Adapting Physics-Informed Neural Networks to Improve ODE Optimization in Mosquito Population Dynamics

Dinh Viet Cuong[1*], Branislava Lalić[2], Mina Petrić[3], Nguyen Thanh Binh[4], Mark Roantree[5]

[1] School of Computing, Dublin City University, Dublin, Ireland
[2] Faculty of Agriculture, University of Novi Sad, Serbia
[3] Avia-GIS NV, Zoersel, Belgium
[4] University of Science, Ho Chi Minh City, Vietnam
[5] Insight Centre for Data Analytics, Dublin City University, Dublin, Ireland

* Corresponding author:
Email: dinhviet.cuong@dcu.ie

Abstract

Physics informed neural networks have been gaining popularity due to their unique ability to incorporate physics laws into data-driven models, ensuring that the predictions are not only consistent with empirical data but also align with domain-specific knowledge in the form of physics equations. The integration of physics principles enables the method to require less data while maintaining the robustness of deep learning in modelling complex dynamical systems. However, current PINN frameworks are not sufficiently mature for real-world ODE systems, especially those with extreme multi-scale behavior such as mosquito population dynamical modelling. In this research, we propose a PINN framework with several improvements for forward and inverse problems for ODE systems with a case study application in modelling the dynamics of mosquito populations. The framework tackles the gradient imbalance and stiff problems posed by mosquito ordinary differential equations. The method offers a simple but effective way to resolve the time causality issue in PINNs by gradually expanding the training time domain until it covers entire domain of interest. As part of a robust evaluation, we conduct experiments using simulated data to evaluate the effectiveness of the approach. Preliminary results indicate that physics-informed machine learning holds significant potential for advancing the study of ecological systems.

Introduction

Arboviruses can spread quickly and cause major disease epidemics. Mosquitoes are vectors of some of the worlds most severe diseases such as malaria, dengue, Zika, Chikungunya, and West Nile Virus disease [1–5]. Different modelling approaches have been developed to simulate the abundance and seasonal dynamics of mosquito vectors and support disease risk prediction. They can be viewed in two broad categories: mathematical and statistical models. Mathematical models rely on laboratory and field data for the parameterisation of key life history traits such as the development and mortality rates of different stages in the mosquito life cycle [6–11]. On the other hand, statistical models use correlative and other machine learning (ML) techniques to infer the relationship between vector abundance and a set of abiotic factors [12–18]. These models typically require multi-year time-series of mosquito surveillance data (derived from labour intensive longitudinal field studies) to produce accurate outputs, and are often subject to various sources of variability that could lead to biased results.

In this paper we explore the feasibility of using Physics Informed Neural Networks (PINNs) trained on an ordinary differential equation (ODE) mosquito population dynamics model to bridge the gap between conventional mathematical modelling and data-science approaches while conserving the physical and biological constraints and consistencies that govern these systems and processes.

Physics Informed Neural Networks

Recent advancements in computational capabilities and the exponential growth in data availability have made data-driven analytics one of the predominant strategies in both research and practical applications. Deep learning using different forms of neural networks is central to this development and has been extensively applied across various domains such as computer vision [19, 20], natural language processing [21, 22], genomic prediction in plants [23] and finance [24, 25]. In data-driven approaches, neural networks are trained to minimize discrepancies between model predictions and observed data. However, this purely data-driven approach has a number of limitations, including poor interpretability [26], poor out-of-distribution generalization [27], and the requirement for substantial amounts of training data [28].

Physics-informed neural networks (PINNs) [29] have emerged as an alternative for scenarios where data is governed by underlying physical laws expressed through differential equations. This approach integrates domain-specific knowledge into machine learning by incorporating physical laws as additional objective loss functions alongside the traditional data-fitting loss functions. This multi-task optimization strategy not only attempts to align with observational data but also to approximate the governing differential equations. As a result, PINNs adhere to physical laws and thereby, enhance model generalizability and also uncover latent patterns within empirical data. This type of framework also facilitates an uncomplicated solution to both forward and inverse problems, where we can simultaneously learn the system state (forward problem) and the system's parameters (inverse problem). Successful deployments of PINNs have been demonstrated across various fields, as surveyed in [28, 30–34].

However, despite this potential, PINN deployments continue to face significant challenges, particularly in training models that involve multi-scale and stiff solutions [28, 35, 36]. In recent years, numerous advancements have been made to enhance the foundational framework initially proposed in [29]. These enhancements include the development of innovative neural network architectures [35–38] and novel adaptive activation functions [39, 40]. Several studies have focused on optimizing the multi-task training process by adaptively adjusting the weights of different loss components [36, 41, 42]. Others have explored modifications in the distribution of collocation points [43–46]. Additionally, some researchers have adopted a sequential learning approach, where training is conducted on one subdomain at a time before progressing to the next [47–49], thereby preserving the causality within the system [50]. Moreover, there are efforts where the input domain is divided into smaller subdomains with PINNs trained separately on each subdomain, significantly enhancing the model's convergence and accuracy [51–53].

Contribution and Paper Structure

While the majority of research on PINNs has concentrated on enhancing techniques for the effective training of Partial Differential Equations (PDEs), there has been less focus on customizing these methods for multi-variate, multi-equation Ordinary Differential

Equation (ODE) systems. In this study, we explore the applicability and effectiveness of existing PINN methodologies to ODE systems. We introduce specific adjustments designed to optimize the training of ODE systems, including the individual normalization and loss weight balancing tailored for each variable and equation involved in the system. Moreover, we implement two additional steps in the training process to enhance both the effectiveness and accuracy of the models, and to simplify domain decomposition for practical application scenarios. We evaluate our modified framework using the Lorenz system, a classical model in dynamical systems theory, to demonstrate its applicability and value. Finally, we apply our approach to the modelling of mosquito population dynamics to validate its effectiveness in a practical biological context.

The contributions of this research can be articulated as follows:

- The development of a systematic framework for training physics-informed neural networks (PINNs) on real-world ordinary differential equation (ODE) systems. This framework incorporates a range of customized techniques, including ODE normalization, gradient balancing, causal training, and domain decomposition, to address common challenges in training PINNs with ODE systems.
- Our method includes a comprehensive normalization not only for inputs and outputs but also ODEs, ensuring more stable and accurate training processes in multi-variate multi-equation systems. In this respect, we implement an adaptive re-weighting of loss functions that individually adjusts the ODE loss weights, thus ensuring balanced training across physics constraints and data loss. Model training is further enhanced by incorporating a 3-phase progressive learning approach that respects temporal causality. This begins with data fitting for initialization, followed by cumulative training across sub-domains of increasing size, and informed neural networks guide mechanistic modelling from sparse experimental data finally tuning across the entire domain. In addition, we simplify domain decomposition, enforcing equality of boundary values at the interface between sub-domains and as a result, avoid extensive calculations to achieve continuity in high-order derivatives.
- A robust 2-step validation is carried out, firstly through an ablation study involving the Lorenz system and secondly, using mosquito population dynamical modelling, to validate the effectiveness of our approach and demonstrate its potential for continued study.

The remainder of this paper is structured as follows: section Related Work reviews the literature relevant to our study; section Methods introduces our PINN framework as applied to a system of ODEs and including our proposed extensions; in section Ablation Study with Lorenz System, an ablation study using the Lorenz system is used to evaluate the effectiveness of each component in our framework; section Case Study-Based Validation examines the wider impact and applicability of our approach by modelling mosquito populations; and finally in section Discussion and Future Work, we conclude with some limitations of our current methods with suggestions for future work in this important research area.

Related Work

Mosquito Population Dynamics Modelling

The main methods that are currently being used in mathematical modelling of mosquito dynamics are linear and nonlinear systems of coupled ordinary (ODE) [6–9, 11] or delayed differential equations (DDE) [10, 54, 55]. These models are compartmental, stage-structured models which divide the population into sub-groups corresponding to the developmental stages of the mosquito vector. They typically include at least four compartments separating the aquatic or immature stages (egg, larva and pupa) from the airborne adult stages, but are usually more complex, including additional adult stages to accurately simulate resting, feeding and gestating females, as well as density dependent competition within the immature stages [56]. The rate at which individuals progress from one stage to the other is simulated by the species-specific development and mortality rates, which depend on micrometeorological variables including air temperature, precipitation and relative humidity, as well as a complex set of interactions between the individuals of the same and competing species, breeding site availability etc. [9, 57–59]. These dependencies introduce temporal constraints to the system in terms of stiffness which has an effect on the overall stability and the numerical integration [60–62]. The magnitude of the stiff problem is defined by the time scale and variability of the driving biotic and abiotic processes. This determines the choice of the time-differencing scheme used to solve the equations, number of steps, local accuracy and length of the numerical integration [61, 62]. A common issue in current approaches is the lack of experimental data for the accurate calibration of the development and mortality parameters, as well as applicability to locations with different ecoclimatic settings [63]. Although there has been recent research on the use of physics-informed neural networks in the field of biological systems [64, 65], this is, to the best of our knowledge, the first paper to investigate the feasibility of applying PINNs to an ODE mosquito population dynamics model. In this study, we aim to provide a first step toward an integrated deep learning vector population dynamic modelling framework.

Normalization

Normalization is a crucial yet often overlooked step in the training of PINNs. In [53], researchers employed a strategy that involved dividing the input domains and applying individual input normalization alongside a unified global output normalization within their model computations. In [66], the authors suggested not only normalizing the inputs and outputs of the models but also non-dimensionalizing the differential equations integrated into the objective functions. In both [34, 65] and [64], both approaches added input- and output- scaling layers that multiply the inputs and outputs with their average magnitudes. However, this is carried out at the model level and thus, still affects the objective functions and training efficiency. Despite recognizing the advantages of these approaches, there remains a lack of a systematic methodology for normalization, particularly in the context of dynamical systems characterized by multivariate system states and numerous differential equations. To address this in our research, we developed a normalization procedure that applies the MIN-MAX scheme to both the inputs and outputs of the neural networks, while appropriately transforming the system of ODEs. This procedure aids not only in aligning with the assumptions of model initialization but also in addressing the challenges associated with multi-scale and stiff issues commonly encountered in training PINNs.

Loss Re-weighting

PINNs operate as a multi-task learning framework that incorporates distinct losses for data fidelity and adherence to physical laws. The different scaling and convergence rates of these losses can lead to imbalances, potentially direct the model towards incorrect solutions as one objective may disproportionately influence the training process. A common solution is the re-weighting of losses to achieve a more balanced training. [36] demonstrated that one of the primary training pathologies in PINNs is the imbalance in gradients propagated from the different losses. These authors proposed an adaptive method that adjusts the weights of the losses based on the ratio between the maximum gradient magnitude of the physics task loss and the mean gradient magnitude of the data task loss, with respect to the model parameters. The authors in [66] used a similar approach utilizing the ratio of the L_2-norm of the gradients instead while in [41], researchers opted to balance the variances of the gradients instead. In [42], they applied the Neural Tangent Kernel to demonstrate that losses from physical laws converge substantially faster than those from initial or boundary conditions, proposing an algorithm to equalize the convergence rates by monitoring the kernels of the losses. We differ to the above approaches, in that while we adopt the strategy proposed by [36], we did so with several modifications. Similar to [34], we set the weight for the data loss at a fixed value of 1.0 and adjusted the weights for the physics law losses, which exhibited more significant variability. And we assigned individual weights to different differential equations, providing the flexibility needed to accommodate the diverse scales and behaviors of these equations.

Collocation points

Collocation points (residual points), where physics constraints are minimized, are traditionally selected uniformly at random across the domain. However, this uniform approach may not be optimal for systems exhibiting steep derivatives. A method known as residual-based adaptive refinement (RAR) proposed by [43] enhances this process by adding new collocation points with the highest differential equation residuals every few iterations, enabling models to focus adaptively on the most challenging areas during training. In [44], the authors adopt a similar strategy by sampling collocation points based on a probability distribution proportional to these residuals. Elsewhere [45], researchers used importance sampling technique to derive a distribution proportional to the 2-norm of the gradient of the loss function, approximated by the loss value to lower computational demands. In [46], they also derived an improved collocation point distribution but use a generative deep learning model to approximate the distribution.

In [48], the authors implemented a sequential training approach, dividing the input domain into subdomains and using predictions from one as initial conditions for the next. Conversely, researchers in [49] leverage predictions from all prior subdomains to enable a single global approximation network. A different approach is presented in [47, 67] where the authors introduce a progressive learning approach with residual points drawn uniformly from a dynamically expanding subdomain, starting from a single point and growing to cover the desired domain. This way, the training process starts by solving the ODEs at earlier in time before moving to what happens later. This technique respects the time causality essential for accurately predicting dynamical systems' evolution. The authors in [50] also emphasize time causality by weighting the residuals to prioritize earlier time points.

We build on the methods presented in [47, 67] but crucially add two new steps: one for initial data fitting to improve model initialization (such as [34, 64]) and include a final step which tunes across the entire domain.

Domain Decomposition

When dealing with extremely large input domains, convergence in training PINNs can be particularly challenging. A domain decomposition approach addresses this by dividing the domain into smaller subdomains and training PINNs for each. To maintain solution continuity and smoothness across these subdomains, additional objective functions, known as interface losses are integrated into the optimization. In [51], the authors introduce two interface conditions: one ensures the alignment of solution values at the interfaces, while the other enforces the consistency of conservation laws across these boundaries. For inverse problems, these conditions also extend to parameter values at the interfaces. The authors in [52] further expanded on these interface conditions to accommodate arbitrary differential systems by ensuring the continuity of the differential equations at the interfaces. In [53, 68], they adopt an implicit approach by employing a gating function. For real-world applications where a high degree of smoothness in the solution is negligible, we simplify the approach by [52]. Instead, we opt to implement only the value enforcement at the interfaces, which serves both as the initial condition and as a means to ensure continuity across the subdomains.

