Machine Learning (2025) 114:180
https://doi.org/10.1007/s10994-025-06808-y

Physics encoded blocks in residual neural network architectures for digital twin models

Muhammad Saad Zia[1] · Corentin Houpert[1] · Ashiq Anjum[1] · Lu Liu[1] · Anthony Conway[1] · Anasol Peña-Rios[2]

Received: 19 March 2025 / Revised: 19 March 2025 / Accepted: 26 May 2025 /
Published online: 10 July 2025

Abstract
Physics Informed Machine Learning has emerged as a popular approach for modeling and simulation in digital twins, enabling the generation of accurate models of processes and behaviors in real-world systems. However, existing methods either rely on simple loss regularizations that offer limited physics integration or employ highly specialized architectures that are difficult to generalize across diverse physical systems. This paper presents a generic approach based on a novel physics-encoded residual neural network (PERNN) architecture that seamlessly combines data-driven and physics-based analytical models to overcome these limitations. Our method integrates differentiable physics blocks–implementing mathematical operators from physics-based models–with feed-forward learning blocks, while intermediate residual blocks ensure stable gradient flow during training. Consequently, the model naturally adheres to the underlying physical principles even when prior physics knowledge is incomplete, thereby improving generalizability with low data requirements and reduced model complexity. We investigate our approach in two application domains. The first is a steering model for autonomous vehicles in a simulation environment, and the second is a digital twin for climate modeling using an ordinary differential equation (ODE)-based model of Net Ecosystem Exchange (NEE) to enable gap-filling in flux tower data. In both cases, our method outperforms conventional neural network approaches as well as state-of-the-art Physics Informed Machine Learning methods.

Keywords Physics-informed machine learning · Physics-informed neural networks · Hybrid neural architectures · Residual learning · Differentiable physics modules · Digital twins

1 Introduction

Digital twins are virtual counterparts of any real-world system or entity synchronized with the system or entity through real-time data. A central concept in digital twins is the modelling and simulation of processes and behaviour associated with the real-world entity. These

Editor: Myra Spiliopoulou.

Extended author information available on the last page of the article

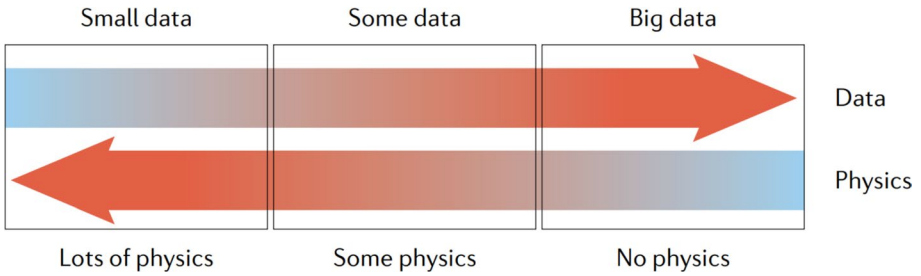


Fig. 1 Three possible categories of physical problems and associated available data and knowledge of physics (Figure from Lu et al. 2021)

models allow users to investigate behaviour in the alternative *'what-if'* scenarios and can be divided into two fundamental categories: *physics-based models* and *data-driven models*.

While physics-based models offer better accuracy and reliability of predictions, they are often computationally expensive and inflexible to be used in near-real-time simulations - especially in the context of Digital Twins (Willcox et al., 2021; Wang et al., 2022). On the other hand, data-driven models provide flexible and adaptive solutions, especially with a limited understanding of underlying physics. However, they popularly suffer in terms of reliability and interpretation (Rasheed et al., 2020). The benefits of combining physics with data-driven models to target these limitations are found in several studies, especially in the context of developing simulation models in situations with limited availability of prior physics knowledge and real-world data (Rasheed et al., 2020; Thelen et al., 2022).

Modelling of physical systems can be categorized into three regimes based on the availability trade-off between data and known physics, as illustrated in Fig. 1. As data increases, the balance shifts from leveraging prior physics knowledge to data-driven extraction of physical relationships. In the ubiquitous middle ground, some physics and data are known, with potentially missing parameters and initial/boundary conditions, and this is the most common scenario in models for digital twins in general (Lu et al., 2021).

Physics-informed machine learning is a recent domain that offers a streamlined approach to incorporate both data and the fundamental laws of physics as prior knowledge, even when dealing with models with incomplete physical information. This integration, or *'bias'*, is achieved by regularizing the loss function of the learning algorithms to adhere to the constraints defined by the respective physics models and equations. More specialized approaches involve specifically designed neural networks, which ensure that the predictions generated adhere to the governing physical principles. One of these approaches recently gaining traction in the research community is known as 'differentiable physics'. This approach allows differentiable physics models to be directly baked into neural network architectures. An example is the recent work on new connections between neural network architectures and viscosity solutions to Hamilton-Jacobi PDEs (Darbon & Meng, 2021) as shown in Fig. 2.

While existing work mainly focuses on isolated physical phenomena (Li et al., 2019; Toussaint et al., 2018; Sanchez-Gonzalez et al., 2018), and lacks the generalization of the approaches to wider problems, we propose Physics Encoded Residual Neural Network (PERNN), a general framework to capture complex behaviour of real-world

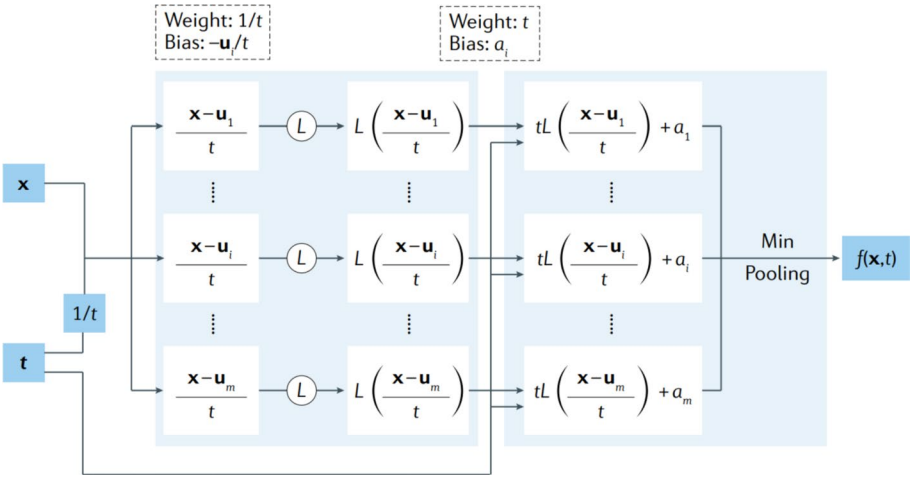


Fig. 2 A neural network with the Lax-Oleinik formula baked into the architecture. Here, f represents the solution of the Hamilton-Jacobi partial differential equations, while x and t correspond to the spatial and temporal variables, respectively. The activation function L is convex and Lipschitz. The parameters $a_i \in \mathbb{R}$ and $u_i \in \mathbb{R}^n$ are specific to the neural network. Image courtesy of J. Darbon and T. Meng, Brown University and cited here from Lu et al. (2021)

systems. Our approach aims to target scenarios where data availability from the target systems is limited to specific operating scenarios and known physics only captures specific regions of the behaviour. Our approach defines a hybrid neural network which comprises feature extraction blocks to compute unknown variables (learning blocks) and combines them with differentiable physics operator layers (physics blocks) to model the complete behaviour of the target system. To facilitate the convergence of the model, we introduce intermediate 'residual' layers between the physics and the learning blocks. We also make our code available publicly on GitHub.[1]

In this way, domain knowledge can inform the underlying neural network architecture while the gradient descent optimization using data can fill in gaps (Seo et al., 2022; Stewart & Ermon, 2019). The goal is to leverage interpretable mechanistic modelling to improve generalization and avoid relying solely on empirical correlations (Greydanus et al., 2019; Raissi et al., 2019). This approach could benefit several complex systems, including robotic platforms (Cranmer et al., 2020; Mrowca et al., 2018), autonomous vehicles (Ajay et al., 2018), and aircraft swarms (Mohan et al., 2021).

In this paper, we present two concrete applications to evaluate our method. The first application focuses on modeling dynamic steering for autonomous grounded vehicles. In this case, we develop a hybrid neural network architecture that integrates differentiable geometrical and kinematic operators from the well-established pure pursuit algorithm with learning blocks that discover intermediate unknown variables in the driving environment. The model is trained on observation data from an expert driving system in a simulation environment to learn the expert's behavior or *policy* using supervised learning. This forms a Behaviour Cloning problem, which involves transforming expert demonstrations into i.i.d. state-action pairs and learning to *imitate* the expert actions given each recorded state.

[1] https://github.com/saadzia10/Physics-Encoded-ResNet.git.

In the context of digital twins, this approach is particularly useful because it models the exact behavior of a real-world system using observational data.

The second application demonstrates our approach in a real-world digital twin setting for climate modeling. Net Ecosystem Exchange (NEE), a key metric representing the net CO_2 exchange between ecosystems and the atmosphere, is typically measured by flux towers that often suffer from data gaps due to instrumentation failures and other issues. To address this challenge, we model NEE using an ordinary differential equation (ODE) derived from an Arrhenius-type formulation of ecosystem respiration. Our Physics Encoded Residual Neural Network (PERNN) for NEE explicitly incorporates the NEE ODE into the network via a dedicated physics block, while learning and residual blocks predict critical intermediate variables such as temperature sensitivity (E_0), base respiration (r_{b_night}), and the rate of change of ambient air temperature $\left(\frac{dT}{dt}\right)$. This integration enables accurate gap-filling and forecasting of NEE, leading to improved digital twin fidelity for climate monitoring and forecasting.

This paper presents the following key contributions:

- It is the first (to our understanding) generic framework for embedding physics models into neural networks for end-to-end training on observation data.
- It is a significantly more interpretable hybrid neural network comprising learning blocks that predict human-understandable variables and physics blocks that predict the actual target variables.
- It allows the implicit discovery of unknown environmental variables that are not measured or observed.
- It outperforms conventional neural networks regarding key driving metrics on unseen road scenarios in simulation environments.
- It uses significantly fewer observation data, with fewer parameters, than conventional neural networks while outperforming key driving metrics.
- It achieves superior gap-filling performance in digital twin applications for climate modeling by accurately forecasting Net Ecosystem Exchange (NEE) from flux tower data, outperforming state-of-the-art methods such as Random Forest and XGBoost in key error metrics.
- It provides a robust digital twin framework for environmental monitoring by integrating an ODE-based physics model of ecosystem respiration directly into the neural network architecture.

2 Related work

Physics Informed Machine Learning (PIML) is a recent approach that combines prior physics knowledge with conventional machine learning approaches to form hybrid learning algorithms of processes. Approaches like Physics-Informed Neural Networks (PINNs) encode analytical equations as loss terms to constrain deep learning models. Making a learning algorithm physics-informed refers to applying appropriate observational, inductive, or learning biases that direct the learning process towards identifying physically consistent solutions (Lu et al., 2021).

Several methods have been proposed to incorporate observational bias in the form of input data generated from variable fidelity physical systems as a weak form of prior knowledge (Lu et al., 2021; Kashefi et al., 2021; Li et al., 2021). However, abundant data must

be available, especially for large deep-learning models, making scaling these approaches to other areas difficult. Another approach to PIML involves incorporating a learning bias into the learning model by penalizing the cost function of deep learning algorithms regarding adherence to a physical constraint, such as satisfying a partial differential equation (PDE) associated with the modelled process. In this way, the learning model attempts to fit both the observational data and the constraining equation or rule, for example, conservation of momentum and mass (Raissi et al., 2019; Lagaris et al., 1998; Zhu et al., 2019; Geneva & Zabaras, 2020; Wu et al., 2020). This, however, only adds a soft constraint to the learning model as the physics no longer plays a direct, explicit role after the training of the model is complete. A more direct approach to PIML is incorporating an inductive bias into the learning model through crafted architectures of neural networks (NNs) to embed prior knowledge and constraints related to the given task, such as symmetry and translation invariance. For differential equations in particular, several approaches have been investigated to modify NN architectures to satisfy boundary conditions or encode a priori parts of PDE solutions (Lagaris et al., 1998; Beidokhti & Malek, 2009; Wang et al., 2020; Cai et al., 2020; Darbon & Meng, 2021). However, these models can struggle with extremely complex emergent behaviours arising from unknown dynamics outside the scope of hand-coded physics (Miles et al., 2020).

Differentiable physics-based PIML methods are based on incorporating inductive biases into the neural network architectures. This has allowed the integration of physical simulators directly into deep learning architectures through automatic differentiation (Degrave et al., 2019; Holl et al., 2020). But most of these applications have focused on modelling isolated phenomena rather than full-system behaviours and provide particular architectures suited to only the respective problem spaces (Li et al., 2019; Toussaint et al., 2018; Sanchez-Gonzalez et al., 2018). A generic framework for integrating physics models into learning-based data-driven models remains an open challenge. Furthermore, applying these methods in digital twin scenarios where the observational data and prior knowledge are limited and only partially known is restrictive and requires thorough exploration. The method defined in this paper herewith effectively addresses these issues.

3 Knowledge blocks with residual learning

To address the problem of learning the behaviour of a target system with limited observation data and partially known physics, we propose a novel approach to integrate physics models into traditional neural networks to form modular architectures with ‘knowledge blocks’. These blocks are of two categories:

1. Physics blocks: representations of known physics equations as computational graphs comprising non-trainable parameters and static operators to reflect the mathematical operations of the equations.
2. Learning blocks: conventional neural network layers comprising trainable parameters (weights).

