Computer Methods in Applied Mechanics and Engineering 452 (2026) 118723

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

PACMANN: Point adaptive collocation method for artificial neural networks

Coen Visser [a], Alexander Heinlein [b,*], Bianca Giovanardi [a,c]

[a] Faculty of Aerospace Engineering, Delft University of Technology, 2629 HS Delft, The Netherlands
[b] Delft Institute of Applied Mathematics, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands
[c] Delft Institute for Computational Science and Engineering, Delft University of Technology, 2628 CD Delft, The Netherlands

ARTICLE INFO

Keywords:
Differential equations
Physics-informed neural networks (PINNs)
Adaptive sampling
Collocation points
Residual gradient

ABSTRACT

Physics-Informed Neural Networks (PINNs) have emerged as a tool for approximating the solution of Partial Differential Equations (PDEs) in both forward and inverse problems. PINNs minimize a loss function which includes the PDE residual determined for a set of collocation points. Previous work has shown that the number and distribution of these collocation points have a significant influence on the accuracy of the PINN solution. Therefore, the effective placement of these collocation points is an active area of research. Specifically, available adaptive collocation point sampling methods have been reported to scale poorly in terms of computational cost when applied to high-dimensional problems. In this work, we address this issue and present the Point Adaptive Collocation Method for Artificial Neural Networks (PACMANN). PACMANN incrementally moves collocation points toward regions of higher residuals using gradient-based optimization algorithms guided by the gradient of the PINN loss function, that is, the squared PDE residual. We apply PACMANN to several forward and inverse problems, including one with a low-regularity solution and 3D Navier Stokes, and demonstrate that this method matches the performance of state-of-the-art methods in terms of the accuracy/efficiency tradeoff for the low-dimensional problems, while outperforming available approaches for high-dimensional problems. Key features of the method include its low computational cost and simplicity of integration into existing physics-informed neural network pipelines. The code is available at https://github.com/CoenVisser/PACMANN.

1. Introduction

Physics-Informed Neural Networks (PINNs) build upon the ability of deep neural networks to serve as universal function approximators, as established by Cybenko [1] and Hornik et al. [2] in 1989. Based on these findings, several methods were developed to solve Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) using neural networks, originally proposed by [3,4]. Supported by these developments and recent advances in computational tools, notably automatic differentiation in 2015 [5], Raissi et al. [6] proposed the name and framework of *Physics-Informed Neural Networks* and their use to approximate the solution of PDEs in both forward and inverse problems; their work was published in 2019. Since then, PINNs have been applied in a variety of fields [7,8], such as fluid dynamics [9–11], heat transfer [12,13], material sciences [14,15], and electromagnetism [16,17].

* Corresponding author.
E-mail address: A.Heinlein@tudelft.nl (A. Heinlein).

https://doi.org/10.1016/j.cma.2025.118723
Received 3 October 2025; Received in revised form 26 November 2025; Accepted 30 December 2025
Available online 14 January 2026

In their classic form, PINNs approximate the solution of differential equations by minimizing a loss function incorporating boundary conditions, initial conditions, and the PDE residual sampled over a set of collocation points. In 2020, Mao et al. [9] explored the impact of collocation point placement on prediction accuracy for solutions exhibiting discontinuities. They demonstrated that, when discontinuities are known a priori, manually increasing the density of points near these regions improves prediction accuracy. To address scenarios where solution features are unknown before training, adaptive algorithms for collocation point selection were developed. For instance, Lu et al. [18] introduced Residual-based Adaptive Refinement (RAR) in 2021, the first adaptive sampling algorithm, which places additional points in regions with the largest PDE residuals. RAR improves prediction accuracy, capturing features such as a discontinuity better than a static grid for the Burgers' equation. Instead of sampling points only where the residual is the largest, Nabian et al. [19] proposed to randomly resample all points in the domain based on a Probability Density Function (PDF) proportional to the loss function. This approach samples a higher density of points in high residual areas, resulting in accelerated convergence of the PINN. Based upon these studies, Wu et al. [20] presented two additional sampling algorithms, Residual-based Adaptive Distribution (RAD) and Residual-based Adaptive Refinement with Distribution (RAR-D). In RAD, all collocation points are resampled using a PDF defined by a nonlinear function of the PDE residual. RAR-D is a combination of RAR and RAD, where collocation points are sampled in addition to the existing ones according to the same probability density function used by RAD. Both approaches lead to a higher prediction accuracy, specifically for PDEs with complex solutions due to, for example, steep gradients. Moreover, RAD was found to outperform the method proposed by Nabian et al. [19].

Several adaptations to the aforementioned PDF-based sampling algorithms have been proposed. Guo et al. [21], for example, propose an adaptive causal sampling method, which decomposes the domain into subdomains where the ratio of points sampled in each subdomain is based on the PDE residual and a temporal weight, ensuring temporal causality. This approach was found to enhance the prediction accuracy and computational efficiency of PINNs in problems with nonlinear PDEs containing higher-order derivatives. Hou et al. [22] propose an adaptive collocation method that samples points according to a PDF proportional to the PDE residual or gradient of the solution and further combine this with an adaptive loss weighting strategy. Numerical experiments demonstrate that the resulting method improves prediction accuracy and generalization ability over vanilla PINNs. Furthermore, Mao et al. [23] consider the gradient of the solution by sampling additional points in subdomains with large residuals and large solution gradients. Liu et al. [24] propose to add points with large residual gradients to the set of collocation points used for training. Both Mao et al. and Liu et al. report an improvement in accuracy for problems with solutions exhibiting steep gradients.

While the aforementioned collocation point sampling methods have proven effective in low-dimensional problems, these approaches to resampling are computationally expensive for high-dimensional problems, as reported by Wu et al. [20]. Specifically, RAD or RAR are computationally expensive for these problems due to the cost of evaluating the residual at a sufficiently large number of points, either to construct the probability density function or to identify additional points for inclusion in the training process. Other approaches have been proposed to sample collocation points in high-dimensional problems. For instance, Tang et al. [25] propose the DAS-PINNs approach, which samples according to a Deep Adaptive Sampling (DAS) method and uses KRnet [26], a deep generative model, to approximate the PDF proportional to the residual. Similarly, Gao et al. [27] also use a dual-network approach in their EEMS-PINNs (Energy-Equidistributed Moving Sampling-PINNs) framework. This adaptive method relocates collocation points in conservative PDE problems by training a second neural network to solve a moving-mesh PDE where the energy density serves as the monitor function. They report that this method improves solution accuracy and stability in long-time simulations. However, these approaches are not straightforward to integrate into existing PINNs pipelines due to their dual-network frameworks.

In this work, we present a collocation point resampling method that scales to higher dimensions more efficiently without introducing significant computational overhead while maintaining the accuracy improvements achieved by previous approaches. We propose the Point Adaptive Collocation Method for Artificial Neural Networks (PACMANN), which uses the gradient of the squared residual to move collocation points toward areas with a large residual. In this approach, collocation point resampling is formulated as a maximization problem of the squared residual. First, the PINN is trained on a static grid of collocation points. After a certain number of iterations, this process is paused and the gradient of the squared residual is determined for the input coordinates of each collocation point. Based on the magnitude and direction provided by these gradients, points are moved to maximize the squared residual using established optimization methods. Since the residual landscape is static while training is paused, the process of moving points may be repeated iteratively. PACMANN includes four main hyperparameters: the resampling period, the optimizer for moving the collocation points, the stepsize, and the number of steps taken by the optimization algorithm. Key features of the method include its low computational cost and simplicity of integration in existing physics-informed neural network pipelines. Our approach builds on the work of Wang et al. [28], who, independently of the aforementioned developments, found that iteratively updating the placement of collocation points by applying gradient ascent over the L^{∞} physics-informed loss results in a greater prediction accuracy for the Hamilton-Jacobi-Bellman equation.

First, we investigate the performance of PACMANN in combination with a variety of optimization algorithms for two low-dimensional problems: the one-dimensional Burgers' and Allen-Cahn equations. We then perform sensitivity studies on the number of collocation points and the method's hyperparameters. In addition, we demonstrate the suitability of PACMANN for high-dimensional and inverse problems. As test cases, we consider an inverse problem based on the 2D Navier-Stokes equations, the Poisson's equation in five dimensions, and the 3D Navier-Stokes equations. Finally, we apply PACMANN to two geometrically challenging problems involving a re-entrant corner in a disk and a plate with holes, showing the effectiveness of our approach for problems with low-regularity solutions and non-convex domains. For all problems under consideration, we compare the performance of our method in terms of prediction accuracy and computational cost to state-of-the-art adaptive and non-adaptive sampling methods. Notably, our results show that our method matches the performance of state-of-the-art methods in terms of the accuracy/efficiency tradeoff for low-dimensional problems while efficiently scaling to high-dimensional problems, where it outperforms state-of-the-art methods.

This paper is organized as follows: In Section 2, we briefly review the PINNs framework, followed by a description of PACMANN. Next, in experimental studies in Section 3, we compare the accuracy and computational cost of PACMANN to other state-of-the-art sampling methods for five forward problems and an inverse problem. Finally, in Section 4, we summarize our findings.

2. Methodology

This section begins with a brief review of PINNs based on the framework presented by Raissi et al. [6] in 2019. Next, we propose the novel PACMANN.

2.1. Physics-informed neural networks (PINNs)

PINNs approximate the solution of PDEs using neural networks. Generally, we consider PDEs of the form: find $\boldsymbol{u}$ such that

$$\boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}] = 0, \quad \boldsymbol{x} \in \boldsymbol{\Omega}, \quad t \in [0, T], \tag{1}$$

with the initial and boundary conditions

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{x}, 0) &= h(\boldsymbol{x}), & \boldsymbol{x} &\in \boldsymbol{\Omega}, \\ \mathcal{B}[\boldsymbol{u}](\boldsymbol{x}, t) &= 0, & \boldsymbol{x} &\in \partial\boldsymbol{\Omega}, \quad t \in [0, T], \end{aligned}$$

where $\mathcal{N}[\cdot]$ is a linear or nonlinear differential operator, and $\mathcal{B}[\cdot]$ is a boundary operator corresponding to a set of boundary conditions. In addition, $\boldsymbol{x} \in \boldsymbol{\Omega} \subset \mathbb{R}^d$ and $t \in [0, T]$ denote the spatial and temporal coordinates, respectively, and we write $\partial\boldsymbol{\Omega}$ for the boundary of $\boldsymbol{\Omega}$. We denote the space-time domain by $\mathcal{D} := \overline{\boldsymbol{\Omega}} \times [0, T]$.