Methods

In this section, we formulate the problem and over a number of steps, present our proposed methodology which creates a solution for problems employing Ordinary Differential Equations (ODEs).

PINN Structure

Consider $u(t) = \left(u^{(1)}, u^{(2)}, \ldots, u^{(V)}\right)$ as a V-dimensional vector representing the state of a dynamical system at any given time t, where t ranges from 0 to T. This dynamical system u is governed by a set of F ODEs as shown in Eq (1), where ...

$$\frac{du}{dt} = f^{(i)}\left(t, u, \theta\right), \quad i = 1, 2, \ldots, F. \tag{1}$$

In these equations, the system's behavior over time is shaped by a set of P parameters $\theta = \left(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(P)}\right)$, which might or might not be known in advance. The functions $f^{(i)}$ are known functions defining the system's dynamics. Suppose that we have some observations of the system at different times, $\mathcal{D}_u = \{(t_1, u_1), (t_2, u_2), \ldots, (t_j, u_j), \ldots\}$. Our objective is to find a solution u and possibly θ that simultaneously matches these observations and is consistent with the ODEs.

In the standard PINN framework [29], a neural network U (parameterized by W_U), is used to approximate the solution u. This network, as a function defined in $[0, T]$, tries to estimate u at any given time t, with W_U being the trainable parameters of the network. In inverse problems where a few or all the parameters θ is not available, we can use a neural network, $\Theta^{(l)}$, to predict the unknown values θ_l. For the sake of simplicity, we denote the set of all parameters, including the known or ones to be learnt by neural networks, as Θ and regard it as neural networks. The parameters Θ, parameterized by W_Θ, are also defined as functions in the domain $[0, T]$. If we let $W = \{W_U, W_\Theta\}$, then the task now becomes an optimization problem when determining the parameters W as shown in Eq (2), which aims to minimize the multi-task objective function defined in Eq (3).

$$W = \text{argmin}_W \mathcal{L} \tag{2}$$

$$\mathcal{L} = \mathcal{L}_{data} + \frac{1}{F}\sum_{i=1}^{F} \lambda_i \mathcal{L}_{f^{(i)}} \tag{3}$$

where

$$\mathcal{L}_{\text{data}} = \frac{1}{N_u} \sum_{(t_i,u_i)\in\mathcal{D}_u} (U(t_i) - u_i)^2 \tag{4}$$

$$\mathcal{L}_{f^{(i)}} = \frac{1}{N_f} \sum_{j} \left| \frac{dU}{dt} - f^{(i)}\left(t_j, U(t_j), \Theta(t_j)\right) \right|^2 \tag{5}$$

Where $U(t_j), \Theta(t_j)$ are the output values of the neural networks valuated as t_j. By minimizing the objective $\mathcal{L}_{\text{data}}$, we decrease the discrepancy between the network predictions and the observed data. Likewise, by minimizing the residuals with $\mathcal{L}_{f^{(i)}}$, the neural network U aligns with the differential equations at low errors. Here, λ_i are balancing factors between fitting to the data and adhering to the dynamics, and N_f is the number of residual points randomly sampled from a distribution μ, typically uniform, in the domain $[0, T]$. We resample the residual points every step to ensure the losses are minimized everywhere in the entire period. Overall, by minimizing this overall loss function $\mathcal{L}$, the network can fit observed data while also approximately follows the dynamic rules at the same time.

The loss function is minimized using gradient-based algorithms. The methods iteratively update the parameters W in the directions of reducing the loss function, based on its gradients with respect to the parameters, namely, $\nabla_W \mathcal{L}$. Specifically, the updates are performed according to the following rule

$$W_{\text{new}} = W_{\text{old}} - \eta \nabla_W \mathcal{L} = W_{\text{old}} - \eta \left(\nabla_W \mathcal{L}_{data} + \sum_{i=0}^{F} \lambda_i \nabla_W \mathcal{L}_{f^{(i)}} \right) \tag{6}$$

where η is the learning rate, which dictates how big of a step to take in the direction opposite to the gradient. To carry out this optimization, the Adam algorithm [69], a variant of the gradient descent algorithm that has been widely used on training PINNs, is used. The calculations of gradients, whether it is the network U with respect to time t or the loss function $\mathcal{L}$ with respect to W, can rely on automatic differentiation supported by popular deep learning frameworks such as Pytorch [70], Tensorflow [71] or JAX [72].

According to the universal approximation theorem [73], multi-layer perceptrons (MLP) [74] can approximate any continuous function on a given domain provided the MLP has sufficient complexity, as represented by the number of hidden layers and parameters in those hidden layers. It is shown that challenges in PINN may not be due to the capabilities of MLP [48]. For an MLP with L layers, the mathematical formulation that describes its operation, layer by layer, is presented in Eq (7) where x is the input to the neural network, $h^{(l)}$ is the hidden state at layer l, $W^{(l)}$ and $b^{(l)}$ are the parameters of the layer l, and σ is the activation function which provides non-linearity for the model.

$$\begin{aligned} h^{(0)} &= x \\ h^{(l)} &= \sigma(W^{(l)} h^{(l-1)} + b^{(l)}), l = 1, \ldots, L-1 \\ h^{(L)} &= W^{(L)} h^{(L-1)} + b^{(L)} \end{aligned} \tag{7}$$

The GELU activation function [75] is employed for its smooth properties which is essential for differential problems, offering an advantage over the ReLU function. For the initialization of the MLP's weights and biases, the Glorot scheme [76] is utilized.

The "vanilla" PINN framework described above can approximate well when the ODE systems are relatively simple. However with more complex systems, especially ones that exhibit extreme stiffness, chaotic and multi-scale behavior, the basic setup tends to have difficulties in converging to a satisfactory local minimum. For this reason, we now proceed to describing our approaches to aid PINN training for both forward and inverse problem involved ODE systems.

ODE Normalization

Under normal circumstances, NNs' output is in the range [-1,1] and so for output values outside this range, data normalization plays a crucial role in the machine learning workflow, ensuring that both input and output variables remain within a reasonable range to enhance model convergence and accuracy. In the PINN context, this normalization process requires careful consideration, as any transformation results in modification to the corresponding ODE systems. In [34, 53], the authors consider normalization and de-normalization as parts of the overall model. However, this type of approach retains the original variable scales of the variables, potentially leading to significant imbalances in the objective function. [66] suggests normalizing the differential equations as well but its generalization is omitted. In this paper, we propose a systematic approach to do normalizations using PINNs involving ODEs. In particular, we utilize the MIN-MAX normalization scheme for both input and output variables, i.e. t, u and θ. To normalize the input variable, i.e. time t, we employ the transformation shown in Eq (8), where $[T_{\min}, T_{\max}]$ represents the time domain in which the models are trained. This will have the effect of normalizing the time variable t to the range of $[-1, 1]$.

$$t' = 2 \cdot \frac{t - T_{\min}}{T_{\max} - T_{\min}} - 1 \tag{8}$$

To normalize the outputs from the neural networks, we define $\mathfrak{L}_u, \mathfrak{U}_u, \mathfrak{L}_\theta$, and $\mathfrak{U}_\theta$ as the lower and upper bounds for u and θ, respectively. When the difference between the lower and upper bounds is small, we adjust the bounds around the mean as $\mathfrak{L} = \min(\mathfrak{L}, \mathfrak{M} - 1)$ and $\mathfrak{U} = \min(\mathfrak{U}, \mathfrak{M} + 1)$ where $\mathfrak{M} = \frac{\mathfrak{L}+\mathfrak{U}}{2}$ is the middle point. It is important to note that these bounds are specific to each dimension in the dynamical system. These bounds can be estimated through collected data, inferred from domain knowledge, or estimated through approximate simulations when data is scarce. The normalization of the variables u and θ at a normalized time t' is then obtained using Eqs (9) and (10) respectively.

$$u'(t') = \frac{u(t') - \mathfrak{L}_u}{\mathfrak{U}_u - \mathfrak{L}_u} \cdot 2 - 1 \Longleftrightarrow u(t') = (u'(t') + 1)\frac{\mathfrak{U}_u - \mathfrak{L}_u}{2} + \mathfrak{L}_u \tag{9}$$

$$\theta'(t') = \frac{\theta(t') - \mathfrak{L}_\theta}{\mathfrak{U}_\theta - \mathfrak{L}_\theta} \cdot 2 - 1 \Longleftrightarrow \theta(t') = (\theta'(t') + 1)\frac{\mathfrak{U}_\theta - \mathfrak{L}_\theta}{2} + \mathfrak{L}_\theta \tag{10}$$

Following the normalization above, the transformed inputs and outputs, t', u' and θ', now vary within the range [-1, 1]. Thus, instead of having neural networks approximating u and θ, it is instead more beneficial to use U and Θ as surrogate models for u' and θ'. Following the normalization transformation in Eqs (9) and (10), it is necessary to adapt the loss functions accordingly. When computing $\frac{du'}{dt}$ and applying Eq (1), we arrive at the result shown in Eq (11).

$$\frac{du'}{dt} = \frac{2}{\mathfrak{U}_u - \mathfrak{L}_u}\frac{du}{dt} = \frac{2}{\mathfrak{U}_u - \mathfrak{L}_u} f^{(i)}\left(t, u, \theta\right) \tag{11}$$

Instead of using objective functions (4) and (5), we now consider the alternative Eqs (12) and (13) as these will provide re-scaling of the losses to the magnitude order of u and $\frac{du}{dt}$, which are more similar to each other than the original objective functions are.

$$\mathcal{L}_{\text{data}} = \frac{1}{N_u} \sum_j \left(U(t'_j) - \frac{u_j - \mathfrak{L}_u}{\mathfrak{U}_u - \mathfrak{L}_u} \cdot 2 - 1 \right)^2 \tag{12}$$

$$\mathcal{L}_{f^{(i)}} = \frac{1}{N_f} \sum_j \left| \frac{dU}{dt} - \frac{2}{\mathfrak{U}_u - \mathfrak{L}_u} f^{(i)} \left(t_j, (U(t'_j) + 1) \frac{\mathfrak{U}_u - \mathfrak{L}_u}{2} + \mathfrak{L}_u, (\Theta(t'_j) + 1) \frac{\mathfrak{U}_\theta - \mathfrak{L}_\theta}{2} + \mathfrak{L}_\theta \right) \right|^2 \tag{13}$$

The modifications to the objective functions, as detailed above, not only trying to balance the data loss and ODE terms, and also across the ODE components. This re-scaling is particularly beneficial in scenarios where variables exhibit significant differences in scale, arising from their inherent characteristics or the units used for measurement. By normalizing these variables, we prevent any single variable from dominating others, balancing the impact of each terms the training process and hence enhancing the convergence. This approach is also aligned with the assumptions of Glorot initialization for neural networks [76]. However, despite these normalizations, $\frac{dU}{dt}$ might not be completely re-scaled and could still reflect the inherent stiffness of the system. Further measures need to be taken, such as domain decomposition and weight re-balancing discussed in other subsections.

Gradient Balancing

As this is a multi-task problem, we have different objective functions (e.g. Eq (3)) as we are measuring different constraints. Previous research [36] highlighted that one point of failure in PINN training is the imbalance in gradients $\nabla_W \mathcal{L}_{data}$ and $\nabla_W \mathcal{L}_{f^{(i)}}$ in the update rule (6). It is observed that the differential equation residual loss dominates the overall loss due to the *stiffness* of the system. This makes the model prioritise optimizing the ODE constraint over matching the initial, boundary condition or data observations, leading the model to converge to a trivial solution, i.e. the *null solution*, violating conditions for the data. We address this issue by adopting a similar approach to [36] where balancing adjusts the weights λ in the objective function based on statistics from the gradients $\nabla_W \mathcal{L}_{data}$ and $\nabla_W \mathcal{L}_{f^{(i)}}$. We extend this further to ODE systems by separately and individually assigning and adjusting the weights to each and every differential equation. In objective functions (3) and (6), the weights for data are set to 1 with component weights λ_i each adjusted individually, to ensure that every equation in the system is given equal *importance* while aligning with the *importance* given to data. This extension to previous work is crucial because the scale and complexity of each equation are different and require different treatment during training.

Algorithm 1 gradient_balancing()

```
Require: step ← current training step; $\alpha$ smoothing factor; updates are made every $N$ steps
Ensure: re-calculating $\lambda_i$ such that gradients from different loss terms are balanced
  if step = 0 then
      Initialize $\lambda_i \leftarrow 1, \forall i = 1, \ldots, F$
  end if
  if step mod$N = 0$ then
      Compute $\hat{\lambda}_i$ by
          $$\hat{\lambda}_i \leftarrow \frac{\overline{|\nabla_W \mathcal{L}_{data}|}}{\max\left\{\left|\nabla_W \mathcal{L}_{f^{(i)}}\right|\right\}}, i = 1, \ldots, F \qquad (14)$$
  where $\overline{|\nabla_W \mathcal{L}_{data}|}$ is the average of the absolute gradients over all model parameters $W$.
      Adjust the weights $\hat{\lambda}_i$ by
          $$\lambda_i \leftarrow \alpha \cdot \lambda_i + (1 - \alpha)\hat{\lambda}_i, i = 1, \ldots, F \qquad (15)$$
  end if
```

This gradient-balancing algorithm is described in Algorithm 1, where all weights λ_i are initialized to 1 and are updated every N steps. We compute the weight $\hat{\lambda}_i$ by calculating the ratio between the mean of absolute values of gradient $\nabla_W \mathcal{L}_{data}$ and the maximum of the absolute values for the gradients of $\nabla_W \mathcal{L}_{f^{(i)}}$. Due to the stochastic nature of gradient descent, the weight from these calculations can be highly volatile and thus, we update the weights λ_i using the moving average formula in Eq (15). The recommended values for hyper-parameters α and N in the original study [36] are $\alpha = [0.5, 0.9]$. However, we set $N = 100$ and tune α in extreme cases $\alpha = 0.99, 0.9, 0.5$, or set $N = 1$ with $\alpha = 0$.