The physics blocks provide an a priori understanding of the system’s fundamental behaviour. The learning blocks can learn unknown, intermediate variables in the input space required by the physics blocks to predict the target variable(s). Collectively, the connected blocks form a neural network that can be trained end to end on observation data generated

by the target system. For the scope of this work, we assume that the physics block encoding the known physics equation(s) of the target system can be represented as a differentiable physics model. This is necessary to define the physics equation(s) as a computational graph that can be integrated with the conventional learnable layers of a neural network.

To explain the proposed architecture, we define a Behaviour Cloning problem where expert demonstrations from the target system are used to learn a policy to model observed states (inputs) to actions (target values) taken. For a set of expert demonstrations T from the target system, we assume i.i.d state-action pairs as:

$$T = \left\{(\mathbf{x}_1, a_1), (\mathbf{x}_2, a_2) \ldots (\mathbf{x}_n, a_n)\right\}$$

We define a learning block as a set of fully connected layers $\mathcal{L}(\mathbf{x};\theta)$ that takes observed state vector $\mathbf{x}$ and outputs an intermediate state vector $\mathbf{l}$. The learning block has a set of trainable weights ϕ.

$$\mathbf{l} = \mathcal{L}(\mathbf{x};\phi)$$

This intermediate state could include unknown variables from the environment or parts of the behaviour that are not directly defined by prior physics and thereby require empirical modelling. We also define our physics block as a computational graph $\mathcal{P}$ that predicts an action $a \in \mathcal{A}$, from the available set of actions $\mathcal{A}$ usually taken by the target system, using the observed state x and the intermediate state $\mathbf{l}$:

$$\hat{a} = \mathcal{P}(\mathbf{x}, \mathbf{l})$$

We can then define our policy function π to predict optimal action given the current state:

$$\hat{a} = \pi(\mathbf{x}, \mathcal{L}, \mathcal{P}) \text{ where } \hat{a} \in \mathcal{A}$$

3.1 Residual learning in knowledge blocks

While the physics blocks provide key a priori knowledge to the model, they pose a significant problem in converging the loss function in the gradient descent algorithm. If the fixed relationships defined by these layers conflict with the learning blocks' modelled patterns, it can result in conflicting gradients. This can make the training process highly unstable, thereby affecting convergence. Furthermore, static layers, such as those representing fixed physics equations, do not have learnable parameters or tweakable weights. When these static layers are integrated into a neural network, they can disrupt the flow of gradients to the learnable layers, often exacerbating the known problems of vanishing and exploding gradients.

To overcome this problem, we propose to add 'residual blocks' (similar to the concept introduced in the original ResNet paper (He et al., 2016) that branch out from the learning blocks to predict residual target value to be added to the physics block output. This introduces alternative pathways for gradients to flow to the learning blocks, mitigating the effect of the static layers within the physics blocks. Figure 3 shows an example case with two target intermediate variables and two target values to be predicted by the model. Intermediate, unknown variables l_1 and l_2 are predicted (discovered) by the learning blocks and passed to the physics block as inputs alongside the original input vector X. Intermediate feature vectors V_1 and V_2 are passed from the learning blocks to the residual blocks to predict residual

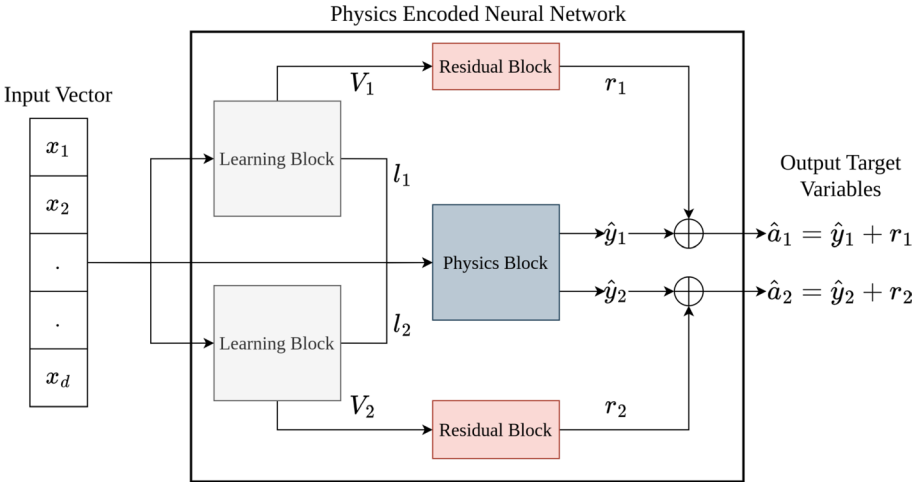


Fig. 3 Proposed Physics-Encoded Neural Network Architecture with Residual Blocks. The example shows a problem case with two target intermediate variables and two target values to be predicted. Intermediate, unknown variables l_1 and l_2 are predicted (discovered) by the learning blocks and passed to the physics block as inputs alongside the original input vector X. Intermediate feature vectors V_1 and V_2 are passed from the learning blocks to the residual blocks to predict residual target values r_1 and r_2. The residuals eventually add up with the physics-predicted target values y_1 and y_2 to output the final target values a_1 and a_2

target values r_1 and r_2. The residuals eventually add up with the physics-predicted target values y_1 and y_2 to output the final target values a_1 and a_2.

Consequently, we can update the following in our earlier example of a Behavioral Cloning problem. An intermediate layer output vector V from the learning block $\mathcal{L}$ will be passed to a separate residual block $\mathcal{R}$, which also comprises trainable parameters ϕ_r.

$$r = \mathcal{R}(V;\phi_r)$$

The residual target value r will then be added to the output from the physics blocks to predict the action $\hat{a}$.

$$\hat{a} = \mathcal{P}(\mathbf{x}, \mathbf{l}) + \mathbf{r}$$

Thereby, the final policy function to the Behaviour Cloning problem will become:

$$\hat{a} = \pi(\mathbf{x}, \mathcal{L}, \mathcal{P}, \mathcal{R}) \text{ where } \hat{a} \in \mathcal{A}$$

The gradient descent algorithm optimises any suitable loss function $L(a, \hat{a})$ that defines the difference between a and $\hat{a}$. Since the policy function is differentiable and comprises the affine connection of $\mathcal{L}$ and $\mathcal{P}$, the gradients from ∇L will flow through the chain of static operators from the physics block $\mathcal{P}$ to optimize the weights in the learning block $\mathcal{L}$, thereby modulating the learning of intermediate unknown variables using known physical constraints of the system. This essentially gives the final model the capacity to learn unknown, empirical relationships in data while maintaining the fundamental structure of the physics of the behaviour being modelled.

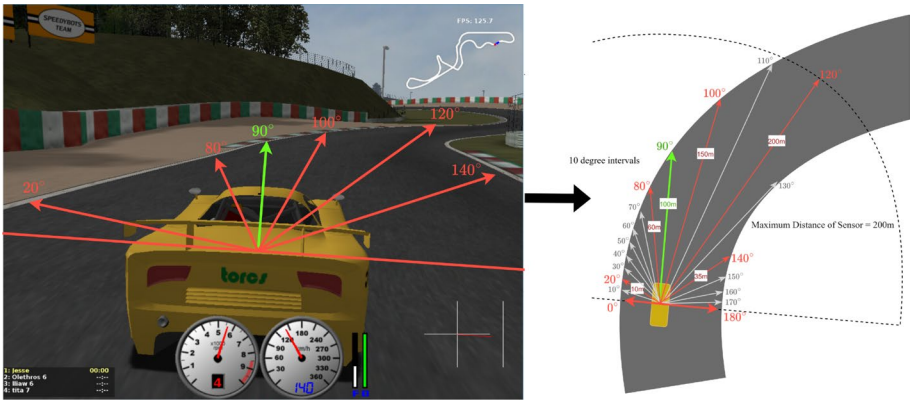


Fig. 4 Illustration of the 19 range finder sensors (R) comprising distances to track edges, spanning clockwise from 0 to 180 degrees to the car axis at 10-degree intervals. The image from the TORCS simulation on the left shows a sample of sensors at different angles depicted with red arrows, which is further elaborated on in the figure on the right side. The maximum range of the sensors is 200 m. A similar sensor setup defines distances to surrounding traffic vehicles (O)

4 Experimental study

We validate the effectiveness of our approach in two distinct experimental setups. The first study involves modeling steering dynamics for autonomous vehicle simulation. In this scenario, we investigate our method within a simulated digital twin problem where the available prior physics is relatively simple and only partially defines the system's dynamics. The second study addresses a more complex, real-world digital twin problem for modeling CO_2 emissions and gap-filling of Net Ecosystem Exchange (NEE) in flux data. Here, we examine the ability of our approach to integrate an ordinary differential equation that partially characterizes the behavior of emissions with data-driven learning of the remaining dynamics from flux measurements, thereby enabling a forecasting model for gap-filling in emissions data–an important application in the climate digital twin domain

4.1 Case study 1: steering dynamics for autonomous vehicle simulation

We now apply this framework to a more complex digital twin problem where the available data and prior physics may not be substantially available to completely model the system dynamics independently. We investigate our approach in an autonomous driving task of steering a vehicle in a simulation as it manoeuvres around a race track. Using data generated from an existing driving agent within the simulation (treated as a reference agent), we train a feed-forward neural network architecture that combines fully connected layers that capture perceptual understanding with relevant mathematical operators from the physics-based steering model known as Pure Pursuit. Figure 7 illustrates the proposed architecture and is explained in detail in Sect. 4.1.2.

We use the TORCS open-source driving simulator as our experimental environment due to the tracks' diversity and rich non-visual sensory and perceptual information (Espié et al., 2005). The input parameter space offered to the model consists of a vehicle's state variables and perception information from the surrounding environment. Table 1 summarizes

Table 1 System and actuator variables in the driving environment

Variable	Range (unit)	Description
State variables		
α_{axis}	$[-\pi, +\pi]$ (rad)	Angle between the car direction and the track axis
D_{center}	$(-\infty, +\infty)$ (m)	Normalized distance from the car to the track axis (0 on axis, −1 at the right edge, +1 at the left edge)
z	$(-\infty, +\infty)$ (m)	Vertical distance of the car's mass centre from the track surface
$V_{x,y,z}$	$(-\infty, +\infty)$ (km/h)	Speed vector in $\mathbb{R}^3$
R	[0, 200] (m)	19 range finder sensors: each returns the distance from the car to the track edge, sampling every 10° from −90° to +90° relative to the car's axis
O	[0, 200] (m)	19 range finder sensors for traffic vehicles with the same angular sampling
Actuator variables		
δ	$[-1, +1]$	Steering value: −1 and +1 represent the right and left bounds, corresponding to 0.366519 rad
a	[0, 1]	Virtual acceleration pedal (0 = no gas, 1 = full gas)
b	[0, 1]	Virtual braking pedal (0 = no brake, 1 = full brake)

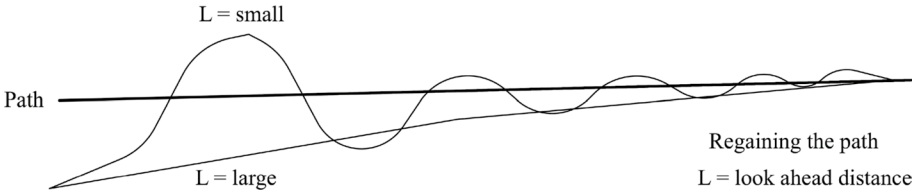


Fig. 5 The effect of look-ahead distance magnitude on the behaviour of path convergence (Degrave et al., 2019)

the subset of the available variables from the simulation environment used as input parameters and actuators/actions for our model.

The vector of 19 range finder sensors (R) comprises the distance between the track edge and the car centre within a range of 200 ms. Each value represents clockwise distances from 0 to 180 degrees to the car axis at 10-degree intervals. This is similar to how an automotive RADAR sensor is used to localize objects in the long-range vicinity of the autonomous vehicle. Figure 4 illustrates this sensor setup. The same methodology defines the distances to surrounding traffic vehicles to define vector O, as Table 1 explains.

4.1.1 Physics-based pure pursuit steering model

The pure pursuit algorithm is a path-tracking algorithm that steers a vehicle along a desired path, given a set of reference points to follow. The algorithm calculates the steering angle

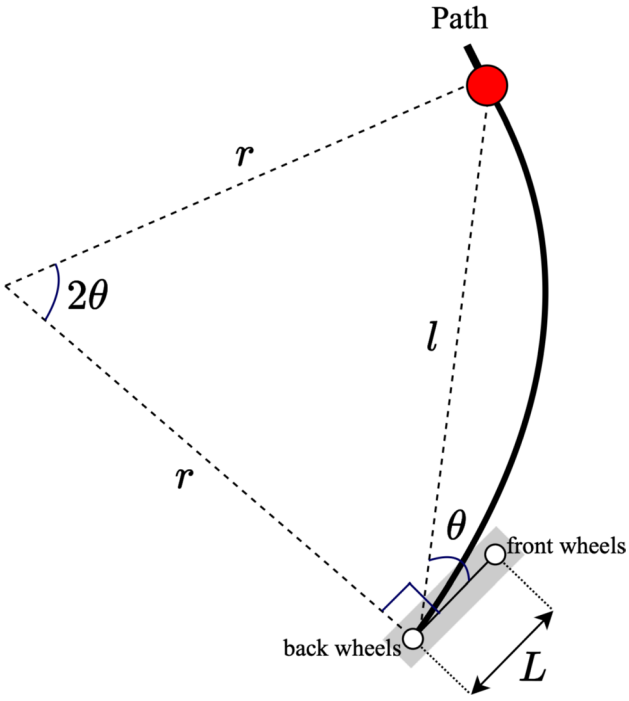


Fig. 6 Geometric relationship between the circle of rotation, wheelbase of the vehicle (L), lookahead distance (l), and the heading difference (θ)

required to follow the path based on the vehicle's current position and velocity (Darbon & Meng, 2021). It operates based on the concept of "pursuit," where a hypothetical point, known as the look-ahead point, is established ahead of the vehicle on the desired path. The objective is to steer the vehicle towards this point, thereby ensuring path tracking. The algorithm computes the optimal steering angle required to reach the look-ahead point by analyzing the geometric relationship between the vehicle and the path.

