case in practice. Based on the continuity of u_θ and the target function, the only fair assumption is that the PINN prediction will shortly follow the analytic function. The results plotted in Fig. 3 show the behavior described above, along with the expected degradation of prediction outside the training area.

Predictions are extremely accurate in the training area and maintain precision close to its boundaries. Indeed, since the network can properly approximate the derivatives of the target function at some point, then it can also approximate the target function properly in a ball around said point. However, the prediction quickly becomes unreliable and subject to stochasticity further away from the training domain. The average PINN prediction suggests that an accurate function evaluation can be done with confidence only in the immediate proximity of the convex hull defined by the collocation points used in training. This concept is aligned with the analytical error bounds in [24, 35], where the tightest error estimates present an exponential increase in time when moving far away from the training area. This does not strictly imply that generalization is not possible far from the boundaries of the training domain but it is consistent in its proximity, which is why it is important to understand whether the behavior of the PINN outside the training domain can be controlled through the hyperparameters set before the training process.

3.2 Statistical analysis

Through a statistical analysis, we evaluate whether generalization to points outside the training domain is possible for a trained PINN architecture. To do so, we test an empirical bound on the probability of a PINN being able to achieve a prescribed precision in some area of size δ outside of the training domain. To capture variability, we study the value of the generalization level obtained before the minimization over u_θ of Definition 4.

$$g_l^\epsilon(u_\theta) = \max_{\Omega_G \subseteq \Omega \setminus \Omega_T} \left\{ \frac{l(\Omega_G)}{l(\Omega_T)} \quad \text{s.t.:} ||u(x) - u_\theta(x)|| \le \epsilon \quad \forall x \in \Omega_G \right\} \tag{8}$$

Hence, we consider the variable g_l^ϵ in Definition 8, as the subject of our statistical analysis. Then, we study the probability $P(g_l^\epsilon \ge \delta)$ with $\delta > 0$. Since g_l^ϵ represents a non-negative length, we can bound the probability $P(g_l^\epsilon > \delta)$ with Markov's inequality for moments of order m, which holds for any value of m. The general formulation is given in Eq. 9, and the classical Markov's inequality is obtained when $m = 1$. Results of the obtained bounds are available in Fig. 4 for the number of parameters—neurons and layers—, number of collocation points, and domain size, with $m = 1, \ldots, 5$, $\delta = 0.25$ and $\epsilon = 10^{-3}$.

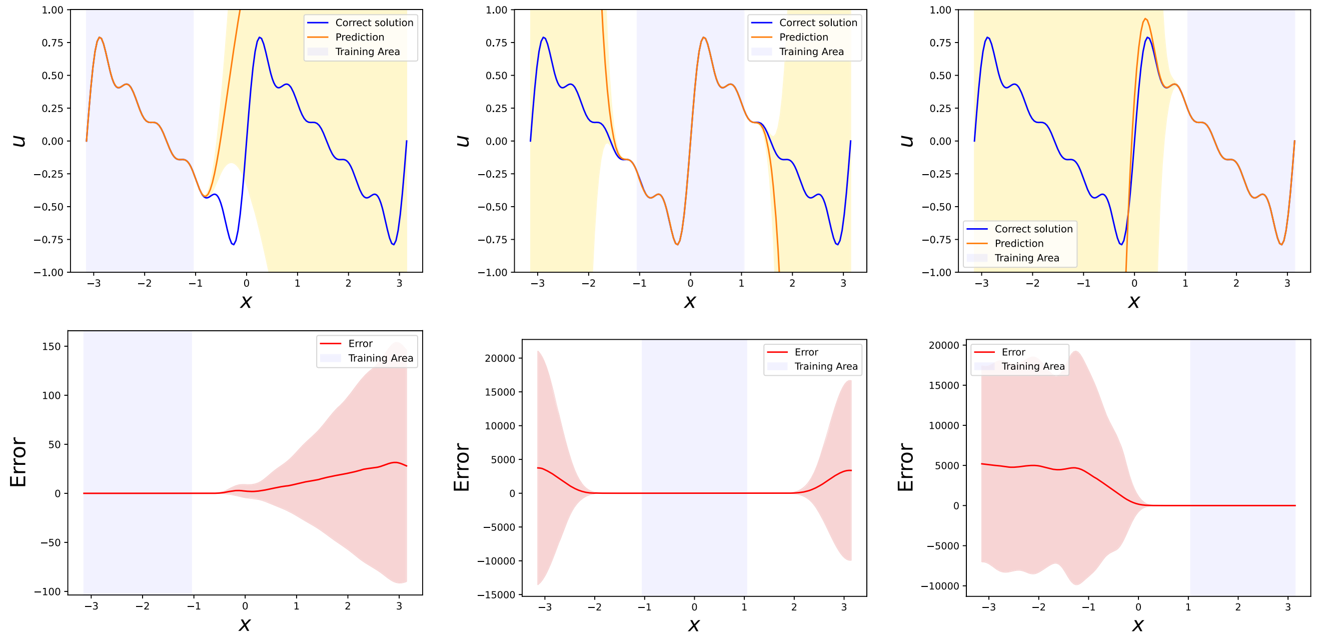

Fig. 3 Prediction and ground truth (top row) and related error (bottom row) for PINNs trained in the respectively light blue area. The shaded gold and red areas represent two times the standard deviation of the prediction and the error of PINNs respectively (colour figure online)

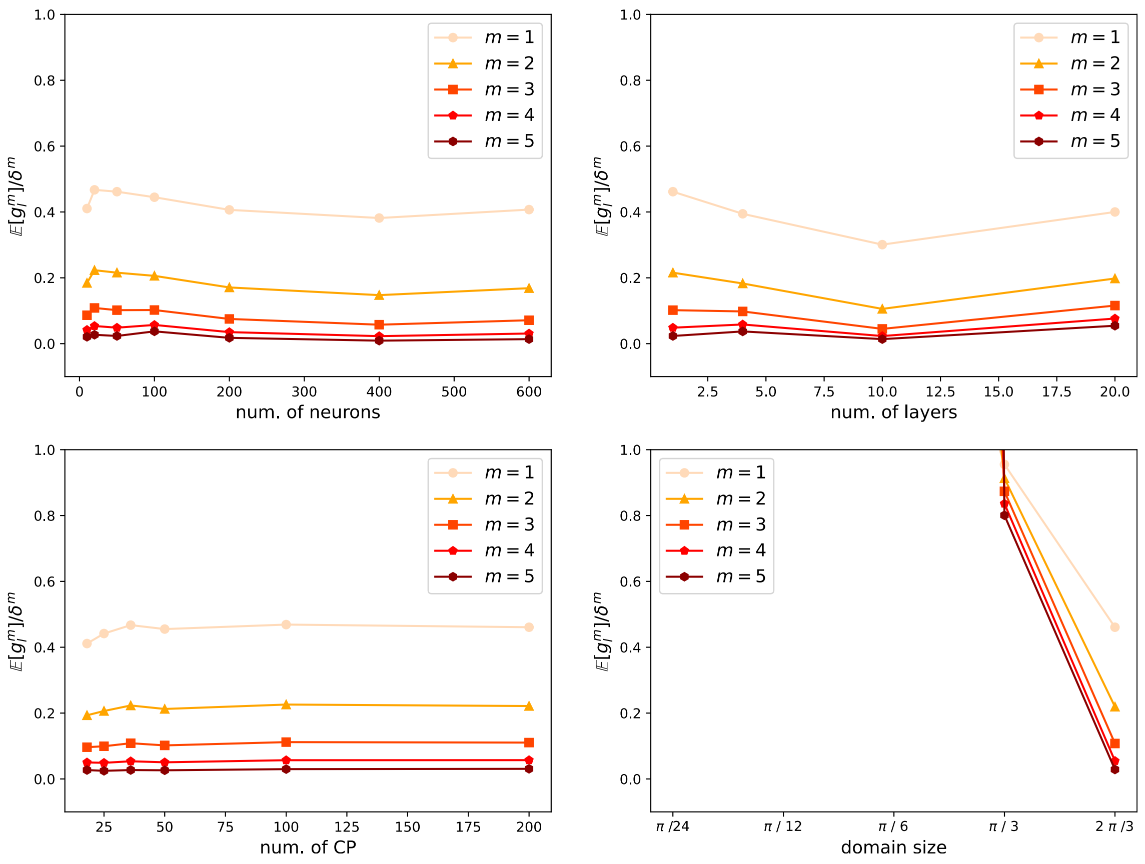

Fig. 4 Bounds on $P(g_l \geq \delta)$ for a different number of neurons and layers (top row), number of collocation points, and domain size (bottom row) for $\delta = 0.25$. Values are limited between 0 and 1, values that falls outside the plots indicate a bound larger than 1

$$P(|g_l| \geq \delta) \leq \frac{\mathbb{E}[|g_l|^m]}{\delta^m} \tag{9}$$

The reasoning behind the choice of Markov's inequality is that for m large enough, the bound will converge to zero if there is no evidence that the generalization level could be greater than δ. This is the case of $\delta = 0.25$ except for small domain sizes, where no relevant bound can be given through Markov's inequality. This suggests that, in general, PINNs fail to extrapolate on large regions. On the other hand, for small values of δ, the obtained bound becomes almost always greater than 1, which does not provide any relevant insight. Indeed, as $\delta \rightarrow 0^+$ the bound will always diverge to infinity as, for a properly trained PINN, $g_l > 0$ due to continuity and so $\mathbb{E}[|g_l|^m]y > 0$ for finite values of m.