Causal Training

When training PINNs, it is important to consider the *order* of causal effect [50], especially when dealing with problems where data is quite sparse. Usually, PINN models are trained to follow differential equations at every point in the input domain at the same time. However a problem can arise where the model starts to follow these rules at later values for t but it has properly adhered to the rule at an earlier point t. This discrepancy leads to a situation where efforts to conform to ODEs at a one point cause violations at other points during the learning process. Furthermore, if the penalty for not following ODEs at the earlier point outweighs the fitting at the later, the model may get stuck in a local minimum, unable to satisfy ODEs over the entire period anymore. To solve this, a *causal* approach to training is recommended. In a causal training, the task of meeting the data conditions is given priority while ensuring that the model complies with ODEs at the earlier values t before attempting compliance later values t.

In this study, we divide the training process into three phases: data fitting, progressive causal training, and final tuning.

- First, in the data fitting phase, we focus on making the model match the data conditions but training only with the data loss term $\mathcal{L}_{\text{data}}$. This helps set a good starting point for the model so that later, it can follow the rules of differential equations more easily.

- In the progressive causal training phase, we take the *growing-interval* approach used in [67]. We *gradually* teach the models to follow the differential equations starting from a small interval and slowly covering more domain as the training progresses. Both data loss term and ODE loss terms are included but the ODE residual points for $\mathcal{L}_{f^{(i)}}$ are drawn from a growing interval. If N_2 is the number of update steps in the second phase, then at the n_2-th step, residual points are uniformly drawn from the interval $\left[0, \frac{n_2}{N_2} \cdot T\right]$.
- In the final tuning phase, we train the model using both data loss and ODE losses, with the ODE residual points drawn at random with uniform distribution across the entire domain. The goal here is to refine and improve the solutions the model has learned so far, ensuring it follows the differential equations more accurately over the entire period.

The training process mainly happens during the second and third stages, with the first stage being the shortest. This is because neural networks can quickly learn to fit the given data too well. We usually limit the first phase, where the model learns to match the initial data, to between 5,000 and 10,000 steps. In the progressive causal phase, the model gradually learns to solve the differential equations over an expanding area. This phase takes longer because the model must successfully reduce its errors to a required level before it can extend its learning to new areas. For systems that do not reliably converge, the number of steps for this phase should be sufficiently high. We set this phase to last between 50,000 and 100,000 steps to ensure thorough learning. Then, in the final tuning phase, the early stopping approach is adopted. The model's performance is checked every 1,000 steps with an evaluation loss, where all weights λs are set to 1.0. If no improvement is recorded in the model's performance after several checks, the training ends. This ensures the model is as accurate as possible without time spent on unnecessary training.

Domain Decomposition

One of the issues when working with PINNs is that the time series may be too large. Domain decomposition is a effective way to train PINN when the solution is overly complex for a particular time interval or the target input domain is too large, which is often the case in real world applications. The method is particularly useful in the extrapolation forward problem where there are little or no data in the period of interest, or the data condition (i.e. initial or boundary condition) needs to rely on predictions from some other domain. In such cases, the shape of the solution relies heavily on training the models to fit the ODE constrains, which is a difficult task due to issues such as causal effect violation [50] or gradient imbalance [36]. The decomposition of domains reduces the domain the models are trained on and subsequently, reduces the complexity of the optimization task.

In particular, the approach divides the input domain into S non-overlapped subdomains, $D_s = [T_{s-1}, T_s], s = 1, \ldots, S$, with $T_1 = 0, T_S = T$. In each subdomain D_s, a neural network U_s is defined, with the overall solution is defined in Eq (16). With this definition, the goal is to train U_s to approximate the solution u in the subdomain D_s.

$$U(t) = \sum_{s} \mathbf{1}_{D_s}(t) \cdot U_s(t), \forall t \in [0, T] \tag{16}$$

Let $D_s^* = [T_{s-1} - O, T_s + O]$ be the extended subdomain of D_s, O be 50% of overlapped size. We treat each subdomain D_s^* as a separate PINN problem, where the model is trained using the framework described in section Methods, including the normalization and gradient balancing measure. Starting from the first subdomain D_0, the model is trained with the initial condition data $\mathcal{D}_u$ provided by users. For subsequent subdomains D_s^*, the model is trained with data $\mathcal{D}_u$ generated by the previously trained model U_{s-1} in the overlapped domain $[T_{s-1}, T_{s-1} + O]$. The volume of data generated could be unlimited, we set the number from 10 to 100, depending on the granularity and the continuity across subdomains required. This process is repeated until the entire domain is covered. The final solution would be the combined predictions in Eq (16).

This divide-and-conquer scheme offer several advantages in training PINNs: it reduces the complexity of the overall problem into many smaller less-complex problems; it reduces the explosion of gradients which has always been an important issue when training PINN; it allows optimized customization of models in each subdomain, enhancing the individual and overall convergence and accuracy.

In this study, we experiment with domain decomposition for the forward problem only. However, this approach could be expanded to inverse problem by adding similar interface conditions on ODE-parameter neural networks.

Ablation Study with Lorenz System

As different NN configurations or structures have been constructed for this task, we now evaluate the efficacy of the proposed PINN methodologies utilizing the Lorenz system, which consists of three coupled, nonlinear differential equations and is the common choice when evaluating ODEs [77]. This system is a classic example of chaotic behavior and is described in Eq (17). Here, x, y, and z represent the state variables of the system at any given time t while σ, ρ, and β are physical parameters of the system. We define two temporal domains for our experiments: a shorter duration with a maximum time of $T = 2.0$ and a longer duration extending to $T = 40.0$.

$$\begin{cases} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{cases} \tag{17}$$

The shorter duration is employed to assess the impact of techniques such as ODE Normalization, Gradient Balancing and Causal Training on the model's performance. Conversely, the longer duration is utilized to investigate the effectiveness of Domain Decomposition in managing extended temporal intervals. For the generation of both training and ground-truth datasets, we employ a well-established numerical method [78] for solving ODE systems, implemented in Python's Scipy library [79].

Normalization is a crucial preprocessing step in machine learning. In our study, we utilize ODE normalization as a baseline model and integrate various additional techniques to evaluate their impact on model performance. We investigate the following models:

1. $\text{OdePINN}_{\text{orig}}$: Original framework without additional techniques.
2. $\text{OdePINN}_{\text{baseline}}$: ODE Normalization only. This model represents a baseline model for the more complex models 3-6 below.
3. $\text{OdePINN}_{\text{grad}}$: ODE Normalization combined with Gradient Balancing.
4. $\text{OdePINN}_{\text{causal}}$: ODE Normalization combined with Causal Training.
5. $\text{OdePINN}_{\text{grad+causal}}$: ODE Normalization, Gradient Balancing, and Causal Training combination.

6. OdePINN$_{\text{grad+causal+domain}}$: The model combines ODE Normalization, Gradient Balancing, Causal Training, and Domain Decomposition.

Where a bespoke technique is not applied, conventional methods are used by default. Specifically, in the absence of ODE normalization, input and output variables retain their original scale; without Gradient Balancing, the weights λ_i are set to a fixed value of 1; without Causal Training, collocation points are uniformly sampled across the domain; and without Domain Decomposition, the model is trained on the entire domain of interest in a single training. In this approach, experiments were conducted on both forward and inverse problems using the Lorenz system, setting a shorter time frame of $T = 2.0$ to test the first five models. The sixth model OdePINN$_{\text{grad+causal+domain}}$ is assessed on the forward problem over a longer duration with $T = 20.0$.

Forward Problem with T=2

Now, we explore how the techniques described earlier work together, particularly on their capacity to extrapolate in forward problem scenarios using the Lorenz system. The data loss condition, $\mathcal{L}_{data}$, involves using only the initial condition as a single data point $(x, y, z) = (1, 1, 1)$ at $t = 0$. The physical parameters are set constant over time, as: $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$. These parameters are all known to the framework.

The framework utilizes an MLP, U, to approximate the solution u. The architecture of U comprises four hidden layers, each consisting of 100 units, with the GELU activation function applied in all hidden layers. For Gradient Balancing, the hyperparameters α and N are tuned across four settings: $(0.99, 100)$, $(0.9, 100)$, and $(0.0, 1)$. The configuration $(0.99, 100)$ is identified as the most effective through this tuning process. In the absence of Gradient Balancing, α defaults to 0 and N to None, maintaining the λ_is at a constant value of 1.0 throughout the training. Causal Training involves three phases: the first phase of 1,000 steps, followed by the second phase of 199,000 steps, and the final phase includes early stopping up to 100,000 steps. Should Causal Training be deactivated, the model bypasses the initial two phases and proceeds directly to the last phase, extending training up to a maximum of 300,000 steps without the implementation of early stopping.

Fig 1 presents the solution approximations produced by the five models, starting with the poorest performing models.

- OdePINN$_{\text{orig}}$ tends to converge towards the null solution due to the dominant factor of ODE loss.
- OdePINN$_{\text{causal}}$ is similar to the *original* model, while capturing the system's dynamics effectively, deviates from the correct initial condition and subsequently converges to the trivial null solution.
- The baseline model manages to obtain the general shape of the solution but has difficulties in matching both the initial condition and the governing differential equations.
- OdePINN$_{\text{grad}}$ achieves a better alignment with the initial condition but it fails to sufficiently satisfy the physics constraints, leading to inaccurate approximation of the correct solution.
- OdePINN$_{\text{grad+causal}}$ demonstrates a close convergence to the true solution, showing superior performance to the other models.

In summary, OdePINN$_{\text{grad+causal}}$ successfully integrates both methodologies, ensuring adherence to the initial condition and minimizing ODE loss. This dual

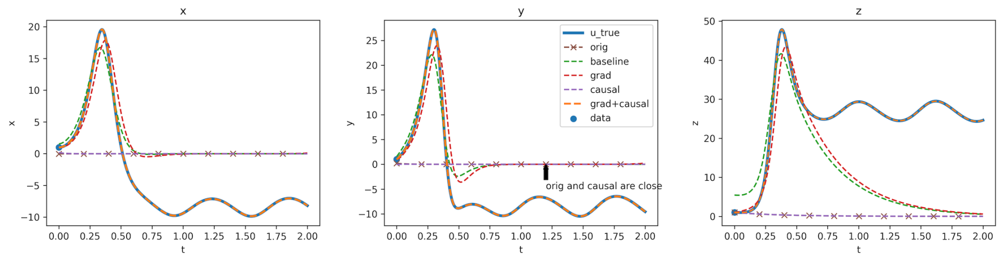

Fig 1. Lorenz ODE system for the forward problem, with U approximation of the system state. The blue line u_true represents the target for each of the 5 models used in the experiment. Graphs illustrate the performance across the x,y and z dimensions.

approach enables the model to more accurately approximate the true solution. These observations are further illustrated in Fig 2. Note that evaluation losses are computed across the entire period, with the weighting coefficients λ_i maintained at 1.0; OdePINN$_{\text{orig}}$ are not shown as the losses are at different scales and its approximation is similar to OdePINN$_{\text{causal}}$. Fig 2a demonstrates that the OdePINN$_{\text{grad+causal}}$ model achieves a significantly low final total loss of $1.6 \cdot 10^{-4}$, which is considerably less than that of the OdePINN$_{\text{causal}}$ at $2.2 \cdot 10^{-3}$. The remaining models display higher losses of 0.046 and 0.056, with the baseline model showing slightly better performance. Comparison of errors across the models indicates that Gradient Balancing substantially enhances convergence towards the data loss term $\mathcal{L}_{data}$. The data losses for OdePINN$_{\text{grad}}$ and OdePINN$_{\text{grad+causal}}$ are recorded at $1.4 \cdot 10^{-8}$ and $8.9 \cdot 10^{-9}$, respectively, which are significantly lower than the approximately 10^{-2} or 10^{-3} observed in the other models. Despite achieving a significantly lower data loss, the total loss of OdePINN$_{\text{grad}}$ is $6.3 \cdot 10^{-2}$, slightly higher than the baseline model's $4.5 \cdot 10^{-2}$, due to a higher ODE loss. This technique enables the neural network to better fit the initial condition but simultaneously makes it harder to satisfy the ODE loss, resulting in higher ODE and overall losses. Greater detail on the usage and analysis of this method is provided in [36]. Ultimately, the root mean squared error between the grad model and the true solution is 11.9, slightly better than the baseline's 12.18, though both remain far from a perfect solution.

