Computer Methods in Applied Mechanics and Engineering 421 (2024) 116805

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

Residual-based attention in physics-informed neural networks

Sokratis J. Anagnostopoulos [a,*,1], Juan Diego Toscano [b,1], Nikolaos Stergiopulos [a], George Em Karniadakis [b,c]

[a] Laboratory of Hemodynamics and Cardiovascular Technology, EPFL, Lausanne, 1015, VD, Switzerland
[b] Division of Applied Mathematics, Brown University, Providence, 02912, RI, USA
[c] School of Engineering, Brown University, Providence, 02912, RI, USA

ARTICLE INFO

Keywords:
Residual-based attention
PINNs accuracy
Adaptive weights
Fast convergence

ABSTRACT

Driven by the need for more efficient and seamless integration of physical models and data, physics-informed neural networks (PINNs) have seen a surge of interest in recent years. However, ensuring the reliability of their convergence and accuracy remains a challenge. In this work, we propose an efficient, gradient-less weighting scheme for PINNs that accelerates the convergence of dynamic or static systems. This simple yet effective attention mechanism is a bounded function of the evolving cumulative residuals and aims to make the optimizer aware of problematic regions at no extra computational cost or adversarial learning. We illustrate that this general method consistently achieves one order of magnitude faster convergence than vanilla PINNs and a minimum relative L^2 error of $\mathcal{O}(10^{-5})$, on typical benchmarks of the literature. The method is further tested on the inverse solution of the Navier–Stokes within the brain perivascular spaces, where it considerably improves the prediction accuracy. Furthermore, an ablation study is performed for each case to identify the contribution of the components that enhance the vanilla PINN formulation. Evident from the convergence trajectories is the ability of the optimizer to effectively escape from poor local minima or saddle points while focusing on the challenging domain regions, which consistently have a high residual score. We believe that alongside exact boundary conditions and other model reparameterizations, this type of attention mask could be an essential element for fast training of both PINNs and neural operators.

1. Introduction

Physics-informed neural networks (PINNs) have emerged as an alternative to traditional numerical methods for solving partial differential equations (PDEs) in forward and inverse problems [1]. In the PINN approach, a neural network is trained to approximate the solution of a PDE by minimizing a composite loss function that includes terms related to the physical laws and data from initial boundary conditions, simulations, or experiments [2]. Keeping a balance between the various and diverse loss terms is crucial to avoid convergence and accuracy issues, especially when dealing with highly nonlinear, multi-scale, or chaotic behavior problems [3–5]. To address this imbalance, researchers have developed several approaches that can be roughly classified into three categories: neural network modifications, expression transformations, and adaptive weighting strategies.

A neural network modification addresses the multi-layer perceptron optimization capabilities via domain decompositions [6], model reparametrizations [7,8], input dimension expansions [9,10], sequential learning [11,12], or adaptive activation functions [13]. On the other hand, expression transformations attempt to simplify the problem by converting the physical laws into

* Corresponding author.
E-mail address: sokratis.anagnostopoulos@epfl.ch (S.J. Anagnostopoulos).
[1] The first two authors contributed equally to this work.

https://doi.org/10.1016/j.cma.2024.116805
Received 27 September 2023; Received in revised form 22 January 2024; Accepted 23 January 2024
Available online 31 January 2024
0045-7825/

their equivalent forms or re-formulating soft constraints into strict ones. For instance, by using alternate or auxiliary physical laws, it is possible to reduce the number of equations or decrease the PDE order [14,15], which eases the optimization process. Similarly, by exactly imposing the boundary conditions, the number of losses can be reduced, and the optimizer can focus on a single condition [16–18].

Adaptive weighting strategies address the loss imbalances by iteratively altering the contribution of particular terms or regions of the domain during the training process. This modification can be done indirectly by adaptively resampling points in crucial areas [19–21] or directly by assigning multipliers that adjust the contribution of each error function. For instance, Wang et al. [7] proposed a learning rate annealing algorithm that balances the losses' weights based on their back-propagated gradients. Wang et al. [3] also defined a causal parameter for time-dependent problems that force the model to learn sequentially based on the time steps. Global multipliers generally modify the average contribution of each loss term [3,7,22,23]. Similarly, these multipliers can be applied locally (i.e., per training point); McCLenny et al. [4] proposed a self-adaptive (SA) method where the individual loss weights are trained via adversarial training. Zhang et al. [24] extended this concept and proposed a differentiable adversarial self-adaptive (DASA) pointwise weighting scheme that uses a subnetwork to find the optimal multipliers. Basir et al. presented a physics and quality-constrained artificial neural network (PECANN) that computes the local weights based on the residuals of the constraints (i.e., initial and boundary conditions) and the augmented Lagrangian method [25]. PECANN was extended in subsequent studies using adaptive (PECANN-AL) and global Lagrange multipliers [15,26]. Likewise, Son et al. [27] proposed an augmented lagrangian relaxation method for PINNs (AL-PINNs), using the initial and boundary conditions as constraints for the optimization problem of the PDE residual. These weighting strategies enhance the capabilities of PINNs and enable them to address challenging problems in diverse fields. Nevertheless, their implementation can be expensive since they may require coupling auxiliary networks [24], training additional multipliers [4], or computing and processing gradients [7]. Moreover, some of these approaches grow unboundedly [4,15], which may result in over-refining some regions while ignoring others [24].

To address these problems, we propose a residual-based attention (RBA) scheme that efficiently computes adaptive weights for the training points using the residuals of their given losses. The loss residuals contain information about the high error regions, so the obtained weights work as an attention mask and help the optimizer focus on capturing the spatial or temporal characteristics of the given problem. The suggested weighting approach considers the neural network training dynamics and adapts itself during the training process.

This paper is organized as follows. In Section 2, we provide an overview of PINNs, introduce the RBA weighting scheme, and review additional modifications that enhance the model performance. Then, in Section 3, we present the implementation of our method through two benchmark problems. To further evaluate our model performance in real-world applications, we combine RBA with artificial intelligence velocimetry (AIV) [28,29] to infer high-resolution velocity and pressure fields in the cerebrospinal fluid in the perivascular brain space using real data. We conclude each problem with an ablation study that helps us interpret the relative contribution of each modification. Finally, in Section 4, we summarize the findings and discuss our future work. In the appendix, we include details regarding the implementation. The relevant code is available on Github: https://github.com/soanagno/rba-pinns.

2. Methods

2.1. Physics-Informed Neural networks

In the context of Physics-Informed Neural Networks (PINNs), a general partial differential equation (PDE) can be expressed as:

$$\mathcal{D}\{u(x,t)\} = f(x,t), \tag{1}$$

where $u(x,t)$ is the unknown function we wish to approximate, $\mathcal{D}$ denotes the differential operator, and $f(x,t)$ is a given source or forcing function that introduces external influences to the system. The differential operator $\mathcal{D}$ depends on the specific PDE under consideration. For instance, in the case of the heat equation, $\mathcal{D} = \frac{\partial}{\partial t} - \kappa\nabla^2$, where κ is the thermal diffusivity and ∇^2 is the Laplace operator.

Additionally, the problem may be constrained by several types of boundary conditions. These conditions are typically categorized into Dirichlet, Neumann, Robin, or Periodic, and are represented as follows:

$$u(x_b,t) = g_d(x_b,t), \qquad \text{(Dirichlet)} \tag{2}$$

$$\frac{\partial u}{\partial n}(x_b,t) = g_n(x_b,t), \qquad \text{(Neumann)} \tag{3}$$

$$\alpha u(x_b,t) + \beta\frac{\partial u}{\partial n}(x_b,t) = g_r(x_b,t), \qquad \text{(Robin)} \tag{4}$$

$$u(x_b + L,t) = u(x_b,t), \qquad \text{(Periodic)} \tag{5}$$

where x_b denotes the boundary points. The Dirichlet condition defines the function value at the boundary, the Neumann condition specifies the normal derivative at the boundary, and the Robin condition is a weighted combination of the function and its normal derivative. For the Periodic condition, the function value repeats after a certain length L.

The loss function in PINNs is designed to encompass the deviation of the neural network solution from the initial/boundary conditions, data and the PDE itself. The influence of these components in the loss function is balanced using weighting multipliers w_{ic}, w_{bc}, w_d and w_r:

$$\mathcal{L} = w_{ic}\mathcal{L}_{ic} + w_{bc}\mathcal{L}_{bc} + w_d\mathcal{L}_d + w_r\mathcal{L}_r, \tag{6}$$

where $\mathcal{L}_{ic}$, $\mathcal{L}_{bc}$, and $\mathcal{L}_r$ represent the residuals associated with the initial conditions, boundary conditions, any available data and the PDE, respectively. These terms are given by:

$$\mathcal{L}_{ic} = \frac{1}{N_{\text{ic}}} \sum_{i=1}^{N_{\text{ic}}} (u_{NN}(x_{i,\text{ic}}, t_{i,\text{ic}}) - u_{i,\text{ic}})^2 \tag{7}$$

$$\mathcal{L}_{bc} = \frac{1}{N_{\text{bc}}} \sum_{i=1}^{N_{\text{bc}}} (\mathcal{B}\{u_{NN}(x_{i,\text{bc}}, t_{i,\text{bc}}), u_{i,\text{bc}}\})^2 \tag{8}$$

$$\mathcal{L}_r = \frac{1}{N_{\text{r}}} \sum_{i=1}^{N_{\text{r}}} (\mathcal{R}\{u_{NN}(x_{i,\text{r}}, t_{i,\text{r}}), x_{i,\text{r}}, t_{i,\text{r}}\})^2, \tag{9}$$

where u_{NN} is the neural network prediction, $u_{i,j}$ is the expected value at collocation point i with spatio-temporal coordinates (x, t), $\mathcal{B}$ is the boundary condition operator, $\mathcal{R}$ is the PDE residual operator and each N is the number of collocation points of each set. Note that Eq. (7) can also apply to data.

In general, a loss term $\mathcal{L}_j$ takes the form:

$$\mathcal{L}_j = \langle (\lambda_{i,j} \cdot e_{i,j}(u_{NN}))^n \rangle_i, \quad for \quad j \in \{ic, bc, d, r\} \tag{10}$$

where $\langle \cdot \rangle$ is the mean operator, $\lambda_{i,j}$, is an attention mask of multipliers [4] and $e_{i,j}$ is a function of the neural network u_{NN}, associated with the error or residual between the prediction and the ground truth.