The PINN consists of a (deep) neural network with the coordinates $(\boldsymbol{x}, t)$ as inputs and $\hat{\boldsymbol{u}}(\boldsymbol{x}, t, \boldsymbol{\theta})$ as output, approximating $\boldsymbol{u}(\boldsymbol{x}, t)$. The trainable parameters $\boldsymbol{\theta}$ of this neural network are trained by minimizing a specific loss function $\mathcal{L}(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{2}$$

The loss function is defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_r \mathcal{L}_r(\boldsymbol{\theta}) + \lambda_{ic} \mathcal{L}_{ic}(\boldsymbol{\theta}) + \lambda_{bc} \mathcal{L}_{bc}(\boldsymbol{\theta}), \tag{3}$$

where

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left(\hat{\boldsymbol{u}}_t\left(\boldsymbol{x}_r^i, t_r^i, \boldsymbol{\theta}\right) + \mathcal{N}[\hat{\boldsymbol{u}}]\left(\boldsymbol{x}_r^i, t_r^i, \boldsymbol{\theta}\right) \right)^2, \tag{4}$$

$$\mathcal{L}_{ic}(\boldsymbol{\theta}) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \left(\hat{\boldsymbol{u}}\left(\boldsymbol{x}_{ic}^i, 0, \boldsymbol{\theta}\right) - h\left(\boldsymbol{x}_{ic}^i\right) \right)^2, \tag{5}$$

$$\mathcal{L}_{bc}(\boldsymbol{\theta}) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \left(\mathcal{B}[\hat{\boldsymbol{u}}]\left(\boldsymbol{x}_{bc}^i, t_{bc}^i, \boldsymbol{\theta}\right) \right)^2 \tag{6}$$

represent the loss terms for the PDE residual, initial conditions, and the boundary conditions, respectively. Furthermore, N_r, N_{ic}, and N_{bc} denote the numbers of collocation points of the aforementioned terms. The hyperparameters λ_r, λ_{ic}, and λ_{bc} are scalar weights used to balance the loss function. Each loss term is evaluated over a set of data points, where $\{(\boldsymbol{x}_r^i, t_r^i)\}_{i=1}^{N_r}$ is a set of collocation points located in the interior of the domain, $\{(\boldsymbol{x}_{ic}^i, 0)\}_{i=1}^{N_{ic}}$ is a set of points sampled at the initial time, and $\{(\boldsymbol{x}_{bc}^i, t_{bc}^i)\}_{i=1}^{N_{bc}}$ is a set sampled along the boundary. These points may be fixed during training [20], resampled through periodic random resampling [20], or resampled using adaptive sampling methods based on guiding information, such as the PDE residual [18–20]. Note that we assume sufficient regularity, existence of a strong-form solution of Eq. (1), for the PINN loss function to be meaningful.

To train the model parameters $\boldsymbol{\theta}$, the gradient of the loss function with respect to the parameters is determined using back-propagation [29]. Next, the model parameters are updated with an optimization algorithm, often based on the gradient descent method, such as the Adam optimizer [30]. Similarly, the derivatives of $\hat{\boldsymbol{u}}(\boldsymbol{x}, t, \boldsymbol{\theta})$ with respect to the input coordinates $(\boldsymbol{x}, t)$ as required by $\mathcal{N}[\cdot]$ and potentially $\mathcal{B}[\cdot]$ in Eqs. (4) and (6) are computed using automatic differentiation; the initial conditions and the corresponding loss function Eq. (5) may also depend on the temporal derivative of $\hat{\boldsymbol{u}}$, but we omit these cases for simplicity.

For problems that incorporate reference data during training, such as inverse problems, an additional loss term $\mathcal{L}_{ref}$ is added to the loss function described by Eq. (3), where

$$\mathcal{L}_{ref}(\boldsymbol{\theta}) = \frac{1}{N_{ref}} \sum_{i=1}^{N_{ref}} \left(\hat{\boldsymbol{u}}\left(\boldsymbol{x}^i, t^i, \boldsymbol{\theta}\right) - \boldsymbol{u}_{ref}\left(\boldsymbol{x}^i, t^i\right) \right)^2.$$

This term corresponds to the mean squared error between the (noisy) observed data $\boldsymbol{u}_{ref}$ at the set of data points $\{(\boldsymbol{x}^i, t^i)\}_{i=1}^{N_{ref}}$ and the approximation $\hat{\boldsymbol{u}}\left(\boldsymbol{x}^i, t^i, \boldsymbol{\theta}\right)$ given by the neural network. Adding this term leads to the following loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_r \mathcal{L}_r(\boldsymbol{\theta}) + \lambda_{ic} \mathcal{L}_{ic}(\boldsymbol{\theta}) + \lambda_{bc} \mathcal{L}_{bc}(\boldsymbol{\theta}) + \lambda_{ref} \mathcal{L}_{ref}(\boldsymbol{\theta}),$$

where λ_{ref} is the additional scalar weight assigned to the reference data loss term.

Algorithm 1: PACMANN with a given optimization algorithm, P, s, and T.

```
1 Sample a set $\boldsymbol{X_r}$ of $N_r$ collocation points $\{\boldsymbol{x}_r^i, t_r^i\}_{i=1}^{N_r}$ using a uniform sampling method;
2 repeat
3     Train the PINN for $P$ iterations;
4     Determine $r^2(\boldsymbol{x}, t) = \left(\boldsymbol{u_t}(\boldsymbol{x}_r^i, t_r^i, \boldsymbol{\theta}) + \mathcal{N}[\boldsymbol{u}](\boldsymbol{x}_r^i, t_r^i, \boldsymbol{\theta})\right)^2$, the squared PDE residual over $\boldsymbol{X_r}$;
5     Find $\nabla_{\boldsymbol{x}} r^2(\boldsymbol{x}, t)$ and $\frac{\partial}{\partial t} r^2(\boldsymbol{x}, t)$, the gradients of the squared PDE residual with respect to the input coordinates of the
        points in $\boldsymbol{X}_r$;
6     Iteratively move the points in $\boldsymbol{X}_r$ according to the chosen optimization algorithm with stepsize $s$ and number of steps $T$;
7     Replace points in $\boldsymbol{X}_r$ outside the domain with points sampled according to a uniform probability distribution;
8 until the total number of iterations reaches the limit;
```

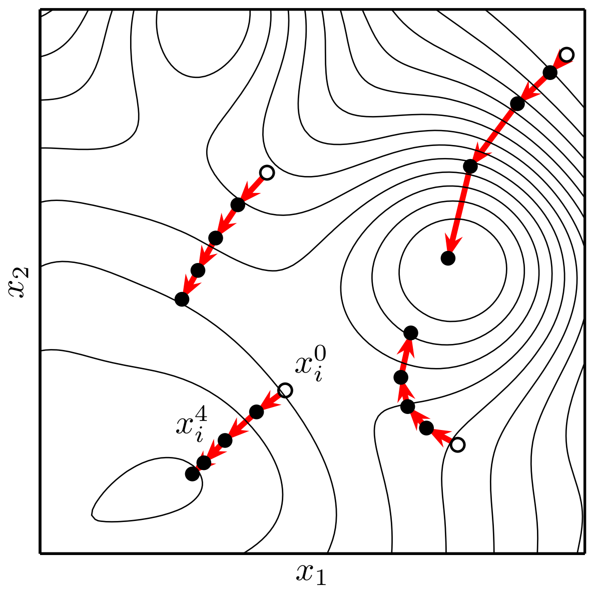


Fig. 1. A schematic of PACMANN with four steps of gradient ascent on a contour plot of the squared residual.

2.2. Point adaptive collocation method for artificial neural networks (PACMANN)

In this work, we propose the *Point Adaptive Collocation Method for Artificial Neural Networks*, which uses the gradient of the squared residual as guiding information to gradually move collocation points toward areas of high residuals; see Algorithm 1 and Fig. 1. In particular, instead of the minimization problem in Eq. (2), we consider the following min-max problem for training the model parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} \left[\lambda_{ic}\mathcal{L}_{ic}(\boldsymbol{\theta}) + \lambda_{bc}\mathcal{L}_{bc}(\boldsymbol{\theta}) + \lambda_{ref}\mathcal{L}_{ref}(\boldsymbol{\theta}) + \lambda_r \max_{\boldsymbol{X_r} \subset \mathcal{D}} \mathcal{L}_r\left(\boldsymbol{X_r}, \boldsymbol{\theta}\right) \right]. \tag{7}$$

Here, only the collocation points $X_r = \{(\boldsymbol{x}_r^i, t_r^i)\}_{i=1}^{N_r}$ are moved, while points such as those sampled along the boundary are fixed in place throughout training. This approach ensures that a sufficient number of points are placed along the initial and boundary conditions to accurately compute the respective loss terms.

Remark 1. The min-max formulation for the residual loss $\mathcal{L}_r$ in Eq. (7) can also be interpreted as optimizing the supremum norm in $\mathcal{D}$. In our numerical experiments, the neural network function will be smooth, due to the use of the hyperbolic activation function, such that the maximum exists in $\mathcal{D}$. Then, we obviously have that

$$\begin{aligned}\|\hat{\boldsymbol{u}}_t + \mathcal{N}[\hat{\boldsymbol{u}}]\|^2_{L^\infty(\mathcal{D})} &= \max_{(\boldsymbol{x}_r, t_r)\in\mathcal{D}} \left(\hat{\boldsymbol{u}}_t\left(\boldsymbol{x}_r, t_r\right) + \mathcal{N}[\hat{\boldsymbol{u}}]\left(\boldsymbol{x}_r, t_r\right)\right)^2 = \frac{1}{N_r}\sum_{i=1}^{N_r} \max_{(\boldsymbol{x}_r, t_r)\in\mathcal{D}} \left(\hat{\boldsymbol{u}}_t\left(\boldsymbol{x}_r, t_r\right) + \mathcal{N}[\hat{\boldsymbol{u}}]\left(\boldsymbol{x}_r, t_r\right)\right)^2 \\ &= \max_{\boldsymbol{X}_r\subset\mathcal{D}} \left[\frac{1}{N_r}\sum_{i=1}^{N_r} \left(\hat{\boldsymbol{u}}_t\left(\boldsymbol{x}_r^i, t_r^i\right) + \mathcal{N}[\hat{\boldsymbol{u}}]\left(\boldsymbol{x}_r^i, t_r^i\right)\right)^2 \right] = \max_{\boldsymbol{X}_r\subset\mathcal{D}} \mathcal{L}_r\left(\boldsymbol{X_r}, \boldsymbol{\theta}\right),\end{aligned}$$

where we have omitted the dependence on the neural network parameters θ for the sake of brevity. In practice, since we never actually attain the maximum, the loss will be significantly lower than the supremum norm, due to the averaging across $\mathcal{D}$.

In the PACMANN algorithm, N_r collocation points are first sampled using a uniform sampling method, such as an equispaced uniform grid or the Hammersley sequence [31, pp. 31–36]. The PINN is then trained on this set of collocation points for a number of P iterations. The number P is a hyperparameter of the method determining the *resampling period*, that is, the period after which the collocation points are resampled. Next, the training iteration is paused and the gradients of the squared residual $r^2(\boldsymbol{x}, t)$ with respect to the input coordinates, given by $\nabla_{\boldsymbol{x}} r^2(\boldsymbol{x}, t)$ and $\frac{\partial}{\partial t} r^2(\boldsymbol{x}, t)$, are determined for each collocation point $(\boldsymbol{x}, t) \in \boldsymbol{X}_r$. The collocation points are subsequently moved in the direction of increasing residual based on

$$\begin{cases} \boldsymbol{x}_r^{i+1} = \boldsymbol{x}_r^i + s\nabla_{\boldsymbol{x}} r^2(\boldsymbol{x}_r^i, t_r^i), \\ t_r^{i+1} = t_r^i + s\dfrac{\partial}{\partial t} r^2(\boldsymbol{x}_r^i, t_r^i). \end{cases} \tag{8}$$

Here, s is a hyperparameter that determines the stepsize of each move; in the machine learning community, this parameter is also often called a *learning rate*. Since the neural network parameters θ are kept constant during the iteration in Eq. (8), the residual landscape is static and the residual gradient can be determined again for the new location of the collocation point, allowing the process to be repeated. Therefore, collocation points can be moved several times, given by the hyperparameter *number of steps* T. If a point moves outside the domain, it is removed from the set $\boldsymbol{X}_r$ and a replacement point is sampled in the domain based on a uniform probability distribution. Next, the neural network is trained for another P iterations, after which the process of moving collocation points for T iterations is repeated.