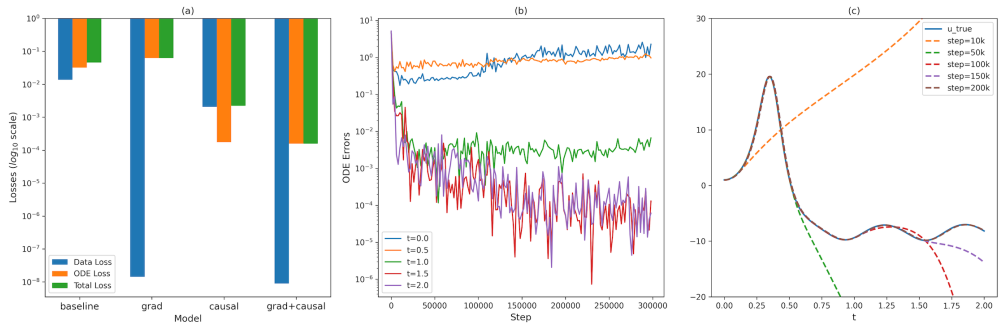

Fig 2. Loss analysis of OdePINN framework with Lorenz system, $t \in [0, 2.0]$. (a) Loss Terms across final models selecting through Early Stopping; the bars are presented in a logscale pointing downwards, with the lower the value the better the performance. (b) ODE errors of the OdePINN$_{\text{grad}}$ model at different time t during the training, motivating the need for Causal Training; (c) The x-value Approximation Solution U of the model OdePINN$_{\text{grad+causal}}$ at different steps during the training.

However, the Gradient Balancing technique on its own does not address the issue of causal effects: Causal Training is essential in this case. Gradient Balancing averages gradients from different times, leading to scenarios in Fig 2b, where at later time $t = 1.5$ or $t = 2.0$, the model can quickly minimize the physics constraints to a negligible level of 10^{-5} at an early stage in the training. Notably, within the domain $t \in [1.0, 2.0]$, the model converges to the null solution (as illustrated in Fig 1), which satisfies the ODE system. However, at earlier times, such as $t = 0$ or $t = 0.5$, the model consistently struggles throughout the training duration. We hypothesize that this premature convergence of the ODE constraints at later times traps the model in a local minimum, preventing it from satisfying the constraints at earlier times and thus resulting in an inaccurate solution approximation. To address this, we employ a Causal Training strategy that trains the physics-laws term using residual points drawn from a progressively expanding interval, starting from the 1,000th to the 200,000th step. This growing interval strategy ensures that the model adheres to the system dynamics at earlier times before progressing to later times, thereby respecting the causal effects. The domain in which the model complies with the differential equations broadens as the training advances, as depicted in Fig 2c. Outside this domain, the model's behavior remains arbitrary. Ultimately, the OdePINN$_{\text{grad+causal}}$ model effectively combines these techniques, effectively minimizing both data and physics losses to achieve a highly accurate solution approximation.

Forward Problem $T = 20$

For a more robust test, we also performed a tougher evaluation attempting to predict all values on a continuous scale between 0 and 20. We demonstrate domain decomposition by solving the Lorenz system over an extended domain, from $t = 0$ to $T = 20.0$. The initial condition is set as $(1, 1, 1)$, with constant, known parameters $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$.

Fig 3 displays the the results when the domain is larger where none of the models achieve a close approximation to the reference solution. The two models OdePINN$_{\text{baseline}}$ and OdePINN$_{\text{grad}}$ demonstrate some level of ability to learn the initial condition and marginally adhere to the ODE equations. However, their errors grow sufficiently large by $t = 0.75$, leading them to diverge significantly from the true solution, eventually converging to a trivial constant solution beyond $t = 3$. Other models, OdePINN$_{\text{orig}}$, OdePINN$_{\text{causal}}$ and OdePINN$_{\text{grad + causal}}$ exhibit a similar pattern as observed in the $T = 2$ experiment, converging to the null solution. The models' inaccuracies can be attributed to the increased domain size, which raises the solution's complexity. Therefore, it was necessary to apply Domain Decomposition to manage the larger domain size.

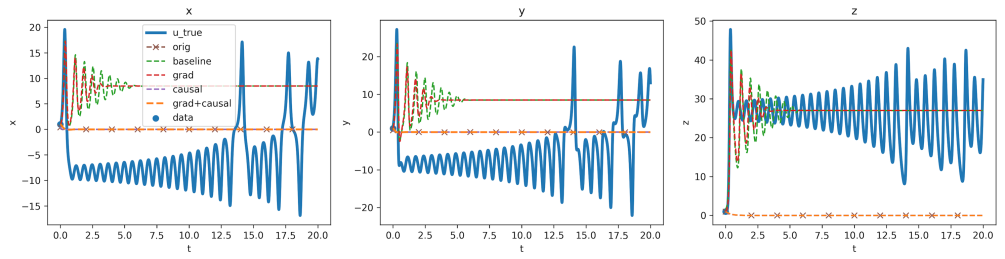

Fig 3. Lorenz ODE system, forward problem, U approximation of the system state from the first 5 models, using the first five models, excluding the Domain Decomposition model.

The domain is divided into 40 subdomains, each with a size of 0.6 and an overlap of 0.05 on both ends. Each subdomain is independently modeled and trained by a distinct neural network. The predictions from the previous subdomain, uniformly distributed over 100 points within the overlapped region, serve as the data conditions for the subsequent subdomain. The hyper-parameters remain identical across all subdomains. The neural network U is structured as an MLP with four hidden layers, each comprising 100 units and utilizing the GELU activation function. The training process is limited to a maximum of 150,000 steps, starting with an initial phase of 5,000 steps dedicated to data fitting, followed by 100,000 steps focused on causal training and the remaining are for final tuning phase. Additionally, the gradient balancing weights, λ_i, are updated every 100 steps, utilizing a smoothing factor of $\alpha = 0.99$.

Fig 4 illustrates the solution approximated by the proposed framework, accompanied by plots of the training losses and Root Mean Squared Error (RMSE) relative to the ground truth data. The model demonstrates a fairly good prediction of the system's evolution. Initially, the RMSE is approximately $5 \cdot 10^{-4}$, but it exponentially increases as t progresses, where the RMSE escalates to 10^{-3} at $t = 8$, to 10^{-1} at $t = 13$, and reaches an error magnitude of 10^{1} by the end of the period. The final four subdomains, as depicted in Fig 4 (12.5 to 20.0), show notable approximation errors. This substantial error accumulation towards the end of the period is understandable given that the Lorenz system is highly sensitive to initial conditions where minor predictive inaccuracies can significantly deviate the future states. As a result, initial training errors rapidly accumulate over time, culminating in an RMSE of up to 10 by the end of the time period. The training data loss remains below 10^{-4} throughout, with many instances dropping to the 10^{-5} level. Furthermore, the errors related to physical constraints are consistently maintained at the 10^{-3} level, indicating that the physical laws are satisfied with ODE loss approaching zero. Notably, the losses peak in regions of the solution characterized by sharp changes, consequently resulting in a steep increase in RMSE. On a positive front, the framework exhibits consistent performance across all subdomains. However, the clear accumulation of errors highlights a potential limitation of the technique, suggesting areas for further improvement.

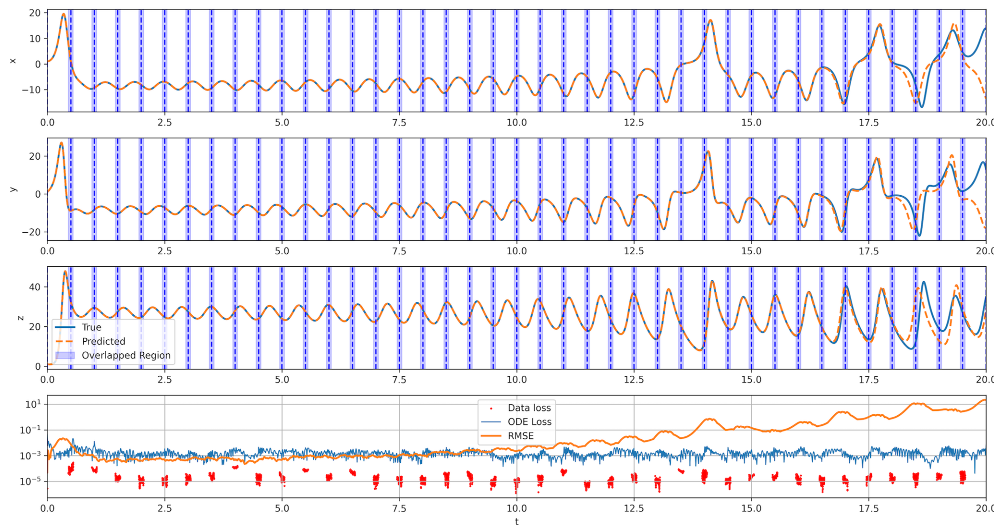

Fig 4. Lorenz ODE system, forward problem, U approximation of the system state, using the domain composition model.

Fig 5 plots the relationship between the number of training steps and the RMSE across a range of subdomains. The training steps increase linearly with the number of subdomains as each subdomain requires approximately 200,000 to 300,000 training steps, equivalent to around 30 minutes on the GPU NVIDIA GeForce RTX 4090 hardware used for all experiments. The figure demonstrates a clear trade-off between the number of subdomains and model accuracy: as the number of subdomains rise, the RMSE decreases exponentially, but at the cost of longer training times. When the number of subdomains is low, such as 1 or 5, the training domain remains too large, causing the model to converge to the null solution, yielding an RMSE of 13.4. With an increase in subdomains, starting from 10, the training domain becomes smaller, reducing the solution complexity within each subdomain. This reduction leads to a RMSE decrease, reaching 4.1 with 10 subdomains and 1.13 with 40 subdomains. These results indicate that cumulative errors can be mitigated by increasing the number of subdomains, allowing the model to focus on improving accuracy at finer levels of detail.

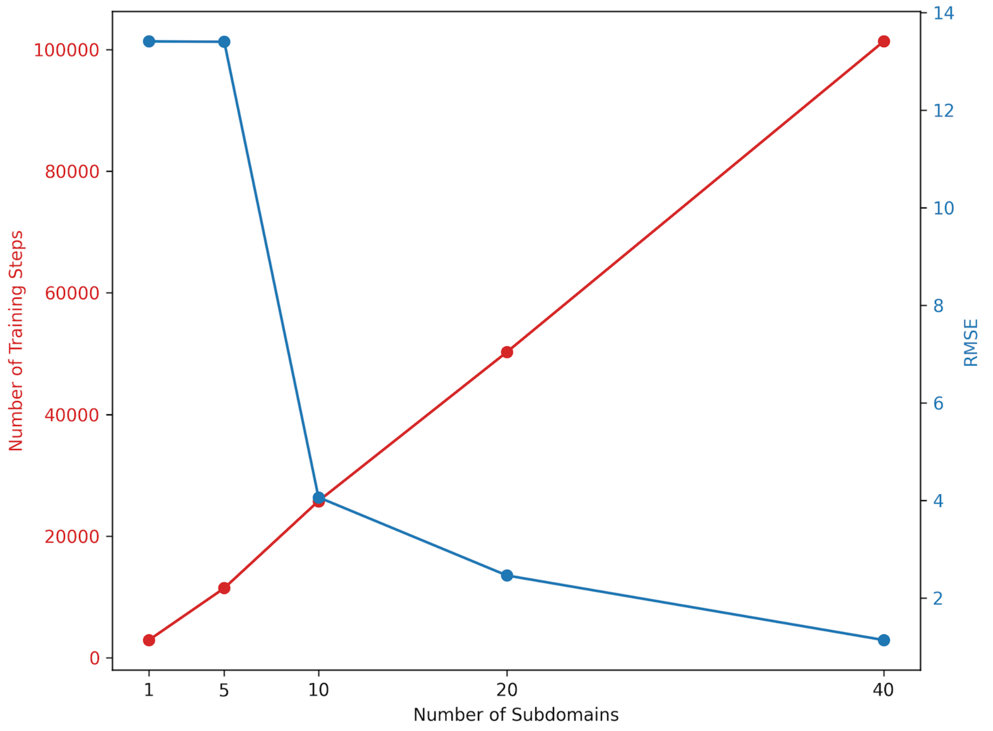

Fig 5. Trade-off between the number of subdomains and accuracy.

Inverse Problem

In this section, we illustrate the application of the proposed framework to an inverse problem scenario, using significantly more data. Here, the framework is employed to predict the physics parameters and simultaneously interpolating the system state from a limited number of observations of the system's state. We conduct these experiments using the Lorenz system over the time domain $t \in [0, 2.0]$, with time-varying physical parameters defined as $\sigma = \frac{10}{2}\sin(2\pi t) + 10$, $\rho = \frac{28}{5}\sin\left(2\pi t + \frac{\pi}{2}\right) + 28$, and $\beta = \frac{8}{3}$. These formulas are unknown to and are to be learnt by the framework. The initial conditions are set to $(1, 1, 1)$, consistent with previous experiments. The dataset for training comprises 21 simulated data points that are evenly distributed across the input domain, resulting in a dataset dimension of $(21, 3)$. Fig 6a presents both the reference solution discussed earlier [78] and the simulated data points.