Firstly, the algorithm estimates the position of the look-ahead point by projecting it along the desired path based on the vehicle's current position. Subsequently, the algorithm computes the curvature of the path at the look-ahead point using mathematical techniques such as interpolation or curve fitting. Finally, utilizing the vehicle's kinematic model, the algorithm calculates the required steering angle to navigate towards the look-ahead point. The look-ahead distance is a control parameter in the context of the algorithm and affects the outcome in two contexts: (1) Regaining a path, i.e., the vehicle is at a "large" distance from the path and must attain the path. (2) Maintaining the path, i.e., the vehicle is on the path and wants to remain on the path. The effects of changing the parameter in the first problem are easy to imagine using the analogy to human driving. The longer look-ahead distances converge to the path more gradually and with less oscillation, as illustrated in Fig. 5. It, therefore, depends on the state of the driving environment at any point in time to optimally set the look-ahead distance to manoeuvre the track safely.

The Pure Pursuit model defines a reference point on the road as a vector of two observed scalar values: the heading difference (angle) to the direction of the reference point and the lookahead distance to the reference point. Formally, this can be represented as:

$$\mathbf{z} = \begin{bmatrix} l \\ \theta_{target} \end{bmatrix}$$

where $\mathbf{z}$ is the reference point, and l and θ_{target} are the lookahead distance and heading difference accordingly. We assume that the vehicle follows the principle of a rigid body moving around a circle and, therefore, compute the required steering angle using this circle's instantaneous centre of rotation. Figure 6 illustrates this concept. Consequently, from the arc of rotation depicted in the figure, we can use the law of sine to calculate the curvature, k, of the path to the selected point as:

$$\begin{aligned} \frac{l}{\sin 2\theta} &= \frac{r}{\sin(\frac{\pi}{2} - \theta)} \\ \frac{l}{\sin\theta\cos\theta} &= \frac{r}{\cos\theta} \\ \frac{l}{\sin\theta} &= 2r \\ k = \frac{1}{r} &= \frac{2\sin\theta}{l} \end{aligned} \tag{1}$$

Here, r is the radius of rotation for the vehicle. Furthermore, we know from the kinematic bicycle model that the radius of rotation has an inverse relationship with the steering vehicle as:

$$r = L/\tan(\delta)$$

Here, δ is the steering angle and L is the wheelbase of the vehicle (horizontal distance between rear and front wheel centers). Therefore, we can substitute r from the equation of curvature (Eq. 1) to get the required steering angle as:

$$\delta = \arctan(\frac{2L\sin\theta}{l}) \tag{2}$$

4.1.2 Physics encoded residual feed-forward network

Based on the approach introduced in Sect. 3, we formulate our physics block as the mathematical operations calculating the target steering angle within the Pure Pursuit algorithm. As previously discussed in Sect. 4.1.1, the selected reference point and look-ahead distance are key parameters that dictate the steering behaviour and are based on the geometry of the driving path ahead. Consequently, we formulate our learning block as a set of fully connected layers to output the optimal look-ahead distance l and heading difference θ_{target} to represent the selected reference point $\mathbf{z}$ on the road to follow (see Eq. 4.1.1). It is important to note that these variables are unknown in the input space (sensor information) and are thereby 'discovered' by the learning blocks.

Algorithm 1 Two-phased behavior cloning using physics-encoded neural network

Input: Demonstration Data 1 ($\mathcal{D}_1$), Demonstration Data 2 ($\mathcal{D}_2$), Loss Threshold (ℓ_{thresh})
Output: Integrated Model (M)
Initialize Integrated Model M comprising three blocks $\mathcal{L}$, $\mathcal{R}$, $\mathcal{P}$
Initialize learning block $\mathcal{L}$ (learnable)
Initialize physics block $\mathcal{P}$ (non-learnable)
Initialize residual block $\mathcal{R}$ (learnable)
Initialize heuristic function H for label generation

Phase I: Warm Start - Train Block $\mathcal{L}$ with Heuristic Labels
repeat
    Sample mini-batch of state-action pairs from $\mathcal{D}_1$
    Generate labels using the heuristic function H
    Forward pass through block $\mathcal{L}$
    Compute loss (ℓ_1) for predicted lookahead distance and target heading differences
    Backward pass and update the weights in block $\mathcal{L}$
until convergence

Phase II: Train Complete Model M
Initialize block layers in $\mathcal{L}$ with weights from Phase I
repeat
    Sample mini-batch of state-action pairs from $\mathcal{D}_2$
    Forward pass through models $\mathcal{R}$ and $\mathcal{L}$ with static block $\mathcal{P}$
    Compute loss (ℓ_2) for predicted steering
    Backward pass and update weights in blocks $\mathcal{R}$ and $\mathcal{L}$
until $\ell_2 < \ell_{\text{thresh}}$

return Trained model M

These variables are then fed to the physics block, a computational graph expressing the steering angle as a geometric combination of the lookahead distance and heading difference from the fully connected layers and the vehicle's wheelbase (available in the input

space). These include the operations in Eq. 2 to calculate the target steering based on the input variables.

To formally define each block, let $\mathbf{x}$ be the vector of input parameters (explained in Table 1). Let us assume $\mathcal{L}(\mathbf{x}, \phi)$ represents the function modelled by the learning block predicting the lookahead distance l and target heading difference θ_{target}, with known input variables x and a set of learnable weights ϕ_{learn}.

$$\mathcal{L} : \mathbb{R}^d \times \mathbb{R}^m \rightarrow \mathbb{R}^2$$
$$\mathcal{L}(\mathbf{x}, \phi_{learn}) = \begin{bmatrix} l \\ \theta_{target} \end{bmatrix}$$

where d is the number of state features in the input vector (from available state parameters shown in Table 1), and m is the total number of weights in the fully connected layers.

Let us also assume our physics block $\mathcal{P}(\mathbf{x}, \mathcal{L})$ to predict the steering value δ_{phys} using the known input state and the intermediate unknown variables predicted by $\mathcal{L}$, using operators from Eq. 2.

$$\mathcal{P}(\mathbf{x}, \mathcal{L}) = \arctan(\frac{2\,L \sin\theta}{\mathcal{L}(x)})$$

We also define a residual block $\mathcal{R}(V;\phi_{res})$ comprising a set of fully connected layers with learnable weights ϕ_{res}, which inputs a feature vector from an intermediate layer from the learning block $\mathcal{L}$ as input. The residual block predicts the residual target value r, which is added to the physics-predicted steering value $\hat{\delta}_{phys}$ to formulate the final steering value from the model.

$$\hat{\delta} = \mathcal{P}(\mathbf{x}, \mathcal{L}) + \mathbf{r}$$

Based on the Behavior Cloning problem discussed in Sect. 3, we can assume a set of demonstration data T from an expert driving agent as i.i.d state-action pairs:

$$T = \{(\mathbf{x}_1, \delta_1), (\mathbf{x}_2, \delta_2)...(\mathbf{x}_n, \delta_n)\}$$

Here, $\mathbf{x}_i$ and δ_i are observed input state and steering values from the expert driving agent. We can then define the predicted steering value (target value) based on the policy function π as:

$$\hat{\delta} = \pi(\mathbf{x}, \mathcal{L}, \mathcal{P}) \text{ where } \hat{a} \in \mathcal{A}$$

The loss function is based on Mean Squared Error (MSE), minimizing the error between the observed (δ) and predicted ($\hat{\delta}$) steering angles. This is defined as:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\delta_i - \hat{\delta}_i)^2$$

We use the *Mini-Batch Gradient Descent* algorithm and *Adam Optimizer* to optimize the loss function, where n is any arbitrary batch size for each sample in the gradient descent algorithm. The gradients for the learnable weights in ϕ_{learn} are calculated by the chain of function derivatives that involve the geometric and kinematic operators from Eq. 1. This allows our feed-forward network to learn the optimal lookahead while being forced to comply with the geometric and kinematic aspects of the steering problem dictated by the pure pursuit algorithm. This is done by constraining the calculation of gradients, which

are calculated as an explicit part of the equations based on the pure pursuit algorithm. On the other hand, the residual block provides an alternative pathway for the gradients to flow back to the intermediate layers of the learning block. This allows the loss function to converge and optimize the weights in the residual and, more importantly, the learning block.

4.1.3 Training on demonstration data

We train our physics-encoded neural network model on the demonstration data T in two sequential phases. The first phase involves a *warm-start* phase where the learning block is trained separately to predict appropriate lookahead distance (l) and target heading difference (θ) values. We generate two batches of data from our simulation (details in Sect. 5.1.1). The first batch is generated from the agent driving on empty segments of tracks, and the second batch is generated from the agent driving amongst other traffic vehicles. The sequence of steps in the training process is defined in Algorithm 1.

Phase I: warm start

In this phase, the learning block is trained separately on the first dataset ($\mathcal{D}_1$) collected from empty road scenarios. We define a heuristic function to provide the learning block with proposed values of lookahead distance (l) and target heading difference (θ_{target}) for each data point. The heuristic function selects the reference point to be in the direction of the furthest point on the road boundary from the vehicle, using the range-finding sensor vector R (refer to Table 1).

Figure 8 illustrates this method. The range finder sensors can be considered to run clockwise from the vehicle's horizontal plane, from the left side of the vehicle, forming a semi-circle axis in front of the vehicle. The sensors capture values every 10 degrees on this axis, clipped to a maximum value of 200 m. These values depict the presence of a road boundary at the corresponding angle value. Formally, for R_t as the vector of range finder sensors from a data point t, we can define:

$$i = argmax(R_t)$$

where i is the index of the maximum distance value in vector R_t. If there are multiple maximum values, the index of the first of these values is picked. Using this index i, we can calculate θ_t (in degrees) and l_t, for any given data point t, as follows:

$$\theta_t = 10i - 90$$
$$l_t = R_t^i$$

We then use the output from the heuristic function to provide target labels for the learning block. The loss function is based on MAE (Mean Absolute Error), minimizing the error between the heuristic output and predicted values from the learning block. This is optimized using a simple Mini-Batch Gradient Descent with *Adam Optimizer* until an arbitrary point of convergence where the mean loss function value across mini-batches falls below a threshold.

Phase II: integrated training

We refer to the complete model M as the connected neural network with the learning ($\mathcal{L}$), learning ($\mathcal{R}$) and learning ($\mathcal{P}$) blocks, as shown in Fig. 7. The learned weights from Phase I are used to initialize the learning block. We then train M on the second dataset ($\mathcal{D}_2$) comprising traffic scenarios, with the observed steering values as target values from which the model can learn. The training is based on the explanation in Sect. 4.1.2.

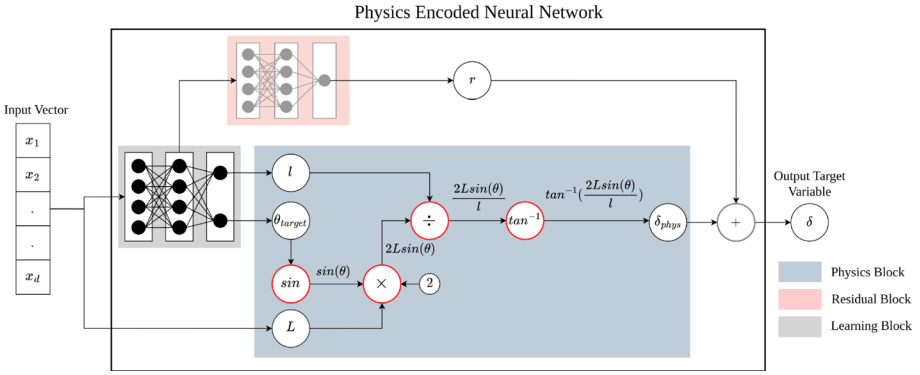


Fig. 7 Physics encoded neural network architecture for the steering model. The learning block predicts two unknown variables from the input space: l (lookahead distance) and θ_{target} (heading difference to selected reference point). These variables are passed as input to the physics block to predict the steering angle δ. The intermediate layer output from the learning block is passed to the residual block to predict the residual value to be added to δ for the final steering value

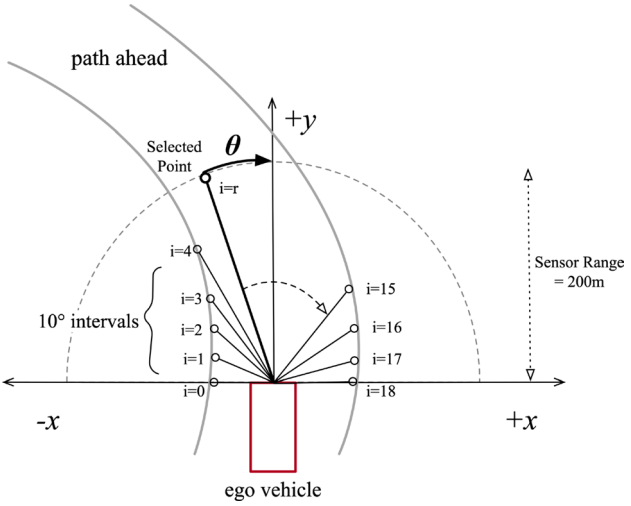


Fig. 8 Proposed method to find the reference direction for vehicle, based on the maximum range finder sensor values

4.2 Case study 2: climate digital twins for CO_2 emissions and gap-filling in net ecosystem exchange modelling

The application of digital twin technology to climate systems has emerged as a promising avenue for enhanced monitoring and forecasting of atmospheric CO_2 dynamics. Digital twins, which provide a virtual representation of real-world processes, are particularly valuable in climate science where continuous, high-quality observational data are critical yet often incomplete. Flux towers continuously monitor key atmospheric scalars such as Net Ecosystem Exchange (NEE), Latent Heat, and Sensible Heat (Zhu et al., 2022) using the

eddy covariance (EC) method. However, these measurements frequently exhibit gaps due to instrumentation failures, power shortages, or environmental interferences (Reichstein et al., 2005; Moffat et al., 2007).