In Eq. (8), points are moved directly based on the gradient vector. This approach is essentially equal to applying gradient ascent. However, other gradient-based optimization algorithms can be applied. The optimization algorithms considered for PACMANN in Section 3 are listed as follows. We also provide the iteration rule for an arbitrary variable x and a function $f(x)$ to be maximized. Other optimization algorithms are also applicable, but we focus on the following algorithms in our numerical experiments in Section 3.

1. **Gradient ascent:** The collocation points are directly moved in the direction of steepest ascent, that is, in the direction of the gradient. This algorithm updates variables using the formula:

   $$x_{i+1} = x_i + sf'(x_i).$$

2. **Nonlinear gradient ascent:** In this algorithm, we apply a nonlinear function to the gradient ascent algorithm to scale down large gradients, preventing points from taking large steps directly out of the domain. We refer to the algorithm as nonlinear gradient ascent. In this work, we use the hyperbolic tangent function, as follows:

   $$x_{i+1} = x_i + s\tanh\left(f'(x_i)\right).$$

3. **RMSprop:** Root Mean Square Propagation (RMSprop) [32] adapts the stepsize by dividing the gradient by a weighted average of previous gradients; this serves to stabilize convergence. The algorithm consists of two steps. First, a parameter S is updated. This parameter consists of a weighted average of previous gradients:

   $$S_{i+1} = \beta S_i + (1-\beta)\left(f'(x_i)\right)^2.$$

   Next, the variable x is updated using:

   $$x_{i+1} = x_i + s\frac{f'(x_i)}{\sqrt{S_{i+1} + \epsilon}}.$$

   To prevent large steps due to small values of S, a small value is added, represented by ϵ.
4. **Momentum:** The momentum optimizer [33] considers a weighted average of previous gradients at each iteration to prevent converging to local minima. First, the weighted average, V, is computed:

   $$V_{i+1} = \beta V_i + (1-\beta)\left(f'(x_i)\right).$$

   After updating V, the variable x is updated:

   $$x_{i+1} = x_i + sV_{i+1}.$$

5. **Adam:** The Adaptive moments (Adam) [30] optimizer combines the concepts behind RMSprop and momentum. First, the value of V is updated:

   $$V_{i+1} = \beta_1 V_i + (1-\beta_1)f'(x_i).$$

   Next, the parameter S is updated:

   $$S_{i+1} = \beta_2 S_i + (1-\beta_2)\left(f'(x_i)\right)^2.$$

   Afterwards, an initialization bias correction is applied to V_{i+1} and S_{i+1}:

   $$\hat{V}_{i+1} = \frac{V_{i+1}}{1-\beta_1^{i+1}}, \quad \hat{S}_{i+1} = \frac{S_{i+1}}{1-\beta_2^{i+1}}.$$

Finally, the variable x is updated as follows:

$$x_{i+1} = x_i + s\frac{\hat{V}_{i+1}}{\sqrt{\hat{S}_{i+1} + \epsilon}}$$

Here, a small regularization parameter ϵ is included to prevent large steps when $\hat{S}_{i+1}$ is small.

6. **Golden section search:** Golden section search [34, pp. 39–42] is a line search method that narrows down the search interval each iteration. By searching along the direction of steepest ascent, the multidimensional optimization problem is reduced to a one-dimensional problem. In this direction, the algorithm searches in an interval $[a_i, b_i]$. In the initial interval, a_0 is equal to the value x_0, for example. We determine b_0 using the stepsize and the gradient:

$$b_0 = a_0 + sf'(x_0).$$

Next, $f(x)$ is evaluated at two points, $x_{l,i}$ and $x_{r,i}$, determined using

$$x_{l,i} = a_i + \alpha(b_i - a_i), \quad x_{r,i} = a_i + \beta(b_i - a_i).$$

The name “golden section search” refers to the golden ratio, defined as $\phi = \frac{1+\sqrt{5}}{2}$, which is incorporated in the values of α and β:

$$\alpha = 1 - \phi^{-1}, \quad \beta = \phi^{-1}$$

If $f(x_{l,i}) > f(x_{r,i})$, then the interval is shortened and shifted to the left:

$$a_{i+1} = a_i, \quad b_{i+1} = x_{r,i}, \quad x_{r,i+1} = x_{l,i}.$$

Otherwise, if $f(x_{l,i}) < f(x_{r,i})$, then the interval is shortened and shifted to the right:

$$a_{i+1} = x_{l,i}, \quad b_{i+1} = b_i, \quad x_{l,i+1} = x_{r,i}.$$

After updating the interval to $[a_{i+1}, b_{i+1}]$, the algorithm is repeated again. After the final iteration, the value of the variable x is found by taking the middle point of the interval $[a_N, b_N]$:

$$x_N = \frac{a_N + b_N}{2}.$$

The golden ratio ensures that either $x_{l,i+1}$ ends up on $x_{r,i}$ or $x_{r,i+1}$ on $x_{l,i}$, depending on the direction of the interval shift. Since $f(x)$ has already been determined for $x_{l,i}$ and $x_{r,i}$ in the previous iteration, $f(x)$ does not have to be evaluated again for these variable values. As a result, for each iteration, the value of $f(x)$ only has to be evaluated once, which is beneficial in terms of the computational cost.

3. Results

In this section, we evaluate the performance of PACMANN in terms of accuracy and computational cost across various PDE examples, including simple model problems, an inverse problem, a problem defined on a re-entrant corner domain, and two high-dimensional problems. Furthermore, we vary the hyperparameters of PACMANN to showcase its behavior and compare our method with other collocation point sampling methods proposed in the related works [18,20].

To gather data on the prediction accuracy and computational cost, each method and set of hyperparameters is run ten times with varying random seeds. The prediction accuracy is compared based on the mean and standard deviation across the ten runs of the test error, measured using the L_2 relative error. The L_2 relative error is determined as follows:

$$\varepsilon_{L_2} := \frac{\|\boldsymbol{u}_{ref} - \boldsymbol{u}_{pred}\|_2}{\|\boldsymbol{u}_{ref}\|_2}, \tag{9}$$

where $\boldsymbol{u}_{ref}$ is the reference solution, which is either an analytical or numerical solution depending on the problem, and $\boldsymbol{u}_{pred}$ is the predicted solution. When an analytical solution is available, we additionally consider the prediction accuracy using the H^1 semi-norm

$$\varepsilon_{H^1} := \frac{|\boldsymbol{u}_{ref} - \boldsymbol{u}_{pred}|_1}{|\boldsymbol{u}_{ref}|_1}, \quad \text{where} \quad |v|_1 := \|\nabla v\|_2, \tag{10}$$

to measure errors in the first derivatives of the solution. To compare $\boldsymbol{u}_{ref}$ and $\boldsymbol{u}_{pred}$ we employ an equispaced uniform grid of 10 000 collocation points. The mean runtime of training over the ten runs serves as an indication of the computational cost of a particular sampling method. Our code is based on the PINNs library DeepXDE [18] using PyTorch [35] version 1.12.1 as the backend. It is publicly available on GitHub at https://github.com/CoenVisser/PACMANN. The models were trained using NVIDIA Tesla V100S GPUs on TU Delft's high-performance computer DelftBlue [36].

For all experiments, the training is split into five phases of 10 000 iterations, consisting of 7000 iterations of Adam with a learning rate of 10^{-3} followed by 3000 iterations of the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm [37], totaling 50 000 iterations. Throughout training, a resampling period P of 50 iterations is maintained. Furthermore, we only resample the collocation points when training the neural network parameters with Adam. Resampling while training with the L-BFGS optimizer would disrupt the convergence of the algorithm due to the change in loss landscape by evaluating the PDE loss term at different

Table 1
Hyperparameter settings for the RMSprop, momentum, and Adam optimizers.

Optimizer	β	V_0	S_0	ϵ
RMSprop	0.999	–	0	10^{-8}
Momentum	0.9	0	–	10^{-8}
Adam	$\beta_1 = 0.9$, $\beta_2 = 0.999$	0	0	10^{-8}

Table 2
Overview of the mean and standard deviation of the test error and the mean runtime for each sampling method for the Burgers' equation. The best result in each column is marked in boldface.

Sampling method	L_2 relative error		Mean runtime [s]
	Mean	1 SD	
Uniform grid	25.9%	14.2%	425
Hammersley grid	0.61%	0.53%	443
Random resampling	0.40%	0.35%	**423**
RAR	0.11%	**0.05%**	450
RAD	0.16%	0.10%	463
RAR-D	0.24%	0.21%	503
PACMANN-Adam	**0.07%**	**0.05%**	461

Table 3
Overview of the mean and standard deviation of the test error and the mean runtime achieved by PACMANN for the Burgers' equation with the optimization methods listed in Section 2. The best result in each column is marked in boldface. For each optimization method, we report the stepsize and number of steps that achieve the lowest test error.

PACMANN optimizer	L_2 relative error		Mean runtime [s]	Hyperparameters	
	Mean	1 SD		Stepsize s	No. of steps T
Gradient ascent	0.30%	0.17%	**436**	10^{-6}	1
Nonlinear gradient ascent	0.10%	0.06%	453	10^{-4}	5
RMSprop	0.10%	**0.03%**	442	10^{-6}	10
Momentum	0.18%	0.24%	448	10^{-6}	5
Adam	**0.07%**	0.05%	461	10^{-5}	15
Golden section search	0.34%	0.17%	460	10^{-7}	5

collocation points. In this work, in all experiments, the hyperbolic tangent is used as the activation function, and all weights λ_i of the individual loss terms are set to 1. In addition, the hyperparameter settings used for RMSprop, momentum, and Adam with PACMANN are given in Table 1. Note that we have not conducted an extensive study varying the neural network architecture. Instead, we have used the architectures provided in the corresponding test cases of DeepXDE, assuming that these were already optimized appropriately.

In the numerical experiments, PACMANN with Adam consistently achieves the lowest error compared to the other optimization algorithms considered. Therefore, to preserve clarity, figures and tables which compare the various sampling methods only contain the results for Adam with our method.

Infrequently, the random neural network weight initialization prevents the PINN from learning the solution, which results in a test error several orders of magnitude larger than the test error obtained with other weight initializations. This has been observed for all sampling methods considered in this study and is characterized by volatile loss behavior or large static loss terms. When this occurs, the corresponding training run is repeated with a different random seed.