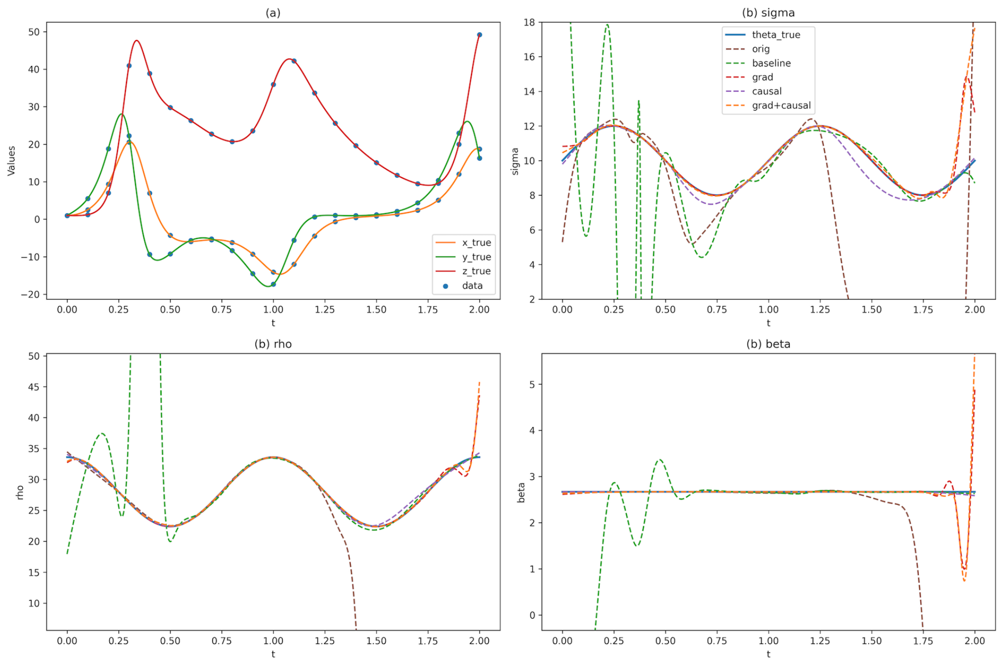

Fig 6. OdePINN framework solves the Lorenz system inverse problem. (a): Ground truth u and data provided to PINN; (b): Θ approximation of the physics parameters (σ, ρ, β).

The configuration of the framework is similar to previously described experiments. The main neural network U retains its structure from earlier experiments, comprising an MLP with four hidden layers, each consisting of 100 units, with GELU activation. Additionally, three smaller, distinct neural networks Θ are implemented to model the three physical parameters of the dynamical system. These networks are each equipped with four hidden layers containing 10 units, using GELU as the activation function for all hidden layers. All networks incorporate the time variable t as a singular input to capture the temporal dynamics of the parameters. In terms of training enhancements, Gradient Balancing is used where the weights of the objective function, λ_i, are adjusted every $N = 100$ steps, employing a smoothing factor of $\alpha = 0.99$. The Causal Training approach involves an initial data fitting phase comprising 10,000 steps, reflecting the increased data availability. This is followed by progressive causal training and a final tuning phase, which are conducted over 200,000 and 100,000 steps, respectively.

Fig 6b displays the dynamical system's parameters as approximated by the neural network Θ. While all models demonstrate an ability to capture the general trends of the parameters, they also exhibit inherent inaccuracies. Notably, the original and baseline models show substantial errors particularly in regions where the derivatives of u with respect to time t are remarkably high, as can be seen within the interval $[0, 0.5]$ or by $t = 2.0$ in Fig 6a. Among the models, OdePINN$_{\text{grad}}$ achieves the most accurate parameter approximations throughout most of the domain. However, it encounters difficulties at the period's end, near $t = 2.0$. In this inverse setting, the system's parameters are not predefined and are subject to variation, leading to non-unique solutions for u and θ in the absence of sufficient data. Consequently, OdePINN$_{\text{grad}}$ tends to favor solutions with smaller magnitude derivatives, which can result in significant discrepancies when compared to the reference solution. Conversely, OdePINN$_{\text{causal}}$ typically exhibits error accumulation towards higher values of t, with noticeable inaccuracies in estimating parameters such as σ and ρ beyond $t = 1.15$, as illustrated in Fig 6b. This pattern of error propagation is a recurring observation throughout our study. The combined model, OdePINN$_{\text{grad+causal}}$, mirrors the performance of OdePINN$_{\text{grad}}$, maintaining reasonable accuracy until the domain's far end. The approximation U by the models can be found in S1 Fig.

Table 1 presents a detailed analysis of the errors associated with both the interpolation of u and the estimation of parameters θ, benchmarked against the ground truth data, errors calculated in metrics presented in S1 Appendix. The *original* model registers highest errors, with RMSE values of 3.9564 for u and 52.4141 for θ, alongside MDAPE of 2.6219 and 24.5114, respectively. Compared to the *original* model, the baseline model shows slight improvements but still incurs relatively high errors. Conversely, OdePINN$_{\text{grad}}$ obtained significant improvements in accuracy, recording the lowest errors across all evaluated metrics for both u and θ. It achieves an RMSE of 0.2525 for u and an impressively low 0.7195 for θ. Additionally, it reports MAE values of 0.0472 and 0.1602, and MDAPE scores below 0.01% for both variables. The causal model records the best RMSE for u at 0.1039, although its performance on other metrics does not reach the levels achieved by the grad model. OdePINN$_{\text{grad+causal}}$, which integrates the approaches of the preceding two models, does not achieve top scores in any specific category but secures the second-best results, including impressive MDAPE values below 0.01% and 0.06%. This final training stage of OdePINN$_{\text{grad+causal}}$ mirrors the training process of the entire OdePINN$_{\text{grad+causal}}$ model, albeit with an improved initialization and a reduced number of training steps.

Table 1. Approximation errors in the inverse problem with Lorenz system where smallest error values are best.

Model	Metric	x	y	z	u	σ	ρ	β	θ
orig		4.4453	4.9325	1.6939	3.9564	5.4294	90.5381	3.8850	52.4141
baseline		3.4995	2.8182	0.6591	2.6219	8.4507	41.5828	1.3683	24.5114
grad	RMSE	0.1011	0.3556	0.2336	$\underline{0.2525}$	0.8727	0.8498	0.2629	**0.7195**
causal		0.1363	0.0949	0.0692	**0.1039**	1.3853	2.4592	0.0740	1.6302
grad+causal		0.0862	0.3760	0.2659	0.2705	1.2593	0.8922	0.3316	$\underline{0.9114}$
orig		1.7002	1.0028	0.3667	1.0232	3.6059	47.0012	1.4000	17.3357
baseline		0.9579	0.6687	0.2430	0.6232	3.1751	10.9855	0.5066	4.8891
grad	MAE	0.0207	0.0759	0.0449	**0.0472**	0.2116	0.2078	0.0611	**0.1602**
causal		0.0844	0.0654	0.0452	0.0650	0.7099	1.3002	0.0472	0.6858
grad+causal		0.0192	0.0904	0.0543	$\underline{0.0546}$	0.2845	0.2590	0.0827	$\underline{0.2087}$
orig		0.0063	0.0064	0.0015	0.0025	0.1271	0.0127	0.0058	0.0200
baseline		0.0019	0.0029	0.0016	0.0019	0.0382	0.0114	0.0086	0.0162
grad	MDAPE	0.0001	0.0002	0.0001	**0.0001**	0.0005	0.0005	0.0003	**0.0004**
causal		0.0004	0.0012	0.0016	0.0013	0.0175	0.0044	0.0074	0.0090
grad+causal		0.0000	0.0001	0.0001	**0.0001**	0.0006	0.0003	0.0007	$\underline{0.0006}$
orig		0.2562	0.2450	0.0703	0.2086	2.7147	16.1675	3.8850	9.7271
baseline		0.2017	0.1400	0.0274	0.1426	4.2253	7.4255	1.3683	4.9955
grad	nRMSE	0.0058	0.0177	0.0097	$\underline{0.0121}$	0.4364	0.1517	0.2629	$\underline{0.3069}$
causal		0.0021	0.0025	0.0008	**0.0019**	0.2210	0.0608	0.0156	**0.1326**
grad+causal		0.0079	0.0162	0.0140	0.0132	0.5519	0.1572	0.3124	0.3772

Bold text highlight the best performing models with respect to a specific metric and $\underline{\text{Underlined numbers}}$ represent the second best performing model.

With an increase in data availability, the preliminary two phases of causal training contribute little to overall improvement in performance. This is mainly because a dataset of sufficient volume can establish a robust initialization, diminishing the relative advantage of the first two phases. On the other hand, OdePINN$_{\text{grad+causal}}$, with its shorter training duration, does not attain performance levels comparable to those of the Gradient-Balancing model.

Case Study-Based Validation

In addition to a theoretical validation, our PINN framework is also validated in the practical environment of dynamical modelling of the mosquito population. Here, our approach is adopted on the ODE-based model of mosquito population dynamics proposed in [80]. The model divides the mosquito life cycle into 10 stages: Egg (E), Larva (L), Pupa (P), Emerging Adults (A_{em}), Nulliparous Bloodseeking Adults (A_{b1}), Nulliparous Gestating Adults (A_{g1}), Nulliparous Ovipositing Adults (A_{o1}), Parous Bloodseeking Adults (A_{b2}), Parous Gestating Adults (A_{g2}) and Parous Ovipositing Adults (A_{o2}). The 10 stages are related via ordinary equations as show in Eq (18), with parameters explained in S1 Table. A diagram illustrating the stages and transitions is available in S2 Fig. System parameters are modelled as a function depending on temperature with data acquired for *Culex pipiens* spieces from [11].

$$\begin{cases} \frac{dE}{dt} &= \gamma_{Ao}(\beta_1 A_{o1} + \beta_2 A_{o2}) - (\mu_E + f_E)E \\ \frac{dL}{dt} &= f_E E - \left(m_L\left(1+\frac{L}{\kappa_L}\right) + f_L\right) L \\ \frac{dP}{dt} &= f_L L - (m_P + f_P)P \\ \frac{dA_{em}}{dt} &= f_P \sigma e^{-\mu_{em}\left(1+\frac{1}{\kappa_P}\right)} P - (m_A + \gamma_{Aem})A_{em} \\ \frac{dA_{b1}}{dt} &= \gamma_{em} A_{em} - (m_A + \mu_r + \gamma_{Ab})A_{b1} \\ \frac{dA_{g1}}{dt} &= \gamma_{Ab} A_{b1} - (m_A + f_{Ag})A_{g1} \\ \frac{dA_{o1}}{dt} &= f_{Ag} A_{g1} - (m_A + \mu_r + \gamma_{Ao})A_{o1} \\ \frac{dA_{b2}}{dt} &= \gamma_{Ao}(A_{o1} + A_{o2}) - (m_A + \mu_r + \gamma_{Ab})A_{b2} \\ \frac{dA_{g2}}{dt} &= \gamma_{Ab} A_{b2} - (m_A + f_{Ag})A_{g2} \\ \frac{dA_{o2}}{dt} &= f_{Ag} A_{g2} - (m_A + \mu_r + \gamma_{Ao})A_{o2} \end{cases} \tag{18}$$

This part of the evaluation comprised two challenges: first, solving the mosquito population dynamics with an initial condition; and second, determining mortality and growth rates over time from available data. We conducted tests under a varying temperature condition, where the temperature changes according to a sine function $\tau = 10\sin\left(2\pi\frac{t}{365}\right) + 10$ with time, t, measured in days.

Forward Problem

For the forward problem, the goal was to solve the mosquito population dynamics using only a single data point at the beginning, known as the initial condition. Specifically, we employ a numerical method [78] for solving ordinary differential equations (ODEs), available in Python Scipy library [79] , to simulate data over three years (t ranging from 0 to 1096 days). We use the data point at $t = 730$ (two years into the simulation) as the starting point and trained a PINN for the period from 730 to $1,096$ days. The simulated data within this period serves as the ground truth solution against which we evaluate the models performance. The advanced techniques presented in the section Methods, including ODE normalization, domain decomposition, gradient balancing and causal training, were implemented.

The time period was separated into 12 subdomains where for each, training was carried out for 10,000 steps to fit the data, followed by 100,000 steps focusing on causal training. Early stopping was used after 100 evaluations without improvement in the final phase. The lower and upper bounds for each domain are acquired from the ground truth data. The solution neural network U was configured as an MLP with 4 hidden layers with 100 units each GELU as activation function for every hidden layer. The subdomains are trained sequentially, with the predictions in the overlapped domain of previous subdomain models used as the initial data condition for the next domain.

The results for solving the mosquito population dynamics as a forward problem are shown in Fig 7, with error measurements provided in Table 2. The trained neural network shows impressive performance in extrapolating and providing a fairly accurate approximation of the solution in Fig 7. However, towards the end of the time period, accumulated error can be seen in Ag1 and Ag2 stages of mosquitoes development.

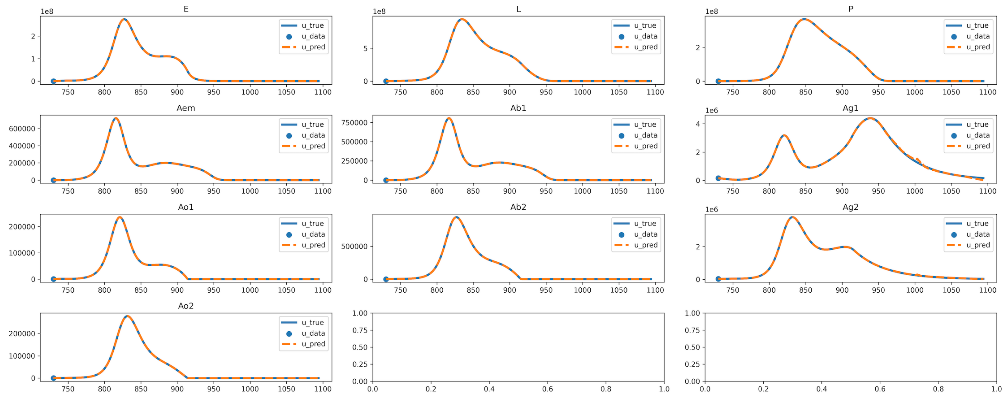

Fig 7. Mosquito ODE system, forward problem, U approximation of the solution.