Having assessed that PINNs can generalize to a relatively small region outside the training area, we proceed to identify whether the network hyperparameters have a statistically significant impact on the generalization potential by performing the Kruskal-Wallis H test for independent samples [36]. The latter sets as the null hypothesis the equality in the median of samples coming from multiple groups having independent probability distribution functions. The Kruskal–Wallis H test is based on the rank of the data samples and rejects the null hypothesis when one of the samples appears to have greater values than the other. For our analysis, the chosen groups are identified by g_l^ϵ values obtained by varying one of the aforementioned hyperparameters.

The test results are presented in Table 1, where the value of the H statistics are shown for each hyperparameter-based group and each value of ϵ. In the aforementioned Table, we highlight the H values in bold when the p value is above 0.01, which represents cases in which the H statistic cannot validate any statistically significant

Table 1 Kruskal–Wallis' H statistic value for all hyperparameters and all values of ϵ studied

Hyperparameter versus ϵ	1e−2	1e−3	1e−4	1e−5
N. of Neurons	238.15	183.46	206.35	235.19
N. of Layers	37.09	98.72	123.91	111.83
N. of CP	16.38	**6.31**	**8.86**	18.53
Domain size	48.68	260.27	324.89	346.30

Values in bold refer to cases where statistically significant differences cannot be validated difference in distribution. Except for the number of collocation points used in training, all hyperparameters present a strong statistical significance, for any ϵ, to reject the null hypothesis of the Kruskal–Wallis test. This means that the values of g_l^ϵ are directly influenced—with very high probability—by each of the hyperparameters considered in our study. To identify where the statistically significant differences appear within each group, we also perform pairwise Mann–Whitney U tests—the two-sample version of the Kruskal–Wallis test—for each of the hyperparameters that have shown statistically significant differences. The results obtained through these post hoc tests can be summarized as follows:

- *Number of neurons* statistically significant differences seem to be present in blocks. Networks with 10–100 neurons appear to present similarities within each other, as do networks with 200–600 neurons. However, the null hypothesis is always rejected between the two groups, regardless of the values of ϵ.
- *Number of layers* similar to the case of the number of neurons, results obtained for architectures with 4 or more layers appear to be correlated, while the null hypothesis is always rejected between the one-layered architecture and the other networks.
- *Domain size* The null hypothesis is always rejected, except for the two configurations with larger domain sizes, which seem to be correlated when ϵ is very small.

Finally, we can conclude that varying the number of collocation points does not significantly influence the generalization level of a PINN. On the other hand, the network complexity and the size of the PDE domain have a statistically significant impact on the value of the generalization level.

4 Hyperparameter study through the generalization level

In this section, we provide numerical results on the power of generalization for the aforementioned hyperparameters of the PINN algorithm. The results obtained for the G_l metric are presented alongside the training time, to provide a more complete understanding of the benefits of specific parametric settings. Moreover, for each hyperparameter, we also provide further visualizations of the behavior of the predictions when the related hyperparameter changes. For all hyperparameter analysis, we perform training of 100 randomly initialized PINN architectures restricted to 10^4 Adam iterations with learning rate 10^{-3} and a subsequent L-BFGS optimizer limited to 10^4 iterations and 10^{-8} tolerance. The results are then expanded in Sect. 5 and thoroughly discussed in Sect. 6.

4.1 Network complexity

The first case taken into consideration is the complexity of the network adopted to define the PINN architecture. The analysis is divided into the number of neurons and the number of layers, to properly grasp the effect of each of the two components.

4.1.1 Number of neurons

We consider the number of neurons of single-layered neural network architectures. We test several PINN architectures characterized by the number of neurons of their hidden layer and evaluate them based on the value of the GL metric. To fully grasp the effect of under and overparametrized regimes, we test architectures with 10, 20, 50, 100, 200, 400, and 600 neurons. The results obtained for this analysis are shown in Fig. 5, along with the average training time per architecture configuration and its variance.

Regarding the value of the generalization metric, the value is always zero for the smallest architecture since, on average, 10 neurons are not enough to learn the target function. Increasing the number of neurons rapidly enhances the generalization power of the architecture, which seems to be saturated as soon as 50 neurons are used. When the number of neurons is further increased, no clear enhancement is obtained in terms of the generalization metric we have defined. The choice of a 50-neuron architecture seems to be the most appealing both in terms of average training time and variability.

In Fig. 6, it is possible to also visualize the impact on predictions when the number of neurons in the hidden layer is increased. Increasing the width of the network seems to generally skew the outside prediction to an overestimation by the PINN, leading to a solution that rapidly departs from the correct one. This effect is not visible for a relatively small number of neurons—10, 20, and 50—and quickly becomes remarkable when the overparametrized regime is reached. It is also interesting to observe that the overparametrized regime does not appear to be beneficial, in terms of the width of the network, to predict the solution of the PDE.

4.1.2 Number of layers

An analogous analysis is conducted for neural architectures with a fixed number of neurons and a variable number of layers. We provide our analysis with as many as 50 neurons per layer and study architectures with 1, 4, 10, and 20

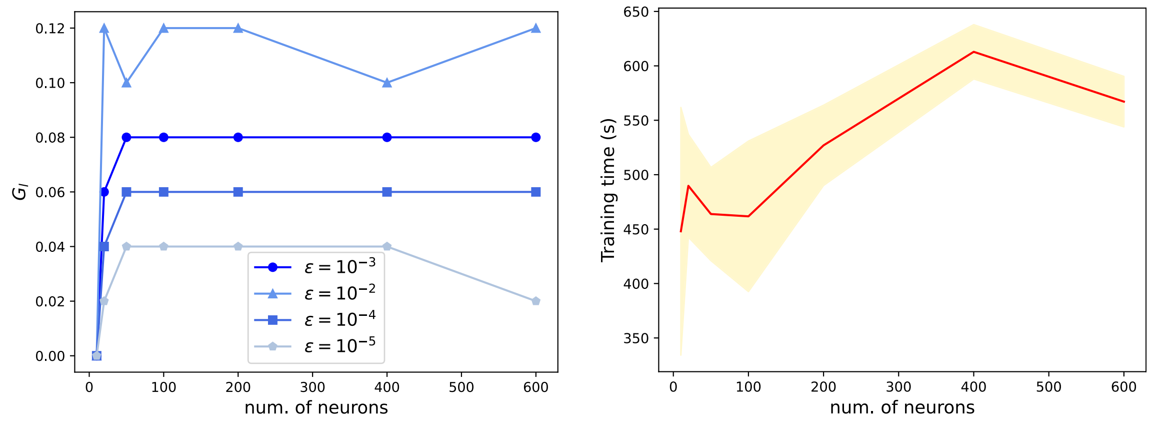

Fig. 5 G_l values for different values of ϵ when the number of neurons is increased (left) and training time (mean and variance) of each training configuration (right)

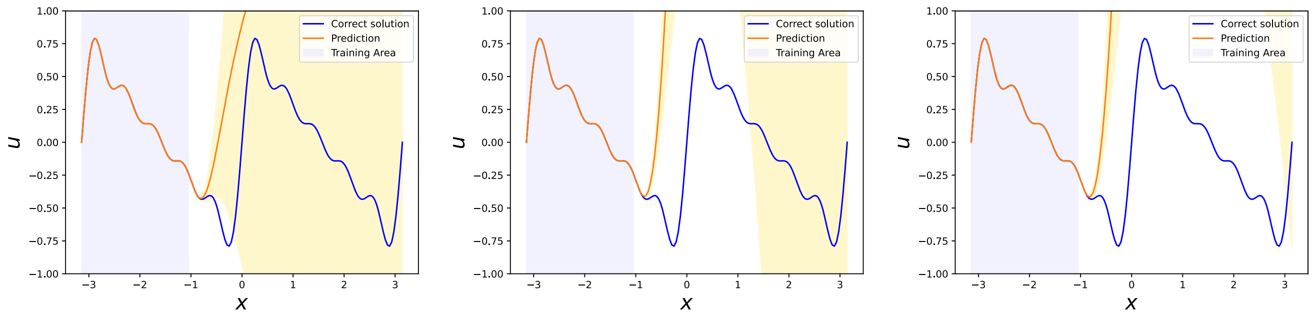

Fig. 6 PINN predictions (with variance) and ground truth for single layer architectures with 20 (left), 400 (center), and 600 (right) neurons, trained in the light blue area (colour figure online)

layers. Once more, the results are presented along with the statistics of the training time per network architecture and can be seen in Fig. 7.