Each of these terms encourages the neural network to conform to the respective physical laws or initial/boundary conditions encoded in the problem. Note that the weighting multipliers can be global (e.g., scalars w balancing each individual loss term) or local (e.g., vectors λ balancing each collocation or boundary point). In the most basic PINN definition, $w = \lambda = 1$ for all loss terms.

The training of the neural network weights is performed using a version of gradient descent (GD). In each iteration k, an update is applied to a network weight θ as follows:

$$\theta^{k+1} = \theta^k - \eta \cdot \nabla_\theta \mathcal{L}, \tag{11}$$

where η is the learning rate, and $\nabla_\theta \mathcal{L}$ is the gradient of the loss function with respect to θ. Each iteration updates the parameters θ iteratively, pushing the solution towards satisfying the given PDE and boundary/initial conditions. For batch GD-methods, each epoch consists of multiple mini-batch iterations.

2.2. Residual-based attention (RBA) scheme

One of the inherent challenges in obtaining accurate results with PINNs is that the residuals at key collocation points can get overlooked by the mean calculation of the objective function (Eq. (10)). Consequently, despite a decrease in total loss during training, certain spatial or temporal characteristics might not be fully captured. This issue becomes particularly pronounced in multiscale problems, where such troublesome areas could not only result in a lack of detail but could also impair the flow of important information from the initial and boundary conditions into the domain of interest, thereby hindering convergence. Although mini-batching techniques present a potential solution to this issue, the difficult task of a priori identifying the problematic regions remains.

A workaround for this problem is selecting a set of weighting multipliers, either global or local, which can be adjustable during training. The primary aim of the global multipliers is to balance different terms of the loss function, while local multipliers aim at weighting the influence of specific collocation points of the domain. Establishing a systematic rule for updating these multipliers can significantly aid the optimizer, with several successful strategies documented in the literature, demonstrating major improvements to the vanilla PINNs performance [3,4,24,27].

Furthermore, as opposed to classical numerical methods, which can guarantee stability, the training convergence of PINNs is affected by the evolution of residuals, which will have some degree of stochasticity, leading to a corresponding degree of instability. Moreover, convergence techniques applied on non-convex loss landscapes can only guarantee a solution as a local optimum, while there might exist other better local (or global) optima. Driven by the aforementioned challenges, the initial objective of this work is the development of a simple, gradient-less attention mask based on the rolling history of the cumulative residuals to enhance the focus of the optimizer. The proposed scheme aims to increase attention to the challenging regions of information propagation in both space and time dimensions defining the problem.

Based on the exponentially weighted moving average (EWMA) formulation of classic optimizers like RMSprop and Adam [30], the update rule for the proposed residual-based multipliers for any collocation point i on iteration k is given by:

$$\lambda_i^{k+1} \leftarrow \gamma \lambda_i^k + \eta^* \frac{|e_i|}{\|e\|_\infty}, \quad i \in \{0, 1, \ldots, N\}, \tag{12}$$

where N is the number of collocation points, e_i is the residual of each loss term for point i, γ is the decay parameter and η^* is the learning rate, respectively. Note that the learning rate of the optimizer η and the learning rate of the weighting scheme η^* are two different hyperparameters.

This is a convergent linear homogeneous recurrence relation. Given that $\frac{|e_i|}{\|e\|_\infty} \in [0,1]$ and $\lambda_i^1 \neq 0, \forall i \in \{0,1,\ldots,N\}$, its bounds are given by:

$$\lambda_i^k \in (0, \frac{\eta^*}{1-\gamma}], \quad as\ k \to \infty \tag{13}$$

These local multipliers scale with the normalized cumulative residuals, but they also behave dynamically during training due to the decay γ as follows: since γ lies between 0 and 1, λ^k will gradually decrease in its contribution to λ^{k+1}, as k increases. Simultaneously, $\eta^* \frac{|e_i|}{\|e\|_\infty}$ is being added in each iteration, leading to an eventual equilibrium. However, as different points i may be targeted depending on the stage of the training process, the distribution of λ_i will dynamically vary across the domain of interest.

The main advantages of the proposed residual weighting scheme are summarized as follows:

1. Deterministic operation scheme, bounded by the γ and η^* parameters, which ensure the absence of exploding multipliers.
2. No training or gradient calculation is involved, leading to negligible additional computational cost.
3. Scaling with the cumulative residuals guarantees increased attention on the solution fronts where the PDEs are unsatisfied in both spatial and temporal dimensions.

The modified training process for an indicative Gradient Descent optimization method is outlined in Algorithm 1. We define the RBA weights as $\lambda_{i,j}$ and initialize them with 0 values for all collocation points, so that at $k = 1$ we get $\lambda_i^1 = \eta^* \frac{|e_i|}{\|e\|_\infty}$. We also test the option where $\eta^* = 1$ only for $k = 1$, so that $\lambda_i^1 = \frac{|e_i|}{\|e\|_\infty}$. Both variations yield similar performance. To avoid having very small λ values in problems where outliers or noisy measurements might be present, an offset term λ_o may be added, which works as a lower bound such that $\lambda_{i,j} \in (\lambda_o, \frac{\eta^*}{1-\gamma} + \lambda_o]$. This was only used in the case of Section 3.3.

Algorithm 1: Residual-based attention Gradient Descent

Data: PDE/Boundary collocation points i, learning rate η, RBA learning rate η_0^*, decay γ, offset term λ_o
Result: Optimized network parameters $\theta_i,\ i \in \{0,1,...,N\}$
Initialization: Xavier [31] $\forall\ \theta_i$ and $0\ \forall\ \lambda_{i,j}$, $init = 1\ or\ 2$;
for *training iterations* k **do**
  for *each loss term* j **do**
    Get the PDE and boundary residuals $e_{i,j}$;
    if $init = 2$ *and* $k = 1$ **then**
      $\eta^* = 1$;
    else
      $\eta^* = \eta_0^*$;
    end
    $\lambda_{i,j}^{k+1} \leftarrow \gamma \lambda_{i,j}^k + \eta^* \frac{|e_i|}{\|e\|_\infty}$;
    $\lambda_{i,j}^* \leftarrow \lambda_{i,j}^{k+1} + \lambda_o$
  end
  $\mathcal{L} = \sum_{j=1}^n \langle (\lambda_{i,j}^* \cdot e_{i,j})^2 \rangle_i$;
  $\theta^{k+1} = \theta^k - \eta \cdot \nabla_\theta \mathcal{L}$;
end

2.3. Additional PINN enhancements

2.3.1. Modified multi-layer perceptrons

For the design of the network architecture, we leverage the recently introduced modified multi-layer perceptron (mMLP) [7], which draws its inspiration from the attention mechanisms prominent in transformers [32]. The mMLP aims to augment the efficacy of PINNs by embedding the input variables x into the hidden states of the network. Initially, these inputs are encoded in a feature space by employing two distinct encoders, U and V, given by:

$$U = \sigma(x^0 \theta^U + b^U), \quad V = \sigma(x^0 \theta^V + b^V) \tag{14}$$

The encoders are then assimilated within each hidden layer of a conventional MLP by point-wise multiplication. Thus, each forward pass becomes:

$$\alpha^l(x) = \alpha^{l-1}(x)\theta^l + b^l, \quad for\ \ l \in \{1,2,\ldots,layers\} \tag{15}$$

$$\alpha^l(x) = \sigma(\alpha^l(x)) \tag{16}$$

$$\alpha^l(x) = (1 - \alpha^l(x)) \odot U + \alpha^l(x) \odot V, \tag{17}$$

where x is the input, α^l and θ^l are the neurons and weights of layer l, σ is the activation function and $\odot$ is the element-wise product.

2.3.2. Weight normalization

Weight Normalization is a reparametrization technique that accelerates convergence in PINNs [33]. In this scheme, the weight vectors' length and direction are decoupled to be trained separately [8]. The computation of each neuron output is defined as follows,

$$\alpha = \sigma(\theta \cdot x + b) \tag{18}$$

$$\theta = \frac{g}{\|\mathbf{v}\|_2}\mathbf{v}, \tag{19}$$

where α is the neuron output, σ is the activation function, x is the input vector, θ is a weight vector, and b is the bias. As shown in Eq. (19), the weight vector θ is redefined in terms of new trainable parameters, $\mathbf{v}$ (direction) and g (length). Notice that $\|\theta\| = g$, so this reparametrization allows us to decouple the weight's length and direction, which speeds up the model convergence. Since g is a scalar, this modification induces minimal computational overhead [8].

2.3.3. Exact imposition of boundary conditions

The inexact imposition of boundary conditions can harm the performance and training stability of neural networks [7,34,35]. Therefore, for our experiments, we consider exactly imposing Dirichlet or Periodic boundary conditions as follows:

Dirichlet boundary conditions. A recent study by Sukumar et al. [16] showed how to impose Dirichlet, Neumann, and Robin boundary conditions in PINNs by using approximate distance functions (ADF). The constrained expression for Dirichlet boundary conditions is defined in Eq. (20).

$$u(\mathbf{x}) = g(\mathbf{x}) + \phi(\mathbf{x})u_{NN}(\mathbf{x}), \tag{20}$$

where $g(\mathbf{x})$ is a function that satisfies u along the boundaries, $u_{NN}(\mathbf{x})$ is the output of our neural network, and $\phi(\mathbf{x})$ is a composite distance function that equals zero when evaluated on the boundaries. If the boundary is composed of M partitions $[S_1, \dots, S_M]$, the composite distance function for Dirichlet boundary conditions can be described as:

$$\phi(\phi_1, \phi_2, \dots, \phi_M) = \Pi_i^M \phi_i, \tag{21}$$

where $[\phi_1, \dots, \phi_M]$ are the individual distance functions. Notice that if $\mathbf{x} \in S_i$, then $\phi(\mathbf{x}) = 0$ and the neural network approximation exactly satisfies the boundary conditions, i.e., $u(\mathbf{x}) = g(\mathbf{x})$.