Table 2. Approximation Errors for Mosquito ODE Solution.

Stage	RMSE	MAE	MDAPE	nRMSE
E	56107.947877	28619.218663	0.001900	0.000407
L	144952.348476	82367.593703	0.000759	0.000308
P	243295.690709	123936.043785	0.002170	0.001325
Aem	398.308537	175.715612	0.001821	0.001106
Ab1	342.155798	161.987210	0.001782	0.000850
Ag1	42095.722155	20403.707941	0.002158	0.019350
Ao1	34.564301	16.196719	0.003921	0.000293
Ab2	102.456176	50.194130	0.002831	0.000217
Ag2	7853.480625	4466.414135	0.001567	0.004155
Ao2	19.229490	9.978838	0.002394	0.000138
Overall	92296.312508	26020.705074	0.001918	0.006291

In Table 2, the overall RMSE is quite high at 92,296 (organisms). Errors vary significantly across different life stages of mosquitoes, ranging from as low as 19 in $Ao2$ stage to as high as 243,295 at the P stage, most likely due to the different scales of these life stages The MAE shows a similar range of variation, from 10 ($Ao2$ stage) to 123,936 (P stage), with an average of 26,021. Despite these high values for RMSE and MAE, the Median Absolute Percentage Error (MDAPE), is relatively low at an average of 0.19 %, which indicates a good performance in general from a machine learning perspective. MDAPE tends to be lower for stages with larger scales, suggesting that stages with larger scales are more tolerant of minor errors. For instance, the smallest MDAPE is 0.0759% for the L stage, which has the second highest values in both RMSE and MAE. Conversely, the highest MDAPE, at 0.3921%, occur in the $Ao1$ stage, which has relatively lower RMSE and MAE.

Inverse Problem

In solving inverse problems, the goal was to predict 10 specific parameters using available data. Three parameters (γ_{Aem}, γ_{Ab} and γ_{Ao}) are treated as constants and are represented by learnable parameters that remain constant over time. The remaining seven parameters (f_E, f_P, f_L, f_{Ag}, m_L, m_P and m_A) vary over time with neural networks using time as input to estimate these parameters. This experiment was conducted under the same conditions and temperatures as previous experiments but here, it was necessary to generate simulation data over a three-year period, from day 0 to day 1096, using the numerical method in [78]. For the data condition loss, we used daily data from $t = 730$ to $t = 1096$, totaling 367 days times 10 data points. Both the solution u and the ODE parameters θ were obtained from the simulation data as ground truth.

With this setup, the high data volumes allow for the use of simpler techniques. Due to significant differences in the number of instances across various stages, ODE normalization and gradient balancing are applied. Techniques such as domain decomposition and causal training were not used (set the number of subdomains to 1 and the number for the first two phases to 0), as the data condition with sufficient data provides a solid base for the framework's convergence. The boundaries for the stages and ODE parameters were determined based on the data from the simulations. For the neural network solution U, an MLP with 4 hidden layers, each containing 100 units, and utilizing the GELU activation function, was created. The constant parameters are represented by individual trainable weights. Time-dependent parameters, on the other hand, are each represented by a smaller MLP, consisting of four hidden layers with 10 units per layer and also using the GELU activation function.

Fig 8 shows the system parameters learnt from the PINN framework. The plots generally illustrate accurate parameter approximations although there are noticeable inaccuracies, particularly for parameters m_L and m_P within the domain $[943, 1064]$. The error metrics detailed in Table 3, reveal an average RMSE of 0.132615, with m_L and m_P contributing the highest errors at 0.286959 and 0.291548, respectively. The lowest RMSE values are for f_{Ag} and f_L at 0.002769 and 0.002816, respectively. The MAE closely follows the RMSE trends, averaging at 0.044757. The highest MAE values correspond to m_L and m_P (0.137099 and 0.139703), and the lowest to f_{Ag} and f_L (0.001982 and 0.001784). The MDAPE averages at 5.9%, with m_A showing the lowest error at 3.3566% and the three constants (γ_{Aem}, γ_{Ab}, and γ_{Ao}) ranging between 3.62% and 3.87%. The errors for m_L and m_P are notably higher at 12.59% and 25.91%, respectively.

We interpret these results as follows: in colder temperatures, within the day range of 943 to 1064, where calculated air temperature is below $5.0°C$, the mosquito population decreases rapidly to zero (with the exception of $Ag1$ and $Ag2$) as can be seen in Fig 7. This situation makes it challenging to gather useful information, negatively affecting the ability to determine several system parameters, especially m_L and m_P as these parameters are primarily derived from equations related to $\frac{dL}{dt}$ and $\frac{dP}{dt}$. Conversely, parameters γ_{Aem}, γ_{Ab} and γ_{Ao} remain constant over time, which makes them simpler to accurately estimate, as seen in their relatively low MDAPE. Meanwhile, parameters like m_A, f_{Ag} which are involved in several differential equations, tend to receive more consistent and stable information and gradients. In fact, the target parameters are not identifiable across the entire input domain due to the system setup (see S3 Fig), which can result in inaccuracies in the approximations made by PINNs.

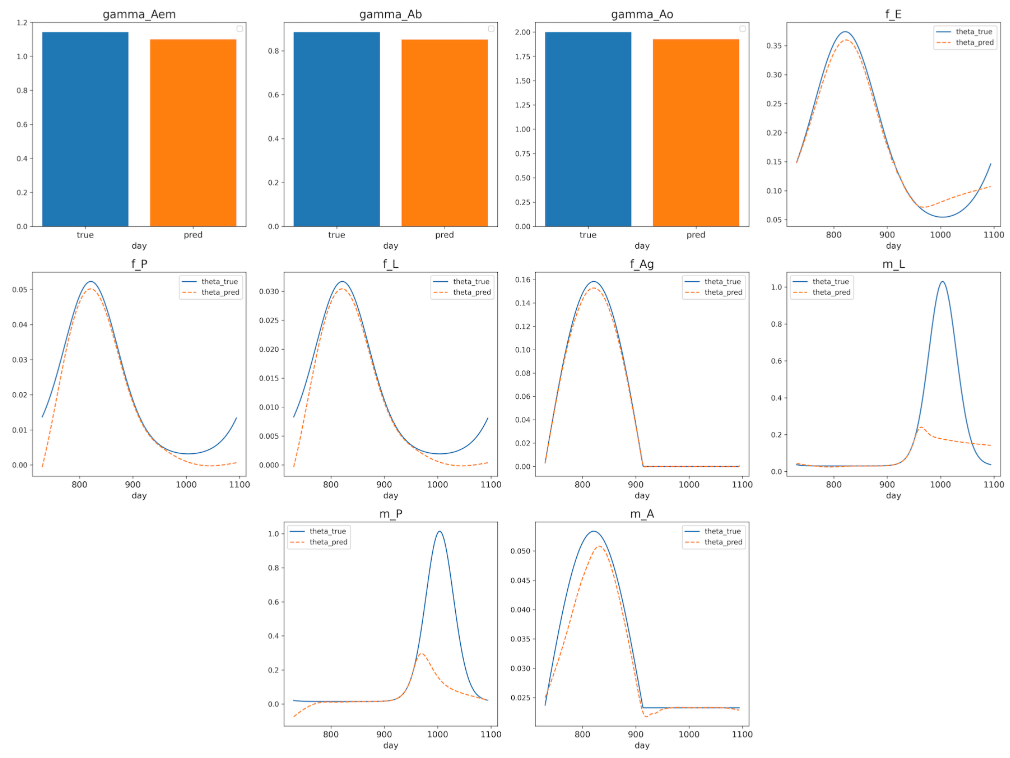

Fig 8. Mosquito ODE system, inverse problem, Θ approximation of the system's parameters.

Table 3. Mosquito ODE System's Parameter Approximation Errors

Parameter	RMSE	MAE	MDAPE	nRMSE
γ_{Aem}	0.042639	0.042639	0.037304	0.042639
γ_{Ab}	0.034216	0.034216	0.038662	0.034216
γ_{Ao}	0.072417	0.072417	0.036208	0.072417
f_E	0.015570	0.012633	0.039964	0.015570
f_P	0.004646	0.003270	0.078140	0.004646
f_L	0.002816	0.001982	0.078126	0.002816
f_{Ag}	0.002769	0.001784	0.094168	0.002769
m_L	0.286959	0.137099	0.125888	0.286959
m_P	0.291548	0.139703	0.259069	0.291548
m_A	0.002848	0.001828	0.033566	0.002848
Overall	0.132615	0.044757	0.059045	0.132615

Discussion and Future Work

In this work, we introduced a hybrid framework based on Physics informed Neural Networks that integrates physical laws into data-driven machine learning models. This particular framework is designed to manage systems of ordinary differential equations, validated using a case study modelling mosquito population dynamics. The approach employs a multi-task learning strategy, incorporating multiple components in the objective function: one for data fitting, several for weakly enforcing physical knowledge, with separate functions for the remaining conditions. The framework includes several advanced techniques such as domain decomposition, ODE normalization, gradient balancing, and causal training. We evaluated the effectiveness of our approach through an ablation study using the Lorenz system before introducing the problem of a model for the dynamics of mosquito populations.

The application of PINNs to mosquito population dynamics represents a significant advancement in ecological modelling for a number of reasons. Traditional data-driven models often require extensive data and are limited to capturing the nonlinear, multi-scale behaviours seen in real-world mosquito populations. However, PINNs integrate domain-specific knowledge directly into the learning process by enforcing ODE constraints. This is crucial for mosquito population dynamics because it ensures that the model respects biological principles even with limited data.

Our findings demonstrate that ODE Normalization and Gradient Balancing played an important role in stabilizing the training process. These techniques ensured that no individual component of the loss function disproportionately influenced the optimization, thus preventing premature convergence to suboptimal solutions. Causal Training preserves the temporal causality inherent in the dynamical system, critical for achieving accurate model predictions, especially in scenarios where extrapolation beyond the scope of the training data is necessary. Domain Decomposition effectively manages significantly large input domains, particularly in forward problem scenarios. The results also confirm the framework's efficacy in modelling mosquito population dynamics, highlighting its potential for application to other ODE-based ecological models, which are commonly applied not only to mosquito populations but also to other vector species, such as ticks [81], also agricultural and forest pests, including *Drosophila suzukii* [82], *Batrocera oleae* [83] and bark beetles [84].

Our research using PINNs revealed certain limitations. In the inverse problem setup, the PINN tended to favor solutions with smaller gradients that still met the data and ODE constraints. Furthermore, the capability of the framework to extrapolate in inverse problems remains uncertain; with lack of appropriate volume of data, the networks may converge to arbitrary solutions. This aspect underscores the need for incorporating additional regularization techniques or constraints in order to guide the network towards more plausible solutions. In the current implementation, the models consider only time as an input, without accounting for external factors. This restriction significantly limits predictive performance in real-world applications, as the coordination inputs and the encoded physical laws may not fully capture the underlying mechanisms of the processes. Allowing for external variables would not only enhance performance but also add flexibility to the framework. Although the framework has shown promising results on test systems, its scalability and ability to generalize to other types of dynamical systems have not been thoroughly evaluated. Our current focus is on the development of a new PINN framework that encodes complex, learned interactions of air temperature, precipitation and relative humidity, to improve and demonstrate the effectiveness of this approach in addressing inverse problems, such as inferring mosquito development and mortality rates. Accurate mosquito population dynamics modelling is essential for predicting the outbreaks of arbovirus diseases. By improving predictive accuracy, PINNs can provide more reliable insights into population trends and help design targeted control measures (e.g., optimal times for pesticide application). This leads to more effective, data-informed vector control strategies that can be adapted across different geographic and climatic contexts.

Supporting information

S1 Appendix. Error metrics.
Let $y_i, i = 1 \ldots, M$ be M reference values and $\hat{y}_i, i = 1 \ldots, M$ are the corresponding predictions. Let $\mathfrak{L}$ and $\mathfrak{U}$ be the pre-defined lower and upper bounds for the values. The

error metrics used in this paper, including Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Median Absolute Percenrage Error (MDAPE) and Root Mean Squared normalized Error (nRMSE) are defined as followed

$$RMSE = \sqrt{\frac{1}{M}\sum_{i}^{M}(y_i - \hat{y}_i)^2} \tag{19}$$

$$MAE = \frac{1}{M}\sum_{i}^{M}|y_i - \hat{y}_i| \tag{20}$$

$$MDAPE = \text{median}_i\left(\frac{|y_i - \hat{y}_i|}{|y_i|}\right) \tag{21}$$

$$nRMSE = \frac{RMSE}{\mathfrak{U} - \mathfrak{L}} \tag{22}$$

S1 Fig. OdePINN framework solves inverse problem with Lorenz system, U approximation of the solution (Fig 9).

S1 Table. ODE Parameters (Table 4).