In the context of a digital twin for climate applications, robust gap-filling methods are essential not only for data completion but also for accurately capturing the underlying physical processes governing NEE. Traditional gap-filling techniques–such as Non-Linear Regression, Look-Up Tables, and Marginal Distribution Sampling–have shown reasonable performance for short-duration gaps, yet they struggle with longer intervals of missing data (Moffat et al., 2007). More recent approaches, such as the Random Forest Robust (RFR) method, have enhanced performance over longer gaps by improving the statistical robustness of the estimates, although challenges remain for nighttime measurements (Zhu et al., 2022).

Integrating the underlying physics of NEE into the gap-filling process can address these challenges. As demonstrated in previous work, NEE dynamics can be modelled as a Stochastic Differential Equation (SDE) that decomposes the signal into a deterministic drift component–governed by temperature-dependent ecosystem respiration following an Arrhenius-type law (Lasslop et al., 2010; Lloyd & Taylor, 1994)–and a stochastic diffusion term representing measurement noise, assumed to be Gaussian (White & Luo, 2008). Embedding this SDE within a digital twin framework allows the inherent physics of ecosystem CO_2 exchange to directly inform the data reconstruction process, leading to more reliable and interpretable predictions.

Our proposed Physics Encoded Neural Network (PERNN) framework naturally extends to this climate modelling use case. In this setting, the physics blocks encapsulate the SDE-based model of NEE, capturing both the deterministic Arrhenius-type response of ecosystem respiration and the inherent stochastic fluctuations. Complementary learning blocks infer latent variables and compensate for unobserved dynamics during periods of data absence. Moreover, residual blocks not only aid gradient flow during training but also facilitate the integration of these physics-informed estimates with empirical corrections, ensuring that the digital twin maintains fidelity to both the established physical laws and the observational data.

This hybrid architecture enhances gap-filling performance while simultaneously improving the interpretability of NEE trends, making it a robust tool for climate digital twins. By directly incorporating physical models into the neural network, our approach bridges the gap between purely data-driven methods and traditional physics-based modeling, thereby providing a scalable and generalizable framework for environmental monitoring and forecasting (Tao et al., 2019; Jones et al., 2020).

4.2.1 Flux tower data

The observational data employed in this study were collected from a flux tower located in East Anglia, UK (Cumming et al., 2020) from 2012 to 2019. The tower is equipped with an eddy covariance (EC) system designed to measure key atmospheric scalars including Net Ecosystem Exchange (NEE), sensible heat flux density (H), momentum flux (Tau), relative humidity (RH), vapor pressure deficit (VPD), global radiation (R_g), friction velocity (Ustar), soil temperatures (Tsoil1 and Tsoil2), and air temperature at 2 m (T_{air}). The dataset spans eight years, from 2012 through the end of 2019, with measurements recorded at 30-minute intervals.

These high-frequency measurements provide a rich temporal resolution of the ecosystem's CO_2 exchange dynamics. However, the dataset also presents common challenges such as missing values caused by power shortages, sensor malfunctions, or adverse weather conditions. These data gaps can range from short intervals to periods extending over

Table 2 List of variables from the flux data Cumming et al. (2020)

Variable	Units	Description
NEE	µmol C $m^{-2}s^{-1}$	Net ecosystem CO_2 exchange flux density before data gap-filling
H	W m^{-2}	Sensible heat flux density
Tau	kg $m^{-1}s^{-2}$	Momentum flux
RH	%	Relative humidity at 2 m
VPD	HPa	Vapor pressure deficit
R_g	W m^{-2}	Global radiation
Ustar	m s^{-1}	Friction velocity
Tsoil1	°C	Soil temperature at a depth of 0.05 m
Tsoil2	°C	Soil temperature at a depth of 0.05 m
T_{air}	°C	Air temperature at 2 m

several months, emphasizing the necessity for robust gap-filling methods that can reliably reconstruct missing measurements while preserving the underlying physical relationships.

A detailed description of the measured variables is provided in Table 2. In this study, particular attention is given to the NEE variable, which is central to understanding the carbon dynamics of the ecosystem. Given that latent heat (L) is measured by the same instrument as NEE, any gaps in NEE often correspond to gaps in latent heat measurements; hence, latent heat is excluded from the analysis to maintain consistency. Additionally, time-based attributes such as season, hour, day of week, month, and day of year are incorporated to account for diurnal and seasonal variations.

4.2.2 The net ecosystem exchange model

The Net Ecosystem Exchange (NEE) quantifies the net flux of CO_2 between an ecosystem and the atmosphere. It is defined as the balance between ecosystem respiration and photosynthetic uptake. Specifically, NEE can be expressed as:

$$\mathrm{NEE}_t = \mathrm{R}_{\mathrm{eco},t} - \mathrm{GPP}_t, \tag{3}$$

where $R_{eco,t}$ represents the rate of CO_2 release due to biological respiration, and GPP_t denotes the Gross Primary Production associated with photosynthesis. By convention, negative NEE values indicate net CO_2 uptake by the ecosystem (Lasslop et al., 2010; Keenan et al., 2019).

For periods when global radiation is insufficient (i.e., $R_g < 20\,\mathrm{W.m^{-2}}$), photosynthesis is assumed to be negligible, resulting in $GPP_t \approx 0$. Under these conditions, the measured NEE predominantly reflects ecosystem respiration. The temperature dependence of ecosystem respiration is commonly modelled using an Arrhenius-type expression. Accordingly, the respiration term at night is formulated as:

$$\mathrm{R}_{\mathrm{eco},t} = r_{night} \exp\left(E_0 \left(\frac{1}{T_{ref} - T_0} - \frac{1}{\mathrm{T}_{\mathrm{air},t} - T_0} \right) \right), \tag{4}$$

where r_{night} (in µmol C $m^{-2}s^{-1}$) is the baseline respiration rate at the reference temperature $T_{ref} = 15°\mathrm{C}$, E_0 (in °C) is a constant that characterizes temperature sensitivity, $T_{air,t}$ is the ambient air temperature at time t, and T_0 is a fixed offset (typically −46.02°C) (Lasslop

et al., 2010). For consistency, the parameter r_{night} is periodically updated–commonly every five days–using estimations derived from 15-day windows of historical observations.

This formulation of NEE as described above forms the foundation of our modelling approach.

4.2.3 ODE-based net ecosystem exchange model

In our approach, the dynamics of Net Ecosystem Exchange (NEE) are modeled using an ordinary differential equation (ODE) that captures the temperature-dependent behaviour of ecosystem respiration. Specifically, the evolution of NEE over time is expressed as

$$\frac{d\,\mathrm{NEE}_t}{dt} = \frac{d}{d\,\mathrm{T}_{\mathrm{air},t}}\mathrm{R}_{\mathrm{eco},t}(\mathrm{T}_{\mathrm{air},t})\,\frac{d\,\mathrm{T}_{\mathrm{air},t}}{dt}, \tag{5}$$

where $\mathrm{R}_{\mathrm{eco},t}(\mathrm{T}_{\mathrm{air},t})$ denotes the ecosystem respiration rate, modeled via an Arrhenius-type function given in Eq. 4.

The derivative of the respiration function with respect to air temperature is given by

$$\frac{d}{d\,\mathrm{T}_{\mathrm{air},t}}\mathrm{R}_{\mathrm{eco},t}(\mathrm{T}_{\mathrm{air},t}) = \frac{E_0}{(\mathrm{T}_{\mathrm{air},t} - T_0)^2}\,\mathrm{R}_{\mathrm{eco},t}, \tag{6}$$

and the rate of change of the ambient air temperature is modeled as

$$\frac{d\,\mathrm{T}_{\mathrm{air},t}}{dt} = \pi\,\frac{\Delta\mathrm{T}_{\mathrm{air},t}}{t_{day}}\,\sin\!\left(2\pi\,\frac{t - t_{T_{\max}}}{t_{day}}\right). \tag{7}$$

By incorporating Eq. 5 into our Physics Encoded Residual Neural Network (PERNN) framework, we directly embed the known temperature-driven dynamics of ecosystem respiration into the digital twin. This integration ensures that while the neural network components can infer latent variables and correct for discrepancies, the overall model remains grounded in the established physical relationships governing CO_2 exchange.

4.2.4 Physics encoded residual neural network for NEE ODE

In our NEE modeling application, we extend the Physics Encoded Residual Neural Network (PERNN) framework to forecast the evolution of Net Ecosystem Exchange (NEE) by integrating the known deterministic dynamics of ecosystem respiration into a digital twin. The goal is to predict the rate of change of NEE, $\frac{d\,\mathrm{NEE}}{dt}$, and subsequently obtain the forecasted NEE at the next time step via

$$\mathrm{NEE}_{t+1} = \mathrm{NEE}_t + \widehat{\frac{d\,\mathrm{NEE}}{dt}} \cdot \Delta t, \tag{8}$$

where Δt is 30 min, consistent with the measurement interval of the flux data.

Let $\mathbf{x} \in \mathbb{R}^d$ denote the vector of input features, where d corresponds to the number of variables defined in Table 1 (e.g., $\mathrm{T}_{\mathrm{air}}$, R_{g}, RH, VPD, etc.). These features encapsulate the meteorological and temporal context required for the NEE model.

The learning block, denoted by $\mathcal{L}(\mathbf{x}, \phi_{learn})$, is implemented as a set of fully connected layers. It processes the input state $\mathbf{x}$ and infers the intermediate variables that are not directly observable:

$$\mathcal{L}(\mathbf{x}, \phi_{learn}) = \begin{bmatrix} E_0 \\ r_{b\_night} \\ \left(\frac{dT}{dt}\right)_{pred} \end{bmatrix}.$$

Here, E_0 represents the temperature sensitivity, r_{b_night} is the base respiration at the reference temperature, and $\left(\frac{dT}{dt}\right)_{pred}$ is the predicted rate of change of ambient air temperature. These variables are subsequently fed to the physics block.

The physics block, $\mathcal{P}(\mathbf{x}, \mathcal{L})$, is a non-trainable computational graph that encapsulates the physical operators derived from the ODE governing NEE dynamics. Using the predicted temperature derivative $\left(\frac{dT}{dt}\right)_{pred}$ from the learning block, the physics block computes the change in NEE based on Eq. 10 as:

$$\mathcal{P}(\mathbf{x}, \mathcal{L}) = \frac{d\,\mathrm{NEE}_t}{dt} = \frac{E_0}{(\mathrm{T}_{\mathrm{air},t} - T_0)^2}\,\mathrm{R}_{\mathrm{eco},t}\left(\frac{dT}{dt}\right)_{pred}. \tag{9}$$

To compensate for any discrepancies between the physics-based prediction and the actual dynamics observed in the data, a residual block $\mathcal{R}(V;\phi_{res})$ is employed. Here, V represents an intermediate feature vector extracted from the learning block. The residual block predicts an additive correction r such that the final prediction of the rate of change in NEE is:

$$\widehat{\frac{d\,\mathrm{NEE}}{dt}} = \mathcal{P}(\mathbf{x}, \mathcal{L}) + r. \tag{10}$$

Ground-truth computation Ground-truth values for the rates of change are computed from the flux tower measurements using a right-sided first-order finite difference approximation. Specifically, for NEE:

$$\frac{d\,\mathrm{NEE}_t}{dt} \approx \frac{\mathrm{NEE}_{t+1} - \mathrm{NEE}_t}{\Delta t}, \tag{11}$$

and similarly for air temperature:

$$\frac{d\,T_{air,t}}{dt} \approx \frac{T_{air,t+1} - T_{air,t}}{\Delta t}, \tag{12}$$

with $\Delta t = 30$ minutes.

Parameter estimation for E_0 ***and*** r_{night} Since the flux tower data does not directly provide ground-truth values for the parameters E_0 and r_{night}, these are estimated using the REddyProc partitioning algorithm (Wutzler et al., 2018). REddyProc applies a non-linear regression approach based on the methodology of (Reichstein et al., 2005) to partition the flux data, particularly during nighttime conditions when photosynthetic uptake is negligible. In this context, the Lloyd-and-Taylor model (Lloyd & Taylor, 1994) is fitted to the scatter of NEE and $\mathrm{T}_{\mathrm{air},t}$ values, providing estimates within established ranges (e.g., $E_0 \in [50, 400]$ and $r_{night} > 0$) as discussed in Lasslop et al. (2010).

Loss function and two-phased training procedure The model is trained to minimize the Mean Squared Error (MSE) loss, with two distinct phases. We use the *Mini-Batch Gradient Descent* algorithm and *Adam Optimizer* to optimize the loss function, where n is any arbitrary batch size for each sample in the gradient descent algorithm.