3.1. 1D Burgers' equation

We first consider the one-dimensional Burgers' equation:

$$\begin{cases} u_t + uu_x = \nu u_{xx}, & x \in [-1,1], \quad t \in [0,1], \\ u(x,0) = -\sin(\pi x), & \\ u(-1,t) = u(1,t) = 0. & \end{cases} \tag{11}$$

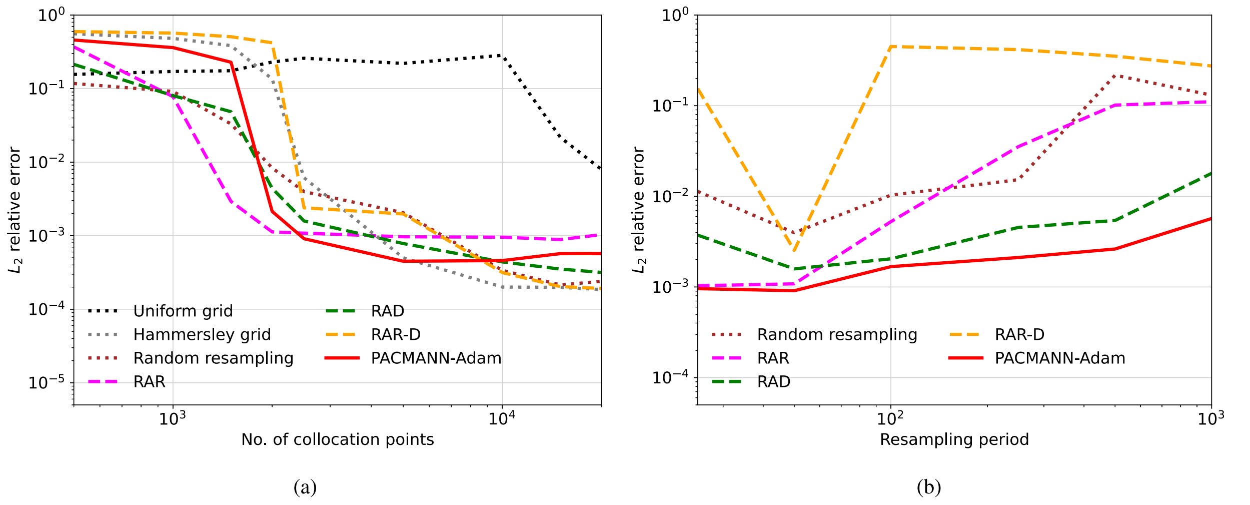


Fig. 2. Mean of the test error for each of the sampling methods for a varying (a) number of collocation points and (b) resampling period for the Burgers' equation example.

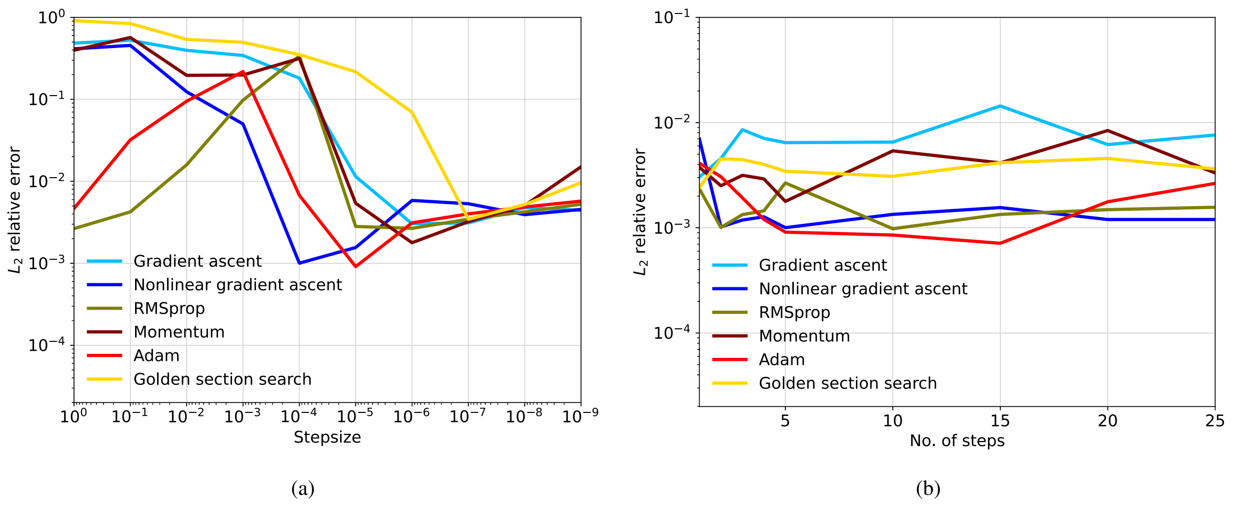


Fig. 3. Mean of the test error for PACMANN with the different optimization algorithms listed in Section 2 for a varying (a) stepsize and (b) number of steps for the Burgers' equation example.

Here, ν is the diffusion coefficient or kinematic viscosity, set at $\nu = 0.01/\pi$. For this problem, we employ 2500 collocation points, 80 boundary points, and 160 initial condition points. The neural network architecture used for this example consists of four hidden layers of 64 neurons. To compute the L_2 relative error, we compare the network prediction to a numerical solution generated using a spectral solver by Raissi et al. [6].

The mean and standard deviation of the test error and the mean runtime for each sampling method are given in Table 2. Out of the non-adaptive sampling methods tested, only the static uniform grid is unable to capture the solution satisfactorily, resulting in a high test error of 25.9%. The other non-adaptive methods, the static Hammersley grid and random resampling, attain a significantly lower test error. Overall, our method, in combination with the Adam optimizer and a stepsize of 10^{-5}, achieves the lowest test error. It achieves a lower error than the next-best sampling method, RAR, at a slightly higher computational cost.

Table 3 compares the performance of the various optimization algorithms for PACMANN in terms of accuracy and efficiency. We note that the nonlinear gradient ascent and the RMSprop optimizers with our proposed method both achieve a competitive test error and computational cost compared to the second best approach in Table 2, that is, RAR.

Next, we test the behavior of the different sampling methods by varying the number of collocation points from 500 to 20 000; see Fig. 2(a). We observe that RAR initially reduces the test error the fastest, but plateaus at a higher error than the other methods under consideration. PACMANN in combination with the Adam optimizer significantly improves the prediction accuracy as the

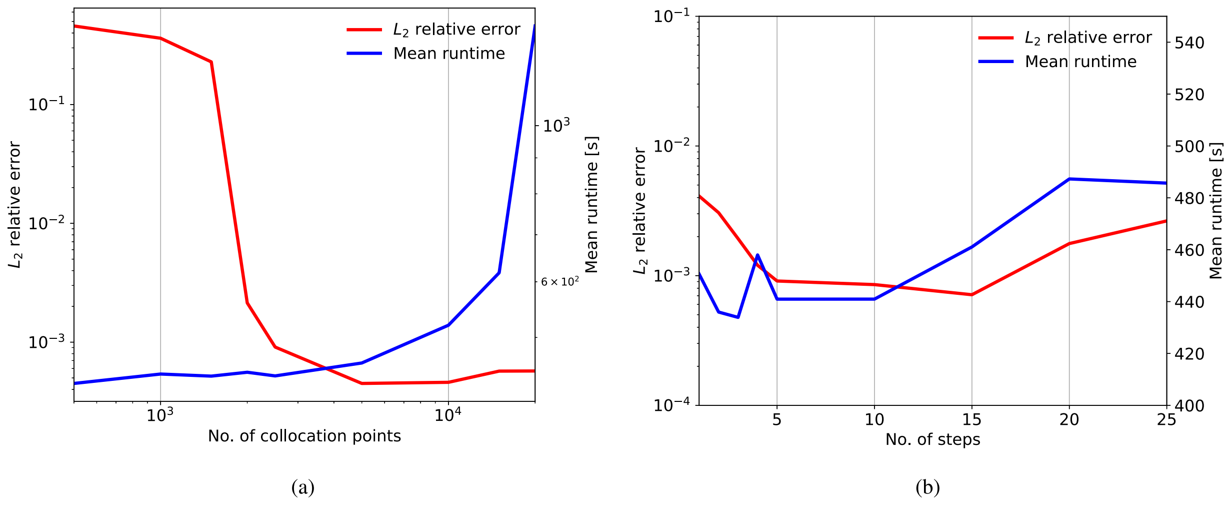


Fig. 4. Mean of the test error and the runtime for PACMANN with Adam for a varying (a) number of collocation points and (b) number of steps for the Burgers' equation example.

number of collocation points is increased from 1500 to 2000, after which it slowly increases the prediction accuracy. Only at a large number of collocation points (10 000) are the adaptive sampling methods slightly outperformed by the static Hammersley grid.

Fig. 2(b) depicts the behavior of the various sampling methods as the resampling period is increased from 25 to 1000. Generally, most sampling methods lose accuracy as the period is increased. Notably, RAR-D performs significantly better at a period of 50 iterations compared to other resampling periods. PACMANN performs best for all resampling periods considered, losing accuracy slower than the other sampling methods.

In the following, we compare the behavior of the optimizers listed in Section 2 when varying the hyperparameters stepsize s and number of steps T. First, in Fig. 3(a), we test the accuracy of the optimizers for different stepsizes s ranging from 1 to 10^{-9}. This figure demonstrates that the stepsize has a significant influence on the prediction accuracy achieved by PACMANN. We note that, depending on the optimizer used, a different stepsize is optimal, such as 10^{-6} for momentum or 10^{-5} for Adam. Furthermore, the behavior of the optimizers at stepsizes near 1 is split into two groups, with Adam and RMSprop gaining accuracy as the stepsize is increased, while others continue to lose accuracy. This phenomenon is explained by the number of collocation points that leave the domain while points are moved by PACMANN. At these large stepsizes, all 2500 collocation points exit the domain when using Adam or RMSprop. Since PACMANN uses a uniform probability distribution to determine the location of the replacement collocation points, the test error approaches the accuracy of random resampling ($0.40\% \pm 0.35\%$); cf. Table 2. In contrast, when applying the other optimization algorithms with these stepsizes, only a small portion of the collocation points exit the domain, of order $\mathcal{O}(10)$. These few points are not sufficient for the random resampling to significantly affect the test error achieved. The difference in the number of points that exit the domain may be explained by the relatively large optimal stepsize and number of steps for RMSprop and Adam compared to the other optimization algorithms.

Importantly, in Fig. 7 we show that the loss during training serves as an indicator for selecting an appropriate stepsize. PACMANN with Adam and a stepsize of 10^{-5} achieves the lowest loss during training, followed by the uniform grid baseline, whereas a stepsize of 10^{-3} exhibits the highest loss. This ranking follows the corresponding L_2 relative error reported for these stepsizes in Fig. 3(a).

We test the influence of the number of steps T by ranging it from 1 to 25. Fig. 3(b) shows that certain optimizers benefit from more steps, such as nonlinear gradient ascent and Adam. Others remain at a near-constant accuracy or lose accuracy with additional steps. Based on Fig. 3(b), we note that the number of steps generally has a smaller impact on the test error achieved compared to the stepsize.

Furthermore, Fig. 4(a) and (b) depict the prediction accuracy and computational cost of PACMANN with the Adam optimizer for varying numbers of collocation points and steps. Fig. 4(a) demonstrates that increasing the number of collocation points reduces the L_2 relative error before reaching a plateau, beyond which the computational cost rises steeply without a further increase in accuracy. Importantly, we point out that the accuracy of Adam with five steps is nearly the same as its accuracy at 15 steps, see Fig. 4(b). Thus, we recommend taking fewer steps to save on computational cost in more complex problems.