The results obtained for the generalization level show on average a decay of the metric value when the number of layers is increased. The exception is the case of five layers or less, where it is possible to notice a slight improvement in the metric for some ϵ. It is also possible to see that the deepest architecture—20 layers—gives almost always zero as generalization metric: This is because the training process consists of a limited number of iterations and is stopped afterward. The number of epochs is not enough to make such a big architecture converge properly. However, we found that even when trained for longer periods, such

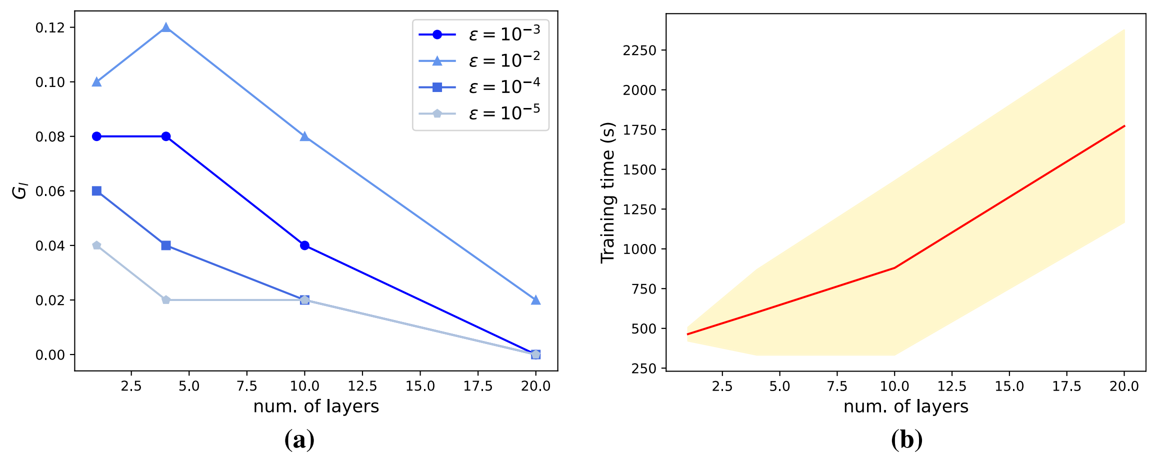

Fig. 7 G_l values for different values of ϵ when the number of hidden layers of the network is increased (**a**) and training time (mean and variance) of each training configuration (**b**)

deep architectures struggle to achieve a suitable solution and do not achieve relevant results in terms of the generalization level.

In Fig. 8, we also provide a visual representation of the effect on the predicted solution of an increasing number of layers. The depth of the PINN appears to smooth out the prediction of the network to a constant value outside the boundary of the training domain. Moreover, this smoothened guess seems to have reduced variability when more layers are added. This impacts the G_l metric negatively since its value is based on the outcome of the worst-performing architectures among all the trained ones.

4.2 Number of collocation points

We perform our routine analysis on single-layered neural network architectures with 20 neurons, which take as training input a variable number of collocation points. In our setting, we test the case of 18, 25, 36, 50, 100, and 200 collocation points randomly sampled in each iteration with the Latin hypercube sampling strategy over the training domain $[-\pi, -\frac{1}{3}\pi]$. The results obtained for the generalization level and the training time are given in Fig. 9.

The resulting values of our generalization metric show that 18 collocation points are not enough to learn the target function. Once the number of collocation points is increased, the value of the generalization metric also increases. As foreshadowed by the preliminary analysis, the generalization level is not largely affected by the number of collocation points used, as long as enough points are provided. Moreover, for the case used in our analysis, 36 collocation points are sufficient to learn the target function and obtain a good value of generalization.

Furthermore, the number of collocation points does not present any major pattern in the prediction of the PDE solution. The only visible effect can be found in the prediction of the second derivative of the target function in the low numbers regime. The latter is provided in Fig. 10, which includes the prediction results of the second derivative for PINNs trained with 18, 25, and 36 collocation points sampled in the training domain. In this visualization, it is possible to see that an increased number of collocation points reduces the variability of the predictions outside the training area. Among those architectures, the latter configuration is the only one that completely reached convergence, as it is possible to identify a non-zero variance in the predictions obtained via performing training with either 18 or 25 collocation points.

4.3 Domain size

With domain size, we refer to the length—or area—of the region where the collocation points are sampled and where the underlying PDE is defined. For the analysis of this section, we consider decreasingly smaller domains that are defined as $\Omega_i = [-\frac{1}{3}\pi - \frac{2}{3\cdot 2^i}\pi, -\frac{1}{3}\pi]$ for $i = 0, 1, 2, 3, 4$. Since we have already studied the effect on the generalization of the density of CP in the training domain, we reduce the number of collocation points picked for training the network coherently with the decrease in the size of the domain. The network architecture, instead, is kept fixed to a single-layered neural network with 20 neurons, to properly grasp the effect in generalization of the domain size. For the sake of comparability, the value of the G_l metric is computed for the right side of the outer domain, which means that it is computed for all training setups in the area where $x \geq -\frac{1}{3}\pi$. The results for the generalization metric and training time are shown in Fig. 11.

Regarding the generalization level, we can see that architectures trained on the small domains provide a consistent generalization to points outside Ω_T. Indeed, the G_l values obtained in our analysis show a clear decay with respect to the size of the domain. On the other hand, the training time presents an interesting behavior: an increased time demand for larger domains on average, but a much larger variability in training time for the intermediate values tested. This is most likely due to the richer variety of functions that exhibit the behavior described by the PDE residuals in the training domain. The case of the smallest domain represents once more an exceptional case, for

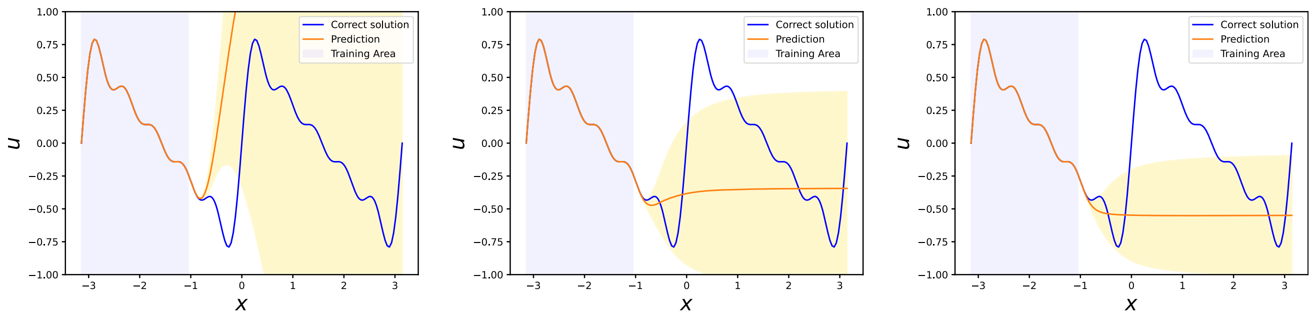

Fig. 8 PINN predictions (with variance) and ground truth for architectures with 1(a), 4(b), and 10(c) layers and 50 neurons per layer, trained in the light blue area (colour figure online)

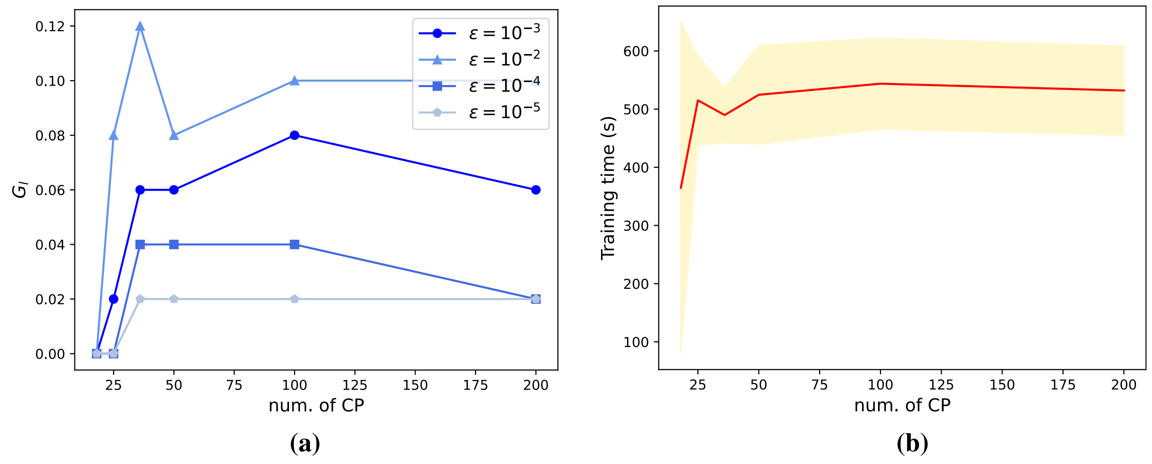

Fig. 9 G_l values for different values of ϵ when the number of collocation points is increased (**a**) and training time (mean and variance) of each training configuration (**b**)

