Computer Methods in Applied Mechanics and Engineering 455 (2026) 118910

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

A flow-aware training strategy for physics-informed neural networks

Pietro Cestola [a,*], Luciano Teresi [b], Antonio De Simone [c,d]

[a] Department of Mathematics and Physics, Roma Tre University, Largo San Leonardo Murialdo 1, Rome, 00146, Italy
[b] Department of Industrial Engineering Electronics and Mechanics, Roma Tre University, Via Vito Volterra 62, Rome, 00146, Italy
[c] Mathematics Area, SISSA-International School for Advanced Studies, Via Bonomea 265, Trieste, 34136, Italy
[d] The BioRobotics Institute, Scuola Superiore Sant'Anna, Viale R. Piaggio 34, Pontedera (PI), 56025, Italy

ARTICLE INFO

Keywords:
Physics-informed neural networks
Scientific machine learning
Numerical methods
Partial differential equations

ABSTRACT

Physics-Informed Neural Networks (PINNs) typically update parameters at all collocation points from the first epoch, even in regions still unreachable from boundary or initial conditions. In these zones, early updates may be uninformative or harmful, as residual gradients often correlate poorly with the true error. We introduce a flow-aware training strategy that delays supervision until the governing physics can propagate meaningful information. The computational mesh is decomposed into geodesic subdomains ranked by distance from an information boundary, where initial or boundary conditions are applied, and progressively activated according to an epoch schedule. This selective exposure concentrates optimization where updates are most effective, preventing wasted capacity and misleading gradients in causally disconnected regions. The method requires no architectural changes and uses the standard PINN loss; only the sampling mask evolves over time. Benchmarks on seven PDE problems show that flow-aware training matches or improves baseline accuracy while reducing computational cost.

1. Introduction

Physics-Informed Neural Networks (PINNs) are neural architectures designed to solve problems governed by Partial Differential Equations (PDEs) by incorporating physical laws, such as governing equations and boundary conditions, directly into the loss function, thus avoiding the discretization schemes used in traditional numerical methods. Since their introduction in [1], PINNs have gained significant attention for modeling physical systems [2]. Efforts to improve their performance have addressed loss term balancing [3], activation functions [4,5], network architectures [6], and constraint enforcement strategies [7]. These studies confirm the potential of PINNs for both forward and inverse problems, while noting challenges such as convergence issues, scalability, and hyperparameter sensitivity.

A commonly observed limitation is that standard training treats all collocation points equally, ignoring the progressive way in which boundary or initial data constrain the solution of the Cauchy problem. Previous research has noted that, even with uniform domain sampling, PINN errors often decrease first near constrained regions before spreading inward, as discussed in the next section. To exploit this property, we propose a flow-aware training strategy in which training begins in regions constrained by boundary or initial data and progressively expands across the domain. By aligning learning with the Cauchy problem natural information flow, the method reduces early-stage complexity and preserves accuracy. Our contributions are as follows:

* Corresponding author.
E-mail addresses: pietro.cestola@uniroma3.it (P. Cestola), luciano.teresi@uniroma3.it (L. Teresi), antonio.desimone@sissa.it (A. De Simone).

https://doi.org/10.1016/j.cma.2026.118910
Received 2 September 2025; Received in revised form 21 December 2025; Accepted 5 March 2026
Available online 17 March 2026

- We introduce a flow-aware training strategy for PINNs that progressively activates regions of the domain according to how PDE constraints become informative throughout the domain.
- We show that the method preserves the accuracy of full-domain training while reducing unnecessary computation.
- We provide an open-source implementation for direct use and further development, available at this GitHub repository.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 introduces the notation, Section 4 presents the proposed method, Section 5 reports the experimental results, and Section 6 discusses the findings.

2. Related work

2.1. Gradient pathologies and optimization challenges

Several works have reported and analyzed the learning difficulties of PINNs. Basir [8] attributes difficulties in solving PDEs, especially without prior knowledge, to non-convex loss landscapes and unstable training from high-order derivatives, which amplify noise and hinder convergence. To mitigate this, the authors propose avoiding direct computation of such derivatives, reducing gradient noise and improving optimization. Their method also emphasizes learning in complex subregions where physics residuals dominate, highlighting the importance of how physical constraints shape training dynamics.

The study in [9] identifies two key failure modes in PINNs: first, that soft PDE regularization can lead to ill-conditioning and hinder optimization; second, that solving the entire space-time domain at once is often inefficient. To mitigate these issues, they propose two solutions: curriculum regularization, which gradually enforces the PDE constraint during training; and a sequence-to-sequence approach, where the model advances step-by-step from initial conditions, predicting short time segments to preserve temporal causality. This reformulation improves convergence and reduces error by aligning training with the natural direction of time.

2.2. Causality and curriculum strategies

Curriculum-based sampling [10] has been proposed to improve training in high-dimensional PINNs by gradually expanding the region where collocation points are placed. Two strategies are explored: one that grows a cuboid along time or space axes, and another that expands cylinders around sparse data. This guides the model from simpler to harder regions, improving convergence. Unlike our method, which follows the flow of information dictated by the Cauchy problem, this approach modulates sampling complexity without explicitly modeling propagation.

[11] further support the causal view by linking PINN failures on chaotic and multi-scale systems to violations of physical causality. Standard losses treat all time steps equally, allowing early errors to persist. Neural Tangent Kernel analysis shows that PINNs prioritize later times, breaking temporal order. To address this, the authors propose a weighted loss that enforces causality by penalizing later time steps until earlier ones are well-learned. This improves performance on benchmarks where standard PINNs fail, highlighting causality as key to convergence.

2.3. Propagation failures and information flow

[12] propose the propagation hypothesis, stating that correct solutions must spread from boundary or initial points into the interior domain to avoid convergence to trivial solutions. When this propagation fails, PINNs may satisfy the PDE residual but violate boundary conditions, leading to propagation failure. These local errors can then spread, disrupting global convergence. To counter this, the authors introduce the Retain-Resample-Release (R3) algorithm, which adaptively concentrates collocation points in regions with high PDE residuals.

Building on the propagation hypothesis, Wu et al. [13] provide a formal architectural explanation for propagation failures in PINNs. The authors argue that such failures arise when supervision from boundary or initial data cannot effectively propagate to the interior domain, due to low gradient correlation between nearby collocation points. Unlike FEMs, which enforce local coupling via mesh connectivity, PINNs lack explicit mechanisms for neighbor interaction during optimization. The authors prove that low gradient correlation is both necessary and sufficient for propagation failure, and propose ProPINN, an architecture that improves local gradient sharing through a multi-region mixing mechanism. This enhances solution propagation across the domain and mitigates propagation failure. These results underscore the need for training dynamics and architectures that support stepwise information transmission aligned with constraint causality.

2.4. Residual-based approaches

A separate line of research enhances PINN training by adaptively changing the distribution of collocation points based on local residual information, see Wu et al. [14] and the references therein. A representative example is the residual-based adaptive refinement (RAR) strategy introduced by Lu et al. [15], in which new points are added in regions exhibiting large residuals. Another closely related approach is the importance-sampling method proposed by Nabian et al. [16], which assigns higher sampling probability to points with large loss or gradient magnitude, thereby increasing the amount of informative gradient signal per iteration. These methods differ conceptually from our flow-aware strategy, as they are driven by error-localization criteria rather than by the analysis of information

propagation across the domain, and rely on solution-dependent adaptive updates that do not provide explicit a priori control over the training cost.