Periodic boundary conditions. Similarly, it is possible to enforce periodic boundary conditions as hard constraints strictly by constructing Fourier feature embeddings of the input data [3,34]. In particular, the periodic constraint of a smooth function $u(x)$ can be encoded in a neural network using a one-dimensional Fourier feature embedding (See Eq. (22)).

$$v(x, y) = [1, cos(\omega_x x), sin(\omega_x x), \dots, cos(m\omega_x x), sin(m\omega_x x)] \tag{22}$$

In the same way, a neural network can incorporate 2D periodic constraints by utilizing a two-dimensional Fourier feature embedding [3].

$$p(x, y) = \begin{bmatrix} cos(\omega_x x)cos(\omega_y y), \dots, cos(n\omega_x x)cos(m\omega_y y) \\ cos(\omega_x x)sin(\omega_y y), \dots, cos(n\omega_x x)sin(m\omega_y y) \\ sin(\omega_x x)cos(\omega_y y), \dots, sin(n\omega_x x)cos(m\omega_y y) \\ sin(\omega_x x)sin(\omega_y y), \dots, sin(n\omega_x x)sin(m\omega_y y) \end{bmatrix} \tag{23}$$

where $\omega_x = \frac{2\pi}{P_x}$, $\omega_y = \frac{2\pi}{P_y}$, m, n are non-negative integers, P_x and P_y are the periods in the x and y directions. Dong et al. [34] proved that any network representation such as $u_{NN}(v(x))$ or $u_{NN}(p(x, y))$ is periodic in the x and x, y directions, respectively.

3. Results

3.1. Dynamic case: 1D Allen–Cahn equation

The Allen–Cahn equation is a compelling benchmark for PINNs due to its challenging characteristics. It is regarded as a "stiff" PDE, a term that describes equations that require careful numerical handling to avoid instability in their solutions. This complexity arises from its ability to produce solutions with pronounced, sharp transitions in both spatial and temporal dimensions. The general PDE form for the 1D Allen–Cahn is given by:

$$\frac{\partial u}{\partial t} = \epsilon \frac{\partial^2 u}{\partial x^2} + f(u), \tag{24}$$

where $\epsilon > 0$ is the diffusion coefficient and $f(u)$ is a non-linear function of u. For this study, we set $\epsilon = 10^{-4}$ and $f(u) = -5u^3 + 5u$. The initial and periodic boundary conditions are given by:

$$u(0, x) = x^2 \cos(\pi x), \quad \forall x \in [-1, 1] \tag{25}$$

$$u(t, x + 1) = u(t, x - 1), \quad \forall t \geq 0 \quad \text{and} \quad x \in [-1, 1] \tag{26}$$

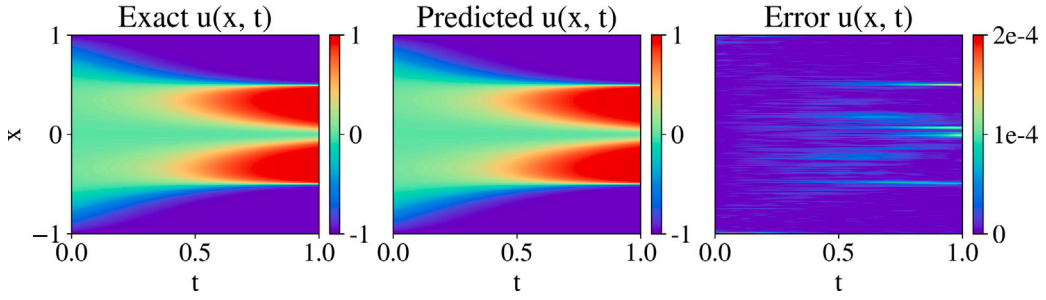

Fig. 1. Exact solution of the 1D Allen–Cahn with the corresponding network prediction and the absolute error difference.

To obtain the solution of Allen–Cahn PDE in Fig. 1, we train a PINN for $3 \cdot 10^5$ iterations with the standard Adam optimizer [30] and an exponential learning rate scheduler on 25,600 collocation points. For the periodic boundary conditions, we use the formulation of Section 2.3.3 and set $w_{ic} = 100$, $w_r = 1$, following the benchmark setup of the recent literature [3]. We also apply RBA only for the PDE residual term $\mathcal{L}_r$, while the rest of the parameters are mentioned in the appendix. The benchmark comparison of the final relative L^2 error obtained by different methods is presented in Table 1, where the error for the RBA weights represents the average of five runs with random seeds. It should be noted that only the best performing cited methods of Table 1 [3,4,24] utilize the PINN enhancements discussed in Section 2.3. Specifically for SA weights, we perform our own 5-run average based on [4], while also including our PINN enhancements for the sake of comparison. For all examples, the relative L^2 is defined as:

$$relative\ L^2 = \frac{\|Exact - Predicted\|_2}{\|Exact\|_2} \tag{27}$$

Table 1
Relative L^2 comparison between different methods of the literature for the 1D Allen–Cahn equation. The RBA run represents the average of five randomly chosen seeds. Note that the relative L^2 of SA weights is obtained by our own 5-run average based on [4], including the enhancements of Section 2.3 for fairness.

Method	Relative L^2 error
Vanilla PINN [1]	4.98e−1
Time marching [36]	1.68e−2
Causal training [3]	1.39e−4
DASA-PINNs [24]	8.57e−5
SA weights [4]	1.11e−4 ± 0.46e−5
RBA weights	**5.8e−5** ± 0.74e−6

3.1.1. Ablation study for Allen–Cahn

An ablation study involves measuring the performance of a system after removing one or more of its components to measure the relative contribution of the ablated components [37]. To investigate the effect of each component of the ensemble on the relative L^2 convergence, we perform a single-component ablation study and report the results in Table 2. The best-performing seed of Table 1 is selected to represent the full model. For each run, a component is removed to identify the effect of the omitted method on the performance, as shown in the convergence histories of Fig. 2. As evident from these results, the Fourier feature embedding is the most important component to achieve low relative L^2. Moreover, when coupled with the RBA weights or the modified MLP, they reach a relative L^2 of 10^{-4}. The full model leads to the best L^2 at $4.57 \cdot 10^{-5}$. Moreover, the RBA weights initiate a steep convergence trajectory at 20,000 iterations, achieving a final L^2 of $5.2 \cdot 10^{-3}$ without any additional component. The noise associated with the best-performing methods indicates that the optimizer is "jumping" over local minima of the loss landscape and effectively not getting stuck in a sub-optimal solution.

Table 2
Final relative L^2 error for each case of the single-component ablation study for the ensemble of the 1D Allen–Cahn.

Method	Relative L^2 error
RBA	5.2e−3
RBA + mMLP	1.58e−3
RBA + Fourier	1.0e−4
mMLP + Fourier	1.0e−4
RBA + mMLP + Fourier	4.57e−5

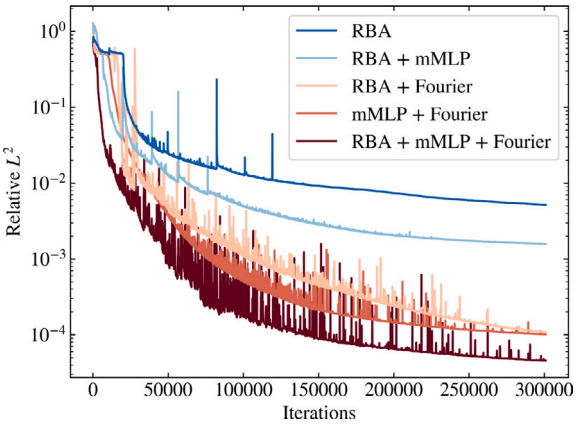

Fig. 2. Ablation study convergence for the 1D Allen–Cahn: Progression of convergence for each component of the ensemble. The results clearly demonstrate that the integration of the RBA approach and the Fourier feature embedding is crucial for attaining a minimal relative L^2. When combined with a modified MLP architecture, an optimal L^2 of $4.57 \cdot 10^{-5}$ is reached. Additionally, the RBA weights address the instability issues in the standard PINN, achieving a relative L^2 of $5.2 \cdot 10^{-3}$ without supplementary components. The noise linked with the top-performing methods suggests that the optimization process successfully avoids becoming trapped in sub-optimal solutions by effectively "leaping" past local minima in the loss landscape.

3.2. Static case: 2D Helmholtz equation

The Helmholtz equation is commonly employed to model wave and diffusion phenomena, capturing changes over time in either a spatial or combined spatial–temporal domain. The 2D Helmholtz PDE is expressed as follows:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + k^2 u - q(x,y) = 0 \tag{28}$$

$$\begin{aligned} q(x,y) = &-(a_1\pi)^2 sin(a_1\pi x) sin(a_2\pi y) \\ &-(a_2\pi)^2 sin(a_1\pi x) sin(a_2\pi y) \\ &+ k sin(a_1\pi x) sin(a_2\pi y), \end{aligned} \tag{29}$$

where $q(x,y)$ is the forcing term which leads to the analytical solution $u(x,y) = sin(a_1\pi x) sin(a_2\pi y)$ [4]. The boundary conditions are defined as follows:

$$u(-1,y) = u(1,y) = u(x,-1) = u(x,-1) = 0 \tag{30}$$

To allow a direct comparison with previous studies, we set $a_1 = 1$, $a_2 = 4$, $k = 1$ and follow the experimental setup described in [24], setting $w_{bc} = 100$. We again apply RBA only for the PDE residuals $\mathcal{L}_r$. We initially apply a combination of Dirichlet and Periodic boundary conditions, with the latter enforced using the Fourier feature embedding. However, to avoid encoding the PDE analytical solution (i.e., $\sin(\pi x)\sin(4\pi y)$), we expand our inputs (i.e., x, y) using a truncated embedding with $m = n = 5$ (Eq. (31)):

$$p(x,y) = \begin{bmatrix} cos(\omega_x x)cos(\omega_y y), \ldots, cos(n\omega_x x)cos(n\omega_y y) \\ cos(\omega_x x)sin(\omega_y y), \ldots, cos(n\omega_x x)sin(n\omega_y y) \\ sin(\omega_x x)cos(\omega_y y), \ldots, sin(n\omega_x x)cos(n\omega_y y) \end{bmatrix} \tag{31}$$

Then, we approximate the solution of the PDE as $u_F = u_{NN}(p(x,y))$, where u_{NN} is the modified MLP. Additionally, to demonstrate the flexibility of RBA weights, we examine an additional case applying only Dirichlet boundaries with approximate distance functions (ADF). Towards this end, we split the BCs into four partitions, i.e., $y = 1$, $y = -1$, $x = 1$, and $x = -1$, and use $\phi = (x^2 - 1)(y^2 - 1)$ as the composite distance function. Based on Eq. (30) we define $g(\mathbf{x}) = 0$ and approximate the solution of the Helmholtz equation as:

$$u_{ADF} = (x^2 - 1)(y^2 - 1) u_{NN}(x,y), \tag{32}$$

which exactly satisfies the BCs. We train our models (i.e., u_F and u_{ADF}) using the Adam optimizer for $2 \cdot 10^4$ iterations, followed by L-BFGS for $2 \cdot 10^3$ iterations. The exact solution of the 1×4 Helmholtz PDE, alongside the corresponding prediction obtained from the PINN is shown in Fig. 3. To allow a direct comparison with the self-adaptive weights (SA), we also train a second model with identical hyperparameters and update the local weights using gradient ascend as defined in [4]. The average L^2 errors for the RBA and SA weights with Fourier and ADF are shown in Table 3, along with the results reported by the current state-of-the-art methods.