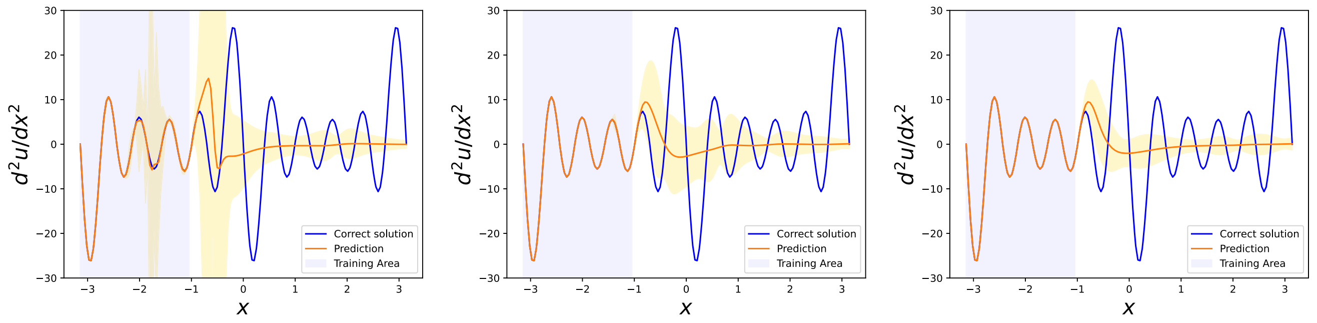

Fig. 10 PINN predictions (with variance) and ground truth of the second derivative of the PDE solution for architectures trained with 18 (left), 25 (center), and 36 (right) collocation points distributed in the light blue area (colour figure online)

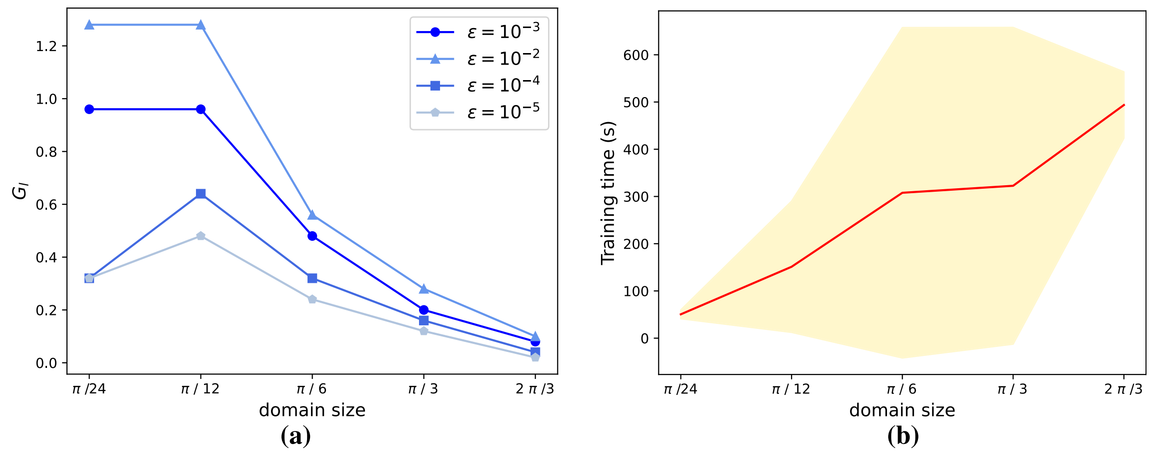

Fig. 11 G_l values for different values of ϵ when the size of the training domain is increased (**a**) and training time (mean and variance) of each training configuration (**b**)

which the training time presents a substantially small variance.

The case of the domain size also represents one of the most interesting impacts on the outcome of the prediction. For visual purposes, we present the effect of the domain size in terms of the prediction of the second derivative of the PDE solution. The results can be seen in Fig. 12 for decreasing domain sizes. The main impact on predicting in small domains is given by reduced variability of the prediction outside of the training area. Since the metric we defined evaluates the best–worst result, the variability of the prediction is a very impactful factor in its value.

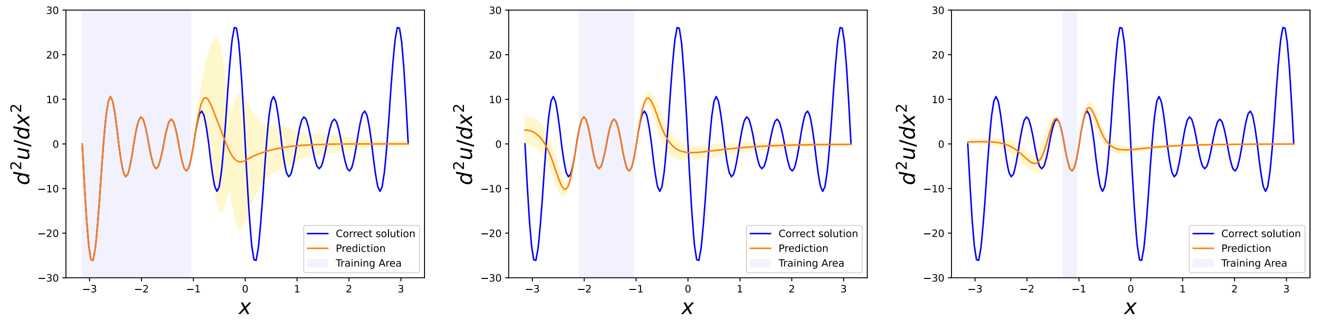

Fig. 12 PINN predictions (with variance) and ground truth of the second derivative of the PDE solution for architectures trained on the domain highlighted by the light blue area (colour figure online)

4.4 Activation function and training algorithm

The activation function and training algorithm are two core components of a neural network that should be treated carefully. In particular, we opt to evaluate the two components simultaneously since the gradients computed by the training optimizer are heavily impacted by the activation function chosen. Moreover, training time is not considered for evaluation since the results obtained with two different optimizers are influenced by the efficiency of the implementation. As for the optimization algorithm, we compare 10^6 iterations of the first-order Adam optimizer against 10^3 iterations of the second-order Levenberg–Marquardt and use them to train 20 models with hyperbolic tangent, sine, rectified linear unit (ReLU), and sigmoid linear unit (SiLU) as activation functions.

The results obtained for the training algorithm and the activation function are shown in Fig. 13 along with the value of the L^2 error achieved on the test set by the trained PINNs. It is clear from the results that ReLU is not a suitable activation function for learning the solution of the Poisson equation. Indeed, PINNs with ReLU as activation do not converge to the correct solution, which is indicated by the G_l values obtained and visible in Fig. 14. On the other hand, the L^2 error on the test set erroneously indicates the ReLU as the best choice for the chosen problem. Indeed, for the PDE instance chosen, the trivial solution $u_\theta(x) = 0$ is not far from the correct solution, despite not capturing any behavior of the ground truth. Regarding the remaining activation functions, the G_l values do not show large discrepancies, except for the sinus, which depicts a different behavior between models trained with first or second-order methods.

Regarding the two optimization algorithms considered, the values achieved are roughly the same. However, PINNs trained with second-order methods tend to achieve a better performance. It is worth noticing that similar generalization performance is obtained by the trained PINNs despite them reaching very different loss values during training. In particular, models trained with a second-order method achieve valuable results in generalization despite "overfitting" the test set. Moreover, models trained with LM indicate the sine as the most suitable activation function. This is coherent with the fact that the correct solution is written as a sum of sine functions.

The predictions showcased in Fig. 14 provide a hint on the previously discussed motivation of the low L^2 error on the test set obtained by models with ReLU as activation. On the other hand, models that use the sine as activation function struggle to converge. On the other hand, models with sine as activation function and trained with LM show an average solution that contains frequencies very similar to that of the analytical extension of the correct solution. The same does not happen when using a first-order method due to the spectral bias of PINNs, which is alleviated when adopting second-order optimization methods [37].

5 Extension to different cases and higher dimensions

The results obtained in this paper can be extended to any kind of partial differential equations. In particular, if the considered PDE is monodimensional, it is rather straightforward to repeat the tests performed in our analysis. To validate our findings in more general scenarios, we provide the values of the generalization level for two additional well-known PDEs: the damped harmonic oscillator and the equation of motion of the pendulum. The goal is to determine whether adding different derivatives and/or nonlinearities carries large differences with respect to the results obtained so far. For the sake of compactness, we collect the results for the aforementioned PDEs in Figs. 15 and 16.