2.5. Domain decomposition approaches

Recent work has explored domain decomposition to improve PINN training. The Progressive Domain Decomposition (PDD) method [17] partitions the domain based on residual loss behavior, assigning subnetworks to subregions and progressively training them. Well-converged areas are frozen to focus computation on harder regions, enabling localized refinement and better scalability. Unlike methods that follow the natural flow of information through expanding domains, PDD uses static partitions and independently trained subnetworks, introducing challenges like interface continuity and optimal decomposition.

XPINN [18] extends domain decomposition by dividing the space-time domain into irregular, non-overlapping subregions, each handled by a subnetwork adapted to local solution complexity. Interface conditions (e.g., residual and flux continuity) are added to the loss to ensure coherence. This approach allows for parallel training and scalability in complex or multi-scale problems.

A recent work [19] offers a unified view of causality in PINNs, distinguishing between hard causality, where information must strictly follow initial or boundary data, and soft causality, where such structure is encouraged but not enforced. Within this framework, common training failures are framed as results of poor information flow. To address this, the authors propose causal XPINNs, where subnetworks over time intervals are activated progressively, allowing information to propagate before training the next. Combined with transfer learning and adaptive sampling, the method improves accuracy and efficiency on problems where standard PINNs and XPINNs underperform.

3. Notation

Definition 1 (Feedforward Neural Network)**.** A feedforward neural network is a parameterized family of functions $\{u_\theta : \mathbb{R}^n \to \mathbb{R}^m : \theta \in \mathbb{R}^d\}$, for some $n, m, d \in \mathbb{N}$, defined by the following recurrence on $\ell = 1, \dots, L-1$ on a point $\mathbf{x}$:

$$\mathbf{z}^{(0)} = \mathbf{x}, \qquad \mathbf{z}^{(\ell)} = \sigma\big(W^{(\ell)}\mathbf{z}^{(\ell-1)} + \mathbf{b}^{(\ell)}\big), \qquad u_\theta(\mathbf{x}) = W^{(L)}\mathbf{z}^{(L-1)} + \mathbf{b}^{(L)}. \tag{1}$$

Here, $W^{(\ell)} \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}^{(\ell)} \in \mathbb{R}^{N_\ell}$ are the weights and biases of the ℓth layer; all such parameters are collectively denoted by θ. The activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is applied element-wise; unless otherwise specified, the hyperbolic tangent is used.

Definition 2 (Self-Supervised Training Loss)**.** Given a feedforward neural network u_θ and a function $\ell : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}_+$, differentiable with respect to the first argument (which measures how well the network satisfies intrinsic constraints defined on the input domain), the self-supervised loss is defined as:

$$\mathcal{L}(\theta; S) = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \ell(\theta, \mathbf{x}), \tag{2}$$

for each finite, nonempty subset $S \subset \mathbb{R}^n$ of points. Training consists in minimizing this loss over a fixed dataset D by iteratively updating the model parameters θ.

We now introduce a unified notation for physics-informed learning, including stationary and time-dependent problems. We denote by $\Omega \subset \mathbb{R}^n$ the spatial domain, assumed relatively compact and connected, and by $[0, T]$ with $T \in \mathbb{R}_+$ the temporal domain. The problem is formulated over a generalized computational domain $\mathcal{X}$, with the variable $\mathbf{x} \in \mathcal{X}$ collecting all independent coordinates: $\mathbf{x} = x \in \Omega$ in the stationary case, and $\mathbf{x} = (x, t) \in \Omega \times [0, T]$ in the time-dependent case.

Definition 3 (Physics-Informed Neural Network)**.** Let us fix a computational domain $\mathcal{X}$, a convex function $\phi : \mathbb{R} \to [0, +\infty)$ and a generalized Cauchy problem of the form:

$$\begin{cases} \mathcal{R}(u, \nabla u, \dots, \nabla^k u; \mathbf{x}) = 0 & \text{for } \mathbf{x} \in \mathcal{X} \\ \mathcal{B}_j(u, \nabla u, \dots \nabla^l u; \mathbf{x}) = 0 & \text{for } \mathbf{x} \in \Gamma_j, \quad \forall j = 1, 2, \dots, m \end{cases} \tag{3}$$

where $\Gamma_j \subset \partial\mathcal{X}$ for all j. Then the associated physics-informed neural network is a feedforward neural network u_θ together with the following physics-informed training loss:

$$\mathcal{L}(\theta; S) = \mathcal{L}_\mathcal{R}(\theta; S \cap \mathcal{X}) + \sum_j \lambda_j \mathcal{L}_j(\theta; S \cap \Gamma_j), \tag{4}$$

where the weights $\lambda_j \in \mathbb{R}$ are hyperparameters used to balance the contribution of different boundary conditions. The individual loss terms are defined respectively as:

$$\mathcal{L}_\mathcal{R}(\theta; S) = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \phi \circ \mathcal{R}(u_\theta, \nabla u_\theta, \dots, \nabla^k u_\theta; \mathbf{x}), \tag{5}$$

$$\mathcal{L}_j(\theta; S) = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \phi \circ \mathcal{B}_j(u_\theta, \nabla u_\theta, \dots, \nabla^l u_\theta, \mathbf{x}). \tag{6}$$

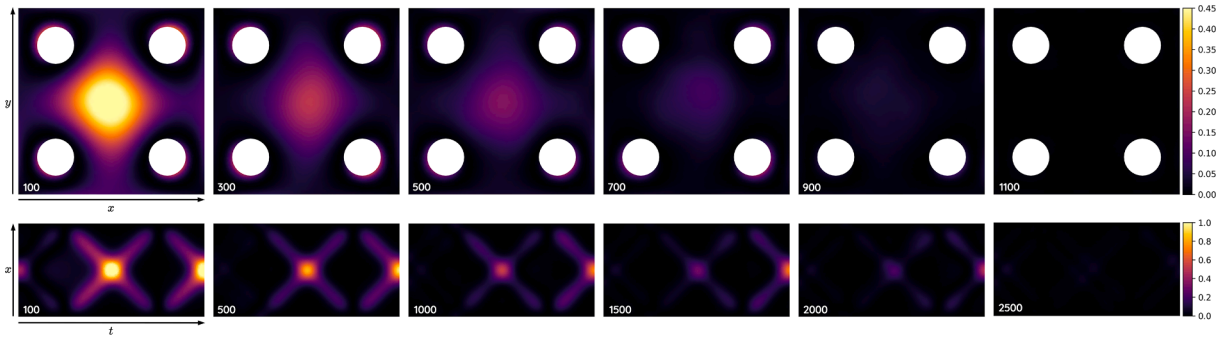

Fig. 1. Pointwise absolute difference between the PINN output and the ground truth at successive training epochs. Top: Laplace equation on a domain with holes. Bottom: one dimensional wave equation. The epoch is shown in white in the bottom-left corner of each image. Despite full-domain training, learning proceeds in a constraint-consistent manner.

Definition 4 (Standard Training Procedure for PINNs)**.** Given a physics-informed neural network and a finite, nonempty set $\mathcal{D} \subset \mathcal{X}$, the standard training procedure is defined as the iterative update of the parameters θ according to the rule:

$$\theta_{k+1} = \theta_k - \eta_k \cdot \mathcal{G}\big[\nabla_\theta \mathcal{L}\big](\theta_k; \mathcal{D}), \tag{7}$$

where η_k is the learning rate at iteration k, and $\mathcal{G}$ denotes the update transformation associated with a first-order optimizer.