- **Phase I: warm start:** In this phase, the learning block $\mathcal{L}$ is pre-trained using heuristic labels generated by inverting Eqs. 11 and 12. The MSE loss is computed solely on the intermediate variables:

$$L_1 = \frac{1}{n}\sum_{i=1}^{n}\left[\left(\widehat{E}_0^{(i)} - E_0^{(i)}\right)^2 + \left(\widehat{r}_{b\_night}^{(i)} - r_{b\_night}^{(i)}\right)^2 + \left(\widehat{\left(\frac{dT}{dt}\right)}^{(i)} - \left(\frac{dT}{dt}\right)^{(i)}\right)^2\right], \tag{13}$$

  where n is the number of samples in the mini-batch.
- **Phase II: integrated training:** In the second phase, the full model M – comprising the learning block $\mathcal{L}$, the fixed physics block $\mathcal{P}$, and the residual block $\mathcal{R}$ – is trained end-to-end. The MSE loss in this phase is computed on both the intermediate variables and the forecasted NEE_{t+1}:

$$\begin{aligned} L_2 = \frac{1}{N}\sum_{i=1}^{N}\Bigg\{ & \left(\widehat{\text{NEE}}_{t+1}^{(i)} - \text{NEE}_{t+1}^{(i)}\right)^2 \\ & + \left(\widehat{E}_0^{(i)} - E_0^{(i)}\right)^2 \\ & + \left(\widehat{r}_{b\_night}^{(i)} - r_{b\_night}^{(i)}\right)^2 + \left(\widehat{\left(\frac{dT}{dt}\right)}^{(i)} - \left(\frac{dT}{dt}\right)^{(i)}\right)^2\Bigg\}. \end{aligned} \tag{14}$$

  where $\widehat{\text{NEE}}_{t+1}^{(i)}$ is the forecasted value computed via Eq. 8.

Figure 9 illustrates the overall PERNN architecture for modeling the NEE ODE, and Algorithm 2 details the two-phased training procedure.

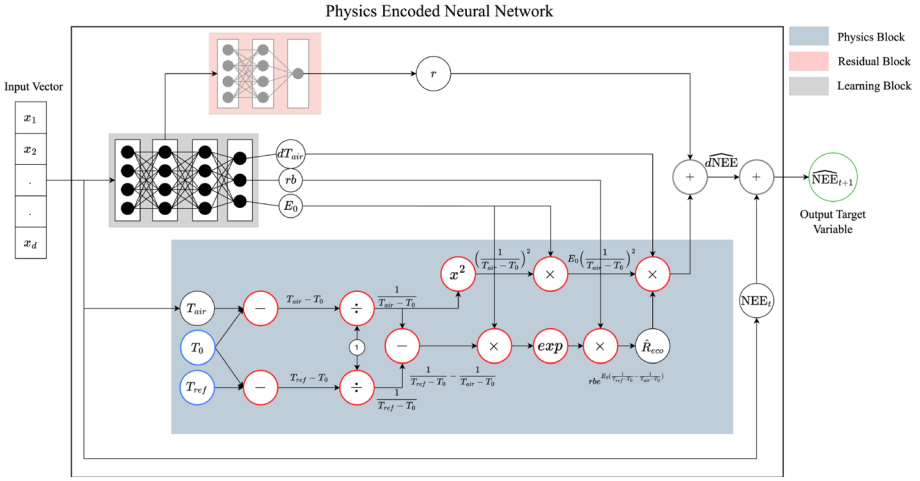


Fig. 9 Physics Encoded Neural Network architecture for NEE forecasting. The learning block predicts three intermediate variables: E_0 (temperature sensitivity), rb_{night} (base respiration) and dT_{air} (change in air temperature). These variables are passed as input to the physics block based on the NEE ODE to predict dNEE (change in NEE). The intermediate layer output from the learning block is passed to the residual block to predict the residual value to be added to $d\text{NEE}$ for the final prediction of the time derivative $\widehat{d\text{NEE}}$. This is then added to the current NEE_t to output $\widehat{\text{NEE}}_{t+1}$

Algorithm 2 Two-phased training for PERNN for NEE

Input: Training dataset $\mathcal{D}$, Loss threshold ℓ_{thresh}
Output: Trained PERNN model M
Initialize model M with:

- Learning block $\mathcal{L}$ (learnable)
- Physics block $\mathcal{P}$ (non-learnable)
- Residual block $\mathcal{R}$ (learnable)

Phase I: Pre-train Learning Block
repeat
  Sample a mini-batch from $\mathcal{D}$
  Forward pass through $\mathcal{L}$ to predict intermediate variables: $\widehat{E_0}$, $\widehat{r}_{b\_night}$, $\widehat{\left(\frac{dT}{dt}\right)}$
  Compute MSE loss L_1 on these variables:

$$L_1 = \text{MSE}\left(\begin{bmatrix} \widehat{E_0} \\ \widehat{r}_{b\_night} \\ \widehat{\left(\frac{dT}{dt}\right)} \end{bmatrix}, \begin{bmatrix} E_0 \\ r_{b\_night} \\ \left(\frac{dT}{dt}\right) \end{bmatrix} \right)$$

  Backpropagate and update weights in $\mathcal{L}$
until L_1 converges

Phase II: Train Integrated Model
repeat
  Sample a mini-batch from $\mathcal{D}$
  Forward pass through $\mathcal{L}$, then through $\mathcal{P}$ (fixed), and finally through $\mathcal{R}$ to obtain forecasted $\widehat{\text{NEE}}_{t+1}$
  Compute MSE loss L_2 on both the intermediate variables and $\widehat{\text{NEE}}_{t+1}$:

$$L_2 = \text{MSE}\left(\begin{bmatrix} \widehat{E_0} \\ \widehat{r}_{b\_night} \\ \widehat{\left(\frac{dT}{dt}\right)} \\ \widehat{\text{NEE}}_{t+1} \end{bmatrix}, \begin{bmatrix} E_0 \\ r_{b\_night} \\ \left(\frac{dT}{dt}\right) \\ \text{NEE}_{t+1} \end{bmatrix} \right)$$

  Backpropagate and update weights in $\mathcal{L}$ and $\mathcal{R}$
until $L_2 < \ell_{\text{thresh}}$
return Trained model M

5 Experiments

In this section, we present a comprehensive set of experiments and analyses that build upon the theoretical foundations described in the preceding sections. For the simulation-based autonomous vehicle steering dynamics use case–where the available prior physics is simple

and only partially known–we evaluate the performance of our approach in terms of generalizability, data efficiency, and model complexity. In the climate digital twin application, we assess our method's ability to accurately model Net Ecosystem Exchange (NEE) behavior over time and effectively perform gap-filling, comparing its performance against state-of-the-art techniques. These experiments serve to validate the advantages of our approach in both controlled simulation environments and real-world scenarios.

5.1 Autonomous vehicle simulation

In this section, we present details on the experimentation and results on the problem of modelling steering dynamics in a simulated autonomous vehicle problem. We compare our Physics Encoded Residual Neural Network (PERNN) approach against two methods. The first is based on traditional fully connected neural networks (FCNN) to serve as a baseline for a purely data-driven method. The second method is based on physics-regularized neural networks which are currently one of the most popular generic physics-informed machine learning methods in literature. This is based on applying soft constraints to the learning process by appropriately penalizing the loss function of neural networks. Assuming an external physics model P, input variables $\mathbf{x}$, ground truth target variable y and any feed-forward neural network f, the loss function of such a physics-informed network is:

$$L = L_{data} + L_{phy}$$

where L_{data} can be any function measuring the similarity between the predicted variable $f(\mathbf{x})$ and the target variable y, while L_{phy} can be any function measuring the similarity between the predicted variable $f(\mathbf{x})$ and the output from the physics model $P(\mathbf{x})$. L_{phy} can also represent the degree to which the neural network output satisfies the constraint on the system, which can be an equation such as a partial differential equation. The loss function can also be represented as a weighted sum of L_{data} and L_{phy}, where the weight value can be represented as a hyperparameter or learnable parameter. In the context of the experimentation in this section, we do not employ a weighted sum to calculate the loss term. Since it can be argued that the extra loss term L_{phy} "informs" the neural network of the physical properties, we will refer to this approach as Physics Informed Neural Network (PINN) throughout the experimentation section ahead.

The comparisons between the methods presented herewith are drawn in terms of regression metrics on the test dataset and driving performance inside the simulation on unseen tracks. We also analyse the unknown intermediate variables: lookahead distance and heading difference to reference points chosen by our trained PERNN-based agent in different driving scenarios.

5.1.1 Data generation

To generate data in TORCS as expert demonstrations for our experimentation, we use the implementation of an advanced heuristic-based agent provided by the simulation based on *Ahura: A Heuristic-Based Racer for the Open Racing Car Simulator* (Bonyadi et al., 2017). The details of this implementation are irrelevant to the approach explained in this paper since, theoretically, this expert system can be any high-fidelity and accurate driving model. We generate state and actuator values at each time instance on each of the 36 race tracks in the simulation, corresponding to two driving scenarios: empty road and traffic bourne. We refer to these two collected datasets as $\mathcal{D}_1$ and $\mathcal{D}_2$.

The race tracks are divided into three categories: road, dirt and oval speedway tracks based on variations in track geometry. Most of the oval speedway tracks offer relatively simple geometry compared to the more complex geometry on the road tracks in terms of sharpness and variation in the directions of turns. The dirt tracks offer significantly tougher terrain owing to sharper turns and high variation in the altitude values leading to dips and troughs, represented as the z value in Table 1. Figure 10 shows a sample of maps representing geometrical variations in each of the three track categories.

We reserve the data from a set of tracks as our test cases from both $\mathcal{D}_1$ and $\mathcal{D}_2$, where we analyze the behaviour of the trained models in the simulation. These include all dirt tracks and several road tracks. The goal is to test how well each approach performs in navigating new geometrical scenarios and their consequent capability for error recovery.

5.1.2 Experimental setup

We test four different agents in our experimentation. The first agent is based on the model trained using our Physics Encoded Residual Neural Network (PERNN) explained earlier in the paper. The second agent is based on the same framework as our method but without the residual block. This will help us illustrate the effectiveness of the residual-based architecture. We will refer to this model as Physics Encoded Neural Network (PENN). The third agent, as a benchmark, is based on a model trained using a conventional feed-forward neural network (FCNN) with a set of fully connected layers and ReLU activation functions. Finally, the fourth type of agent is trained using a physics-informed neural network (PINN) having the same base architecture as the FCNN. The output from all networks is a single steering value based on the state information provided to the model.

For the PINN approach, the Pure Pursuit model serves as the physics model regularizing the loss function (discussed at the beginning of Sect. 5.1). The Pure Pursuit model takes as input appropriate values of lookahead distance (l) and target heading difference (θ_{target}), and since there is no concept of learning blocks predicting intermediate variables in the PINN approach, we use our heuristic function from Sect. 4.1.3 to generate these values for each data point in the training data. This means that the regularizing term L_{phy} is calculated for empty road scenarios where the logic behind the heuristic function holds. For the traffic-based data points, L_{phy} is set to zero to not influence the loss value for traffic-based scenarios.

For experimental coherence, we use a simple PID controller to control the speed and brake of the vehicle and a basic functional logic to control the gears based on the vehicle's speed. The controller's implementation details are irrelevant to the experiment results since they are kept constant for both driving agents.

We compare three different FCNN models with our PERNN model regarding mean absolute error (MAE) on the test dataset, average distance travelled, and average jerk in predicted steering on the test tracks during live simulation. The three FCNN models are trained on the collective data from $\mathcal{D}_1$ and $\mathcal{D}_2$, comprising a single phase of training, as opposed to the two-phased setup for the PERNN model (see Sect. 4.1.3). The three FCNN models differ in architectural complexity and training data size (number of tracks comprised in training data) as shown in Table 3. Since the size of the training data and the number of parameters in the model are related in terms of the bias-variance trade-off, each of the three FCNN models consumes a differing, dedicated size of training data for fair comparison.

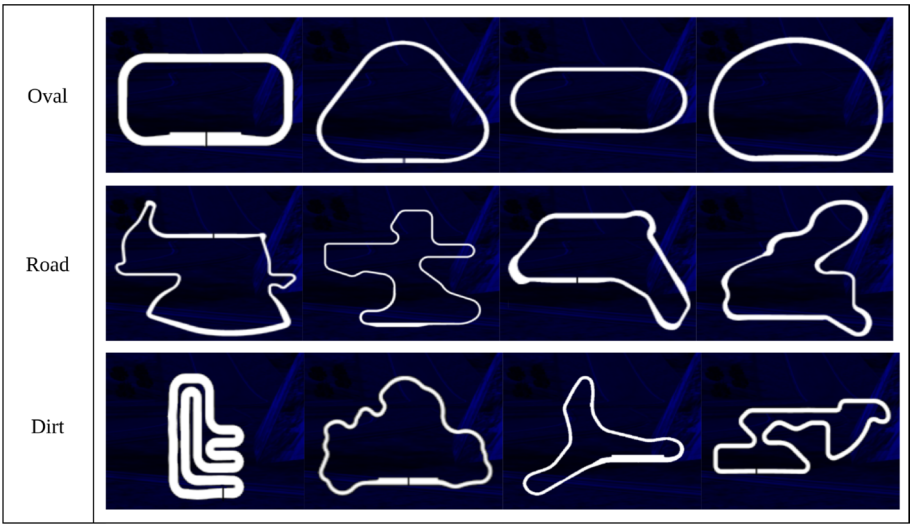


Fig. 10 Sample track maps representing geometrical variations in each of the three track categories

We also compare the performance of the agents using two key driving metrics. The first metric is a macro-level statistic of the average distance travelled by the agents on the test tracks before they hit the road boundaries. The second, more micro-level metric is based on the *smoothness* factor of the driving trajectory. This is shown through the jerk and entropy values associated with the predicted steering angle during the test runs. Jerk is a third-order derivative of the predicted steering angle. If we refer to the predicted steering angle as δ, then the steering velocity is calculated as δ', steering acceleration as δ'' and finally the jerk value as δ'''.

To keep experimental consistency, the PENN architecture for the learning and physics blocks is kept the same as that in the PERNN model. In addition, the PINN-based model architecture is kept the same as the smallest FCNN model for a direct and fair comparison of the effectiveness of the method in comparison to our approach.

In summary, we investigate the effectiveness of the PERNN model against the FCNN and PINN counterparts based on the two areas:

1. Data requirements include the number of unique tracks as part of the training data.
2. The complexity of the architecture in terms of the number of parameters in the neural network to capture driving behaviour from the expert demonstrations.