5.1 Damped harmonic oscillator

The longitudinal position over time $u : \mathbb{R} \rightarrow \mathbb{R}$ of the mass of a damped harmonic oscillator is described by the following PDE:

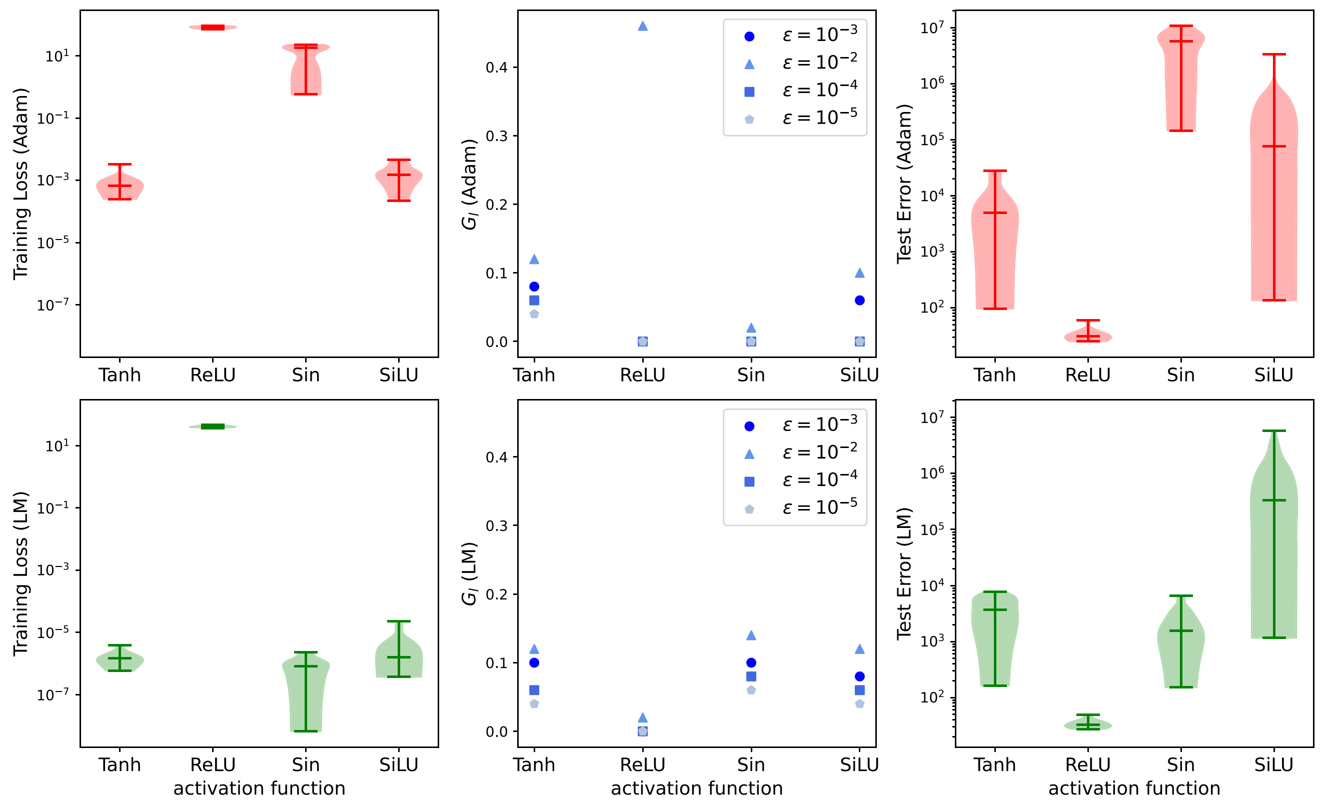

Fig. 13 Violin plot of the Training Loss achieved (left), G_l values for different values of ϵ (center) and violin plot of the Test Error (right) for PINNs with different activation functions trained with Adam (top row) and Levenberg–Marquardt (bottom row)

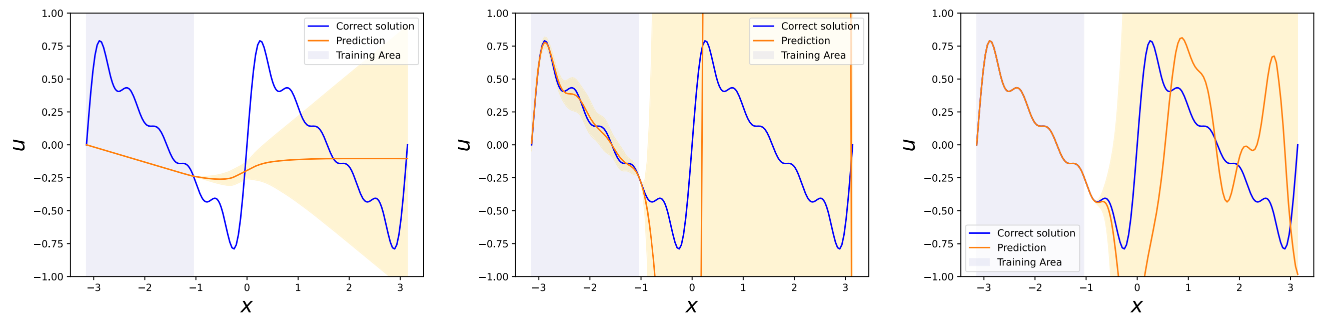

Fig. 14 PINN predictions (with variance) and ground truth of the PDE solution for architectures trained with Adam (left, center) and LM (right) on the domain highlighted by the light blue area. The activation function of the models is the ReLU (left) and the Sinus function (center, right) (colour figure online)

$$\begin{cases} m\dfrac{\partial^2 u}{\partial x^2} + \mu\dfrac{\partial u}{\partial x} + ku = 0 \quad x \in [0,1] \\ u(0) = 1 \\ \dfrac{\partial u}{\partial x}(0) = 0 \end{cases} . \tag{10}$$

The correct solution in the underdamped regime is defined as:

$$u(x) = 2Ae^{-\delta t}\cos(\omega t + \phi) \tag{11}$$

where the parameters introduced in the equation of the solution are computed as follows:

$$\delta = \frac{\mu}{2m}, \quad \omega = \sqrt{\frac{k}{m}}, \quad \phi = \arctan\left(-\frac{\delta}{\omega}\right), \quad A = \frac{1}{2\cos\phi}$$

For the analysis performed in this paper, we choose unitary mass, damping coefficient $\mu = 4$, and reactivity $k = 400$ to ensure the underdamped regime, which is richer in terms of dynamics.