Choices for $\mathcal{G}$ include Stochastic Gradient Descent, Adam, or AdamW. In practice, adaptive methods like Adam are commonly preferred due to their robustness to gradient scaling and their ability to handle heterogeneous curvature in the loss landscape [20].

4. Method

4.1. Observing information flow in PINNs

A well-documented empirical behavior in PINNs is the nonuniform convergence of the learned solution across the domain during training. Rather than reducing the error uniformly over the entire domain, PINNs typically exhibit progressive solution refinement that reflects the causal or propagative structure induced by the PDE's constraint system. In time-dependent problems, the solution typically emerges from the initial conditions and propagates forward as training progresses. Similarly, in steady-state problems, information from boundary conditions gradually constrains interior regions, and the solution develops accordingly. This behavior reflects how PINNs assimilate constraints, starting from strongly enforced regions and extending outward during optimization.

As an illustrative example, Fig. 1 shows how this behavior manifests in both elliptic and hyperbolic settings. Despite uniform sampling and full-domain training, the pointwise error between the PINN prediction and the ground truth reveals a clear pattern: the solution first develops near the boundary or initial conditions, and error reduction progressively spreads across the domain over successive epochs.

This training dynamic suggests that standard PINN training, based on a global loss function that aggregates residuals and constraints over the entire domain, does not effectively account for the directional flow of information intrinsic to the Cauchy problem. Several authors [8,9,11–13] argue that incorporating physical causality into the optimization procedure by employing methods such as curriculum learning, adaptive sampling, or causal reweighting can improve convergence speed, stability, and final accuracy.

Our work focuses on the observation that minimizing the PDE residual at a given point does not necessarily correspond to minimizing the true error at that point, especially during the early stages of training. Residual-based loss serves as a proxy for physical consistency rather than a direct measure of accuracy; as a result, its gradient with respect to the network parameters may be poorly aligned with the true error gradient. In other words, the optimizer's efforts to reduce the residual may fail to reduce the true error and in some cases may even increase it.

This suggests that training over the entire domain from the outset can be inefficient or even counterproductive, since the model may allocate capacity in directions that are not related to actual error reduction. In contrast, progressive or localized strategies can steer optimization toward regions where supervision is more meaningful, making better use of the model's capacity in early training stages.

To formalize this concept, we draw inspiration from the theory of neural tangent kernels introduced by [21]. Given a convex function $\phi : \mathbb{R} \to [0, +\infty)$, we define the pointwise error of the PINN at location $x \in \mathcal{X}$ with parameters θ as:

$$\mathcal{E}(\theta; x) := \phi(u_\theta(x) - u(x)), \tag{8}$$

where $u(x)$ is the exact solution of the problem. Now consider a generic first-order optimization scheme, and let $\dot{\theta}$ denote the parameter update produced by the optimizer. The instantaneous variation of the pointwise error at location x is then given by:

$$\dot{\mathcal{E}}(\theta; x) = \nabla_\theta \mathcal{E}(\theta; x) \cdot \dot{\theta}. \tag{9}$$

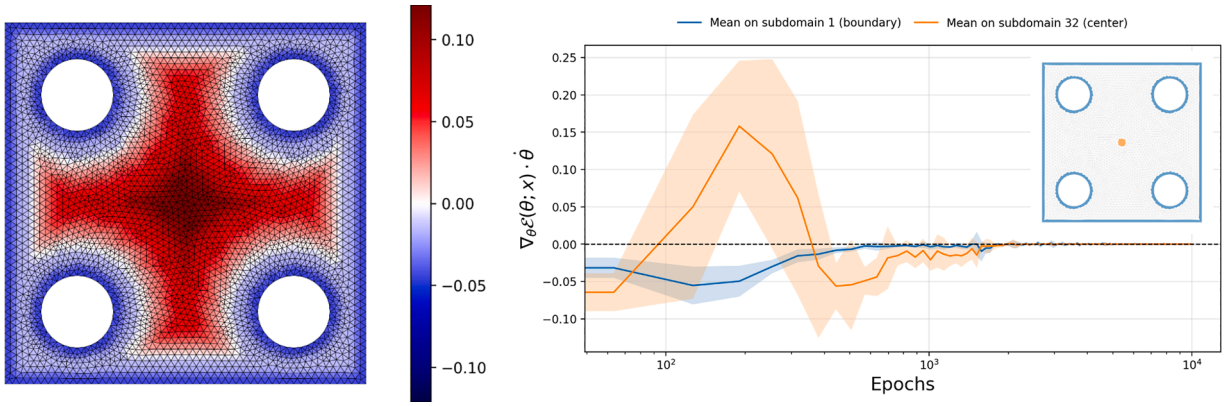

Fig. 2. Visualization of the inner product in (9). Left: spatial distribution of its mean value over geodesic subdomains at an early fixed training epoch. Right: temporal evolution of the mean ± standard deviation, across multiple runs, of the same subdomain-averaged inner product for the boundary subdomain (1) and a central interior subdomain (32).

This inner product quantifies the degree of alignment between the descent direction induced by the optimizer and the direction that would locally increase the pointwise error. Negative values indicate locally beneficial updates, while positive values correspond to harmful ones. Fig. 2 illustrates how this alignment evolves during training.

4.2. Flow-aware training

Definition 5 (Nested Domain Sequence)**.** Let $\mathcal{X}$ be a computational domain and Γ a distinguished portion of its boundary, called the information boundary, representing the region from which information is expected to propagate during training. A nested domain sequence is a collection of subdomains $\{\mathcal{X}_i\}_{i=1}^n$ and a corresponding epoch schedule $\{e_i\}_{i=1}^n$ such that:

$$\Gamma \subset \mathcal{X}_1 \subset \mathcal{X}_2 \subset \cdots \subset \mathcal{X}_n = \overline{\mathcal{X}}, \quad 0 = e_0 < e_1 < \cdots < e_n \in \mathbb{N}. \tag{10}$$

Definition 6 (Flow-Aware Training)**.** Given a PINN u_θ, a finite nonempty subset $D \subset \mathcal{X}$ and a nested domain sequence $(\{\mathcal{X}_i\}, \{e_i\}, \Gamma)$, the flow-aware training procedure is defined as the iterative update of the parameters θ according to the rule:

$$\theta_{k+1} = \theta_k - \eta_k \cdot \mathcal{G}\big[\nabla_\theta \mathcal{L}\big](\theta_k; D \cap \mathcal{X}_i), \quad \text{for } k \in \{e_{i-1}, \ldots, e_i - 1\}. \tag{11}$$

To implement flow-aware training on general geometries, we introduce a decomposition method based on geodesic distances on meshes.