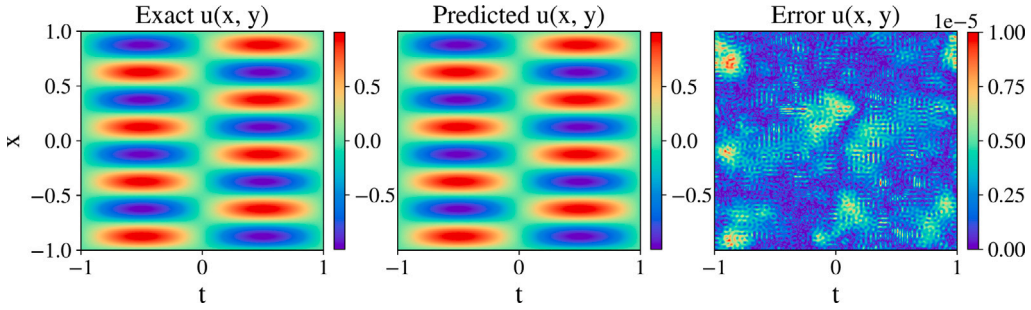

Fig. 3. Analytical solution for the 2D Helmholtz equation and the corresponding network prediction with the absolute error difference.

Table 3
Relative L^2 comparison between different methods of the literature for the 2D Helmholtz equation. The RBA and SA values represent the average result of five randomly chosen seeds. The relative L^2 of SA weights is obtained by our own 5-run average based on [4], including the enhancements of Section 2.3 for fairness.

Method	Relative L^2 error
Vanilla PINN [7]	8.14e−2
Learning-rate annealing [7]	1.49e−3
PECANN-AL [15]	2.77e−4
AL-PINNs [27]	7.46e−4
DASA-PINNs [24]	5.35e−5
SA weights (ADF) [4]	1.97e−4 ± 4e−5
SA weights (Fourier) [4]	4.80e−5 ± 9e−6
RBA weights (ADF)	7.73e−5 ± 1e−5
RBA weights (Fourier)	**8.21e−6 ± 3e−6**

3.2.1. Ablation study for Helmholtz

In this ablation study for the 2D Helmholtz equation, we present the convergence histories for each ablated system (Fig. 4) and summarize the final L^2 errors in Table 4. For the first case (i.e., u_F), the Fourier feature embedding has the most significant influence in reaching a minimized relative L^2. The accuracy further improves paired with either the RBA scheme or the mMLP, while the best-performing seed leads to an L^2 of $5.18 \cdot 10^{-6}$; however, we note that this method is employed for comparison purposes since the Fourier features incorporate the trigonometric traits of the analytical solution. Therefore, we also perform the same study using distance functions (u_{ADF}). In general, exactly imposing the boundary conditions is important for accuracy, as it reduces the number of loss terms and allows the optimizer to focus on minimizing a single loss objective.

Table 4
Final relative L^2 error for each case of the ablation study for the 2D Helmholtz ensemble using either the Fourier feature embeddings for exact periodic BC or ADF for exact Dirichlet BC.

Method	Relative L^2 error
RBA	3.66e−3
RBA + mMLP	5.91e−4
mMLP + Fourier	1.50e−5
RBA + Fourier	5.57e−5
RBA + mMLP + Fourier	5.18e−6
mMLP + ADF	6.34e−5
RBA + ADF	1.52e−3
RBA + mMLP + ADF	5.96e−5

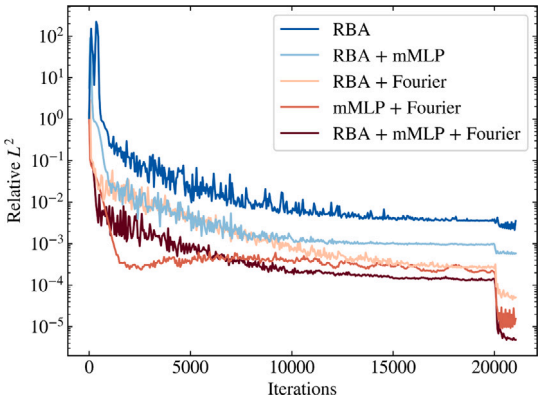

Fig. 4. Ablation study convergence for the 2D Helmholtz: The convergence trajectory for each experiment is illustrated here. For this case, the Fourier feature embedding is critical in achieving a low relative L^2. Coupled with the RBA weights and a modified MLP architecture, the model reaches an optimal L^2 of $5.18 \cdot 10^{-6}$ after $2 \cdot 10^4$ Adam and $2 \cdot 10^3$ L-BFGS training steps.

3.3. Inverse case: 3D Navier–Stokes

To further investigate the effectiveness of our approach in real-world applications, we combine RBA with artificial intelligence velocimetry (AIV) to model the cerebrospinal fluid flow (CSF) in the perivascular spaces (PVS). Accurate quantification of CSF velocity and pressure is critical for understanding the glymphatic system (GS), which is related to the brain's waste removal and nutrient delivery. Building on the work from Boster et al. [28], we infer 3D high-resolution velocity pressure fields from 3D boundaries (BCs) and 2D plane velocities collected from the PVS of a murine's brain. The data utilized in this study comes from the boundary conditions and internal flow velocity measurements. Specifically, the internal flow velocity measurements are obtained using particle tracking velocimetry (PTV), and the velocities of the boundary conditions are determined from the geometry, assuming non-slip boundary conditions.

To obtain the PTV data, Boster et al. [28] injected one-micron microsphere tracers into the CSF, enabling the visualization and quantification of flow within the PVS. The microsphere motion was captured using PTV, and the PVS geometry was reconstructed from a 3D volume scan (see Fig. 5(a)). Subsequently, [28] identified a PVS subdomain from which they isolated the boundary conditions (BCs) and collected 2D velocity data points from three different planes, as illustrated in Fig. 5(b) and (c). Finally, to enhance the temporal resolution of the PTV dataset, the authors consolidated PTV data from multiple cardiac cycles into a single (phase-merged) cardiac cycle of $T = 0.303s$. In this study, to evaluate the PDE residuals during training, we generated collocation points by shrinking and combining the boundary conditions as shown in (Fig. 5(d)), which allowed us to control the spatial distribution of points within our domain.

Generally, obtaining an attention mask requires processing at least as many multipliers as training points, which may hinder their optimal application for large datasets due to their computational cost. To address this issue, we propose using ordered mini-batches, which enable us to use a single attention mask for every minibatch. Specifically, we partition our collocation points and boundary conditions (BCs) into G_c and G_b training groups based on their spatial and temporal coordinates (see Fig. 5(e)). Mini-batches are then created by selecting one point from each training group. Following this approach, we can share the same multiplier along the training group (Fig. 6(b)), reducing the computational cost and updating the attention mask at the same rate as the network parameters (i.e., per mini-batch). Given the large amount of collocation and BC points required to capture the 3D flow over T, we train our model using ordered mini-batches for the BCs and the system of PDEs while for the PTV data we use the entire dataset.

The governing physical laws for this problem are the Navier–Stokes (NS) equations, given by:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{v} \tag{33}$$

$$\nabla \cdot \mathbf{v} = 0, \tag{34}$$

hence, following previous studies [14,28,29,33], we use a neural network to obtain a NS solution (i.e., 3D velocities and pressure) that follows the flow defined by the PTV data and BCs (Fig. 6(a)). However, since the 3D velocities and pressure (u, v, w, p) have different scales, we redefine them using the scalars (u_s, v_s, w_s, p_s) that ensure the network outputs have a similar order of magnitude. In particular, we approximate u, v, w and p as follows:

$$u = u_s u_{NN}(t, x, y, z) \tag{35}$$

$$v = v_s v_{NN}(t, x, y, z) \tag{36}$$

$$w = w_s w_{NN}(t, x, y, z) \tag{37}$$

$$p = p_s p_{NN}(t, x, y, z), \tag{38}$$

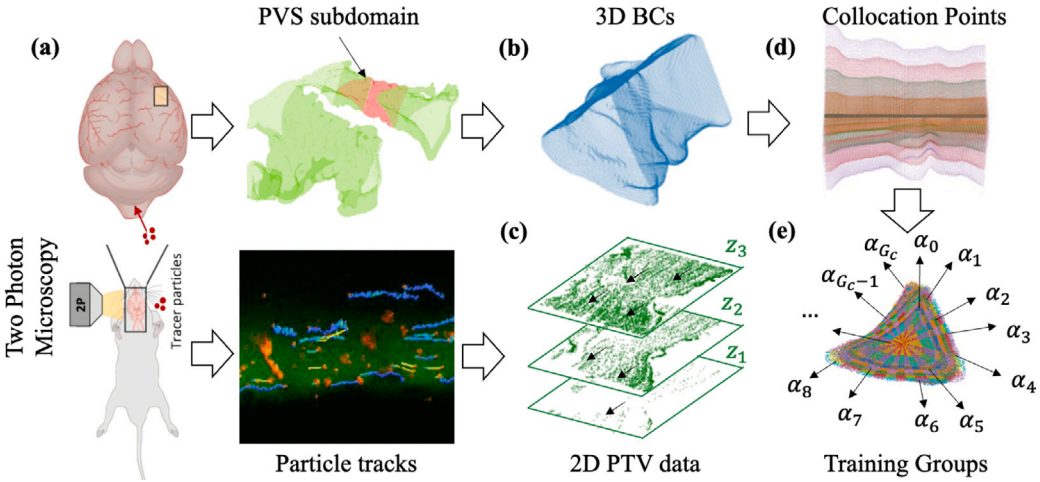

Fig. 5. Artificial intelligence velocimetry data collection and generation: (a) Tracer particles and one-micron microspheres are injected into a murine's brain to quantify and visualize the cerebrospinal flow in the perivascular spaces (PVS). The particle tracks are identified via two-photon microscopy, and their corresponding velocities are determined using particle tracking velocimetry. The PVS geometry is inferred from a volume scan. (b) 3D boundary conditions are specified from a PVS subdomain [28]. (c) 2D velocity data is obtained from the particle tracks in three different planes. (d) The collocation points are generated by shrinking and combining the boundary conditions. (e) The collocation points are classified into G_c training groups (α_i) based on the spatial and temporal coordinates.

where t, x, y, z are the inputs (i.e., time and position) and $u_{NN}, v_{NN}, w_{NN}, p_{NN}$ are the neural network outputs. Finally, to stabilize the training process and speed up convergence, we reparameterize our network using weight normalization (WN) [8].