Next, we compare visually the distribution of collocation points after training. Fig. 5(a) shows the locations before training when the collocation points are laid out based on the Hammersley sequence. Fig. 5(b) to (d) show the locations of the collocation points after training for RAR, RAD, and PACMANN with Adam, respectively. While RAR clusters the points at the steepest region of the solution, RAD and our method tend to create several smaller clusters. In contrast to RAD, our method also forms clusters of points in regions with typically lower residuals, see Fig. 6(a), indicative of local maxima of the squared residual. Fig. 6(b) shows that, after

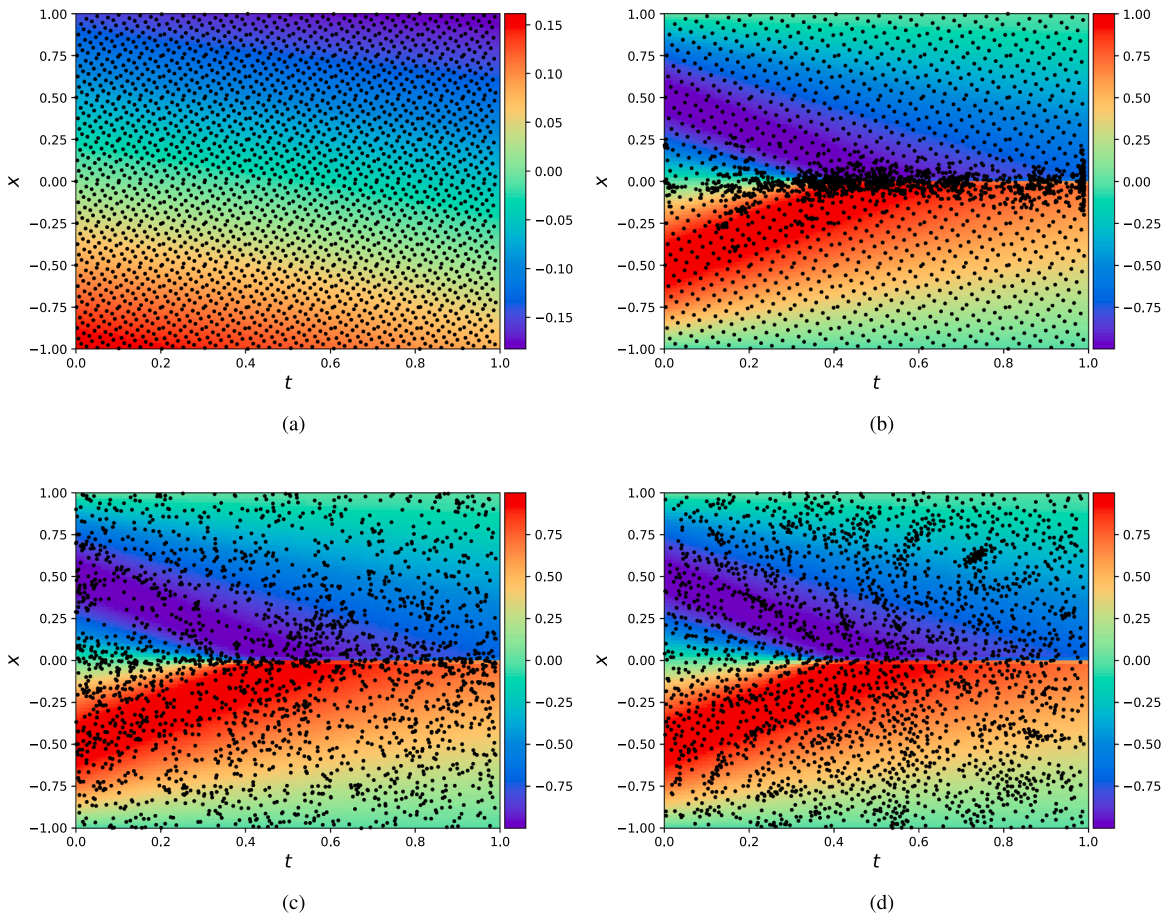


Fig. 5. Location of the 2500 collocation points (a) before training, and after training with (b) RAR, (c) RAD, and (d) PACMANN with Adam for the Burgers' equation example. The color indicates the values of the predicted solution.

Table 4
Overview of the mean and standard deviation of the test error and the mean runtime for each sampling method for the Allen-Cahn equation example. The best result in each column is marked in boldface.

Sampling method	L_2 relative error		Mean runtime [s]
	Mean	1 SD	
Uniform grid	44.34%	18.58%	634
Hammersley grid	0.47%	0.26%	**591**
Random resampling	0.42%	0.28%	592
RAR	0.44%	0.27%	576
RAD	0.93%	0.69%	655
RAR-D	0.28%	0.13%	632
PACMANN-Adam	**0.16%**	**0.07%**	632

training the PINN, these previously observed local maxima have reduced. For our approach, we note the similarity between the shape of the clusters and the solution itself.

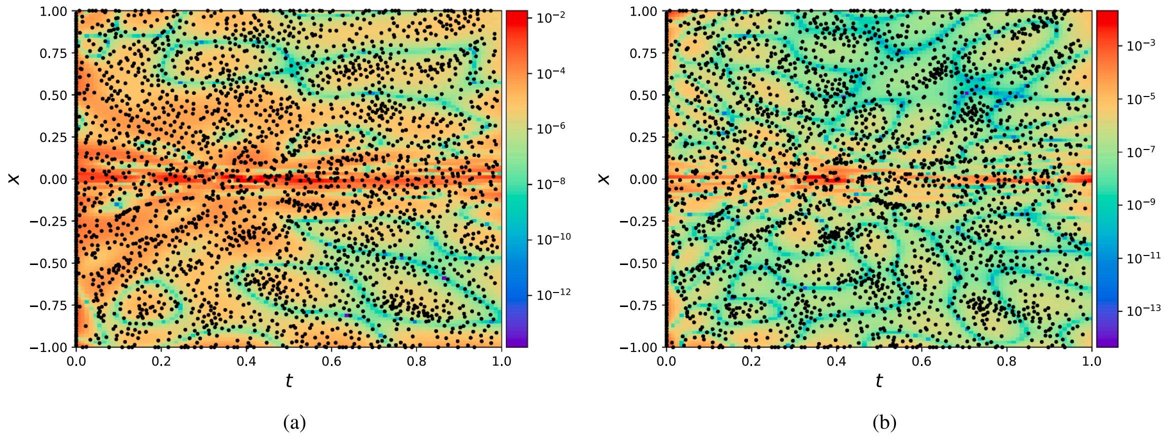


Fig. 6. Location of the collocation points (a) mid-training at 25 000 iterations, and (b) after training using PACMANN with Adam for the Burgers' equation example. The color indicates the values of the squared residual.

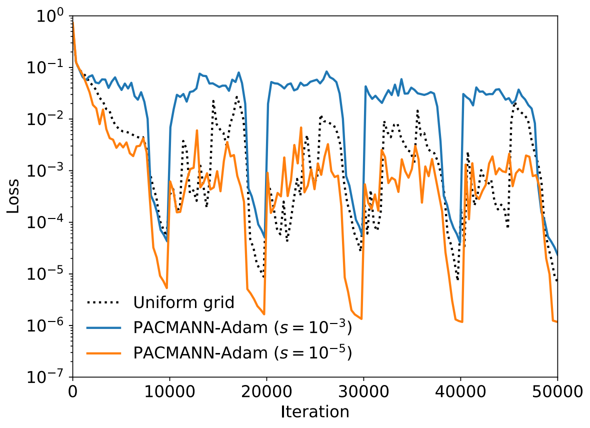


Fig. 7. Mean training loss over ten runs for the Burgers' equation example, comparing a uniform grid and PACMANN using Adam. For PACMANN, we report the mean loss for stepsizes 10^{-3} and 10^{-5}, which correspond to the least and most accurate stepsize choices, respectively, in terms of the L_2 relative error.

Table 5
Overview of the mean and standard deviation of the test error and the mean runtime achieved by PACMANN for each optimization algorithm listed in Section 2. The best result in each column is marked in boldface. We also include the optimal values for the stepsize and the number of steps, for the Allen-Cahn equation example.

PACMANN optimizer	L_2 relative error		Mean runtime [s]	Hyperparameters	
	Mean	1 SD		Stepsize s	No. of steps T
Gradient ascent	0.46%	0.24%	574	10^{-8}	5
Nonlinear gradient ascent	0.42%	0.24%	602	10^{-7}	5
RMSprop	0.29%	0.20%	595	10^{-6}	5
Momentum	0.36%	0.17%	**567**	10^{-7}	5
Adam	**0.16%**	**0.07%**	632	10^{-5}	5
Golden section search	0.37%	0.29%	635	10^{-7}	15

3.2. 1D Allen-Cahn equation

In the following example, we consider the one-dimensional Allen-Cahn equation:

$$\begin{cases} u_t = du_{xx} + 5(u - u^3), \quad x \in [-1, 1], \quad t \in [0, 1], \\ u(x, 0) = x^2 \cos(\pi x), \\ u(-1, t) = u(1, t) = -1. \end{cases} \tag{12}$$

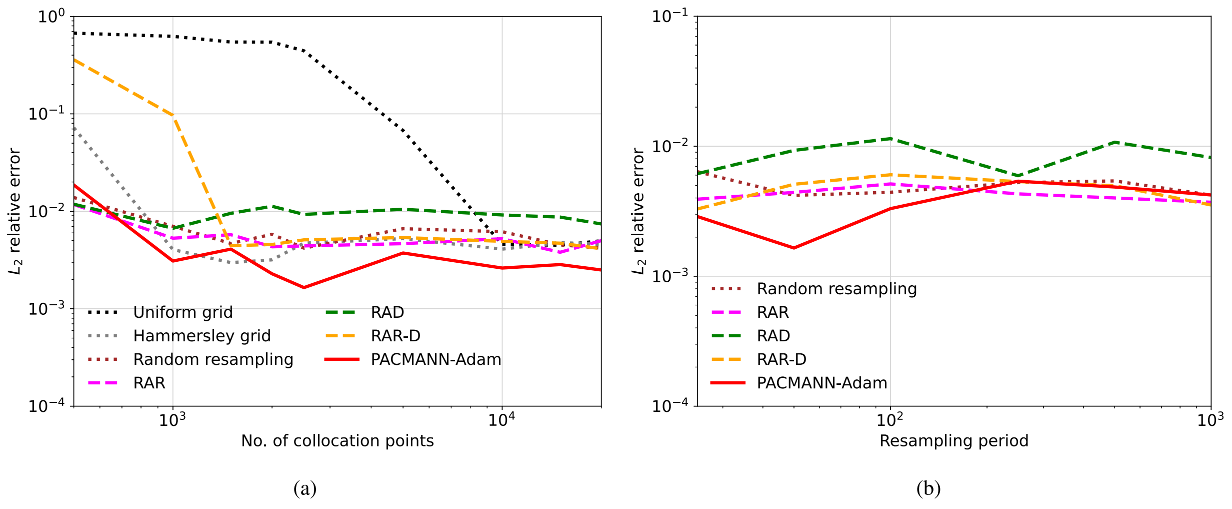


Fig. 8. Mean of the test error for each of the sampling methods for a varying (a) number of collocation points and (b) resampling period for the Allen-Cahn equation example.