Definition 7 (Geodesic Distance on a Triangular Mesh)**.** Let $\mathcal{M} = (V, E, F)$ be a triangular mesh. Here, V, E, and F denote the sets of vertices, edges, and triangular faces. The geodesic distance d between two vertices is defined as the length of the shortest path along mesh edges connecting them, computed as the sum of Euclidean distances between consecutive vertices along that path. The distance from a vertex $v \in V$ to a subset $S \subset V$ is defined as:

$$d(v, S) := \min_{w \in S} d(v, w). \tag{12}$$

Definition 8 (Geodesic Decomposition)**.** Let $\mathcal{M} = (V, E, F)$ be a triangular mesh discretizing a computational domain $\mathcal{X}$. Given $\Gamma \subset \partial\mathcal{X}$, define the scalar field $\phi : V \to [0, 1]$ as:

$$\phi(v) := d(v, V \cap \Gamma) / \max_{w \in V} d(w, V \cap \Gamma), \tag{13}$$

where d denotes the geodesic distance as in Definition 7. This scalar field induces the following partition of the mesh:

$$S_i := \{\sigma \in F \ : \ \min_{v \in \sigma} \phi(v) \in \big[(i-1)/n, i/n\big)\}, \quad i = 1, \ldots, n. \tag{14}$$

The corresponding collection of subdomains is defined as $\mathcal{X}_k := \bigcup_{i=1}^k S_i \cup \Gamma$.

We implemented three scheduling strategies to assign training epochs to each subdomain: a linear schedule, where each subdomain receives the same number of epochs; a proportional schedule, where epochs are assigned proportionally to the number of collocation points; and an exponential schedule, where epochs per subdomain increase exponentially with the index. In all experiments reported in this work, we adopt the exponential schedule, which consistently achieved the lowest error as shown in Table A.1, while the other strategies remain available in our public implementation Fig. 3.

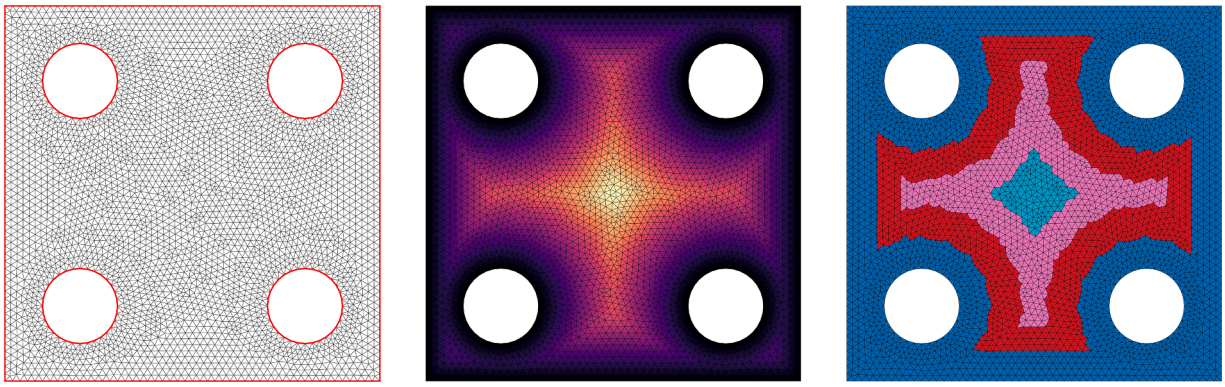

Fig. 3. Example of geodesic decomposition (Definition 8) on a mesh with holes. Left: input mesh with information boundary Γ in red. Center: normalized geodesic scalar field. Right: decomposition of the domain into disjoint subregions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Definition 9 (Exponential Scheduling)**.** Indicating with e_{tot} the total number of epochs, the number of epochs per subdomain increases exponentially with the index:

$$e_0 := 0, \quad e_i = e_{i-1} + \left\lfloor \frac{\exp(i/n) e_{\text{tot}}}{\sum_{j=1}^{n} \exp(j/n)} \right\rfloor. \tag{15}$$

To implement the geodesic decomposition of Definition 8, we adopt a vertex-based assignment strategy that partitions the training dataset according to normalized geodesic distance from the information boundary.

Let $\mathcal{M} = (V, E, F)$ be the discretized domain, $\mathcal{D} \subset \mathcal{X}$ the set of collocation points, and n the number of geodesic layers. Geodesic distances from the boundary are computed at mesh vertices using a multi-source Dijkstra algorithm and transferred to data points via nearest-vertex assignment. Algorithm 1 The resulting partition has computational complexity

$$\mathcal{O}((|V| + |E|) \log |V| + |\mathcal{D}| \log |V|). \tag{16}$$

Algorithm 1 GeodesicDecomposition.

Require: Mesh $\mathcal{M} = (V, E, F)$, boundary Γ, dataset $\mathcal{D}$, number of layers n
Ensure: Decomposed dataset $\{\mathcal{D}_i\}_{i=1}^{n}$
1: $\Gamma_{\mathcal{M}} \leftarrow \text{MeshBoundary}(\mathcal{M}, \Gamma)$
2: $\phi(V) \leftarrow \text{GeodesicDistance}(V, \Gamma_{\mathcal{M}})$
3: $\phi(\mathcal{D}) \leftarrow \phi(\text{NearestVertex}(\mathcal{D}, V))$
4: **for** $i = 1$ to n **do**
5: $\quad \mathcal{D}_i \leftarrow \{x \in \mathcal{D} \mid \phi(x) \in [\frac{i-1}{n}, \frac{i}{n})\}$
6: **end for**
7: **return** $\{\mathcal{D}_i\}_{i=1}^{n}$

5. Experiments and results

This section presents the experimental setup and results for evaluating our training scheme on selected two-dimensional PDEs from the PINNACLE benchmark [22], focusing on problems that are purely spatial or have one spatial and one temporal dimension.

5.1. Evaluation metrics and reference solutions

We evaluate model performance using the Mean Absolute Error (MAE), Mean Squared Error (MSE), Maximum Absolute Error (MXE), and the relative ℓ^1 and ℓ^2 errors (L1Re, L2Re), whose explicit definitions are provided in (17)–(19), and are computed by comparing the model predictions against high-fidelity numerical solutions generated with COMSOL Multiphysics (version 6.2.0.658) [23] under identical settings as those used during training. Additional technical details and reproducibility scripts are provided in the project's GitHub repository.

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |u_\theta(\mathbf{x}_i) - u(\mathbf{x}_i)|, \qquad \text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(u_\theta(\mathbf{x}_i) - u(\mathbf{x}_i)\right)^2 \tag{17}$$

Table 1
Comparison of training configurations on the 1D Burgers benchmark. Bold values highlight the best result per metric. Best performance is achieved with 32 subdomains, improving all error metrics and reducing cost by 10.13% compared to the baseline.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	5102	0.0120685	0.001329	0.398796	0.0226243	0.0528953
2	4926	0.0133732	0.00196318	0.557234	0.02507	0.0691733
4	4745	0.0127339	0.00178789	0.489471	0.0238715	0.0618617
8	4654	0.0118363	0.00110555	0.397349	0.022189	0.0503826
16	4607	0.0127292	0.00169588	0.511074	0.0238627	0.0630617
32	4585	**0.0113681**	**0.000811437**	**0.337013**	**0.0213111**	**0.0427174**

$$\mathrm{MXE} = \max_i \left|u_\theta(\mathbf{x}_i) - u(\mathbf{x}_i)\right|, \qquad \mathrm{L1Re} = \frac{\sum_{i=1}^N \left|u_\theta(\mathbf{x}_i) - u(\mathbf{x}_i)\right|}{\sum_{i=1}^N \left|u(\mathbf{x}_i)\right|} \tag{18}$$

$$\mathrm{L2Re} = \sqrt{\frac{\sum_{i=1}^N \left(u_\theta(\mathbf{x}_i) - u(\mathbf{x}_i)\right)^2}{\sum_{i=1}^N \left(u(\mathbf{x}_i)\right)^2}} \tag{19}$$

These error metrics are adopted for empirical comparison of training strategies, rather than for establishing theoretical error bounds, which are studied under different assumptions in the PINN literature [24].