During training, we apply global weights ($w_{\beta=\{b,d,n\}}$) to balance the contribution of the multiple loss terms, namely four NS (i.e., conservation of mass and conservation momentum in x, y, and z), two PTV data (i.e., velocity in x and y), and three non-slip BCs (i.e., velocity in x, y and x). Additionally, we apply RBA weights to control the interplay inside each loss term. In this step, we share RBA weights along the training groups for the residuals of the BC (e_{bh}) and NS (e_{nj}) while we assign individual multipliers for the PTV data (e_{di}) (Fig. 6(b)). We initialize the RBA with zero and update them using Eq. (12). However, given the uncertainty in the BCs [28], we constrain them lightly by assigning them a lower RBA learning rate (i.e., η^*), which induces a smaller upper bound and within a smaller contribution. Finally, we define the total loss function as $\mathcal{L} = \mathcal{L}_{bc} + \mathcal{L}_d + \mathcal{L}_{ns}$, where the combined loss terms for BCs ($\mathcal{L}_{bc}$), PTV data ($\mathcal{L}_d$) and NS ($\mathcal{L}_{ns}$) are defined as follows:

$$\mathcal{L}_{bc} = \sum_{b=1}^{3} w_b \langle (\lambda_h \cdot e_{h,b})^2 \rangle_h \tag{39}$$

$$\mathcal{L}_d = \sum_{d=1}^{2} w_d \langle (\lambda_i \cdot e_{i,d})^2 \rangle_i \tag{40}$$

$$\mathcal{L}_{ns} = \sum_{n=1}^{4} w_n \langle (\lambda_j \cdot e_{j,n})^2 \rangle_j, \tag{41}$$

here, $e_{h,b}, e_{i,d}, e_{j,n}$ are the BC, data and NS residuals for points h, i, and j (see Fig. 6(a)), w_b, w_d, w_n are global weights applied to each loss term, λ_h, λ_j are the RBA assigned for the BCs and NS training groups and λ_i are the RBA assigned for each PTV point. We note that the BCs and NS are averaged over their corresponding batch sizes, while the data terms are averaged over the PTV points.

To evaluate the AIV performance, we split our PTV data into 60% for training and 40% for validation and train the model for $2.5 \cdot 10^5$ iterations using Adam optimizer [30]. Additional implementation details are described in A.3.1. Once the model is trained, we get a continuous and differentiable function for the velocity ($\mathbf{v} = [u, v, w]$) and pressure (p). Thus, following [28] we use automatic differentiation to compute the vorticity ($\boldsymbol{\omega}$) and shear stress (τ) as follows,

$$\boldsymbol{\omega} = \nabla \times \mathbf{v} \tag{42}$$

$$\tau = \mu \dot{\gamma} \ , \tag{43}$$

where μ is the viscosity, and $\dot{\gamma}$ is the shear rate computed from the stress–strain rate tensor ($\mathbf{E}$) described as,

$$\dot{\gamma} = \sqrt{2\mathbf{E} : \mathbf{E}} \tag{44}$$

$$\mathbf{E} = \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T) \tag{45}$$

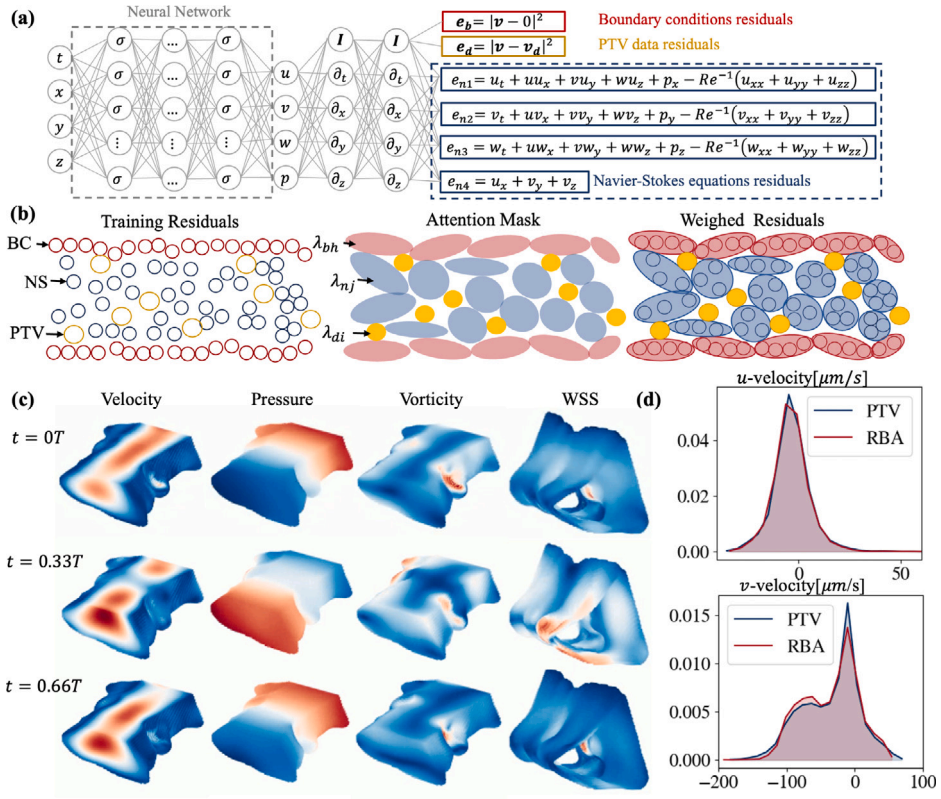

Fig. 6. Artificial intelligence velocimetry framework: (a) Schematic diagram of the AIV neural network. (b) Each loss term is computed by element-wise multiplication between the training residuals and their corresponding RBA. The RBA is shared for the NS and boundary residuals. However, individual RBA weights are assigned for each data point. (c) Inferred 3D high-resolution velocity, pressure, vorticity magnitude, and wall shear stress (WSS) at $t = 0T, 0.33T$, and $0.66T$. Notice that the proposed approach successfully captures the back-flow occurring at $t = 0.33T$. (d) Histograms of the velocity components for the PTV validation dataset during the entire cardiac cycle (T).

Table 5
Relative L^2 comparison for the validation dataset between the original study [28] and our approach using vanilla PINN and combining weight normalization with SA or RBA weights. The weighted PINN, RBA, and SA values represent the average result of five randomly chosen seeds.

Method	Relative $L^2(u)$	Relative $L^2(v)$
Learning-rate annealing [28]	4.21e−1	2.11e−1
PINN + global weights	3.31e−1	2.67e−1
SA weights [4]	2.94e−1 ± 2.5e−3	2.48e−1 ± 2.3e−3
RBA weights	**2.43e−1 ± 1.6e−3**	**2.10e−1 ± 1.5e−3**

As shown in Fig. 6(c), the proposed method can reconstruct 3D high-resolution velocity, pressure, vorticity, and wall shear stress (WSS) fields from 3D BCs and 2D plane velocities. Notice that our model successfully captures the backflow occurring at $t = 0.33T$. Similarly, it can be observed that the irregular regions in the BCs induce high vorticity and WSS. Finally, Fig. 6(d) shows that the predicted 2D velocities (i.e., u and v) follow the same distribution as the PTV data from the validation dataset. Additional results for different time-steps along the cardiac cycle T are shown in Fig. A.13.

To allow a direct comparison with the self-adaptive weights (SA), we train a second model with identical hyperparameters and update the local weights using gradient ascend as defined in [4]. Table 5 shows the relative L^2 error in the validation dataset for the 2D PTV velocity (u, v) in x and y directions. For reference, we also include the results obtained by [28], where the authors trained their model using Learning-rate annealing [7] for $5 \cdot 10^5$ iterations (i.e., twice as many as us). Notice that applying global weights along with local multipliers is essential to achieve a low relative L^2 error, especially for the velocity in the x direction.

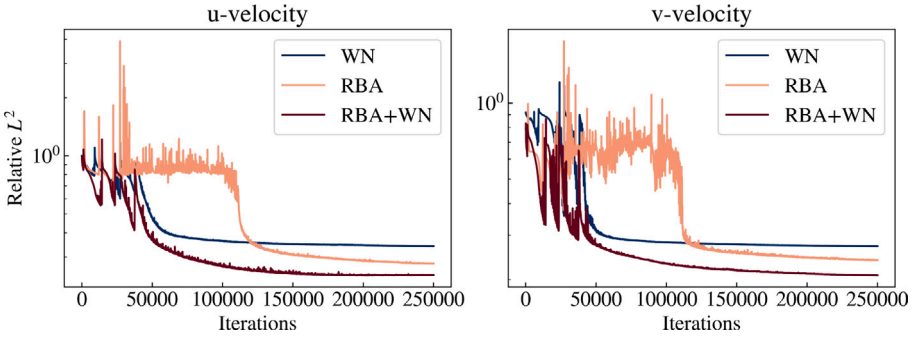

Fig. 7. Ablation study convergence for AIV: Weight normalization speeds up the model convergence. However, applying RBA multipliers is critical to achieving a low relative L^2. The full model, which combines RBA with weight normalization, reaches an optimal L^2 of $2.42 \cdot 10^{-1}$ and $2.08 \cdot 10^{-1}$ for the velocity in x and y (i.e., u and v) after $2.5 \cdot 10^4$ Adam training iterations.

Table 6
Final relative L^2 error for each case of the ablation study on the AIV for the u, v velocity using the validation dataset.