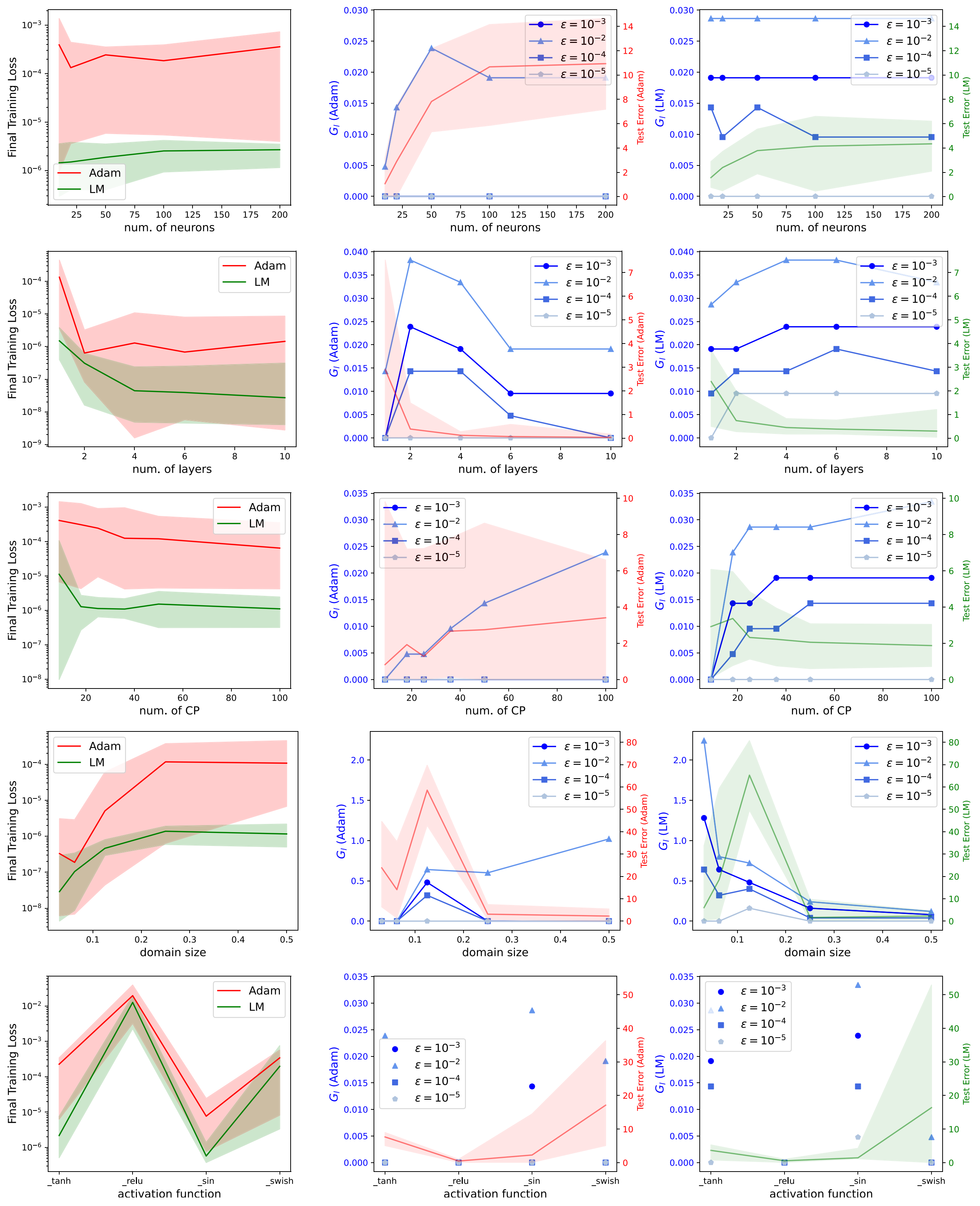

Fig. 15 Results on the Damped Harmonic Oscillator: Training Loss (left) and Generalization level along with L^2 Test Error for models trained with Adam (center) and LM (right) for the algorithmic parameters of a PINN

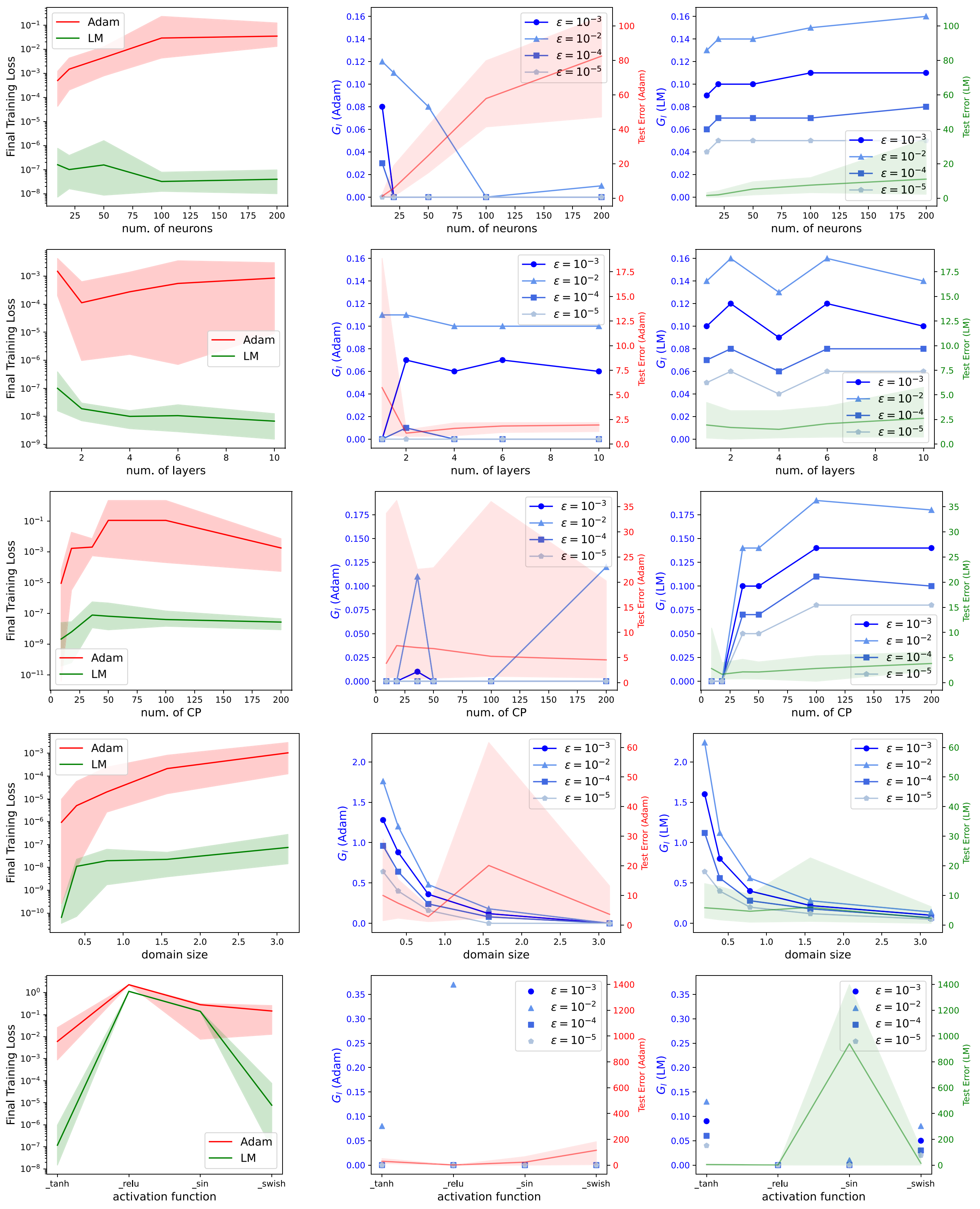

Fig. 16 Results on the nonlinear regime of the Pendulum equation: Training Loss (left) and Generalization level along with L^2 Test Error for models trained with Adam (center) and LM (right) for the algorithmic parameters of a PINN

5.2 Pendulum equation

The variation of the angle of a pendulum oscillating is often studied in the linear regime through the small-oscillation assumption, which is based on the approximation $x \approx \sin(x)$. When said approximation is not reasonable, the angular position $u : \mathbb{R} \to \mathbb{R}$ of a pendulum is described by the following nonlinear PDE:

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} + \frac{g}{l}\sin(u) = 0 \quad x \in [0, 2\pi] \\ u(0) = \frac{\pi}{2} \\ \frac{\partial u}{\partial x}(0) = 0 \end{cases} . \tag{12}$$

In the above equation, g is the gravitational acceleration and l is the length of the string to which the mass of the pendulum is attached. For the specific PDE instance used in this study, we pick $g/l = 10$. The correct solution is given by:

$$u(x) = 2\sin^{-1}\left(\frac{\mathrm{cd}\left(\sqrt{10}x \middle| \frac{1}{2}\right)}{\sqrt{2}}\right) \tag{13}$$

where $\sin^{-1}$ denotes the inverse of the sinus function—arcsin—and cd indicates the Jacobian elliptic function cd [38].

Overall, the additional scenarios considered emphasize the most relevant findings deducted from the previous analysis. In particular:

- A very large number of network parameters (both width and depth) does not seem to carry relevant advancements when training a PINN with first-order optimizers. Similarly, when using a second-order method, overparametrization does not enhance the generalization of the solution obtained;
- The number of collocation points should be sufficient to cover the whole training area properly, increasing the number of points further does not seem to provide large benefits, unless for second-order methods.
- The choice of the activation function should be tailored to the problem at hand. The classical choice of the hyperbolic tangent indicated to be a suitable choice for all the PDEs considered in our study. However, activation functions that mimic the behavior of the solution can carry consistent advantages in terms of both accuracy and generalization;
- The use of a second-order method such as Levenberg–Marquardt consistently provides better results in terms of accuracy, generalization, and, based on our experience, training time.

5.3 Higher dimensions

On the other hand, extending our analysis to PDEs in higher dimensions can be non-trivial conceptually and, more specifically, computationally. Conceptually speaking, the proposed generalization level G_l represents the value of some distance d of data points from the convex hull of the training samples. When new unseen samples are fed to the model, the predicted value is guaranteed to be precise—up to a suitable threshold ϵ–, provided that the new samples are at most G_l away from the training data. In machine learning applications, it is fundamentally hard to guarantee a certain precision, as the underlying problems are typically affected by stochasticity. Thus, any guarantees are typically possible only with high probability and are not necessarily true in general. However, stochasticity should not appear when the underlying problem is the solution of a PDE, which is why we adopted a definition for G_l that should minimize the stochasticity intrinsic in a neural network architecture.