5.2. Model architecture and training details

For each experiment, we employed a feedforward neural network with three hidden layers of 32 neurons each, trained using the Adam optimizer with momentum coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate decays linearly from 10^{-2} to 10^{-4} over 10,000 training epochs. These hyperparameters, shared by the baseline and the proposed method, were determined through preliminary tuning on the baseline configuration and were then kept fixed across all experiments, avoiding asymmetry in hyperparameter optimization and ensuring a fair comparison. The subdomain progression schedule is the only method-specific parameter introduced by the proposed approach and is fixed according to the discussion in Section 4.2. All experiments were performed on an NVIDIA Tesla P100-SXM2 (16 GB) GPU.

5.3. Governing equations and results

The results in this subsection are presented in tabular form, with each table corresponding to a specific PDE benchmark. For every problem, we report the compute cost measured in giga floating-point operations (GFLOPs) and the model's predictive performance across different subdomain configurations, highlighting how the proposed training scheme affects accuracy. To account for the inherent stochasticity in the training process, each experiment was repeated 10 times with different random seeds to mitigate variability resulting from random initialization and the optimizer's internal dynamics. The coefficient of variation (std/mean) is reported in Table A.3 as a compact measure of variability.

Burgers' Equation (1D).

$$\begin{cases} u_t + uu_x = \nu u_{xx}, & (x,t) \in [-1,1] \times [0,1], \\ u(-1,t) = u(1,t) = 0, & t \in [0,1], \\ u(x,0) = -\sin \pi x, & x \in [-1,1], \end{cases} \tag{20}$$

where $\nu = 0.01/\pi$ and the information boundary is chosen as $[-1,1] \times \{0\}$.

Wave Equation (1D).

$$\begin{cases} u_{tt} = u_{xx}, & (x,t) \in [-2,2] \times [0,4], \\ u(-2,t) = u(2,t) = 0, & t \in [0,4], \\ u(x,0) = \exp(-4x^2), u_t(x,0) = 0, & x \in [-2,2], \end{cases} \tag{21}$$

where we set the information boundary as $[-2,2] \times \{0\}$.

Laplace Equation (2D). The computational domain is $\Omega = [-4,4]^2 \setminus \bigcup_{k=1}^4 B_k$ where the circular holes are given by:

$$B_1 = \left\{\mathbf{x} : \|\mathbf{x} - (2.4,\ 2.4)\|_2 \le 0.8\right\}, \qquad B_2 = \left\{\mathbf{x} : \|\mathbf{x} - (-2.4,\ 2.4)\|_2 \le 0.8\right\},$$
$$B_3 = \left\{\mathbf{x} : \|\mathbf{x} - (2.4,\ -2.4)\|_2 \le 0.8\right\}, \qquad B_4 = \left\{\mathbf{x} : \|\mathbf{x} - (-2.4,\ -2.4)\|_2 \le 0.8\right\}.$$

Table 2
Comparison of training configurations on the 1D wave equation. Bold values highlight the best result per metric. Best performance is achieved with 32 subdomains, improving all metrics except MXE and reducing cost by 37.22% compared to the baseline.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	2117	0.068729	0.007742	**0.283225**	0.277750	0.265107
2	1850	0.067279	0.007463	0.283374	0.271891	0.260195
4	1575	0.068657	0.007858	0.310665	0.277458	0.266335
8	1435	0.067555	0.007465	0.284223	0.273006	0.260027
16	1364	0.067617	0.007422	0.285076	0.273258	0.259444
32	1329	**0.066275**	**0.007146**	0.284733	**0.267833**	**0.254608**

Table 3
Comparison of training configurations on the 2D Laplace equation. Bold values highlight the best result per metric. Best performance is achieved with 16 subdomains, improving all metrics except MXE while reducing cost by 9.17% compared to the baseline.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	2149	0.004309	0.000043	0.039003	0.009685	0.012386
2	2093	0.004134	0.000039	0.038676	0.009293	0.011881
4	2026	0.004247	0.000041	0.040912	0.009546	0.012207
8	1979	0.004177	0.000040	0.039182	0.009389	0.012029
16	1952	**0.003948**	**0.000036**	0.037774	**0.008874**	**0.011450**
32	1936	0.004080	0.000037	**0.036714**	0.009170	0.011592

Table 4
Comparison of training configurations on the lid-driven cavity flow. Bold values highlight the best result per metric. The best performance is achieved with 32 subdomains, improving all error metrics while reducing cost by 17.83% compared to the baseline.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	5653	0.016581	0.000634	0.239550	0.223828	0.185130
2	5367	0.015640	0.000570	0.236036	0.211116	0.174531
4	5029	0.014385	0.000493	0.226760	0.194185	0.160624
8	4818	0.013753	0.000399	0.222969	0.185648	0.153661
16	4701	0.013311	0.000371	0.221870	0.179681	0.147728
32	4645	**0.012911**	**0.000353**	**0.216928**	**0.174280**	**0.143013**

Let $\Gamma = \partial B_1 \cup \partial B_2 \cup \partial B_3 \cup \partial B_4$, then we have:

$$\begin{cases} \Delta u = 0, & (x, y) \in \Omega, \\ u(x, y) = 1, & (x, y) \in \partial\Omega \setminus \Gamma, \\ u(x, y) = 0, & (x, y) \in \Gamma. \end{cases} \tag{22}$$

Here the information boundary is taken as $\partial\Omega$.

Lid-Driven Cavity Flow (2D). The computational domain is $\Omega = [0, 1]^2$ with the top boundary denoted by Γ_1, and the left, right, and bottom boundaries denoted collectively by Γ_2. The governing equations are:

$$\begin{cases} u \cdot \nabla u + \nabla p - \dfrac{1}{\mathrm{Re}} \Delta u = 0, & (x, y) \in \Omega, \\ \nabla \cdot u = 0, & (x, y) \in \Omega, \end{cases} \tag{23}$$

where $u = (u_1, u_2)$ is the velocity field, p is the pressure, and $\mathrm{Re} = 100$ is the Reynolds number. The boundary conditions are specified as:

$$\begin{cases} u(x, y) = (4x(1 - x), 0), & (x, y) \in \Gamma_1, \\ u(x, y) = (0, 0), & (x, y) \in \Gamma_2, \\ p(0, 0) = 0. \end{cases} \tag{24}$$

The information boundary is chosen as $\partial\Omega$.