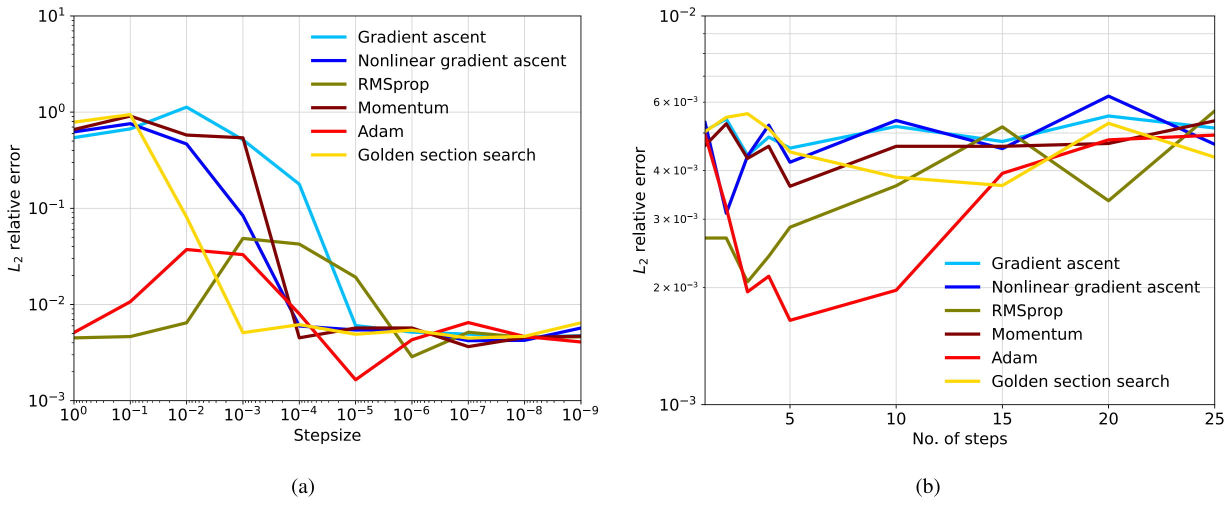


Fig. 9. Mean of the test error for PACMANN with the different optimization algorithms listed in Section 2 for a varying (a) stepsize and (b) number of steps for the Allen-Cahn equation example.

We choose a diffusion coefficient of $d = 0.001$. Similar to the Burgers' equation problem in the previous section, the number of collocation points is set to 2500, the number of boundary points to 80, and the number of initial condition points to 160. The network architecture used for this example consists of four hidden layers of 64 neurons. To compute the L_2 relative error, we compare the network prediction to a numerical solution generated using a spectral solver and made available via the DeepXDE library [18].

The mean and standard deviation of the test error and the mean runtime for each sampling method are given in Table 4. As for the Burgers' equation example, the static uniform grid fails to learn the solution satisfactorily, and the static Hammersley grid and random resampling offer a significant improvement in accuracy. Out of all sampling methods considered, PACMANN in combination with the Adam optimizer and a stepsize of 10^{-5} achieves the lowest test error. The next-best method, RAR-D, attains a lower prediction accuracy at the same computational cost. Table 5 demonstrates that Adam results in the lowest mean and standard deviation of the test error for our method in comparison to the other optimization algorithms considered.

Fig. 8(a) shows the behavior of the different sampling methods when varying the number of collocation points. We find that PACMANN with the Adam optimizer converges the fastest and outperforms the other sampling methods for nearly all numbers of collocation points considered. Most sampling methods stagnate above 2500 collocation points, except for the static uniform grid, which requires up to 10 000 collocation points to compete with the accuracy of the other sampling methods.

Moreover, Fig. 8(b) depicts the accuracy of the sampling methods considered for a range of resampling periods. In contrast to our findings in the previous Burgers' equation example, the resampling period has a reduced influence on the test error in this example. Notably, most sampling methods plateau with an increasing resampling period.

Table 6
Overview of the mean and standard deviation of the test error for λ_1 and λ_2 and the mean runtime for each sampling method for the inverse Navier-Stokes equations example. The best result in each column is marked in boldface.

Sampling method	L_2 relative error				Mean runtime [s]
	λ_1		λ_2		
	Mean	1 SD	Mean	1 SD	
Uniform grid	0.05%	**0.01%**	0.72%	0.43%	1506
Hammersley grid	0.08%	0.04%	0.89%	0.52%	**1492**
Random resampling	0.12%	0.05%	0.65%	0.46%	1514
RAR	0.30%	0.06%	1.44%	0.90%	1520
RAD	0.23%	0.06%	1.38%	0.79%	1583
RAR-D	0.08%	0.05%	0.84%	0.57%	1525
PACMANN-Adam	**0.03%**	0.03%	**0.53%**	**0.19%**	1559

Furthermore, we test the behavior of PACMANN with the different optimization algorithms listed in Section 2 by varying the stepsize and the number of steps hyperparameters. Similarly to the results in Section 3.1 for the Burgers' equation, the stepsize hyperparameter has a significant influence on the test error, as demonstrated by Fig. 9(a). We note, again, the formation of two groups, which occurs due to the random resampling of points that have been moved outside the domain. When using RMSprop and Adam combined with large stepsizes, all 2500 collocation points leave the domain. As a result, the test error converges toward the error of $0.42\% \pm 0.28\%$ found when using random resampling; cf. Table 4.

Finally, we also vary the number of optimization steps for resampling. Fig. 9(b) illustrates that the test error of most optimization algorithms is not significantly impacted by changing the number of steps. Fig. 9(a) and (b) support our earlier finding that the stepsize plays a dominant role in the test error obtained compared to the limited influence of the number of steps.

Based on the observations made in the Burgers' and Allen-Cahn equation examples, we recommend fixing the number of steps to five, which removes a hyperparameter from PACMANN. Furthermore, we recommend five steps as it takes advantage of the increased accuracy achieved by our method with the Adam optimizer at multiple steps (see Figs. 3(b) and 9(b)), while keeping the computational cost low; see Fig. 4(b).

3.3. 2D Navier-Stokes equations (inverse)

Next, we consider an inverse problem based on the two-dimensional Navier-Stokes equations describing the flow of an incompressible fluid past a cylinder discussed by Raissi et al. in [6], given by:

$$\begin{cases} \dfrac{\partial \boldsymbol{v}}{\partial t} + \lambda_1 \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\nabla p + \lambda_2 \nabla^2 \boldsymbol{v}, \quad (x, y) \in [1, 8] \times [-2, 2], \quad t \in [0, 7], \\ \qquad \nabla \cdot \boldsymbol{v} = 0. \end{cases} \tag{13}$$

Here, u and v are the x- and y-components of the velocity field, and p denotes the pressure. The scalar parameter λ_1 scales the convective term, and λ_2 represents the dynamic (shear) viscosity. In this example, we are interested in learning the values of λ_1 and λ_2 based on a data set created by Raissi et al. [6] using a spectral solver. The data set contains the values of u, v, and p determined for a large set of points (x, y, t). The true values of λ_1 and λ_2 are 1 and 0.01, respectively. For this inverse problem, we train the PINN on 7000 randomly selected points from this data set. In addition, we sample 700 collocation points, 200 points on the boundary condition, and 100 points on the initial condition. The network architecture consists of six hidden layers containing 50 neurons each.

The mean and standard deviation of the test error for both λ_1 and λ_2 and the mean runtime for each of the sampling methods are given in Table 6. In this inverse problem, PACMANN in combination with Adam at a stepsize of 10^{-2} achieves the lowest test error for λ_1 and λ_2 at a slightly higher computational cost compared to the second best adaptive method, RAR-D. Furthermore, we note that the non-adaptive sampling methods generally outperform the other adaptive methods considered in this study, both in the mean and standard deviation of the test error.

Qualitatively, we observe similar behavior of the different sampling methods when varying the number of collocation points and the resampling period for the 2D Navier-Stokes equations example and the following examples, as observed in the Burgers' equation example. This observation also applies to the behavior of PACMANN when changing the stepsize and the number of steps hyperparameters, namely, that the number of steps generally has a smaller impact on the prediction accuracy than the stepsize. Therefore, for the sake of conciseness, we do not repeat the analysis of the behavior of the different sampling methods for this two-dimensional Navier-Stokes equations example and the following examples.

Table 7
Overview of the mean and standard deviation of the L_2 relative error and the H^1 semi-norm, and the mean runtime for each sampling method for the Poisson's equation example. The best result in each column is marked in boldface.

Sampling method	L_2 relative error		H^1 semi-norm		Mean runtime [s]
	Mean	1 SD	Mean	1 SD	
Uniform grid	40.32%	1.18%	62.58%	1.74%	**744**
Hammersley grid	82.64%	2.95%	90.48%	2.20%	752
Random resampling	11.47%	1.13%	13.36%	1.66%	751
RAR	62.00%	7.13%	69.97%	6.62%	773
RAD	11.69%	1.78%	13.19%	1.31%	841
RAR-D	89.31%	1.98%	93.83%	2.05%	784
PACMANN-Adam	**8.35%**	**0.54%**	**10.32%**	**0.43%**	786

Table 8
Overview of the mean and standard deviation of the L_2 relative error for u, v, and w and the mean runtime for each sampling method for the 3D Navier-Stokes equations example. The best result in each column is marked in boldface.

Sampling method	L_2 relative error						Mean runtime [s]
	u		v		w		
	Mean	1 SD	Mean	1 SD	Mean	1 SD	
Uniform grid	4.89%	2.47%	5.82%	4.10%	4.24%	3.03%	**1458**
Hammersley grid	3.14%	0.27%	**2.99%**	**0.28%**	2.97%	0.24%	1482
Random resampling	3.97%	0.31%	3.71%	0.42%	3.37%	0.28%	1478
RAR	4.86%	0.54%	4.63%	0.37%	4.12%	0.29%	1496
RAD	3.56%	0.33%	3.63%	0.33%	3.03%	**0.21%**	1591
RAR-D	3.48%	0.53%	3.38%	0.49%	3.17%	0.39%	1500
PACMANN-Adam	**3.01%**	**0.25%**	3.09%	**0.28%**	**2.50%**	**0.21%**	1559

3.4. 5D Poisson's equation

In the following problem, we apply PACMANN to the Poisson equation in five dimensions:

$$\begin{cases} -\Delta v(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in [-1,1]^5, \\ v(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \partial\boldsymbol{\Omega}. \end{cases} \tag{14}$$

We choose the right-hand side function f based on the manufactured solution

$$v(\boldsymbol{x}) = \prod_{i=1}^{5} \sin(\pi x_i), \tag{15}$$

where x_i is the ith component of $\boldsymbol{x}$. For this example, we sample 750 collocation points and 750 points for the boundary condition. Moreover, the network architecture consists of four hidden layers of 64 neurons each.