5.1.3 Results

Table 3 presents the results of this analysis. It can be seen that the PERNN consumes much less training data (only 6 out of 36 tracks used in training data) compared to the FCNN models in general and is composed of significantly fewer model parameters. Despite this, the PERNN model converges better than the small and medium FCNN models on the test dataset on MAE (Mean Absolute Error). It is important to note that the PENN model comprising no residual blocks struggles to converge on the test dataset owing to restricted

Table 3 Numerical results of the experiments comparing the performance of PERNN against FCNN-based agents

Model	Tracks in train set	Neural configuration	Parameters	Test MAE (*rad*)	Avg. distance(*m*)	Avg. jerk ($rads^{-3}$)
FCNN-small	6	[64, 32, 16, 8, 4, 1]	6977	0.0287	1157	0.0014
FCNN-medium	14	[256, 128, 64, 32, 16, 8, 4, 1]	60,801	0.024	2762	0.008
FCNN-large	25	[1024, 512, 256, 128, 64, 32, 16, 8, 4, 1]	767,617	**0.01**	**3803**	0.0347
PINN	6	[64, 32, 16, 8, 4, 1]	6977	0.077	1958	0.0001
PERNN	**6**	Learning Block: [32, 16, 8, 2] + Residual Block: [8, 4, 1] + PhysBlock	5611	0.0214	**3209**	**0.00003**
PENN	6	Learning Block: [32, 16, 8, 2] + PhysBlock	2794	0.0364	275	0.0016

gradient flow. The agent based on this model also fails to generalize the learned driving behaviour in the live simulation with a significantly low average distance travelled on the test tracks.

Figure 11 elaborates on the results. Figure 11a compares the MAE scores to the number of parameters for each model in the experimentation. The PERNN model shows comparable MAE scores on the test dataset to the FCNN-large model but with approximately 260 times fewer model parameters. Similarly, Fig. 11b compares MAE scores to training data requirements for each model in the experimentation. Our PERNN model shows comparable MAE scores on the test dataset to the FCNN-large model while using five times less training data in the form of unique tracks in the generated observation data. Both of these observations are a testament to the effectiveness of the physics blocks in the PERNN architecture in providing prior steering knowledge to the model as an inductive bias. Similarly, Fig. 12 compares the average distances travelled without collision during the live simulation on test tracks by each agent. PERNN-based agent achieves higher distances compared to FCNN-small, FCNN-medium models and the PINN model while using less training data and comprising fewer parameters.

Figure 13 shows the comparison of the distance successfully manoeuvred by the agent (before the collision with the road boundary) against the jerk produced by the steering (lower the jerk, smoother the driving trajectory). The analysis is carried out for the driving agent based on each model in the experimentation. The PERNN model outperforms FCNN-small and FCNN-medium models regarding both higher average distance travelled and significantly less jerk in predicted steering. The large version of FCNN exhibits better MAE values on the test set compared to the PERNN model and eventually surpasses it in navigating longer patches of tracks during the live simulation. It however suffers from higher values of steering jerk due to the model attempting to recover from *cascading errors*, a known issue in these methods under Behavioral Cloning settings. Since the PERNN model inherently encodes the basic error recovery function of following the reference point from any arbitrary state, it offers smaller jerk values and a smoother driving trajectory without explicit variations in training data.

It is important to understand the inherent limitation of the PINN approach which is overcome by our PERNN method, allowing better accuracy. Figure 14 illustrates the difference in the distribution of the ground-truth steering values and the steering values generated from the physics model (Pure Pursuit) for empty road scenarios. This forms decoupled targets for the model to learn, which is depicted in the graph to the right of the distribution showing the validation dataset loss values failing to converge anywhere close to the other methods. Despite this, it outperforms FCNN-small (with equal size of training data and number of model parameters) on live simulation on the test tracks in terms of both average distance travelled and average jerk values. This illustrates the benefit in terms of incorporating prior physics knowledge of driving in the model. PERNN takes a step ahead and provides the capability to learn from both the data and the physics using the concept knowledge blocks, facilitated by residual blocks and therefore does not suffer from the decoupled behaviour in physics and data.

5.1.4 Analysis of unobserved intermediate variables

Another key benefit of the PERNN model is that it predicts the intermediates variables, the lookahead distance and heading difference values which are not explicitly part of the observation data from the driving environment. We analyse the distribution of these variables

for a set of road scenarios to analyse the agent's decision-making based on the PERNN model predictions. This is specifically defined using the range finder sensors R and O (see Table 1), for filtering examples based on the relative distances from the track edge and traffic vehicles on either side of the vehicle in the driving space ahead. Index value 9 in both R and O suggests distances to objects in the direction of the vehicle's heading (straight ahead).

We divide the road scenarios in the test data into three simplified categories:

- Category A: Scenarios with relatively more space towards the right of the driving space ahead. The data points in the set follow the principle that for distances defined in vectors R and O:

$$R_{10} > R_8 + 20$$

$$O_{10} > O_8 + 20$$

  This implies that the track boundary or vehicle at 10 degrees to the left of the agent vehicle is at least 20 m further compared to the track boundary or vehicle at 10 degrees to the right of the vehicle's direction.
- Category B: Scenarios with relatively more space towards the left of the driving space ahead. The data points in the set follow the principle that for distances defined in vectors R and O:

$$R_8 > R_{10} + 20$$

$$O_8 > O_{10} + 20$$

  This implies that the track boundary or vehicle at 10 degrees to the right of the agent vehicle is at least 20 m further compared to the track boundary or vehicle at 10 degrees to the left of the vehicle's direction.
- Category C, where there is limited space on either side, and a traffic vehicle is situated at the center in front. This makes scenarios where overtaking is improbable. The data points in the set follow the principle that for distances defined in vectors R and O:

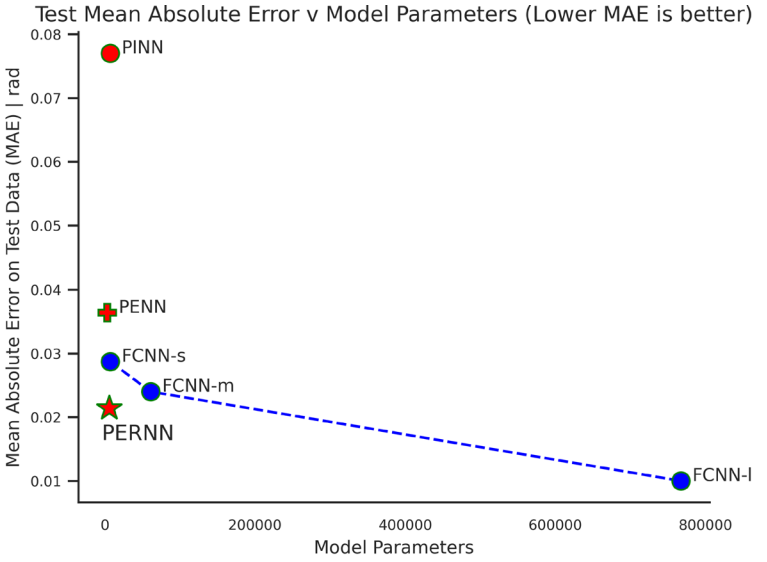
$$R_8 < 40 \quad \text{and} \quad R_{10} < 40$$

$$O_8 < 40 \quad \text{and} \quad O_{10} < 40$$

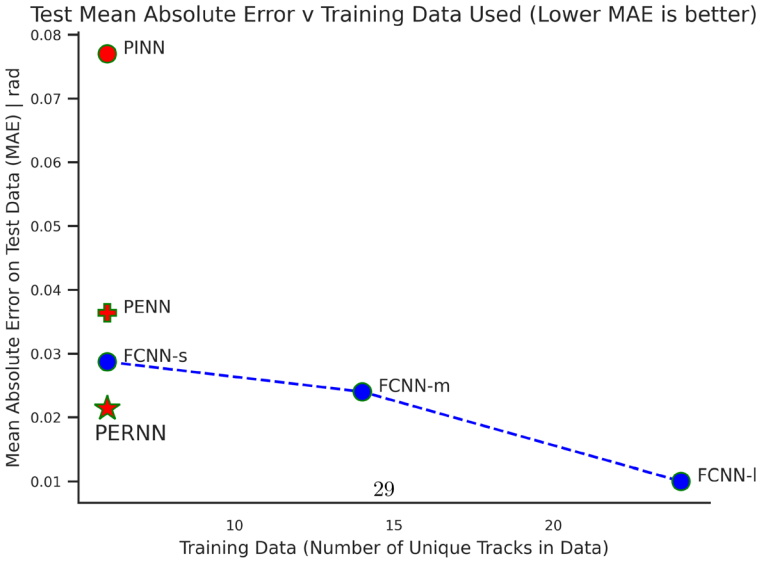
$$O_9 < 70$$

  This implies that the track boundary or vehicle at 10 degrees to the right and left of the agent vehicle's direction of motion is less than $40m$ and that there is a vehicle in front at a distance of less than $70m$.

The values for the distance bounds are empirically found to depict the behaviour most vividly and can be set as any arbitrary value based on the concept of the three category divisions. Figure 15 illustrates the results of this analysis. The distribution of the predicted heading angle for the Category A dataset clearly shows a right skew, suggesting that the model mostly selected points that were to the right side of the driving space in the front. Similarly, for Category B, the left skew is vividly depicted, which is understandable based on the specifications of the scenarios. The heading difference in Category C is slightly

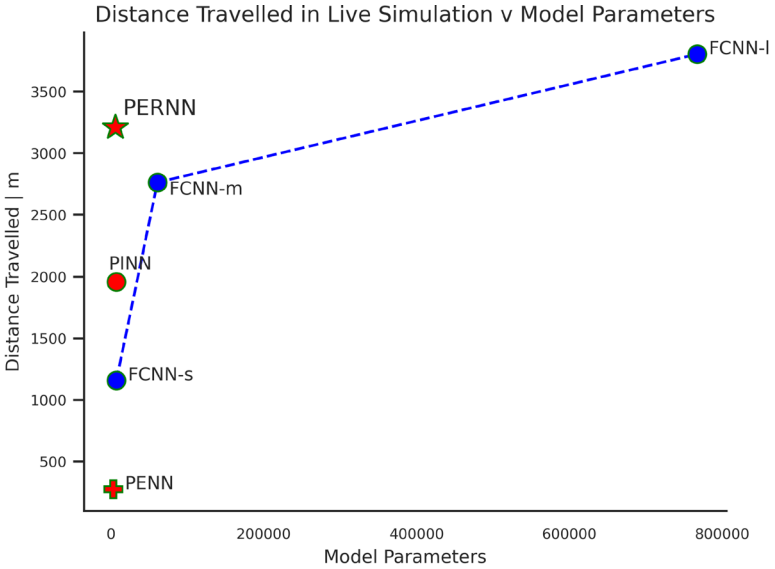


(a) Comparison of MAE score to model parameters for each model in the experimentation. Our PERNN model shows comparable MAE scores on the test dataset to the FCNN-large model but with approximately 260 times fewer model parameters. This is due to the physics blocks providing prior steering knowledge to the model as an inductive bias

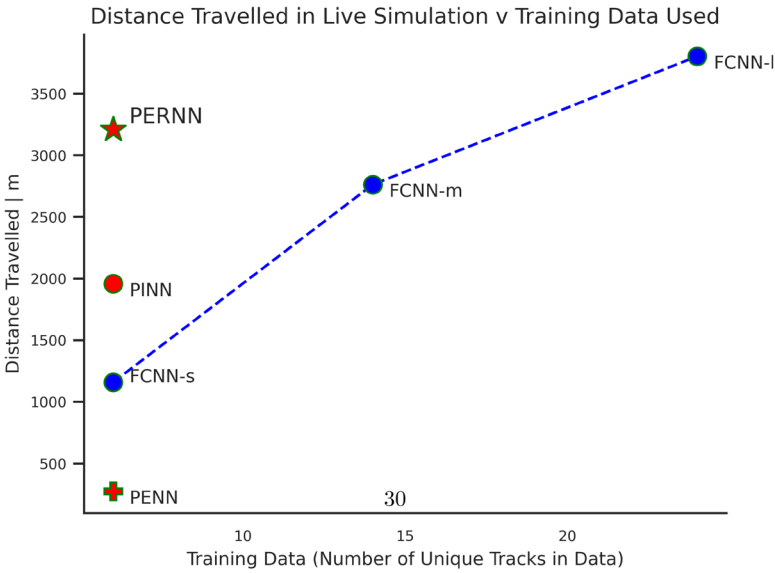


(b) Comparison of MAE score to training data requirements for each model in the experimentation. Our PERNN model shows comparable MAE scores on the test dataset to the FCNN-large model while using five times less training data in the form of unique tracks in the generated observation data. This is due to the physics blocks providing prior steering knowledge to the model as an inductive bias.

Fig. 11 Comparison of model accuracy (Mean Absolute Error), number model of parameters and training data requirements



(a) Comparison of distance successfully manoeuvred by the agent (before the collision with road boundary) on test tracks against model parameters for each model in the experimentation. The PERNN model outperforms all other approaches except FCNN-large with lesser model complexity.



(b) Comparison of distance successfully manoeuvred by the agent (before the collision with road boundary) on test tracks against unique tracks used in training data for each model in the experimentation. The PERNN model outperforms all other approaches except FCNN-large with lesser training data.