Table 5
Comparison of training configurations on the backward-facing step flow. Bold values highlight the best result per metric. Best performance is achieved with 32 subdomains, improving all error metrics while reducing cost by 15.84% compared to the baseline.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	5705	0.035940	0.002675	0.193743	0.201651	0.152960
2	5504	0.035230	0.002531	0.188542	0.197667	0.148889
4	5184	0.034052	0.002366	0.182776	0.191057	0.144146
8	4978	0.033680	0.002316	0.182088	0.188974	0.142723
16	4860	0.034029	0.002377	0.184404	0.190932	0.144436
32	4801	**0.032793**	**0.002193**	**0.176649**	**0.183997**	**0.138736**

Table 6
Comparison of training configurations on the Eikonal equation benchmark. Bold values highlight the best result per metric. Best performance is achieved with 32 subdomains, improving all error metrics while reducing cost by 15.93% compared to the baseline.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	2115	0.009652	0.000154	0.038768	0.030217	0.031948
2	2018	0.009402	0.000146	0.037802	0.029434	0.030989
4	1907	0.008367	0.000113	0.032909	0.026193	0.027084
8	1837	0.008758	0.000133	0.036206	0.027419	0.029293
16	1798	0.008506	0.000115	0.030892	0.026629	0.027483
32	1778	**0.007501**	**0.000089**	**0.028159**	**0.023482**	**0.024330**

Backward-Facing Step Flow (2D). We fix $\Omega = [0,4]\times[0,2]\setminus[0,2]\times[1,2]$ with the left boundary denoted by Γ_{in}, and the right boundary denoted by Γ_{out}. The information boundary is chosen as $\partial\Omega$. The governing equations are:

$$\begin{cases} u\cdot\nabla u + \nabla p - \dfrac{1}{\text{Re}}\Delta u = 0, & (x,y)\in\Omega, \\ \nabla\cdot u = 0, & (x,y)\in\Omega, \end{cases} \tag{25}$$

where $u=(u_1,u_2)$ is the velocity field, p is the pressure, and the Reynolds number is $\text{Re}=100$. The boundary conditions are:

$$\begin{cases} u(x,y) = (4y(1-y),0), & (x,y)\in\Gamma_{\text{in}}, \\ p(x,y) = 0, & (x,y)\in\Gamma_{\text{out}}, \\ u(x,y) = (0,0), & (x,y)\in\partial\Omega\setminus(\Gamma_{\text{in}}\cup\Gamma_{\text{out}}). \end{cases} \tag{26}$$

5.4. Additional benchmark problems

In addition to the previous benchmarks, we evaluate the Eikonal equation, which models minimal distance propagation from the boundary and naturally aligns with the proposed method, making it a suitable test for causally ordered training. The experimental setup remains unchanged.

Laplacian-Regularized Eikonal (2D). We set $\Omega=\{\mathbf{x}:\|\mathbf{x}\|_2\le 1\}$,

$$\begin{cases} -\varepsilon\Delta u + f(x,y)\|\nabla u\| = 1, & (x,y)\in\Omega, \\ u(x,y) = 0, & (x,y)\in\partial\Omega, \end{cases} \tag{27}$$

where $\varepsilon=0.01$, we set the information boundary as $\partial\Omega$, and

$$f(x,y) = 1 + 0.5\sin(2\pi x)\sin(2\pi y). \tag{28}$$

To stabilize convergence, we adopt a linear ε schedule during the first 500 epochs, decreasing ε from 1 to 0.01.

Eikonal (2D). Setting $\varepsilon=0$ (exact front arrival time) yields the classical Eikonal equation:

$$\begin{cases} f(x,y)\|\nabla u\| = 1, & (x,y)\in\Omega, \\ u(x,y) = 0, & (x,y)\in\partial\Omega. \end{cases} \tag{29}$$

To ensure convergence to the positive solution, we adopt a linear ε schedule, decreasing it from 1 to 0 over the first 500 epochs. Training then continues up to a total of 20,000 epochs, during which the learning rate is linearly reduced from 10^{-2} to 10^{-6}.

Table 7 shows that the 32-layer configuration performs better than the baseline, reducing computational cost by 22.61% while also improving all error metrics.

Fig. 4 illustrates the qualitative behavior of the two methods. With the standard approach, the propagating branches meet at the disk's center, producing an incorrect merging. In contrast, the proposed method avoids propagation failure: the two branches remain separated throughout the evolution, in agreement with the reference solution.

Table 7
Metrics for the Eikonal equation, comparing the baseline to the 32 subdomains configuration. The latter reduces cost by 22.61% and improves all error measures.

Subdomains	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
1	2715	0.019151	0.000563	0.079807	0.060900	0.062538
32	2101	**0.012031**	**0.000241**	**0.048463**	**0.038258**	**0.040918**

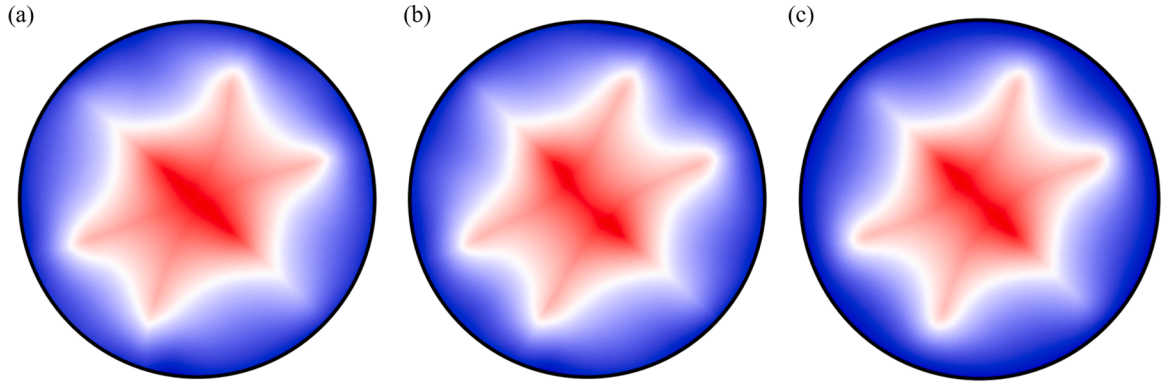

Fig. 4. Solution of the Eikonal Eq. (29), trained with extended scheduling and ϵ-decay. Panels: (a) baseline prediction, (b) 32-layer prediction, (c) ground-truth solution.

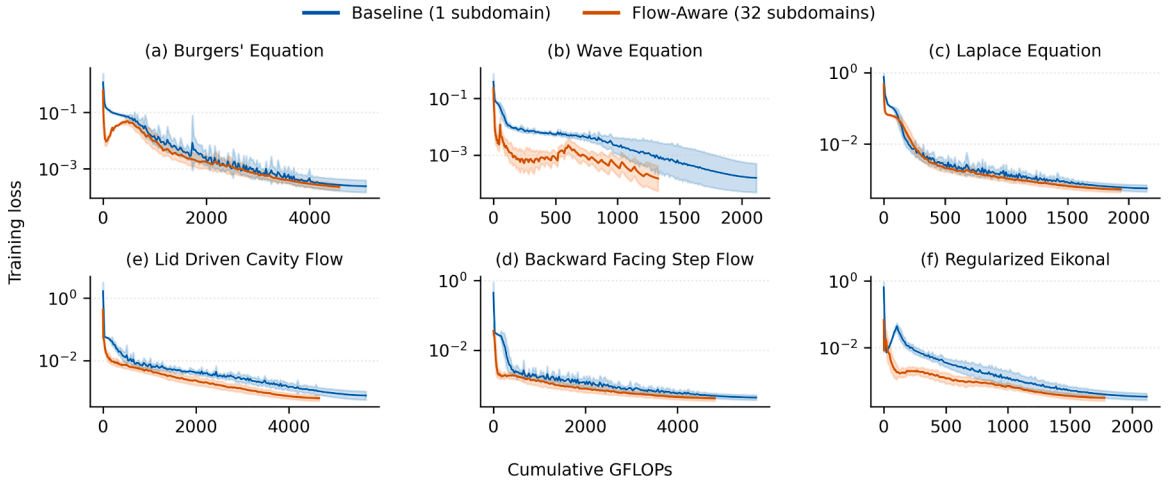

Fig. 5. Cumulative GFLOPs versus training loss for the benchmarks in Section 5.3. Curves report the mean ± one standard deviation (log-scale).