The mean and standard deviation of the L_2 relative error and the H^1 semi-norm, and the mean runtime for each of the sampling methods are given in Table 7. In contrast to the previous examples, the adaptive methods RAR and RAR-D fail to improve the prediction accuracy. PACMANN in combination with the Adam optimizer and a stepsize of 10^{-2} achieves the lowest mean and standard deviation of the L_2 relative error and the H^1 semi-norm. Moreover, we point out the ability of this method to efficiently scale to high-dimensional problems. We find that our method with the Adam optimizer is cheaper at a mean runtime of 786 s compared to RAD, the next-best adaptive sampling method, at 841 s.

3.5. 3D Navier-Stokes equations

We next apply PACMANN to the incompressible Navier-Stokes equations in a cube:

$$\begin{cases} \dfrac{\partial \boldsymbol{v}}{\partial t} + Re\, \boldsymbol{v} \cdot \nabla \boldsymbol{v} + Re\, \nabla p - \nabla^2 \boldsymbol{v} = f(\boldsymbol{x}, t), \quad \boldsymbol{x} \in [-1,1]^3, \quad t \in [0,1], \\ \nabla \cdot \boldsymbol{v} = 0, \\ \boldsymbol{v}(\boldsymbol{x}, \boldsymbol{t}) = \boldsymbol{v}_{ref}(\boldsymbol{x}, t), \quad p(\boldsymbol{x}, t) = p_{ref}(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \partial\boldsymbol{\Omega}, \\ \boldsymbol{v}(\boldsymbol{x}, 0) = \boldsymbol{v}_{ref}(\boldsymbol{x}, 0), \quad p(\boldsymbol{x}, 0) = p_{ref}(\boldsymbol{x}, 0), \quad t = 0. \end{cases} \tag{16}$$

Table 9
Overview of the mean and standard deviation of the H^1 semi-norm for u, v, and w and the mean runtime for each sampling method for the 3D Navier-Stokes equations example. The best result in each column is marked in boldface.

Sampling method	H^1 semi-norm						Mean runtime [s]
	u		v		w		
	Mean	1 SD	Mean	1 SD	Mean	1 SD	
Uniform grid	26.22%	18.11%	34.25%	28.57%	17.17%	12.32%	**1458**
Hammersley grid	10.89%	1.38%	10.62%	1.10%	10.95%	0.90%	1482
Random resampling	10.24%	0.81%	9.48%	1.17%	10.36%	0.99%	1478
RAR	15.97%	1.34%	15.84%	1.54%	15.21%	1.27%	1496
RAD	**7.78%**	1.57%	**7.71%**	1.32%	**7.66%**	1.41%	1591
RAR-D	12.50%	2.21%	11.94%	2.23%	11.78%	1.65%	1500
PACMANN-Adam	8.26%	**0.59%**	8.19%	**0.48%**	7.70%	**0.68%**	1559

Table 10
Overview of the mean and standard deviation of the L_2 relative error for p and the mean runtime for each sampling method for the 3D Navier-Stokes equations example. The best result in each column is marked in boldface.

Sampling method	L_2 relative error		Mean runtime [s]
	Mean	1 SD	
Uniform grid	**10.77%**	2.64%	**1458**
Hammersley grid	24.25%	0.77%	1482
Random resampling	28.19%	1.67%	1478
RAR	29.02%	2.94%	1496
RAD	29.23%	**0.57%**	1591
RAR-D	24.61%	0.62%	1500
PACMANN-Adam	31.48%	4.07%	1559

Here, we set the Reynolds number to $Re = 10^3$. Furthermore, we set initial conditions, Dirichlet boundary conditions, and right-hand side based on the manufactured solutions $\boldsymbol{v}_{ref}(\boldsymbol{x}, t)$:

$$\begin{cases} u_{ref}(\boldsymbol{x}, t) = \cos(x)\sin(y)\sin(z)\, e^{-t}, \\ v_{ref}(\boldsymbol{x}, t) = \sin(x)\cos(y)\sin(z)\, e^{-t}, \\ w_{ref}(\boldsymbol{x}, t) = -2\cos(x)\cos(y)\cos(z)\, e^{-t}, \end{cases} \tag{17}$$

and $p_{ref}(\boldsymbol{x}, t)$:

$$p_{ref}(\boldsymbol{x}, t) = \cos(x)\cos(y)\cos(z)\, e^{-t}. \tag{18}$$

For this problem, we sample 250 collocation points, and we employ 1000 points for the boundary conditions and 250 points for the initial conditions. The network architecture used for this problem consists of four hidden layers of 64 neurons.

The mean and standard deviation of the L_2 relative error and the H^1 semi-norm for u, v, and w, and the mean runtime for each of the sampling methods are given in Tables 8 and 9. In this problem, PACMANN in combination with Adam at a stepsize of 10^{-4} achieves the lowest L_2 relative error for u, v, and w while maintaining a computational cost comparable to that of other adaptive sampling methods. RAD achieves the lowest H^1 semi-norm, closely followed by PACMANN. The mean and standard deviation of the L_2 relative error for p for each sampling method are given in Table 10, where the uniform grid achieves the lowest error.

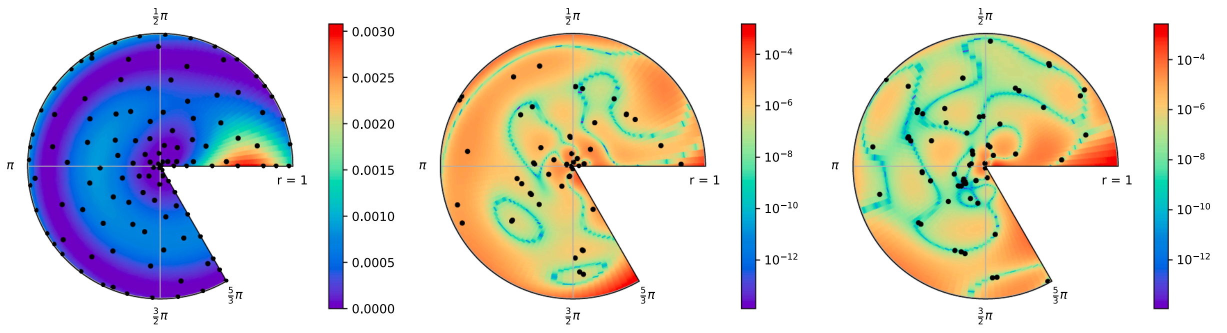
3.6. Re-entrant corner in a disk

In the following problem, we consider a problem involving a domain with a re-entrant corner, where the solution is expected to be less regular. Specifically, we consider the Laplace equation in polar coordinates over a sector:

$$\begin{cases} u_{rr} + \frac{1}{r}u_r + \frac{1}{r^2}u_{\theta\theta} = 0, & r \in [0,1], \quad \theta \in \left[0, \frac{5\pi}{3}\right], \\ u(r,\theta) = h(r,\theta), & (r,\theta) \in \partial\boldsymbol{\Omega}. \end{cases} \tag{19}$$

The exact solution is used for the boundary condition $h(r,\theta)$ and is given by

$$h(r,\theta) = r^{\frac{3}{5}} \sin\left(\frac{3}{5}\theta\right),$$



(a) (b) (c)

Fig. 10. Location of the collocation points (a) before training, (b) mid-training at 25 000 iterations, and (c) after training using PACMANN with Adam for the Laplace equation over a re-entrant corner in a disk example. The color indicates the values of the squared residual.

Table 11
Overview of the mean and standard deviation of the L_2 relative error and the H^1 semi-norm, and the mean runtime for each sampling method for the Laplace equation over a re-entrant corner in a disk example. The best result in each column is marked in boldface.

Sampling method	L_2 relative error		H^1 semi-norm		Mean runtime [s]
	Mean	1 SD	Mean	1 SD	
Uniform grid	6.32%	1.46%	32.36%	6.50%	**514**
Hammersley grid	0.47%	**0.17%**	5.90%	0.91%	539
Random resampling	0.49%	0.18%	5.45%	**0.82%**	536
RAR	0.58%	**0.17%**	6.12%	1.00%	538
RAD	0.48%	0.25%	**4.46%**	1.31%	594
RAR-D	0.96%	0.74%	9.39%	6.27%	543
PACMANN-Adam	**0.43%**	**0.17%**	5.00%	1.03%	544

which has an algebraic singularity at the origin; cf. [38, pp. 110–116]. In this example, we sample 75 collocation points and we set the number of points for the boundary condition to 75. The network architecture used for this problem consists of four hidden layers of 64 neurons.

The mean and standard deviation of the L_2 relative error and the H^1 semi-norm, and the mean runtime for each of the sampling methods are given in Table 11. In this example, PACMANN in combination with the Adam optimizer at a stepsize of 10^{-2} achieves the lowest L_2 relative error at a computational cost comparable to that of the adaptive sampling methods RAR and RAR-D, while RAD achieves the lowest H^1 semi-norm. Furthermore, we observe a collective movement of points toward the origin, where the algebraic singularity is located; see Fig. 10.

3.7. Two-hole perforated plate

Finally, we demonstrate PACMANN with Adam for a linear elasticity problem. The governing equation is the momentum balance equation:

$$\nabla \cdot \boldsymbol{\sigma} = \mathbf{0}, \tag{20}$$

where σ denotes the Cauchy stress tensor. For infinitesimal deformations, the stress tensor is related to the linear strain tensor $\boldsymbol{\varepsilon}$ through Hooke's law:

$$\boldsymbol{\sigma} = \lambda\,\mathrm{tr}(\boldsymbol{\varepsilon})\,\mathbf{I} + 2\mu\,\boldsymbol{\varepsilon}, \tag{21}$$

with λ and μ the Lamé parameters. In turn, the linear strain is defined in terms of the displacement field $\boldsymbol{u}$ as:

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right). \tag{22}$$

We consider the unit square domain $\Omega = [0,1]^2$ containing two holes of radius 0.15 centered at $(0.25, 0.25)$ and $(0.75, 0.75)$. Roller boundary conditions are applied at the left and bottom edges, traction-free conditions on the right edge, and a prescribed vertical

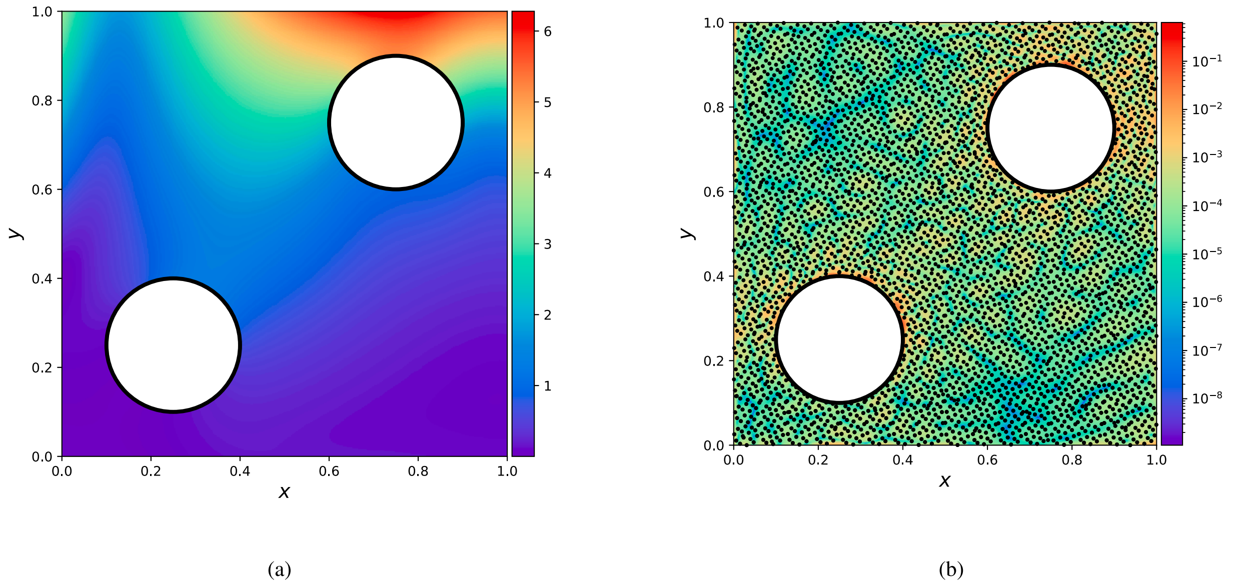


Fig. 11. Plots of the (a) predicted displacement $\|u\|$ of the linear elasticity problem for the two-hole perforated plate and (b) the squared residual alongside the location of the collocation points after training with PACMANN and Adam at a stepsize of 10^{-6}. The color indicates the values of the (a) displacement and (b) squared residual.