Based on the considerations above, it is possible that for different settings it is better to consider an alternative formulation of the generalization level that does not focus on the maximal set where a suitable generalization is achieved, but rather on the distance from the training set within which every prediction remains accurate. From the perspective of Definition 4, instead of searching for the maximal set $\Omega_G \subseteq \Omega \subseteq \Omega_T$, the aforementioned alternative should focus on a maximal distance value d_M, computed through some point-to-set distance d such as the Wasserstein distance. Such a metric can be defined as in Definition 14.

$$G^{\epsilon}_{l,\mathrm{alt}}(\Theta, H) = \min_{u_\theta}\left\{\max_{d_M}\left\{d_M \quad \text{s.t.:}\forall x \quad \frac{d(x,\Omega_T)}{l(\Omega_T)} \le d_M \Rightarrow ||u(x) - u_\theta(x)|| \le \epsilon\right\}\right\} \tag{14}$$

Using either formulation of G_l is equivalent when the underlying problem is monodimensional. However, for higher dimensions, the two metrics can return different results as Definition 4 allows for arbitrary complex shapes of Ω_G, whose area represents the metric value, while Definition 14 substantially limits the shape of Ω_G to that of a ball around Ω_T. However, the latter formulation represents a computational advantage over the former since it does not require approximating the area of an arbitrarily complex shape.

6 Discussion on the effect of the algorithmic setup

In this section, we analyze the results obtained in this paper to provide further insights into the effect of the investigated hyperparameters on the generalization of a PINN. We first briefly comment on the results obtained in our preliminary analysis of Sect. 3 and then focus on the effect on the generalization level of each of the hyperparameters individually. We present the aforementioned results along with additional findings that were noticed during the training of all the PINN architectures used in this paper and in comparison with the research literature.

Our preliminary analysis showcases how a PINN solution represents an accurate approximation of the target PDE solution u, including its high-order derivatives. This property of PINNs is particularly interesting because it represents the most consistent difference between a PDE solution obtained with a classical solver. On the other hand, our analysis clearly shows that a PINN understandably fails to learn the analytical extension of the PDE solution in regions that are far from the collocation points used during training. However, the architecture maintains the accuracy reached inside the training domain when predictions are made in the proximity of its boundaries.

6.1 Network complexity

We conclude that increasing the width of the network skews the prediction to overestimate results far from the training area. However, the value of the generalization metric is not largely affected by the width of the network. Generally, when training an arbitrary neural network, it is common practice to overparametrize the latter to be sure that the network can capture the complexity of the target function. This is a recently studied behavior of neural networks in the overparametrized regime which is regarded in the scientific community as the double descent phenomena [27]. Curiously, the latter is not visible when considering the generalization level introduced in this project. Neural networks are also subject to implicit bias [39] which skews them toward learning simple solutions. However, the results obtained in our analysis hint that a smaller architecture is as suitable for training a PINN. Moreover, when adopting methods that employ a soft-domain decomposition, largely overestimating the target solution could cause numerical instabilities since the gating mechanism might not ensure that the prediction of a model in a subdomain vanishes far from its domain of evaluation.

Increasing the depth of the network skews the PINN to return a flat prediction outside the training domain, which suggests that no information is somehow gathered from the inner domain. Moreover, the generalization level is worsened by the depth, and training time is visibly increased without additional benefits. However, a flat prediction outside of the training domain could be preferable for cases of soft-domain decomposition. Therefore, the outcome of our analysis does not imply that a smaller network is always a more suitable solution, but rather that it is possible and advantageous—in terms of generalization level and training time—to compute relatively simple PDE solutions via a PINN with few trainable parameters, encouraging approaches of domain decomposition.

6.2 Number of collocation points

The generalization level is not largely affected by the number of collocation points. This implies that if the collocation points sampled are enough to correctly predict the target function, there is no benefit in adding more, except for cases of adaptive sampling [30]. Moreover, it is also worth noting the unexpected increase in the variability of training time when few collocation points are given. This is most likely related to the fact that using few points generates several local minima in the loss landscapes which are seen as acceptable to the algorithm. However, the lack of impact in training time is most likely because the computational overhead created by just 200 collocation points is negligible for a use case like the one we consider, for which the computation of the residuals is extremely fast. Nevertheless, this is not always the case, especially for multidimensional and highly nonlinear equations such as Navier–Stokes'.

6.3 Domain size

In terms of generalization level, we notice that training on a small domain appears to be beneficial for the PINN architecture. In particular, the numerical value of the generalization level obtained for small domain sizes is rather surprising. Indeed, the values obtained indicate that the longest segment in which all architectures generalize well can be just as big as the domain used for training—for the Poisson equation—. This outcome deeply encourages the combination of domain decomposition approaches with PINNs, which is already a common practice in several successful publications [21, 40] and a well-established practice in the field of PDE solution.

The results obtained are also coherent with additional available literature: indeed, PINNs are known to struggle when learning functions defined in big domains [41, 42] and their training can be eased by learning on collocation points which are given sequentially to the algorithm [43, 44] or selected during training through ensemble agreement [45].

Learning on smaller domains requires a surprisingly short training time and also presents low variability.

The advantages of training in small domains can also be explained through the study proposed in [46], where the authors noticed that learning on small domains reduces the number of fixed points in the loss landscapes, which carries substantial benefits as better iterative training speed and precision. Surprisingly, our study also emphasizes that reducing the number of fixed points is beneficial also for the generalizability of the PINN model, as it helps avoid local minima of the loss landscape which quickly depart from the ground truth solution when evaluated outside the training domain.

6.4 Activation function and training method

The activation function chosen for a PINN does not seem to largely influence the generalization of the model, provided that the network can converge to the correct solution. Indeed, alternative metrics for generalization, like the L^2 error on the test set, can be strongly impacted by the choice of the activation function. This is mainly motivated by the fact that evaluation far from the training domain will be guided by how the activation function behaves. Indeed, in this perspective, the SiLU seems to be the worse-performing activation since it is unbounded and does not converge to a trivial solution as in the case of ReLU. On the other hand, the generalization level with $\epsilon = 10^{-2}$ appears to be sensitive to the trivial solution, this indicates that it is important to consider different values of ϵ for ensuring a proper evaluation of the models.

The study on the training method, on the other hand, carries the more relevant insights in our study. In particular, when a neural network needs to generalize to new data points, a well-known good practice in general machine learning is to stop the training early so that the training set is not overfitted. However, the results obtained with a second-order training method indicate that training a PINN to a very high accuracy helps the model to generalize well to the test set. This also holds for test sets lying inside the training domain, as training losses on the order of 10^{-7} generally identify a PINN that accurately fits the correct PDE solution in the training domain. Moreover, we confirm that the training time and number of parameters required to train a PINN are both substantially reduced when adopting a second-order method.

7 Conclusion

The analysis provided in this paper shows a collection of relevant outcomes. First of all, fully converged PINN architectures have been shown to provide a limited but consistent potential to return accurate predictions when evaluated in the immediate proximities of the training domain. This region might seem negligible, but it represents a consistent size for several applications in which the PDE solution is computed on very small domains [8], domains which can be approximated as low-dimensional [16–19], or where one needs to exploit the solution of the PDE at its boundaries, the latter being the case of shape optimization and multi-body interaction problems.

From the perspective of the hyperparameters of the PINN model, despite their effects being relatively small, the changes in the neural network's output are often remarkable. In particular, we found that enlarging the number of hyperparameters of the neural networks employed does not seem to carry any meaningful advantage. This indicates that the concept of overparametrization of neural networks might not be the most efficient choice when applying PINNs. Moreover, our analysis highlighted a strong overestimation for wide architectures, a flatter prediction for deep ones, and a consistent extrapolation for networks trained in small areas. All these factors should be considered when combining domain decomposition or ensemble training [45] techniques with PINNs.

Finally, the most relevant outcome of this paper is how the concept of generalization in SciML differs from the classical one. Indeed, overfitting and performing highly accurate training proved to be beneficial concepts for PINNs in terms of their generalization capabilities. This outcome encourages future research on the PINN architecture skewed toward parameter-efficient approaches that recall divide-and-conquer techniques as the one described in [47], methods that consider sequential training as intrinsic to the architecture, as in [41] and, most importantly, toward second-order training methods, which are crucial for solving nonlinear PDEs [37]. Indeed, second-order methods are crucial in engineering applications, which are the future of SciML models as PINNs.

Acknowledgements The authors acknowledge the financial support of BMW AG, Digital Campus Munich (DCM), through the ProMotion program. This project is also partially supported by the Research Groups 2022-2024 (IT1504-22); the Elkartek Program from the Basque Government; and the PID2022-137442NB-I00 and PID2023-149195NB-I00 research projects from the Spanish Ministry of Science. Financial support from the BERC 2022-2025 program and by the Spanish State Research Agency through BCAM Severo Ochoa Excellence Accreditation CEX2021-001142-S/MICIN/AEI/10.13039/501100011033 and through the project PID2020-117080RB-C55 ("Microscopic foundations of soft matter experiments: computational nano-hydrodynamics" - acronym "Compu-Nano-Hydro") are also acknowledged.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability Code and Data sets generated during the current study are available from the corresponding author on request.