Method	Relative $L^2(u)$	Relative $L^2(v)$
WN	3.41e−1	2.71e−1
RBA	2.77e−1	2.39e−1
RBA + WN	2.42e−1	2.08e−1

3.3.1. Ablation study for AIV

The final relative L^2 errors for each ablated system on the validation dataset are summarized in Table 6. As shown in Fig. 7, the RBA weights are crucial to achieving a low relative L^2 error; however, the WN is essential for accelerating convergence and stabilizing the training process. By combining RBA with WN, the full model achieves an optimal L^2 of $2.42 \cdot 10^{-1}$ and $2.08 \cdot 10^{-1}$ for the u and v velocity, respectively.

3.4. Comparison of RBA, SA weights and vanilla

Both RBA and SA weights are based on the same principle: assign higher weight to more important collocation points and increase the attention of the optimizer on certain regions of the domain. The key difference between the two methods is that SA weights accumulate the gradient term $\frac{\partial \mathcal{L}}{\partial \lambda}$ (through gradient ascent), while RBA weights accumulate the normalized residual $\frac{|e_i|}{\|e\|_\infty}$. Thus, RBA does not train any auxiliary λ, instead it directly uses a moving average of the residuals. In this section, we compare the performance of plain RBA with plain SA, against that of the vanilla PINN. Thus, we set the loss term weights w to 1 for all cases and for RBA we set $\eta^* = 0.001 = 1 - \gamma$. For simplicity, we use the same collocation point sampling for Helmholtz as we do for Allen–Cahn and we also drop the periodic Fourier features. The rest of the settings remain the same as in the ablation studies. Note that for SA, we initialize λ_{ic} with (0–100) for Allen–Cahn and λ_{bc} with (0–1) for Helmholtz [4].

The convergence plots of Fig. 8 show that RBA accelerates the convergence of vanilla PINN by one order of magnitude. Note that for Allen–Cahn, the vanilla model is able to converge due to the learning rate scheduler. The SA weights also provide a substantial improvement to the vanilla, which is around four times faster for the given setup. The two initialization options of Algorithm 1 show similar performance. This indicates that the balance between the residuals is what leads to convergence and not a higher w weight for the initial/boundary loss terms. Additionally, both benchmarks of Fig. 9 show relative agreement in the regions of higher attention. However, RBA appear to produce a less noisy attention mask, which is more responsive in the initial iterations. Moreover, during the final iterations RBA becomes more homogeneous, which implies uniform domain convergence. Finally, while SA weights solve a min–max optimization problem (minimize the residuals while maximizing λ), the normalization with $\|e\|_\infty$ ensures that RBA weights are bounded (in this case from 0 to 1).

These results indicate that adaptive weight strategies like RBA and SA, have a significant impact on the vanilla PINN performance. Moreover, even in cases where we used ten times less collocation points, both RBA and SA converged to a reasonable solution, while the vanilla failed to do so. Therefore, we would like to emphasize the importance of local attention mechanisms, which effectively guide the optimizer by focusing on the essential information of the solution space.

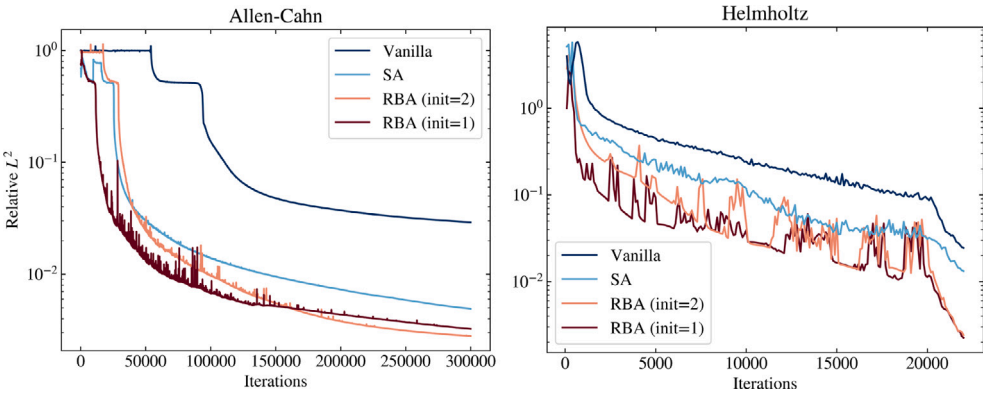

Fig. 8. Convergence for plain RBA, plain SA and vanilla. Allen–Cahn (left) and Helmholtz (right). The RBA method achieves one order of magnitude faster convergence, even without initial/boundary term weights w. The SA method is on average four times faster than vanilla.

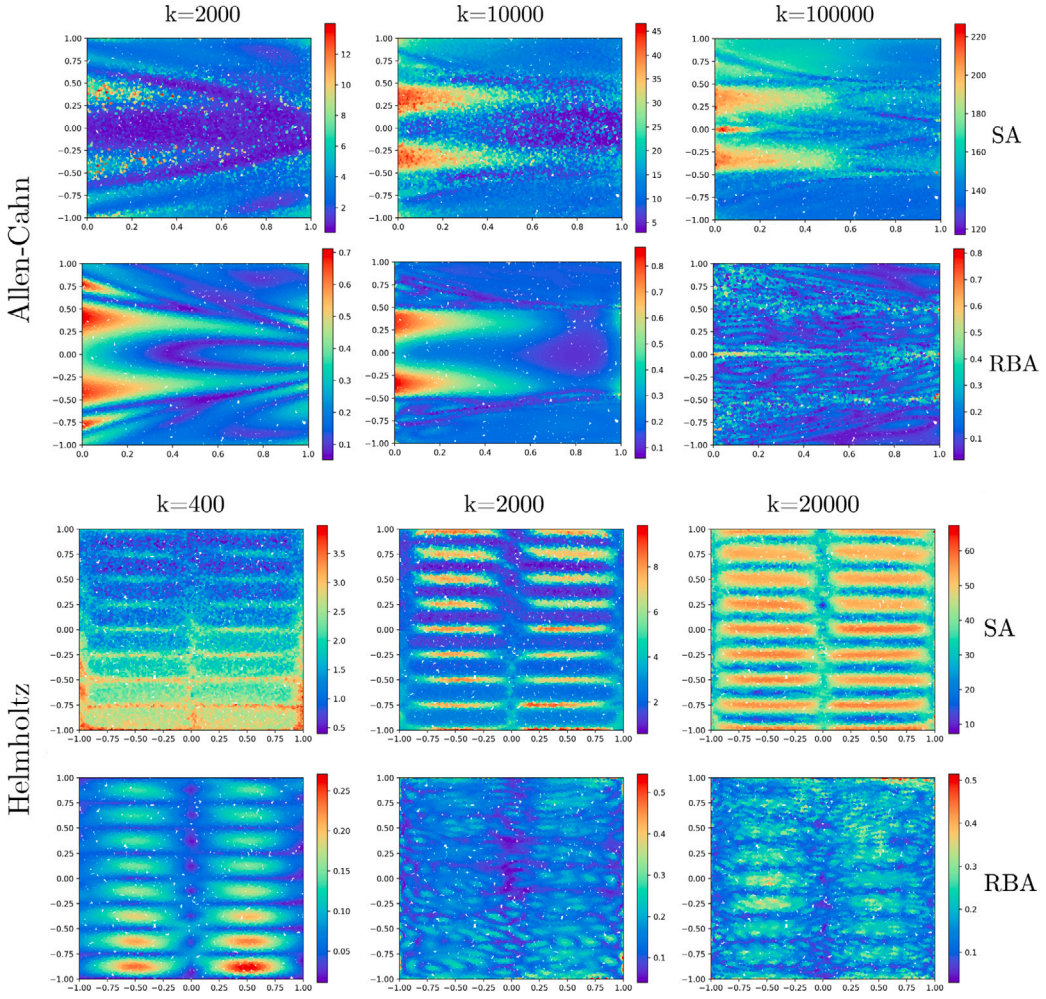

Fig. 9. Weight maps for RBA and SA. Both cases show relative agreement. However, RBA appear to produce a less noisy attention mask, which is more responsive in the initial iterations. Moreover, during the final iterations RBA becomes more homogeneous, which implies uniform domain convergence.

4. Summary

In this work, we propose a fast, residual-based attention scheme to enhance the accuracy of PINNs in both forward (i.e., static/dynamic) and inverse problems, competing with the state-of-the-art performing models of the literature. Additionally, we believe that an equally important element of local-attention strategies is in the realms of convergence. For example, the 1D Allen–Cahn equation is a hard problem to be solved by the vanilla PINN due to its stiff nature, but RBA can initiate an early convergence trajectory, without any empirical loss term weights w. Furthermore, when combined with other PINN enhancements, the ablation study points out that an important aspect of achieving a solution accuracy of $\mathcal{O}(10^{-5})$ is the formulation of exact boundary conditions. The versatility of the proposed method allows for its application with a few additional lines of code and can be extended to complex problems with multiple loss terms. Therefore, we also combine it with artificial intelligence velocimetry (AIV) to model the cerebrospinal fluid flow (CSF) in the perivascular spaces (PVS). Our observations indicate that the RBA weights are critical to reduce the relative L^2 error and speed up convergence. Integrating the proposed attention scheme with existing methodologies can enable more reliable and accurate PINN or neural operator solutions in complex physical modeling scenarios.

CRediT authorship contribution statement

Sokratis J. Anagnostopoulos: Conceptualization, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Juan Diego Toscano:** Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing, Conceptualization. **Nikolaos Stergiopulos:** Methodology, Resources, Supervision, Writing – review & editing, Conceptualization. **George Em Karniadakis:** Methodology, Resources, Supervision, Writing – review & editing, Conceptualization.

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.

Data availability

We have shared a link to the github repository.

Acknowledgments

SJA and NS thank the Swiss National Science Foundation grant “Hemodynamics of Physiological Aging” (Grant nr 205321_197234). JDT and GEK acknowledge support by the DOE SEA-CROGS project (DE-SC0023191), the MURI-AFOSR FA9550-20-1-0358 project, and the ONR Vannevar Bush Faculty Fellowship (N00014-22-1-2795). Finally, we thank Prof. N. Sukumar from the University of California, Davis, for engaging in insightful discussions on the approximate distance functions.