Fig. 12 Comparison of distance travelled by agent in simulation on test tracks, number model of parameters and training data requirements

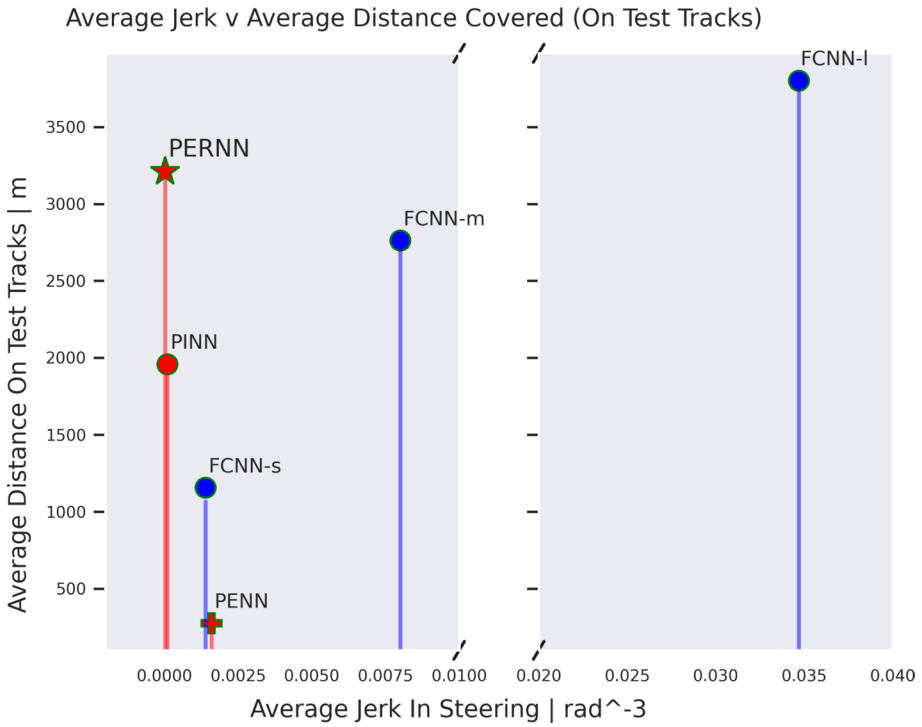


Fig. 13 Comparison of the distance successfully manoeuvred by the agent (before the collision with road boundary) on test tracks against the jerk produced by the steering (lower the jerk, smoother the driving trajectory). The experiment is carried out for each model. The PERNN model achieves a comparable successful distance travelled to the FCNN-large model but with significantly smaller jerk values, indicating a smoother driving trajectory

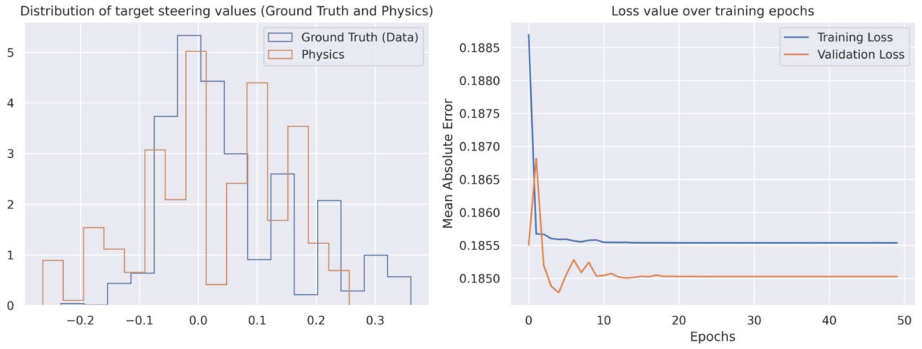


Fig. 14 Distribution of target steering values from the observation (ground-truth) data and the physics model in the PINN approach (shown to the left). The decoupled distributions lead to the loss function struggling to converge on the training and validation dataset (shown to the right)

more evenly distributed despite a right skew. It could be inferred that the reference agent preferred to stay more towards the right in this scenario, which behaviour the PERNN

model captured. It can be seen that the lookahead distance distribution is generally less uniform for Category A and B compared to Category C. This is understandable since Category C comprises more restricted scenarios based on available driving space for the agent.

The encouraging results of this analysis suggest the PERNN model offers significantly more explainability compared to conventional neural networks through the discovery of unknown, human-understandable variables in the environment.

5.2 Net ecosystem exchange modeling

In this section, we present details on the experimentation and results on the problem of NEE modeling for gap-filling in flux-tower observational data. We first present a short analysis on the difference in NEE behavior reflected in the simplistic physics model of NEE based on Eq. 4 and the flux-tower measurements. In addition, similar to Sect. 5.1, we present and compare results for PERNN with PENN (Physics Encoded Neural Network) as a variant of our method without the residual block and PINN (Physics Informed Neural Network) that uses the NEE ODE as a regularization on the loss term only, and finally with a conventioanl FCNN (Fully Connected Neural Network). We also present comparison against Random Forest which serves as the current state-of-the-art method for NEE gap-filling (Zhu et al., 2022) and XgBoost, another decision-tree based algorithm similar to Random Forest for extended comparison.

5.2.1 Analysis of NEE physics model

Since the NEE ODE (based Eq. 10) incorporated in our Physics Block is based on the equation modeled using an Arrhenius-type expression from Equation 4, we compare the NEE dynamics based on this equation with the ground-truth NEE values from the flux-tower observational data. Figure 16 presents four plots from randomly sampled daily cases of NEE observational data. The blue lines represent these observed values, while the red lines represent the values from the NEE physics model calculated using corresponding observed parameter, T_{air} and estimated parameters E_0 and $r_{b,night}$. It can be observed that the underlying NEE physics model (and consequently the NEE ODE) is an over-simplification of the actual NEE trends in the observational data, albeit following the general progression most of the time.

5.2.2 Experimental setup

We split the flux dataset into training and test sets based on the year of recording. The training set comprised years 2012-2017 (inclusive) while the test set was reserved for years 2018 and 2019 for experimental consistency when measuring gap-filling accuracy over different time scales: daily, weekly, monthly, and quarterly.

For all neural network based methods in this study, we keep the core network architecture and hyperparameters same for experimental consistency. The models based on PERNN, PENN, PINN and FCNN all have a set of layers comprising a skip connections and batch normalization and a LeakyReLU activation function. Figure 17 illustrates the difference between conventional fully connected linear layers with activation function, and our skip-connection based layers. These layers involve the concatenation of earlier linear layers with the output of the activation layer (LeakyReLU) followed by a

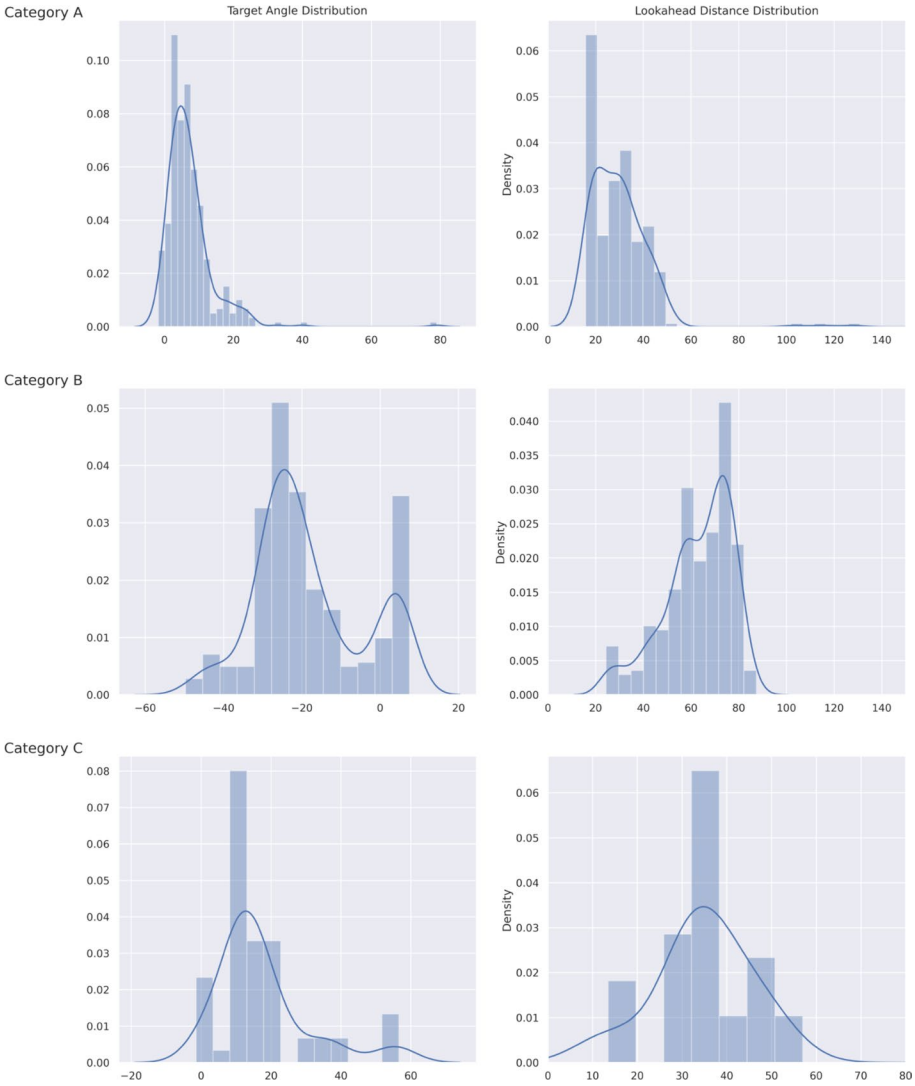


Fig. 15 An analysis of the model rationale in predicting the intermediate variables in the environment. The plot shows distributions of the intermediate variables, target angle (heading difference) and lookahead distance to the selected point on the road by the PERNN-based agent. Each category is represented as a row of two distribution plots

linear projection. We observed that these skip connection based layers allowed better convergence of the model, which can be seen by the results in Table 4 and explained in detail in Sect. 5.2.3. To show the impact of these skip-connection based layers, we also include experiments on the versions of PERNN and PENN with conventional layers, labelled PERNN-NoSkip and PENN-NoSkip respectively.

In addition, Fig. 18 shows the architectures of both PERNN and PINN models to illustrate the difference between the two approaches. In the PERNN model, the NEE

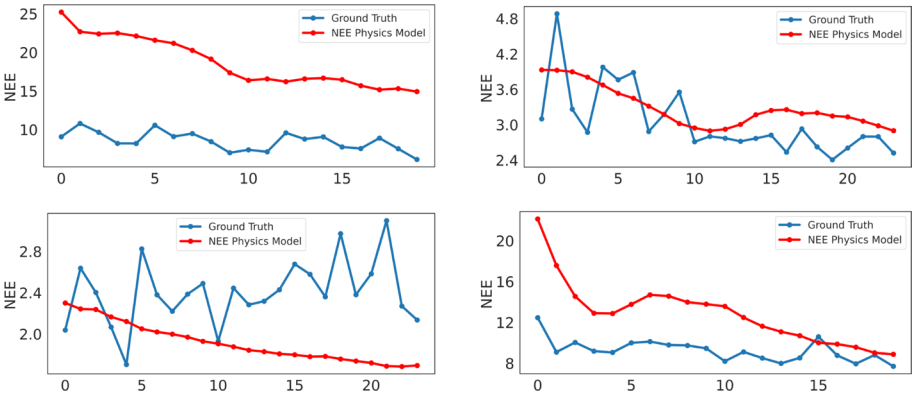


Fig. 16 The comparison of daily NEE values from flux-data and the output from NEE physics model based on Eq. 4 for four randomly sampled daily cases from the test-set. The blue line represents the ground-truth observations and the red line represents the values from the NEE physics model. The top row shows samples from dates 2018-08-18 and 2019-10-24, while the bottom row shows samples from dates 2019-03-04 and 2019-08-21 (left to right) (Color figure online)

ODE is a direct part of the computation graph inside the Physics Block, which outputs the final target variable (NEE_{t+1}). On the other hand, in the PINN model, the learnable layers directly predict the target NEE value (NEE_{t+1}) while the NEE ODE only offers external regularization to the loss function during training. In the PINN model, we present a branched architecture following similar structure to the learning blocks in PERNN. As seen in Fig. 18, two of these branches outputs the parameters for the NEE ODE, while a dedicated branch outputs the target NEE value. During testing in inference mode, only this dedicated branch is active and predicts the target NEE value. This experiment allows us to present the impact of our knowledge blocks in the neural architecture with the Physics Block explicitly housing the NEE ODE as a direct part of the computational graph.

5.2.3 Results

We evaluated the methods using three distribution-based metrics: Mean Maximum Discrepancy (MMD), Wasserstein Distance (Wsstn), and Kullback–Leibler Divergence (KL), to assess how well each technique captures the distribution of the target NEE. Additionally, we measured performance using mean absolute error (MAE) and R2 score to evaluate the fit to target variable. We also evaluate the accuracy of the intermediate variables from the learning blocks E_0, $r_{b,night}$ and dT_{air}, and the predicted time derivative of NEE, dNEE.

Table 4 presents the results on the test set over these metrics for the different methods compared in this study. It can be seen that PERNN surpasses the state-of-the-art Random Forest method by a notable 38.8% improvement in Wassertein Distance, 11% improvement in KL Divergence score while also achieving lower MAE score. PENN also shows comparable performance to the Random Forest albeit falling behind PERNN in all metrics, clearly indicating the advantage in convergence led by the incorporation of residual blocks in the physics encoded neural architecture. It is interesting to note that PINN performs considerably lower on all metrics compared to PERNN and PENN, which could potentially indicate the advantage of a direct integration of the NEE ODE in the computational

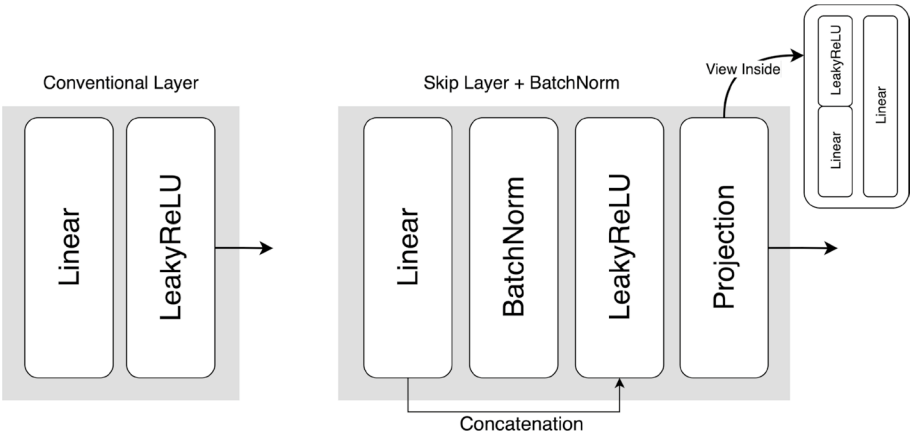


Fig. 17 A comparison of conventional and skip connection with batch normalization layers

graph of the neural network architecture on emprical grounds. Both PERNN and PENN also appear to outperform the FCNN and XGB methods on all metrics. It can also be seen that the PERNN-NoSkip and PENN-NoSkip perform considerably worse than their counterparts, which validates the benefit of the skip-connection based layers introduced in the learning and residual blocks of both PERNN and PENN.