5.5. Additional analyses

Preprocessing Overhead. To assess the overhead introduced by preprocessing, we measure its wall-clock time and compare it with the corresponding training time. As shown in Table A.2, even for the most demanding configuration (32 subdomains) the preprocessing cost remains negligible, accounting for less than 0.2% of the total training time across all benchmarks.

Convergence Analysis. Fig. 5 reports convergence trajectories for the first six benchmarks, comparing the baseline configuration to the 32 subdomains flow-aware setup. Curves are shown as the geometric mean, with shaded bands indicating one standard deviation in log-scale. Training trajectories are reported as a function of cumulative GFLOPs to enable comparison at matched computational effort. These curves focus on optimization dynamics and stability, as loss values are comparable only in the final phase when the full collocation set is used. Overall, the flow-aware approach does not degrade stability and, for flow-dominated problems such as lid-driven cavity and backward-facing step, yields smoother trajectories, particularly during the early and intermediate training phases. For Burgers' and Wave equations, the convergence behavior appears less regular and does not exhibit an improvement. For the regularized Eikonal case, the initial change in slope is due to the linear scheduling of the regularization parameter ϵ during the first training phase; once it reaches its final value, the trajectory stabilizes.

6. Discussion and conclusions

In this work, we proposed a flow-aware training strategy for PINNs, grounded in the dependency structure of the underlying Cauchy problem and its implications for the optimization process. Traditional PINN training treats all collocation points uniformly, disregarding the causal structure imposed by the boundary and initial conditions. Empirical evidence shows that PINNs naturally reduce errors first near boundaries or initial conditions, and then gradually inward. Our flow-aware approach leverages this insight by dynamically controlling the regions of the domain included in the training loss. Initially, training is concentrated in strongly constrained regions close to boundaries or initial conditions, and then gradually expands to the full computational domain as the neural network progressively learns the solution. The experiments performed across PDE benchmarks empirically validate our approach. Our method reduced training cost, while maintaining and in some cases improving accuracy compared to standard PINN training, particularly in problems where the solution structure aligns strongly with the information flow induced by the governing conditions.

Although our experiments are limited to 1D and 2D, the method is dimension-independent, as it only requires an information boundary and a geodesic ordering of points, which extend directly to 3D. Since the underlying rationale does not depend on dimension, similar qualitative behaviour is expected in 3D. The only dimension-dependent effect is geometric: with uniform subdomain spacing, interior regions occupy a larger fraction of the domain in 3D, which proportionally reduces the achievable computational savings; this can be mitigated by using non-uniform spacings.

A limitation of the proposed approach is that the optimal number of subdomains is problem-dependent and accuracy does not always vary monotonically with their number. This sensitivity could be mitigated by automating subdomain activation, for instance using residual-based criteria; however, such mechanisms would remove the explicit a priori control over computational cost provided by the current fixed subdomain scheduling. Developing strategies that balance these aspects is a meaningful direction for future work.

Finally, the proposed training strategy may also be relevant beyond PINNs. Recent foundation models for PDEs, such as the Fourier Neural Operator [25] and PDEformer-1 [26], which adopt a pre-training plus fine-tuning paradigm, could benefit from incorporating information-flow-aware optimization to improve fine-tuning efficiency and convergence.

CRediT authorship contribution statement

Pietro Cestola: Writing – original draft, Visualization, Validation, Software, Methodology, Conceptualization; **Luciano Teresi:** Writing – review & editing, Supervision, Conceptualization; **Antonio De Simone:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization.

Data availability

All code and configuration files required to reproduce the experiments are available at https://github.com/pcestola/Flow-Aware-PINN-Training. Additional data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We gratefully acknowledge the funding support from the Italian Ministry of Research through the project PNRR-M4C2-I1.1-PRIN 2022-PE1-Innovative mathematical models for soft matter and hierarchical materials-F53D2300283 0006, U.E.-NextGenerationEU, Italy. We Acknowledge the support by INdAM-GNFM (Italian Group of Mathematical Physics).

Appendix

Table A.1
For each benchmark, we report the percentage error reduction obtained by the best exponential schedule with respect to the best linear or proportional schedule. Each value is computed as $100 \cdot (1 - \mathrm{err}_{\mathrm{exp}}/\mathrm{err}_{\mathrm{ref}})$, where $\mathrm{err}_{\mathrm{ref}}$ denotes the error of the corresponding linear or proportional schedule. With this definition, positive values indicate that the exponential schedule yields a lower error, while negative values indicate worse performance.

Benchmark	MAE	MSE	MXE	L1Re	L2Re
Table 1 (linear)	1.05%	17.54%	5.23%	1.05%	5.24%
Table 1 (prop)	1.05%	17.54%	2.49%	1.05%	5.24%
Table 2 (linear)	4.63%	13.99%	13.72%	4.63%	6.34%
Table 2 (prop)	3.39%	11.15%	11.32%	3.39%	4.95%
Table 3 (linear)	0.31%	0.55%	1.56%	0.32%	0.66%
Table 3 (prop)	3.63%	7.46%	3.45%	3.64%	3.10%
Table 4 (linear)	24.54%	46.90%	11.50%	24.54%	25.24%
Table 4 (prop)	35.58%	64.23%	17.73%	35.58%	36.83%
Table 5 (linear)	5.08%	10.21%	4.90%	5.08%	5.28%
Table 5 (prop)	6.23%	12.33%	5.58%	6.23%	6.35%
Table 6 (linear)	8.43%	18.78%	14.48%	8.43%	9.02%
Table 6 (prop)	11.76%	24.48%	15.73%	11.76%	12.38%

Table A.2
Average preprocessing and training times for the configuration with 32 layers, computed over the same runs used for the experiments in Section 5.3.

Benchmark	Training time (s)	Preprocessing time (s)	Ratio (%)
Table 1	104.760	0.189268	0.1807
Table 2	86.417	0.134499	0.1556
Table 3	123.086	0.099334	0.0807
Table 4	225.551	0.091987	0.0408
Table 5	229.579	0.120638	0.0525
Table 6	128.057	0.098686	0.0771
Table 7	250.508	0.113420	0.0453

Table A.3
Coefficient of variation (std/mean) over the same runs used for the experiments in Section 5. Lower is better.