Appendix. Implementation details

A.1. RBA weight evolution

This section examines the evolution of the RBA values for the full model of the ablation study for each case. Fig. A.10 shows the evolution of RBA weights for Allen–Cahn and Helmholtz equations. For each run, the maximum value is bounded by Eq. (13), which for the ablation studies of Allen–Cahn and Helmholtz equals 10. The distribution of weights significantly varies depending on the stage of training; hence, the maximum values fluctuate while approaching the upper bound. The fluctuation of the mean values is less pronounced until they finally converge to $\approx 20\%$ of the upper bound. This behavior indicates that, on average, the total magnitude of weights remains constant while their distribution varies as the optimizer focuses on different parts of the domain. Thus, by introducing the decay factor γ, we can effectively prevent exploding weight values. At the same time, this allows the weights to be flexibly distributed, adapting as needed throughout different stages of the solution process.

Similarly, Fig. A.11 shows the variation of the RBA multipliers along the training process for the AIV. For this case, we have nine sets of RBAs (i.e., one per loss term); however, to ease the visualization, we compute the respective metrics (i.e., max, mean, and min) for each loss term group (i.e., BCs, PTV data, and NS). Notice that the larger learning rate (η^*) induces a higher upper bound. Also, the maximum value for the PTV data grows rapidly and remains constant along the training process. This may indicate that, since we are dealing with real data, there may be outliers or contradictory information that the model cannot learn. On the other hand, the BCs and NS RBA weights reach their maximum value in the first iterations and then start decreasing, which may reveal that the difficult regions have already been learned.

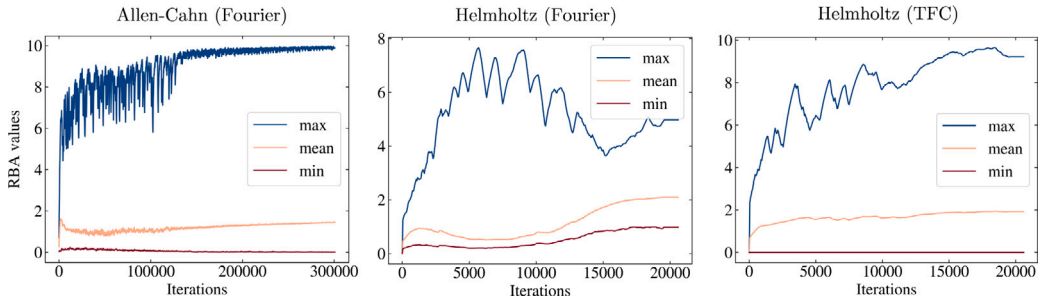

Fig. A.10. Evolution of RBA weights for Allen–Cahn and Helmholtz. For each case, the peak value is bounded by Eq. (13) and equals to 10. Notice that the maxima fluctuate until they approach the upper limit during the final learning stages. On the other hand, the mean values show a less noticeable fluctuation before eventually stabilizing at about $\approx 20\%$ of the upper bound. This trend indicates that, on average, the total magnitude of weights remains constant while their distribution adjusts dynamically as the optimizer shifts its focus across different domain sections.

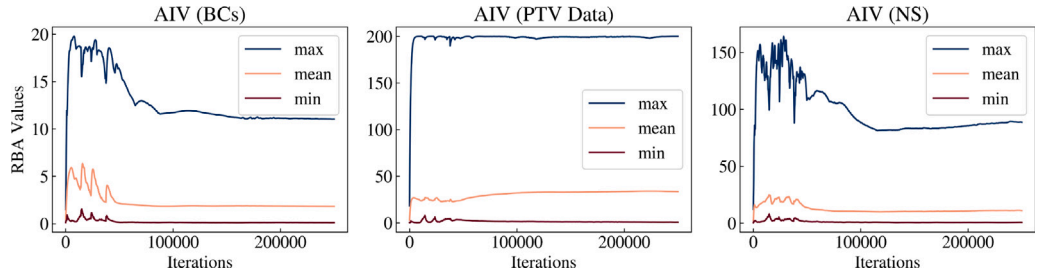

Fig. A.11. Evolution of RBA weights for AIV. We illustrate the RBA multipliers' variation throughout the AIV training process. Although there are nine sets of RBAs (i.e., one for each loss term), we compute each group's maximum, mean, and minimum values (i.e., BCs, PTV data, and NS) to simplify visualization. Notice that the lower learning rate for the BCs leads to a lower upper limit. Also, the maximum RBA for the PTV data increases to its upper limit and remains constant along the training process, which suggests that there are unlearnable points in the dataset (i.e., outliers or contradictory information). On the other hand, the BCs and NS weights reach their max in the initial iterations and then start decreasing, indicating that the difficult points have already been learned.

A.2. Benchmark settings

The network parameters are initialized using Xavier normal distribution [31]. The rest of the case-specific information and their corresponding training time are tabulated below (see Table A.7, Table A.8, Table A.9, Table A.10, Table A.11, Table A.12, Table A.13). Note that the reported training time should only be used as an inner reference since the codes were not optimized for speed.

Table A.7
Allen–Cahn implementation details.

Implementation details	
Sampling method	Latin Hypercube
Number of collocation points	25 600
Number of initial condition points	512
Number of hidden layers	6
Neurons per layer	128
Activation function	tanh
Reparametrization	mMLP
Initial learning rate	1e−3
Decay rate	0.9
Decay step	5000
RBA decay parameter (γ)	0.999
RBA learning rate (η^*)	1e−2
Offset (λ_o)	0
ML framework	PyTorch
GPU	GeForce RTX 3090

Table A.8
Comparison of computational time for different combinations of the 1D Allen–Cahn ablation study.

Method	Training time (ms/iteration)
Vanilla	17
RBA	17
RBA + mMLP	58
RBA + Fourier	18
mMLP + Fourier	58
RBA + mMLP + Fourier	58

Table A.9
Helmholtz (Fourier) ($a_1 = 1$, $a_2 = 4$) implementation details.

Implementation details	
Sampling method	Uniform grid
Number of collocation points	1201
Number of boundary points	200
Number of hidden layers	6
Neurons per layer	128
Activation function	tanh
Reparametrization	mMLP
Initial learning rate	5e −3
Decay rate	0.7
Decay step	1000
RBA decay parameter (γ)	0.9999
RBA learning rate (η^*)	1e−3
Offset (λ_o)	0
ML framework	Jax
GPU	GeForce RTX 3090

Table A.10
Training time for 2D Helmholtz ($a_1 = 1$, $a_2 = 4$).

Method	Training time (ms/iteration)
Vanilla	13.8
RBA	13.8
RBA + mMLP	18.6
RBA + Fourier	22.6
mMLP + Fourier	37.8
RBA + mMLP + Fourier	34.4
RBA + ADF	14.2
mMLP + ADF	18.4
RBA + mMLP + ADF	18.6

A.3. 3D Navier–Stokes settings

A.3.1. Collocation points and training groups generation

We obtain our collocation points by shrinking and combining the boundary conditions as shown in Fig. A.12. First, we split our boundaries into 25 angular (N_θ) and 15 axial (N_y) subregions. Then, we shrink and combine them to get an equally spaced domain of 200 layers. After that, we split our domain into seven radial groups (N_r), tile them 101 times, and split the time domain into 20 temporal subregions (N_t). So, the total number of training groups (G_c) for the NS would be:

$$G_c = N_t \times N_r \times N_\theta \times N_y = 52500$$

Similarly, we split the boundary conditions with $N_\theta = 25$, $N_y = 15$ and $N_t = 15$ as:

$$G_b = N_t \times N_\theta \times N_y = 5625$$

Notice that the total number of collocation $G_c \times Nb = 4.98e6$ and BC points $G_b \times Nb = 5.34e5$ approximately matches the number of training points used in the previous study (i.e., $5e6$ and $5e5$) [28].

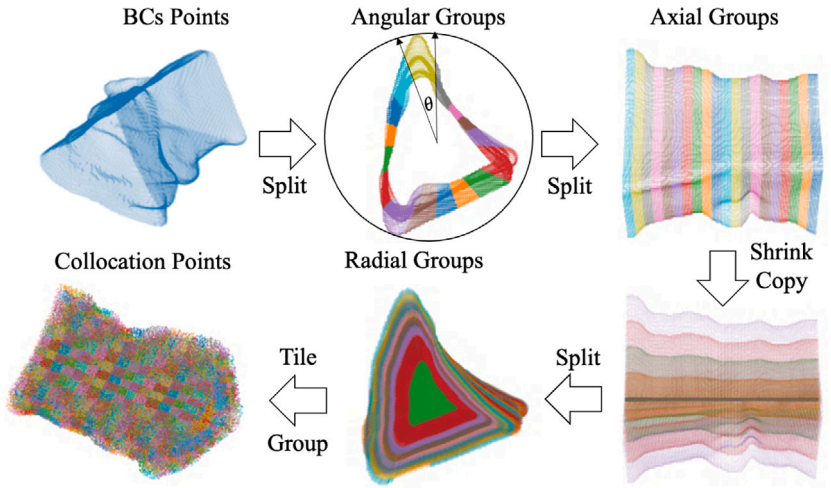

Fig. A.12. Collocation points generation: We split our boundary data points into N_θ angular and N_y axial subgroups. After that, we shrink-copy, combine the boundary points, and classify them in N_r radial sections. Finally, we tile them and classify them into N_t temporal subgroups and group the points into training groups based on their spatial and temporal coordinates.

Table A.11
Global weights w for AIV.

Los term	Global weight
No-slip BC in x	100
Non-slip BC in y	100
Non-slip BC in z	100
PTV velocity in x	50
PTV velocity in y	5
Conservation of Momentum in x	1
Conservation of Momentum in y	1
Conservation of Momentum in z	1
Conservation of Mass	10

Table A.12
AIV implementation details.

Implementation details	
Number of BCs data points	5.34e5
Number of PTV data points (Train)	9.4e3
Number of PTV data points (Validation)	1.4e4
Number of collocation points	4.98e6
BCs batch size	5.63e3
PTV data batch size	9.40e3
Navier–Stokes batch size	5.25e3
Number of hidden layers	8
Neurons per layer	200
Activation function	sin
Reparametrization	WN
Initial learning rate	2e−3
Final learning rate	1.5e−4
Decay rate	0.9
RBA decay parameter (γ)	0.999
RBA Learning rate for PTV and NS (η^*)	2.0e−1
RBA Learning rate for BCs (η^*)	2.0e−2
Offset (λ_o)	1.0
ML framework	Jax
GPU	A100

Table A.13
Computational time for different combinations of the AIV ablation study.