traction on the top edge, namely:

$$\begin{aligned} u_x &= 0, \quad \sigma_{xx} = 0 && \text{on} \quad \partial\Omega_{\text{left}} \\ u_y &= 0, \quad \sigma_{yy} = 0 && \text{on} \quad \partial\Omega_{\text{bottom}} \\ \sigma_{xx} &= 0, \quad \sigma_{xy} = 0 && \text{on} \quad \partial\Omega_{\text{right}} \\ \sigma_{xy} &= 0, \quad \sigma_{yy} = 8\sin(\pi x) && \text{on} \quad \partial\Omega_{\text{top}}. \end{aligned} \tag{23}$$

On the boundaries of both holes, we impose traction-free conditions $\boldsymbol{\sigma n} = \mathbf{0}$, where $\boldsymbol{n}$ the outward unit normal vector.

For this test case, we set $\lambda = \mu = 1$. We solve this problem with PACMANN and Adam with 5000 collocation points internal to the domain, 2000 points on the boundaries, and a stepsize 10^{-6}. The network architecture consists of six hidden layers of 64 neurons. The predicted displacement magnitude $\|\boldsymbol{u}\|$ and the squared residual are shown in Fig. 11(a) and (b).

4. Conclusions

In this work, we presented the Point Adaptive Collocation Method for Artificial Neural Networks (PACMANN), a novel adaptive collocation point sampling method for physics-informed neural networks. This approach uses the gradient of the physics-informed loss terms, that is, of the squared residual, as guiding information to move collocation points toward areas of large residuals. The problem of moving points is formulated as a maximization problem, which can be approached using an optimization algorithm of choice, such as gradient ascent or Adam. Points are moved several times while training is halted. Our approach can be tuned using three additional hyperparameters, namely the resampling period, the size of the step used to move the collocation points, and the number of times that points are moved while training is halted. While this work demonstrates PACMANN for PINNs, we note that the method can also be applied to other collocation-based approaches.

We studied the sensitivity of our method to these hyperparameters, and we observed that the stepsize has a particularly large impact on the solution accuracy. Conversely, we found that five iteration steps are sufficient to achieve a good balance between accuracy and efficiency. We then investigated the accuracy and efficiency of PACMANN in combination with various optimization algorithms and concluded that the Adam optimizer performs the best.

Furthermore, we compared the performance of PACMANN to existing state-of-the-art adaptive and non-adaptive collocation approaches, including random resampling, RAR, and RAD, and demonstrated that our method achieves state-of-the-art performance in terms of the accuracy/efficiency tradeoff for lower-dimensional benchmarks, while outperforming the state-of-the-art for high-dimensional problems and the case of a solution with algebraic singularity. In addition, we showed the effectiveness of our approach in solving inverse problems and problems with non-convex domains.

In particular, the results of our numerical experiments demonstrate that PACMANN achieves high prediction accuracy and scales efficiently to higher dimensions without introducing significant computational overhead across a variety of problems. At the same time, several directions for future work emerge. One concerns problems that exhibit boundary layers, where pronounced gradients near the domain boundary may drive collocation points outside the domain during the point-update step, leading to reduced point density in boundary regions after resampling. Developing mechanisms to retain points near the boundary without increasing algorithmic complexity is a promising extension of this work. Another direction involves the potential influence of the PDE's structural properties, for example whether it is elliptic, parabolic, or hyperbolic, on the tuning and performance of the resampling procedure, an aspect not explored in the present study.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT (OpenAI) in order to improve the readability and language of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

CRediT authorship contribution statement

Coen Visser: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization; **Alexander Heinlein:** Writing – review & editing, Supervision, Resources, Methodology, Investigation, Conceptualization; **Bianca Giovanardi:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Conceptualization.

Data availability

All code required to reproduce the tests is provided in the GitHub repository mentioned in the paper.

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.

Acknowledgments

The authors acknowledge the use of computational resources of the DelftBlue supercomputer, provided by Delft High Performance Computing Centre (https://www.tudelft.nl/dhpc).

References

[1] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314. https://doi.org/10.1007/BF02551274
[2] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366. https://doi.org/10.1016/0893-6080(89)90020-8
[3] M.W.M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201. https://doi.org/10.1002/CNM.1640100303
[4] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000. https://doi.org/10.1109/72.712178
[5] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2015) 1–43. https://doi.org/10.5555/3122009.3242010
[6] 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. https://doi.org/10.1016/J.JCP.2018.10.045
[7] S. Cuomo, V.S. Di 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) 1–62. https://doi.org/10.1007/S10915-022-01939-Z
[8] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440. https://doi.org/10.1038/s42254-021-00314-5
[9] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Eng. 360 (2020) 112789. https://doi.org/10.1016/J.CMA.2019.112789
[10] S. Cai, Z. Wang, F. Fuest, Y.-J. Jeon, C. Gray, G.E. Karniadakis, Flow over an espresso cup: inferring 3D velocity and pressure fields from tomographic background oriented schlieren videos via physics-informed neural networks, J. Fluid Mech. 915 (2021). https://doi.org/10.1017/jfm.2021.135
[11] 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. https://doi.org/10.1016/J.JCP.2020.109951
[12] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks for heat transfer problems, J. Heat Transfer 143 (6) (2021). https://doi.org/10.1115/1.4050542/1104439
[13] S. Amini Niaki, E. Haghighat, T. Campbell, A. Poursartip, R. Vaziri, Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture, Comput. Methods Appl. Mech. Eng. 384 (2021) 113959. https://doi.org/10.1016/J.CMA.2021.113959
[14] K. Shukla, P.C. Di Leoni, J. Blackshire, D. Sparkman, G.E. Karniadakis, Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, J. Nondestr. Eval. 39 (3) (2020). https://doi.org/10.1007/s10921-020-00705-1
[15] E. Zhang, M. Dao, G.E. Karniadakis, S. Suresh, Analyses of internal structures and defects in materials using physics-informed neural networks, Sc. Adv. 8 (7) (2022). https://doi.org/10.1126/SCIADV.ABK0644
[16] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, M. Yano, N. Sakuma, A. Kinoshita, T. Shoji, A. Kato, T. Schrefl, Conditional physics informed neural networks, Commun. Nonlinear Sci. Numer. Simul. 104 (2022) 106041. https://doi.org/10.1016/J.CNSNS.2021.106041

[17] S. Son, H. Lee, D. Jeong, K.Y. Oh, K. Ho Sun, A novel physics-informed neural network for modeling electromagnetism of a permanent magnet synchronous motor, Adv. Eng. Inf. 57 (2023) 102035. https://doi.org/10.1016/J.AEI.2023.102035
[18] 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
[19] 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
[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. Eng. 403 (2023) 115671. https://doi.org/10.1016/J.CMA.2022.115671
[21] J. Guo, H. Wang, S. Gu, C. Hou, TCAS-PINN: physics-informed neural networks with a novel temporal causality-based adaptive sampling method, Chin. Phys. B 33 (5) (2024) 050701. https://doi.org/10.1088/1674-1056/AD21F3
[22] J. Hou, Y. Li, S. Ying, J. Hou, Y. Li, S. Ying, Enhancing PINNs for solving PDEs via adaptive collocation point movement and adaptive loss weighting, Nonlinear Dyn. 111 (2023) 15233–15261. https://doi.org/10.1007/s11071-023-08654-w
[23] Z. Mao, X. Meng, Physics-informed neural networks with residual/gradient-based adaptive sampling methods for solving partial differential equations with sharp solutions, Appl. Math. Mech. 44 (7) (2023) 1069–1084. https://doi.org/10.1007/S10483-023-2994-7
[24] Y. Liu, L. Chen, J. Ding, Y. Chen, An adaptive sampling method based on expected improvement function and residual gradient in PINNs, IEEE Access (2024). https://doi.org/10.1109/ACCESS.2024.3422224
[25] K. Tang, X. Wan, C. Yang, DAS-PINNs: a deep adaptive sampling method for solving high-dimensional partial differential equations, J. Comput. Phys. 476 (2021). https://doi.org/10.1016/j.jcp.2022.111868
[26] K. Tang, X. Wan, Q. Liao, Deep density estimation via invertible block-triangular mapping, Theor. Appl. Mech. Lett. 10 (3) (2020) 143–148. https://doi.org/10.1016/J.TAML.2020.01.023
[27] Q. Gao, L. Xu, D. Wang, R. Zhang, Energy-equidistributed moving sampling physics-informed neural networks for solving conservative partial differential equations, arXiv preprint (2025). https://doi.org/10.48550/arXiv.2508.19561
[28] C. Wang, S. Li, D. He, L. Wang, Is L2 physics-informed loss always suitable for training physics-informed neural network? Neural information processing systems (2022). https://doi.org/10.48550/arXiv.2206.02016
[29] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536. https://doi.org/10.1038/323533a0
[30] D.P. Kingma, J.L. Ba, Adam: a method for stochastic optimization, 3rd international conference on learning representations, ICLR 2015 - conference track proceedings (2015). https://doi.org/10.48550/arXiv.1412.6980
[31] J.M. Hammersley, D.C. Handscomb, Monte Carlo Methods, Chapman and Hall, London, 1 ed., 1964. https://doi.org/10.1007/978-94-009-5819-7
[32] G. Hinton, T. Tieleman, Neural networks for machine learning lecture 6.5 - RMSprop: divide the gradient by a running average of its recent magnitude, 2012.
[33] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: Proceedings of the 30th International Conference on Machine Learning, 2013, pp. 1139–1147.
[34] M.J. Kochenderfer, T.A. Wheeler, Algorithms for Optimization, The MIT Press, 1 ed., Cambridge, 2019.
[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: an imperative style, high-performance deep learning library, Neural Inf. Process. Syst. 32 (2019) 8026–8037.
[36] Delft high performance computing centre (DHPC), DelftBlue supercomputer (Phase 2), 2024, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.
[37] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1-3) (1989) 503–528. https://doi.org/10.1007/BF01589116
[38] M. Dauge, Elliptic Boundary Value Problems on Corner Domains, Springer-Verlag, Berlin, 1 ed., 1988. https://doi.org/10.1007/BFb0086682