Benchmark	Layers	GFLOPs	MAE	MSE	MXE	L1Re	L2Re
Table 1	1	0.002445	0.192778	1.041661	0.583411	0.192778	0.515387
	2	0.002400	0.133956	0.594542	0.326869	0.133956	0.305266
	4	0.002357	0.181441	0.821502	0.536298	0.181441	0.494802
	8	0.002325	0.130504	0.700855	0.471749	0.130504	0.400568
	16	0.002309	0.136753	0.655865	0.372683	0.136753	0.369130
	32	0.002315	0.113979	0.914308	0.470033	0.113979	0.429920
Table 2	1	0.002425	0.011110	0.040229	0.015215	0.011110	0.020198
	2	0.002514	0.026917	0.064360	0.010489	0.026917	0.032396
	4	0.002830	0.075518	0.164365	0.189611	0.075518	0.077627
	8	0.002947	0.048883	0.100915	0.028304	0.048883	0.051150
	16	0.002901	0.034984	0.073783	0.030732	0.034984	0.037167
	32	0.003005	0.027217	0.064356	0.045680	0.027217	0.032081
Table 3	1	0.004147	0.170868	0.316272	0.125328	0.170868	0.150197
	2	0.004060	0.135501	0.256347	0.108962	0.135501	0.126794
	4	0.003896	0.140586	0.245602	0.154770	0.140586	0.120608
	8	0.003946	0.152956	0.308308	0.184522	0.152956	0.151579
	16	0.003928	0.151589	0.277723	0.143409	0.151589	0.137006
	32	0.003932	0.135912	0.245956	0.161616	0.135912	0.119738
Table 4	1	0.003395	0.332837	0.778401	0.164978	0.332837	0.347177
	2	0.003089	0.357754	0.650931	0.169282	0.357754	0.365037
	4	0.003319	0.383197	0.802719	0.166589	0.383197	0.397211
	8	0.003320	0.153955	0.301366	0.081169	0.153955	0.158844
	16	0.003424	0.169757	0.352808	0.102304	0.169757	0.177701
	32	0.003447	0.208188	0.431074	0.102199	0.208188	0.216140
Table 5	1	0.004164	0.094870	0.190069	0.098945	0.094870	0.097647
	2	0.004306	0.087472	0.184731	0.096984	0.087472	0.089383
	4	0.004129	0.075787	0.147956	0.064563	0.075787	0.073592
	8	0.004237	0.063723	0.126222	0.052034	0.063723	0.061741
	16	0.004329	0.073416	0.157403	0.085161	0.073416	0.077994
	32	0.004316	0.075974	0.156203	0.086918	0.075974	0.078552
Table 6	1	0.004281	0.163926	0.301610	0.117432	0.163926	0.154733
	2	0.004163	0.161025	0.339719	0.120882	0.161025	0.164864
	4	0.004108	0.196114	0.426521	0.215643	0.196114	0.217883
	8	0.004337	0.192420	0.448776	0.239865	0.192420	0.215992
	16	0.004307	0.167810	0.360635	0.194813	0.167810	0.175294
	32	0.004258	0.141362	0.230875	0.106839	0.141362	0.120884

References

[1] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

[2] S. Cuomo, V.S.D. Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics-informed neural networks: where we are and what's next, J. Sci. Comput. 92 (3) (2022). https://doi.org/10.1007/s10915-022-01939-z

[3] R. Bischof, M.A. Kraus, Multi-objective loss balancing for physics-informed deep learning, Comput. Methods Appl. Mech. Eng. 439 (2025) 117914 https://doi.org/10.1016/j.cma.2025.117914

[4] A. Farea, M.S. Celebi, Learnable activation functions in physics-informed neural networks for solving partial differential equations, 2025. arXiv:2411.15111.

[5] R. Gnanasambandam, B. Shen, J. Chung, X. Yue, Z.J. Kong, Self-scalable tanh (stan): faster convergence and better generalization in physics-informed neural networks, 2022. arXiv:2204.12589.

[6] H. Saratchandran, S. Chng, S. Lucey, Architectural Strategies for the optimization of Physics-Informed Neural Networks, 2024. arXiv:2402.02711.

[7] Z.K. Lawal, H. Yassin, D.T.C. Lai, A.C. Idris, Physics-informed neural network (pinn) evolution and beyond: a systematic literature review and bibliometric analysis, Big Data Cognit. Comput. 6 (4) (2022). https://doi.org/10.3390/bdcc6040140

[8] S.B.S. Basir, Investigating and mitigating failure modes in physics-informed neural networks (pinns), Commun. Comput. Phys. 33 (5) (2023) 1240–1269. https://doi.org/10.4208/cicp.oa-2022-0239

[9] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, M.W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, in: Advances in Neural Information Processing Systems (NeurIPS), 34, (2021) 26548–26560.

[10] M. Münzer, C. Bard, A curriculum-training-based strategy for distributing collocation points during physics-informed neural network training, 2022. arXiv:2211.11396.

[11] S. Wang, S. Sankaran, P. Perdikaris, Respecting causality for training physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 421 (2024) 116813 https://doi.org/10.1016/j.cma.2024.116813

[12] A. Daw, J. Bu, S. Wang, P. Perdikaris, A. Karpatne, Mitigating propagation failures in physics-informed neural networks using retain-resample-release (R3) sampling, in: Proceedings of the 40th International Conference on Machine Learning (ICML), 2023, p. 2023.

[13] H. Wu, Y. Ma, H. Zhou, H. Weng, J. Wang, M. Long, Propinn: demystifying propagation failures in physics-informed neural networks, 2025. arXiv:2502.00803.

[14] C. Wu, M. Zhu, Q. Tan, Y. Kartha, L. Lu, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 403 (2023) 115671. https://doi.org/10.1016/j.cma.2022.115671

[15] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, Deepxde: a deep learning library for solving differential equations, SIAM Rev. 63 (1) (2021) 208–228. https://doi.org/10.1137/19M1274067

[16] M.A. Nabian, R.J. Gladstone, H. Meidani, Efficient training of physics-informed neural networks via importance sampling, Comput. Aided Civ. Infrastruct. Eng. 36 (8) (2021) 962–977. https://doi.org/10.1111/mice.12685
[17] D. Luo, S.-H. Jo, T. Kim, Progressive domain decomposition for efficient training of physics-informed neural network, Mathematics 13 (9) (2025). https://doi.org/10.3390/math13091515
[18] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (xpinns): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041. https://doi.org/10.4208/cicp.OA-2020-0164
[19] M. Penwarden, A.D. Jagtap, S. Zhe, G.E. Karniadakis, R.M. Kirby, A unified scalable framework for causal sweeping strategies for physics-informed neural networks (pinns) and their temporal decompositions, J. Comput. Phys. 493 (2023) 112464 https://doi.org/10.1016/j.jcp.2023.112464
[20] A. Bihlo, Improving physics-informed neural networks with meta-learned optimization, J. Mach. Learn. Res. 25 (1) (2024) 1–26. https://doi.org/10.5555/3722577.3722591
[21] A. Jacot, F. Gabriel, C. Hongler, Neural tangent kernel: convergence and generalization in neural networks, in: Advances in Neural Information Processing Systems 31, *Red Hook, NY, USA*, Curran Associates, Inc, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.) 2018. 8571–8580
[22] Z. Hao, J. Yao, C. Su, H. Su, Z. Wang, F. Lu, Z. Xia, Y. Zhang, S. Liu, L. Lu, et al., PINNACLE: a comprehensive benchmark of physics-informed neural networks for solving pdes, 2023, arXiv:2306.08827.
[23] C. Ab, S. Stockholm, COMSOL Multiphysics®v6, 2, 2024. https://www.comsol.com.
[24] S. Goraya, N. Sobh, A. Masud, Error estimates and physics informed augmentation of neural networks for thermally coupled incompressible navier Stokes equations, 2022. arxiv:2209.02977
[25] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, 2021. arXiv:2010.08895.
[26] Z. Ye, X. Huang, L. Chen, Z. Liu, B. Wu, H. Liu, Z. Wang, B. Dong, PDEformer-1: a foundation model for one-dimensional partial differential equations, 2025. arXiv:2407.06664.