S2 Fig. Model scheme showing the 10 develpmental stages within the mosquito life cycle (Fig 10): Egg (E), Larva (L), Pupa (P), Emerging Adults (A_{em}), Nulliparous Bloodseeking Adults (A_{b1}), Nulliparous Gestating Adults (A_{g1}), Nulliparous Ovipositing Adults (A_{o1}), Parous Bloodseeking Adults (A_{b2}), Parous Gestating Adults (A_{g2}) and Parous Ovipositing Adults (A_{o2}) (source: [11]).

S3 Fig. Identifiability of parameters over time in the Mosquito inverse problem, the sine-shape temperature (Fig 11). This is achieved by expressing the system as a system of linear equations, the parameters as unknown variables, and analyzing the Reduced row-echelon form of the coefficient matrix, performed separately for each time t. In the figure, identifiable parameters are defined as free parameters which can get arbitrary values. The values are rounded to 6-digit precision, aligning with the PINN's level of precision after training. The figure explains the PINN's inaccuracies shown in Fig 8.

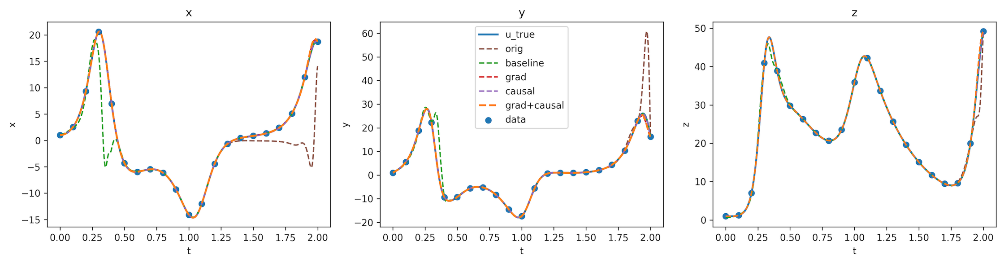

Fig 9. OdePINN framework solves inverse problem with Lorenz system, U approximation of the solution.

Table 4. ODE Model Parameters

Parameter	Description	Value	Unit
τ	Temperature		$^{\circ}C$
γ_{Aem}	Development rate of emerging adults	1.143	$days^{-1}$
γ_{Ab}	Development rate of bloodseeking adults	0.885	$days^{-1}$
γ_{Ao}	Ovipositing adult development rate	2	$days^{-1}$
$f_E(>0)$	Egg development rate	$0.16 \cdot \left(e^{[0.105(\tau-10)]} - e^{\left[0.105(38-10)-\frac{1}{5.007}(38-\tau)\right]}\right)$	$days^{-1}$
f_P	Pupa development rate	$0.021 \cdot \left(e^{[0.162(\tau-10)]} - e^{\left[0.162(38-10)-\frac{1}{5.007}(38-\tau)\right])}\right)$	$days^{-1}$
f_L	Larva development rate	$f_P/1.65$	$days^{-1}$
$f_{Ag}(>0)$	Development rate of gestating adults	$\frac{\tau-9.8}{64.4}$	$days^{-1}$
m_E	Egg mortality rate	$m_E = \mu_E$	$days^{-1}$
m_L	Larval mortality rate	$\exp[-\tau/2] + \mu_L$	$days^{-1}$
m_P	Pupa mortality rate	$\exp[-\tau/2] + \mu_P$	$days^{-1}$
$m_A(>\mu_A)$	Mortality rate of Ab,	$-0.005941 + 0.002965 \cdot \tau$	$days^{-1}$
μ_E	Minimum egg mortality rate	0	$days^{-1}$
μ_L	Minimum larval mortality rate	0.0304	$days^{-1}$
μ_P	Minimum pupa mortality rate	0.0146	$days^{-1}$
μ_{em}	Mortality rate during emergence	0.1	$days^{-1}$
μ_r	Mortality rate during bloodseeking	0.08	$days^{-1}$
μ_A	Minimum adult mortality rate	$\frac{1}{43}$	$days^{-1}$
κ_L	Carrying capacity for larvae	$8 \cdot 10^8$	$days^{-1}$
κ_P	Carrying capacity for pupae	10^7	$days^{-1}$
σ	Sex ratio at emergence	0.5	-
β	Number of eggs per Ao	$\beta_1 = 141(np), \beta_2 = 80(p)$	-

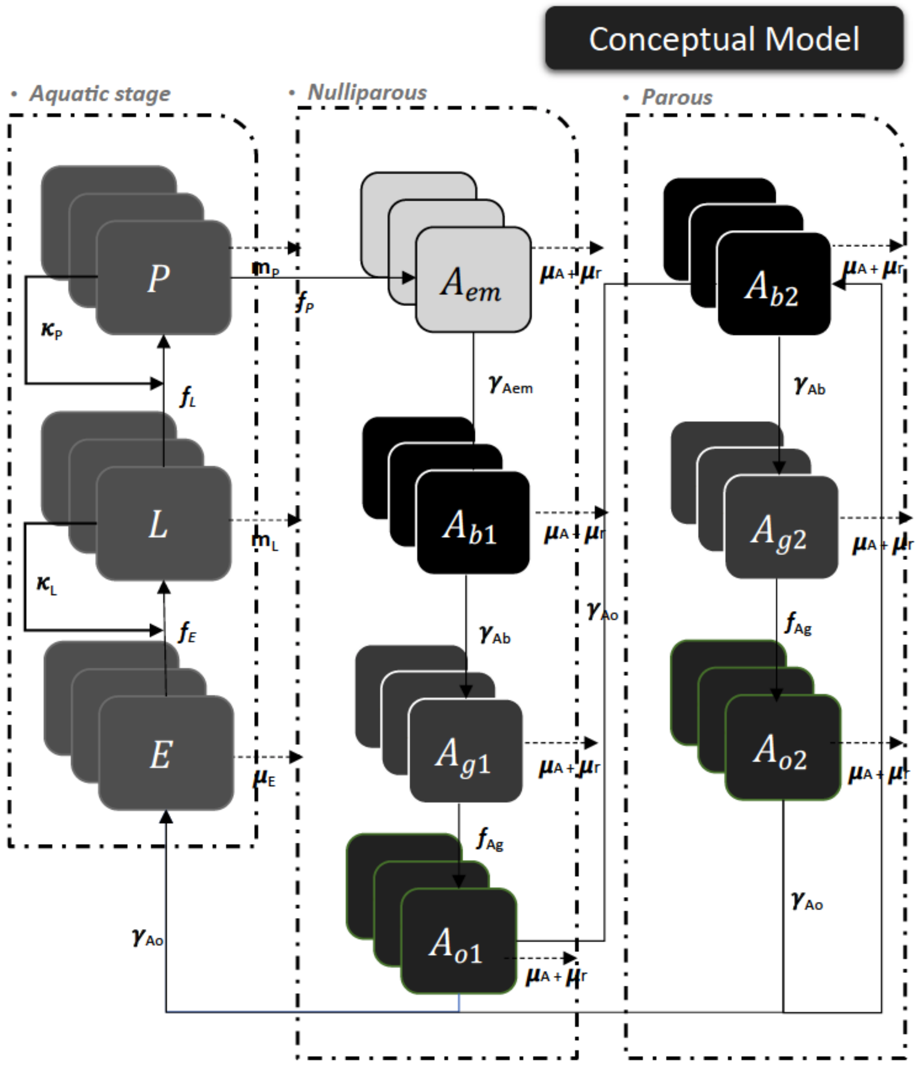

Fig 10. Model scheme showing the 10 develpmental stages within the mosquito life cycle: Egg (E), Larva (L), Pupa (P), Emerging Adults (A_{em}), Nulliparous Bloodseeking Adults (A_{b1}), Nulliparous Gestating Adults (A_{g1}), Nulliparous Ovipositing Adults (A_{o1}), Parous Bloodseeking Adults (A_{b2}), Parous Gestating Adults (A_{g2}) and Parous Ovipositing Adults (A_{o2}) (source: [11])

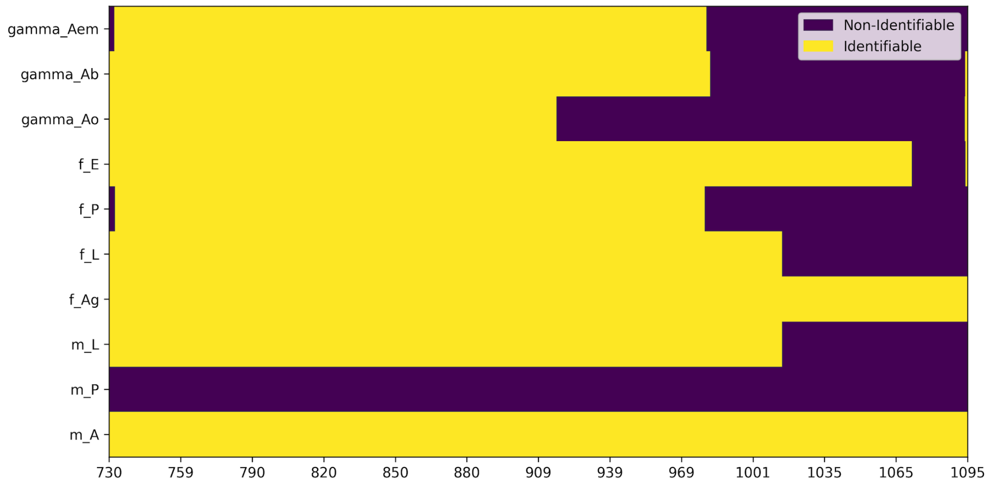

Fig 11. Identifiability of parameters over time in the Mosquito inverse problem, the sine-shape temperature. This is achieved by expressing the system as a system of linear equations, the parameters as unknown variables, and analyzing the Reduced row-echelon form of the coefficient matrix, performed separately for each time t. In the figure, identifiable parameters are defined as free parameters which can get arbitrary values. The values are rounded to 6-digit precision, aligning with the PINN's level of precision after training. The figure explains the PINN's inaccuracies shown in Fig 8.

References

1. Organisation WH. Chikungunya;. https://www.who.int/health-topics/chikungunya.

2. Organisation WH. Dengue and severe dengue;. https://www.who.int/news-room/fact-sheets/detail/dengue-and-severe-dengue.

3. Organisation WH. Fact sheet about malaria;. https://www.who.int/news-room/fact-sheets/detail/malaria.

4. Organisation WH. Zika virus disease;. https://www.who.int/health-topics/zika-virus-disease.

5. Organisation WH. West Nile virus;. https://www.who.int/news-room/fact-sheets/detail/west-nile-virus.

6. Tran A, l'Ambert G, Lacour G, Benot R, Demarchi M, Cros M, et al. A rainfall-and temperature-driven abundance model for Aedes albopictus populations. International journal of environmental research and public health. 2013;10(5):1698–1719.

7. Erickson RA, Presley SM, Allen LJ, Long KR, Cox SB. A stage-structured, Aedes albopictus population model. Ecological Modelling. 2010;221(9):1273–1282.

8. Cailly P, Tran A, Balenghien T, L'Ambert G, Toty C, Ezanno P. A climate-driven abundance model to assess mosquito control strategies. Ecological Modelling. 2012;227:7–17. doi:10.1016/j.ecolmodel.2011.10.027.

9. Marini G, Arnoldi D, Baldacchino F, Capelli G, Guzzetta G, Merler S, et al. First report of the influence of temperature on the bionomics and population dynamics of Aedes koreicus, a new invasive alien species in Europe. Parasites & vectors. 2019;12(1):524. doi:10.1186/s13071-019-3772-5.

10. Qiang L, Wang BG, Zhao XQ. A Stage-Structured Population Model with Time-Dependent Delay in an Almost Periodic Environment. Journal of Dynamics and Differential Equations. 2022;34(1):341–364. doi:10.1007/s10884-020-09827-6.

11. Petrić M. Modelling the Influence of Meteorological Conditions on Mosquito Vector Population Dynamics (Diptera, Culicidae). Ghent University. Belgium; 2020. Available from: `https://biblio.ugent.be/publication/8716105`.

12. Otero M, Solari HG, Schweigmann N. A stochastic population dynamics model for Aedes aegypti: formulation and application to a city with temperate climate. Bulletin of mathematical biology. 2006;68(8):1945–1974.

13. Chuang TW, Ionides EL, Knepper RG, Stanuszek WW, Walker ED, Wilson ML. Cross-correlation map analyses show weather variation influences on mosquito abundance patterns in Saginaw County, Michigan, 19892005. Journal of medical entomology. 2012;49(4):851–858.

14. Edwards CB, Crone EE. Estimating abundance and phenology from transect count data with GLMs. Oikos. 2021;130(8):1335–1345. doi:10.1111/oik.08368.

15. Oluwagbemi OO, Fornadel CM, Adebiyi EF, Norris DE, Rasgon JL. ANOSPEX: a stochastic, spatially explicit model for studying Anopheles metapopulation dynamics. PloS one. 2013;8(7):e68040.

16. Guisan A, Thuiller W. Predicting species distribution: offering more than simple habitat models. Ecology letters. 2005;8(9):993–1009.

17. Da Re D, Van Bortel W, Reuss F, Mller R, Boyer S, Montarsi F, et al. dynamAedes: a unified modelling framework for invasive Aedes mosquitoes. Parasites & Vectors. 2022;15(1):414. doi:10.1186/s13071-022-05414-4.

18. Da Re D, Marini G, Bonannella C, Laurini F, Manica M, Anicic N, et al.. Inferring the seasonal dynamics and abundance of an invasive species using a spatio-temporal stacked machine learning model; 2023. Available from: `https://ecoevorxiv.org/repository/view/6451/`.