Method	Training time (ms/iteration)
Vanilla PINN	325
WN	364
RBA	329
RBA + WN	366

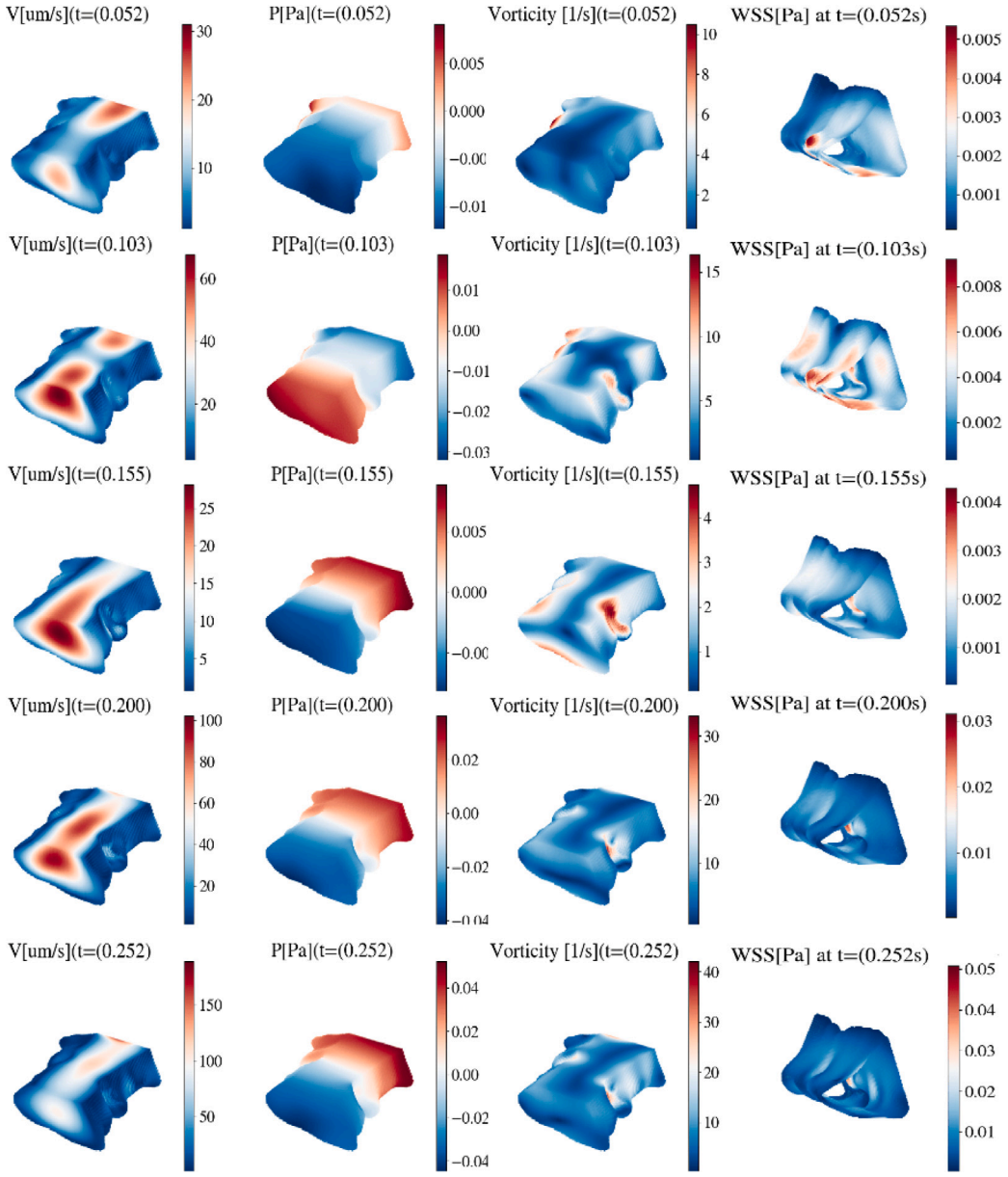

Fig. A.13. AIV Results. Once the model is trained, we get a continuous and differentiable function for the velocity ($\mathbf{v} = [u, v, w]$) and pressure (p). Thus, following [28], we use automatic differentiation to compute the vorticity ($\boldsymbol{\omega}$) and wall shear stress (WSS). We present our results at five different time steps that describe the cardiac cycle T.

A.3.2. Navier–Stokes approximation

We approximate the 3D velocities (u, v, w) and pressure (p) as follows:

$$
\begin{aligned}
u &= u_s u_{NN}(t, x, y, z) \\
v &= v_s v_{NN}(t, x, y, z) \\
w &= w_s w_{NN}(t, x, y, z) \\
p &= p_s p_{NN}(t, x, y, z),
\end{aligned}
$$

where t, x, y, z are the inputs, $u_{NN}, v_{NN}, w_{NN}, p_{NN}$ are the neural network outputs and $(u_s, v_s, w_s, p_s) = (0.1, 1, 0.1, 100)$.

Empirically, we observed that there is a performance improvement by assigning specific global weights to each loss term. The case-specific multipliers used for this problem are shown in Table A.11. The remaining details about the implementation are described in Table A.12. Additional AIV results for different time steps along the cardiac cycle T are shown in Fig. A.13.

References

[1] M. Raissi, P. Perdikaris, G.E. 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.

[2] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: A review, Acta Mech. Sin. 37 (12) (2021) 1727–1738.

[3] S. Wang, S. Sankaran, P. Perdikaris, Respecting causality is all you need for training physics-informed neural networks, 2022, arXiv preprint arXiv:2203.07404.

[4] L. McClenny, U. Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, 2020, arXiv preprint arXiv:2009.04544.

[5] V.A. Es' kin, D.V. Davydov, E.D. Egorova, A.O. Malkhanov, M.A. Akhukov, M.E. Smorkalov, About optimal loss function for training physics-informed neural networks under respecting causality, 2023, arXiv preprint arXiv:2304.02282.

[6] A. Kopaničáková, H. Kothari, G.E. Karniadakis, R. Krause, Enhancing training of physics-informed neural networks using domain-decomposition based preconditioning strategies, 2023, arXiv preprint arXiv:2306.17648.

[7] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.

[8] T. Salimans, D.P. Kingma, Weight normalization: A simple reparameterization to accelerate training of deep neural networks, in: Advances in Neural Information Processing Systems, vol. 29, 2016.

[9] W. Guan, K. Yang, Y. Chen, Z. Guan, A dimension-augmented physics-informed neural network (DaPINN) with high level accuracy and efficiency, 2022, arXiv preprint arXiv:2210.13212.

[10] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 384 (2021) 113938.

[11] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, M.W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, Adv. Neural Inf. Process. Syst. 34 (2021) 26548–26560.

[12] C.L. Wight, J. Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, 2020, arXiv preprint arXiv:2007.04542.

[13] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.

[14] X. Jin, S. Cai, H. Li, G.E. Karniadakis, Nsfnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951.

[15] S. Basir, Investigating and mitigating failure modes in physics-informed neural networks (PINNs), 2022, arXiv preprint arXiv:2209.09988.

[16] N. Sukumar, A. Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, Comput. Methods Appl. Mech. Engrg. 389 (2022) 114333.

[17] C. Leake, D. Mortari, Deep theory of functional connections: A new method for estimating the solutions of partial differential equations, Mach. Learn. Knowl. Extr. 2 (1) (2020) 37–55.

[18] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S.G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM J. Sci. Comput. 43 (6) (2021) B1105–B1132.

[19] 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.

[20] 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. Engrg. 403 (2023) 115671.

[21] B. Zapf, J. Haubner, M. Kuchta, G. Ringstad, P.K. Eide, K.-A. Mardal, Investigating molecular transport in the human brain from MRI with physics-informed neural networks, Sci. Rep. 12 (1) (2022) 15475.

[22] Z. Xiang, W. Peng, X. Liu, W. Yao, Self-adaptive loss balanced physics-informed neural networks, Neurocomputing 496 (2022) 11–34.

[23] D. Liu, Y. Wang, A dual-dimer method for training physics-constrained neural networks with minimax architecture, Neural Netw. 136 (2021) 112–125.

[24] G. Zhang, H. Yang, F. Zhu, Y. Chen, et al., Dasa-pinns: Differentiable adversarial self-adaptive pointwise weighting scheme for physics-informed neural networks, 2023, SSRN.

[25] S. Basir, I. Senocak, Physics and equality constrained artificial neural networks: Application to forward and inverse problems with multi-fidelity data fusion, J. Comput. Phys. 463 (2022) 111301.

[26] S. Basir, I. Senocak, An adaptive augmented Lagrangian method for training physics and equality constrained artificial neural networks, 2023, arXiv preprint arXiv:2306.04904.

[27] H. Son, S.W. Cho, H.J. Hwang, Enhanced physics-informed neural networks with augmented Lagrangian relaxation method (AL-PINNs), Neurocomputing (2023) 126424.

[28] K.A. Boster, S. Cai, A. Ladrón-de Guevara, J. Sun, X. Zheng, T. Du, J.H. Thomas, M. Nedergaard, G.E. Karniadakis, D.H. Kelley, Artificial intelligence velocimetry reveals in vivo flow rates, pressure gradients, and shear stresses in murine perivascular flows, Proc. Natl. Acad. Sci. 120 (14) (2023) e2217744120.

[29] Q. Zhang, C. Wu, A. Kahana, Y. Kim, Y. Li, G.E. Karniadakis, P. Panda, Artificial to spiking neural networks conversion for scientific machine learning, 2023, arXiv preprint arXiv:2308.16372.

[30] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[31] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, vol. 30, 2017.

[33] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030.

[34] S. Dong, N. Ni, A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks, J. Comput. Phys. 435 (2021) 110242.

[35] J. Chen, R. Du, K. Wu, A comparison study of deep Galerkin method and deep ritz method for elliptic problems with different boundary conditions, 2020, arXiv preprint arXiv:2005.04554.

[36] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Comput. Methods Appl. Mech. Engrg. 390 (2022) 114474.

[37] J.D. Toscano, C. Zuniga-Navarrete, W.D.J. Siu, L.J. Segura, H. Sun, Teeth mold point cloud completion via data augmentation and hybrid RL-GAN, J. Comput. Inf. Sci. Eng. 23 (4) (2023) 041008.