Declarations

Conflict of interest The authors declare that there is no conflict of interest regarding the publication of this paper.

References

1. Thiyagalingam J, Shankar M, Fox G, Hey T (2022) Scientific machine learning benchmarks. Nat Rev Phys 4(6):413–420
2. Yang L, Meng X, Karniadakis GE (2021) B-PINNs: Bayesian Physics-informed neural networks for forward and inverse PDE problems with noisy data. J Comput Phys 425:109913
3. Sun Y, Sengupta U, Juniper M (2022) Physics-informed deep learning for simultaneous surrogate modelling and PDE-constrained optimization. Bull Am Phys Soc 67:116042
4. Jeong H, Batuwatta-Gamage C, Bai J, Xie YM, Rathanayaka C, Zhou Y, Gu Y (2023) A complete Physics-Informed Neural Network-based framework for structural topology optimization. Comput Methods Appl Mech Eng 417:116401
5. Dissanayake MWMG, Phan-Thien N (1994) Neural network-based approximations for solving partial differential equations. Commun Numer Methods Eng 10(3):195–201
6. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000
7. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
8. Kissas G, Yang Y, Hwuang E, Witschey WR, Detre JA, Perdikaris P (2020) Machine learning in cardiovascular flows modeling: predicting arterial blood pressure from non-invasive 4D flow MRI data using Physics-informed neural networks. Comput Methods Appl Mech Eng 358:112623
9. Zapf B, Haubner J, Kuchta M, Ringstad G, Eide PK, Mardal K (2022) Investigating molecular transport in the human brain from MRI with Physics-informed neural networks. Sci Rep 12(1):15475
10. Kashinath K et al (2021) Physics-informed machine learning: case studies for weather and climate modelling. Philos Trans R Soc 379(2194):20200093
11. Xu Y, Kohtz S, Boakye J, Gardoni P, Wang P (2022) Physics-informed machine learning for reliability and systems safety applications: state of the art and challenges. Reliab Eng Syst Saf 230:108900
12. Goud JS, Srilatha P, Kumar RSV, Sowmya G, Gamaoun F, Nagaraja KV, Chohan JS, Khan U, Eldin SM (2023) Heat transfer analysis in a longitudinal porous trapezoidal fin by non-Fourier heat conduction model: an application of artificial neural network with Levenberg–Marquardt approach. Case Stud Therm Eng 49:103265
13. Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L (2021) Physics-informed machine learning. Nat Rev Phys 3(6):422–440
14. Tang Z, Dong S, Yang X, Zhang J (2023) Application of a parallel physics-informed neural network to solve the multi-body dynamic equations for full-scale train collisions. Appl Soft Comput 142:110328
15. Li J, Du X, Martins JR (2022) Machine learning in aerodynamic shape optimization. Progress Aerosp Sci 134:100849
16. Abdelsalam SI, Magesh A, Tamizharasi P, Zaher AZ (2023) Versatile response of a Sutterby nanofluid under activation energy: hyperthermia therapy. Int J Numer Methods Heat Fluid Flow 34:408
17. Abdelsalam SI, Alsharif AM, Abd Elmaboud Y, Abdellateef AI (2023) Assorted kerosene-based nanofluid across a dual-zone vertical annulus with electroosmosis. Heliyon 9(5):e15916
18. Weera W, Kumar RSV, Sowmya G, Khan U, Prasannakumara BC, Mahmoud EE, Yahia IS (2023) Convective-radiative thermal investigation of a porous dovetail fin using spectral collocation method. Ain Shams Eng J 14:101811
19. Abdulrahman A, Gamaoun F, Kumar RSV, Khan U, Gill HS, Nagaraja KV, Eldin SM, Galal AM (2023) Study of thermal variation in a longitudinal exponential porous fin wetted with TiO2–SiO2/hexanol hybrid nanofluid using hybrid residual power series method. Case Stud Thermal Eng 43:102777
20. Kharazmi E, Zhang Z, Karniadakis GE (2021) hp-VPINNs: variational Physics-informed neural networks with domain decomposition. Comput Methods Appl Mech Eng 347:1
21. Moseley B, Markham A, Nissen-Meyer T (2021) Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. arXiv preprint arXiv:2107.07871
22. Hu Z, Jagtap AD, Karniadakis GE, Kawaguchi K (2022) Augmented Physics-Informed Neural Networks (APINNs): a gating network-based soft domain decomposition methodology. arXiv preprint arXiv:2211.08939
23. Bishop R, Kraus MA (2022) Mixture-of-experts-ensemble meta-learning for physics-informed neural networks. In: Proceedings of 33. Forum Bauinformatik
24. Mishra S, Molinaro R (2022) Estimates on the generalization error of Physics-informed neural networks for approximating a class of inverse problems for PDEs. IMA J Numer Anal 42(2):981–1022
25. Advani MS, Saxe AM, Sompolinsky H (2020) High-dimensional dynamics of generalization error in neural networks. Neural Netw 132:428–446
26. Olson M, Wyner A, Berk R (2018) Modern neural networks generalize on small data sets. In: Advances in neural information processing systems, vol 31
27. Mei S, Montanari A (2022) The generalization error of random features regression: precise asymptotics and the double descent curve. Commun Pure Appl Math 75(4):667–766

28. Wang Y, Han X, Chang C, Zha D, Braga-Neto U, Hu X (2022) Auto-PINN: understanding and optimizing Physics-informed neural architecture. arXiv:2205.13748
29. Chaudhari M, Kulkarni I, Damodaran M (2021) Exploring Physics-informed neural networks for compressible flow prediction. In: Proceedings of 16th Asian congress of fluid mechanics
30. Wu C, Zhu M, Tan Q, Kartha Y, Lu L (2023) A comprehensive study of non-adaptive and residual-based adaptive sampling for Physics-informed neural networks. Comput Methods Appl Mech Eng 403:115671
31. Shama R, Shankar V (2022) Accelerated training of Physics-informed neural networks (PINNs) using meshless discretizations. In: Advances in neural information processing systems, vol 35, pp 1034–1046
32. Lu L, Meng X, Mao Z, Karniadakis GE (2021) DeepXDE: a deep learning library for solving differential equations. SIAM Rev 63(1):208
33. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. https://arxiv.org/abs/1412.6980
34. Liu DC, Nocedal J (1989) On the limited memory BFGS method for large scale optimization. Math Program 45(1):503–528
35. De Ryck T, Jagtap AD, Mishra S (2022) Error estimates for Physics-informed neural networks approximating the Navier–Stokes equations. arXiv preprint arXiv:2203.09346
36. McKight PE, Najab J (2010) Kruskal-Wallis test. Corsini Encycl Psychol 1:1–10
37. Bonfanti A, Bruno G, Cipriani C (2024) The challenges of the nonlinear regime for Physics informed neural networks. arXiv preprint arXiv:2402.03864
38. Reinhardt WP, Walker PL (2010) Jacobian elliptic functions. In: Olver FWJ, Lozier DM, Boisvert RF, Clark CW (eds) NIST handbook of mathematical functions, Chap 22. Cambridge University Press, Cambridge, UK
39. Valle-Perez G, Camargo CQ, Louis AA (2018) Deep learning generalizes because the parameter-function map is biased towards simple functions. stat 1050:23
40. Jagtap A, Karniadakis GE (2021) Extended Physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In: AAAI spring symposium: MLPS
41. Wang S, Sankaran S, Perdikaris P (2022) Respecting causality is all you need for training Physics-informed neural networks. https://arxiv.org/abs/2203.07404
42. Wang S, Yu X, Perdikaris P (2020) When and why PINNs fail to train: a neural tangent kernel perspective. J Comput Phys 449:110768
43. Mishra S, Rusch TK (2021) Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. SIAM J Numer Anal 59(3):1811–1834
44. Krishnapriyan AS, Gholami A, Zhe S, Kirby RM, Mahoney MW (2021) Characterizing possible failure modes in Physics-informed neural networks. In: Advances in neural information processing systems, vol 34, pp 26548–26560
45. Katsiaryna H, Alexander I (2023) Improved training of Physics-informed neural networks with model ensembles. arXiv preprint arXiv:2204.05108
46. Rohrhofer FM, Posch S, Gößnitzer C, Geiger BC (2022) On the role of fixed points of dynamical systems in training physics-informed neural networks. arXiv preprint arXiv:2203.13648
47. Wang H, Planas R, Chandramowlishwaran A, Bostanabad R (2022) Mosaic flows: a transferable deep learning framework for solving PDEs on unseen domains. Comput Methods Appl Mech Eng 389:114424