19. Tolstikhin IO, Houlsby N, Kolesnikov A, Beyer L, Zhai X, Unterthiner T, et al. MLP-Mixer: An all-MLP Architecture for Vision. In: Ranzato M, Beygelzimer A, Dauphin Y, Liang PS, Vaughan JW, editors. Advances in Neural Information Processing Systems. vol. 34. Curran Associates, Inc.; 2021. p. 24261–24272. Available from: `https://proceedings.neurips.cc/paper_files/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf`.

20. Quach BM, Dinh VC, Pham N, Huynh D, Nguyen BT. Leaf recognition using convolutional neural networks based features. Multimedia Tools and Applications. 2023;82(1):777–801. doi:10.1007/s11042-022-13199-y.

21. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is All you Need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, et al., editors. Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc.; 2017.Available from: `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

22. Tran H, Dinh CV, Phan L, Nguyen ST. Hierarchical Transformer Encoders for Vietnamese Spelling Correction; 2021.

23. Azodi CB, Bolger E, McCarren A, Roantree M, de los Campos G, Shiu SH. Benchmarking Parametric and Machine Learning Models for Genomic Prediction of Complex Traits. G3 Genes—Genomes—Genetics. 2019;9(11):3691–3702. doi:10.1534/g3.119.400498.

24. Liu XY, Xia Z, Rui J, Gao J, Yang H, Zhu M, et al. FinRL-Meta: Market Environments and Benchmarks for Data-Driven Financial Reinforcement Learning. In: Koyejo S, Mohamed S, Agarwal A, Belgrave D, Cho K, Oh A, editors. Advances in Neural Information Processing Systems. vol. 35. Curran Associates, Inc.; 2022. p. 1835–1849. Available from: `https://proceedings.neurips.cc/paper_files/paper/2022/file/0bf54b80686d2c4dc0808c2e98d430f7-Paper-Datasets_and_Benchmarks.pdf`.

25. Nguyen APN, Crane M, Bezbradica M. Cryptocurrency Volatility Index: An Efficient Way toPredict theFuture CVI. In: Longo L, O'Reilly R, editors. Artificial Intelligence and Cognitive Science. Cham: Springer Nature Switzerland; 2023. p. 355–367.

26. Saeed W, Omlin C. Explainable AI (XAI): A systematic meta-survey of current challenges and future opportunities. Knowledge-Based Systems. 2023;263:110273. doi:https://doi.org/10.1016/j.knosys.2023.110273.

27. Yu H, Liu J, Zhang X, Wu J, Cui P. A Survey on Evaluation of Out-of-Distribution Generalization. ArXiv. 2024;abs/2403.01874.

28. Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nature Reviews Physics. 2021;3(6):422–440. doi:10.1038/s42254-021-00314-5.

29. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019;378:686–707. doi:https://doi.org/10.1016/j.jcp.2018.10.045.

30. Banerjee CK, Nguyen K, Fookes C, Karniadakis GE. Physics-Informed Computer Vision: A Review and Perspectives. ArXiv. 2023;abs/2305.18035.

31. Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE. Physics-informed neural networks (PINNs) for fluid mechanics: a review. Acta Mechanica Sinica. 2021;37:1727 – 1738.

32. Hao Z, Liu S, Zhang Y, Ying C, Feng Y, Su H, et al.. Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications; 2023.

33. Banerjee CK, Nguyen K, Fookes C, Raissi M. A Survey on Physics Informed Reinforcement Learning: Review and Open Problems. ArXiv. 2023;abs/2309.01909.

34. Yazdani A, Lu L, Raissi M, Karniadakis GE. Systems biology informed deep learning for inferring parameters and hidden dynamics. PLOS Computational Biology. 2020;16(11):1–19. doi:10.1371/journal.pcbi.1007575.

35. Wang S, Wang H, Perdikaris P. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. Computer Methods in Applied Mechanics and Engineering. 2021;384:113938. doi:https://doi.org/10.1016/j.cma.2021.113938.

36. Wang S, Teng Y, Perdikaris P. Understanding and mitigating gradient pathologies in physics-informed neural networks. ArXiv. 2020;abs/2001.04536.

37. Gao H, Sun L, Wang JX. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. Journal of Computational Physics. 2021;428:110079. doi:10.1016/j.jcp.2020.110079.

38. Ren P, Rao C, Liu Y, Wang JX, Sun H. PhyCRNet: Physics-informed convolutional-recurrent network for solving spatiotemporal PDEs. Computer Methods in Applied Mechanics and Engineering. 2022;389:114399. doi:https://doi.org/10.1016/j.cma.2021.114399.

39. Jagtap AD, Kawaguchi K, Karniadakis GE. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. Journal of Computational Physics. 2020;404:109136. doi:https://doi.org/10.1016/j.jcp.2019.109136.

40. Jagtap AD, Kawaguchi K, Em Karniadakis G. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2020;476(2239):20200334. doi:10.1098/rspa.2020.0334.

41. Maddu S, Sturm D, Mller CL, Sbalzarini IF. Inverse Dirichlet weighting enables reliable training of physics informed neural networks. Machine Learning: Science and Technology. 2022;3(1):015026. doi:10.1088/2632-2153/ac3712.

42. Wang S, Yu X, Perdikaris P. When and why PINNs fail to train: A neural tangent kernel perspective. Journal of Computational Physics. 2022;449:110768. doi:https://doi.org/10.1016/j.jcp.2021.110768.

43. Lu L, Meng X, Mao Z, Karniadakis GE. DeepXDE: A Deep Learning Library for Solving Differential Equations. SIAM Review. 2021;63(1):208–228. doi:10.1137/19M1274067.

44. Wu C, Zhu M, Tan Q, Kartha Y, Lu L. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. Computer Methods in Applied Mechanics and Engineering. 2023;403:115671. doi:https://doi.org/10.1016/j.cma.2022.115671.

45. Nabian MA, Gladstone RJ, Meidani H. Efficient training of physics-informed neural networks via importance sampling. Computer-Aided Civil and Infrastructure Engineering. 2021;36(8):962–977. doi:https://doi.org/10.1111/mice.12685.

46. Tang K, Wan X, Yang C. DAS-PINNs: A deep adaptive sampling method for solving high-dimensional partial differential equations. Journal of Computational Physics. 2023;476:111868. doi:https://doi.org/10.1016/j.jcp.2022.111868.

47. L Wight C, Zhao J. Solving Allen-Cahn and Cahn-Hilliard Equations Using the Adaptive Physics Informed Neural Networks. Communications in Computational Physics. 2021;29(3):930–954. doi:https://doi.org/10.4208/cicp.OA-2020-0086.

48. Krishnapriyan AS, Gholami A, Zhe S, Kirby RM, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks; 2021.

49. Mattey R, Ghosh S. A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations. Computer Methods in Applied Mechanics and Engineering. 2022;390:114474. doi:https://doi.org/10.1016/j.cma.2021.114474.

50. Wang S, Sankaran S, Perdikaris P. Respecting causality is all you need for training physics-informed neural networks. ArXiv. 2022;abs/2203.07404.

51. Jagtap AD, Kharazmi E, Karniadakis GE. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. Computer Methods in Applied Mechanics and Engineering. 2020;365:113028. doi:https://doi.org/10.1016/j.cma.2020.113028.

52. Jagtap AD, Karniadakis GE. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Communications in Computational Physics. 2020;28(5):2002–2041.

53. Moseley B, Markham A, NissenMeyer T. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. Advances in Computational Mathematics. 2021;49:1–39.

54. Huang M, Tang M, Yu J, Zheng B, ,School of Statistics and Mathematics, Guangdong University of Finance and Economics, Guangzhou 510320, China, ,Department of Mathematics, Michigan State University, East Lansing, MI 48824, USA, et al. A stage structured model of delay differential equations for *Aedes* mosquito population suppression. Discrete & Continuous Dynamical Systems - A. 2020;40(6):3467–3484. doi:10.3934/dcds.2020042.

55. Song H, Tian D, Shan C. Modeling the effect of temperature on dengue virus transmission with periodic delay differential equations. Math Biosci Eng. 2020;17(4):4147–4164.

56. Christiansen-Jucht C, Erguler K, Shek C, Basanez MG, Parham P. Modelling Anopheles gambiae ss population dynamics with temperature-and age-dependent survival. International journal of environmental research and public health. 2015;12(6):5975–6005.

57. Roiz D, Neteler M, Castellani C, Arnoldi D, Rizzoli A. Climatic factors driving invasion of the tiger mosquito (Aedes albopictus) into new areas of Trentino, northern Italy. PloS one. 2011;6(4).

58. Carrieri M, Fariselli P, Maccagnani B, Angelini P, Calzolari M, Bellini R. Weather factors influencing the population dynamics of Culex pipiens (Diptera: Culicidae) in the Po Plain Valley, Italy (1997-2011). Environmental entomology. 2014;43(2):482–490.

59. Groen TA, Lambert G, Bellini R, Chaskopoulou A, Petric D, Zgomba M, et al. Ecology of West Nile virus across four European countries: empirical modelling of the Culex pipiens abundance dynamics as a function of weather. Parasites & vectors. 2017;10(1):524.

60. Kass M. An Introduction to Physically Based Modeling: Energy Functions and Stiffness. Tech. rep., Pixar; 1997.

61. Soetaert K, Cash J, Mazzia F. Solving Ordinary Differential Equations in R. In: Soetaert K, Cash J, Mazzia F, editors. Solving Differential Equations in R. Berlin, Heidelberg: Springer; 2012. p. 41–80. Available from: `https://doi.org/10.1007/978-3-642-28070-2_3`.

62. Dahlquist G, Bjrck . Numerical Methods. Courier Corporation; 2003.

63. Erguler K, Mendel J, Petrić DV, Petrić M, Kavran M, Demirok MC, et al. A dynamically structured matrix population model for insect life histories observed under variable environmental conditions. Scientific Reports. 2022;12(1):11587.

64. Ahmadi Daryakenari N, De Florio M, Shukla K, Karniadakis GE. AI-Aristotle: A physics-informed framework for systems biology gray-box identification. PLOS Computational Biology. 2024;20(3):1–33. doi:10.1371/journal.pcbi.1011916.

65. Lagergren JH, Nardini JT, Baker RE, Simpson MJ, Flores KB. Biologically-informed neural networks guide mechanistic modeling from sparse experimental data. PLOS Computational Biology. 2020;16(12):1–29. doi:10.1371/journal.pcbi.1008462.

66. Wang S, Sankaran S, Wang H, Perdikaris P. An Expert's Guide to Training Physics-informed Neural Networks; 2023.

67. Baty H. Solving stiff ordinary differential equations using physics informed neural networks (PINNs): simple recipes to improve training of vanilla-PINNs; 2023.

68. Stiller P, Bethke F, Böhme M, Pausch R, Torge S, Debus A, et al. Large-Scale Neural Solvers for Partial Differential Equations. In: Nichols J, Verastegui B, Maccabe AB, Hernandez O, Parete-Koon S, Ahearn T, editors. Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI. Cham: Springer International Publishing; 2020. p. 20–34.

69. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. CoRR. 2014;abs/1412.6980.

70. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. ArXiv. 2019;abs/1912.01703.

71. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al.. TensorFlow: A system for large-scale machine learning; 2016.

72. Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, et al.. JAX: composable transformations of Python+NumPy programs; 2018. Available from: `http://github.com/google/jax`.

73. Leshno M, Lin VY, Pinkus A, Schocken S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks. 1993;6(6):861–867. doi:https://doi.org/10.1016/S0893-6080(05)80131-5.

74. Haykin S. Neural networks: a comprehensive foundation. Prentice Hall PTR; 1994.

75. Hendrycks D, Gimpel K. Gaussian Error Linear Units (GELUs); 2023.

76. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Teh YW, Titterington M, editors. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. vol. 9 of Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR; 2010. p. 249–256. Available from: `https://proceedings.mlr.press/v9/glorot10a.html`.

77. Lorenz EN. Deterministic nonperiodic flow. Journal of the Atmospheric Sciences. 1963;20:130–141.

78. Petzold L. Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations. SIAM Journal on Scientific and Statistical Computing. 1983;4(1):136–148. doi:10.1137/0904010.

79. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods. 2020;17:261–272. doi:10.1038/s41592-019-0686-2.

80. Cailly P, Tran A, Balenghien T, LAmbert G, Toty C, Ezanno P. A climate-driven abundance model to assess mosquito control strategies. Ecological Modelling. 2012;227:7–17. doi:https://doi.org/10.1016/j.ecolmodel.2011.10.027.

81. Wu J, Zhang X, Wu J, Zhang X. Structured Tick Population Dynamics. Transmission Dynamics of Tick-Borne Diseases with Co-Feeding, Developmental and Behavioural Diapause. 2020; p. 51–77.

82. Rossini L, Rosselló NB, Speranza S, Garone E. A general ODE-based model to describe the physiological age structure of ectotherms: Description and application to Drosophila suzukii. Ecological Modelling. 2021;456:109673.

83. Rossini L, Bruzzone OA, Contarini M, Bufacchi L, Speranza S. A physiologically based ODE model for an old pest: Modeling life cycle and population dynamics of Bactrocera oleae (Rossi). Agronomy. 2022;12(10):2298.

84. Křivan V, Lewis M, Bentz BJ, Bewick S, Lenhart SM, Liebhold A. A dynamical model for bark beetle outbreaks. Journal of theoretical biology. 2016;407:25–37.