Figure 19 shows the visualizations of ground-truth and predicted values for NEE_{t+1} used for gap-filling across four time scales: daily, weekly, monthly and quarterly. Randomly sampled sequences, consistent across methods for fair validation, show that PERNN consistently captures NEE trends better than PENN, PINN, RF and XgBoost methods. It can also be seen that PENN, albeit falling short of PERNN, appears to capture these trends better than the rest of the methods in the experimentation.

6 Conclusion and future work

This paper presents a novel physics-encoded neural network architecture that integrates physics models directly into neural networks through residual knowledge blocks. This approach provides a general framework for incorporating differentiable physics models into data-driven learning algorithms, thereby enhancing reliability and interpretability by enabling the discovery of unobserved environmental variables. Furthermore, the proposed method offers a rapid prototyping capability for digital twin applications in scenarios where both observational data and detailed physical models are limited.

We evaluated our framework in two distinct application domains. The first application involved developing a steering model for autonomous vehicles in the TORCS simulation environment. In this context, our method outperformed pure data-driven fully-connected neural networks and traditional physics-informed neural networks by delivering improved generalizability on unseen road scenarios with significantly reduced model complexity and data requirements. The integration of differentiable geometrical and kinematic operators with learned intermediate variables–facilitated by residual and skip-connection based layers–proved critical for robust performance.

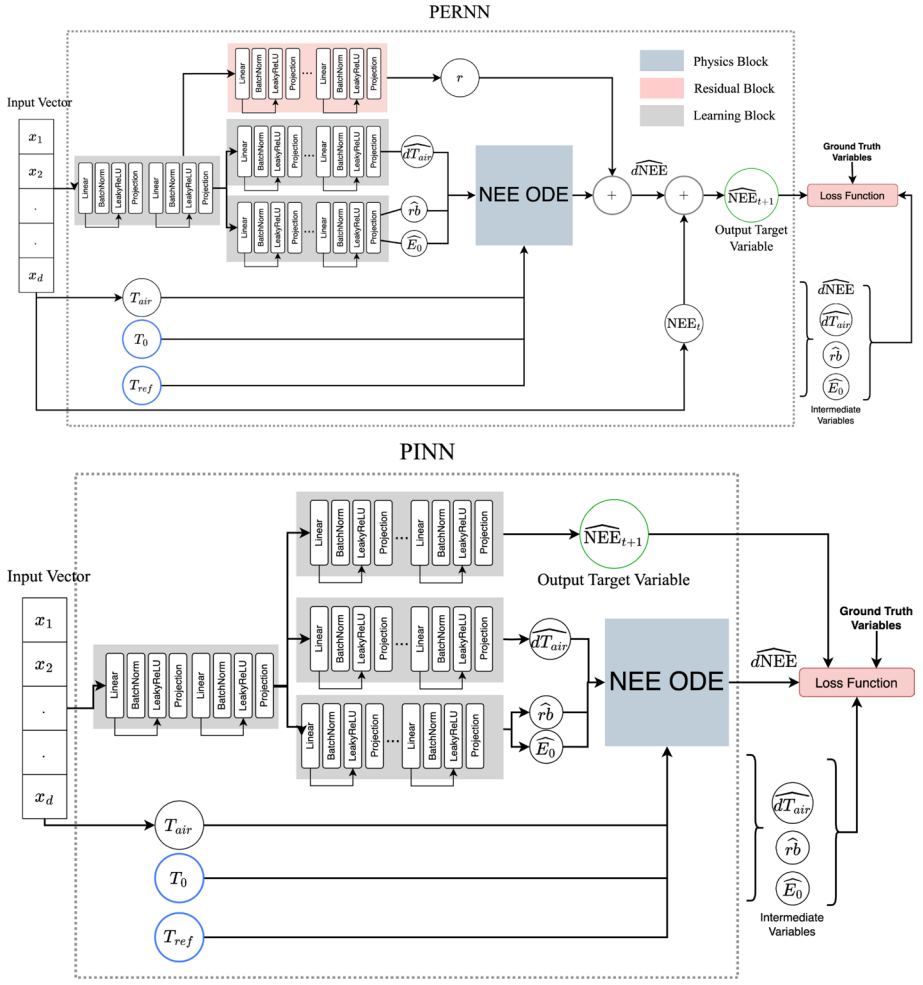


Fig. 18 The architectures for the PERNN and PINN models. Top: PERNN architecture with two learning blocks, a residual block and physics block comprising NEE ODE as a direct part of computation graph. Bottom: PINN architecture with two branches outputting NEE ODE parameters which is used to regularize the loss function during training. A separate branch outputs the actual target NEE value

The second application demonstrated the versatility of our framework in a real-world digital twin setting for climate modeling. Specifically, we modeled Net Ecosystem Exchange (NEE) using an ordinary differential equation (ODE) derived from an Arrhenius-type formulation of ecosystem respiration. By embedding this ODE directly into the network via a dedicated physics block and training the model to predict key intermediate variables (e.g., temperature sensitivity E_0, base respiration r_{b_night}, and the rate of change of ambient air temperature $\frac{dT}{dt}$), our method achieved superior gap-filling performance on flux tower data. Experimental results indicate that our approach outperforms state-of-the-art methods, such as Random Forest Robust and XGBoost, in both error metrics and the ability to capture the distributional characteristics of NEE.

Table 4 Results for NEE prediction on night time data and model experiments. The metrics MMD, Wasstn (Wasserstein Distance), KL (Kullback Leibler Divergence), and MAE (Mean Absolute Error) are expressed as the lower the better

NEE						E_0	r_b	dT_{air}	dNEE
Model	MMD	Wsstn	KL	MAE	R2	MAE			
PERNN	**0.051**	**0.145**	**0.215**	**0.866**	**0.73**	**28.8**	**0.939**	**0.046**	**0.141**
PERNN-NoSkip	0.138	0.396	0.260	1.087	0.67	29.9	1.29	0.036	0.11
PENN	0.086	0.190	0.212	0.986	0.69	30.7	1.66	0.08	0.063
PENN-NoSkip	0.122	0.276	1.149	1.17	0.48	31.06	1.51	0.014	0.053
PINN	0.247	1.043	0.540	1.41	0.55	27.33	1.826	0.029	0.031
FCNN	0.211	0.872	0.726	1.29	0.62	NA	NA	NA	NA
RF	0.055	0.237	0.242	0.901	0.721	NA	NA	NA	NA
XGB	0.052	0.202	0.214	0.988	0.658	NA	NA	NA	NA

Bold values indicate the main method and its significance in terms of its performance improvement over other method (on the metrics quoted)

R2 (score) is expressed as higher the better

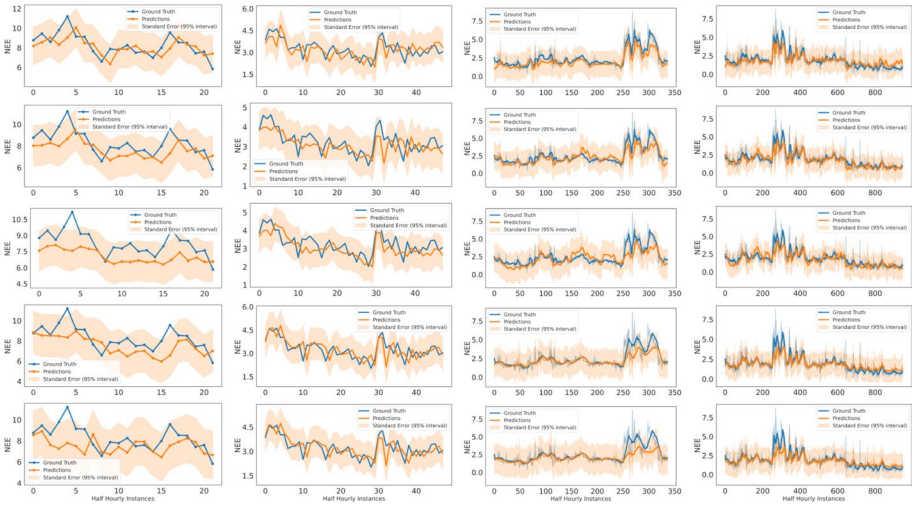


Fig. 19 NEE predictions on test data across different time scales (daily, weekly, monthly, quarterly) for each approach in the experimentation : we create artificial gaps of the corresponding time scales that we fill with the corresponding model. The first column represents results for a single night in test set (2019-09-22), the second column represents results for a single week (2018-11-06 till 2018-11-11), the third columns represents results for a single month (2019-01-01 till 2019-01-29) and fourth column represents results for a single quarter (2019-01-01 till 2019-12-30). Row 1 shows results based on the PERNN model, Row 2 shows results based on the PENN model, Row 3 shows results from the PINN model, Row 4 shows results from the RF model, and Row 5 shows results from the XGB model respectively. The sequences illustrated in the graphs are randomly sampled from the test dataset and are kept consistent for each approach for fair validation

In summary, our contributions demonstrate that directly incorporating physical models into neural network architectures not only enhances interpretability and prediction accuracy but also reduces reliance on extensive datasets. Future work will explore extending

this framework to a broader array of digital twin applications where data scarcity and incomplete physics are prevalent. Additionally, further studies on gradient flow and the development of more robust residual blocks will be pursued to tackle increasingly complex physics-based models.

Author Contributions Muhammad Saad Zia is the primary author who contributed to the main ideation and experimentation in this manuscript. Corentin Houpert provided mathematical formulation for the Net Ecosystem Exchange ODE. Ashiq Anjum is the primary supervisor and reviewer on the work in this manuscript. Lu Liu is an academic reviewer and collaborator on the work in this manuscript. Anthony Conway and Anasol Peña-Rios are industrial collaborators on the work in this manuscript.

Funding The research presented in this manuscript is supported by BT (British Telecom) PLC, and two Engineering and Physical Sciences Research Council (EPSRC) grants: AI-driven Digital Twins for Net Zero (EP/Y00597X/1) and Clinical Care (EP/Y018281/1)

Availability of data and materials The simulation and agents used to generate data and test agents in this research are available online. The references and citations are provided in this manuscript.

Declarations

Conflict of interest The IP for this work is owned by BT PLC, and as part of the agreement, the patent for this work has been filed before submitting the document to the journal.

Ethics approval The experiments were performed on data not related to living beings. No human beings and animals were involved in conducting the experiments. Thus, approval from an ethical committee is not required.

Consent to participate No living beings are involved in conducting the experiments for this research. Thus, consent to participate is not required.

Consent for publication The authors involved in conducting this research give their consent for the publication of the article titled "Physics Encoded Blocks in Residual Neural Network Architectures for Digital Twin Models".

Code availability The code has been made available on GitHub. https://github.com/saadzia10/Physics-Encoded-ResNet.git

References

Ajay, A., Wu, J., Fazeli, N., Bauza, M., Kaelbling, L. P., Tenenbaum, J. B., & Rodriguez, A. (2018). Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *IEEE/RSJ international conference on intelligent robots and systems (IROS)*.

Beidokhti, R. S., & Malek, A. (2009). Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. *Journal of Franklin Institute, 346*, 898–913.

Bonyadi, M. R., Michalewicz, Z., Nallaperuma, S., & Neumann, F. (2017). Ahura: A heuristic-based racer for the open racing car simulator. *IEEE Transactions on Computational Intelligence and AI in Games, 9*(1), 54–66. https://doi.org/10.1109/TCIAIG.2016.2565661

Cai, W., Li, X., & Liu, L. (2020). A phase shift deep neural network for high frequency approximation and wave problems. *SIAM Journal on Scientific Computing, 42*, 3285–3312.

Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D., & Ho, S. (2020). Lagrangian neural networks. In *International conference on learning representations (ICLR)*.

Cumming, A. M. J., Newman, T. R., Benson, S. J., Balzter, H., Evans, C., Jones, D., Kaduk, J., Morrison, R. D., & Page, S. E. (2020). Eddy covariance measurements of carbon dioxide, energy and water flux at an intensively cultivated lowland deep peat soil, East Anglia, UK, 2012 to 2020. *NERC Environmental Information Data Centre*. https://doi.org/10.5285/13896773-01e5-48e6-bfab-c319de46b221

Darbon, J., & Meng, T. (2021). On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton–Jacobi partial differential equations. *Journal of Computational Physics, 425*, Article 109907.

Degrave, J., Hermans, M., Dambre, J., & Wyffels, F. (2019). A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics, 13*, 42.

Espié, E., Guionneau, C., Wymann, B., Dimitrakakis, C., Coulom, R., & Sumner, A. (2005). TORCS, the open racing car simulator. Available online.

Geneva, N., & Zabaras, N. (2020). Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics, 403*, Article 109056.

Greydanus, S., Dzamba, M., & Yosinski, J. (2019). Hamiltonian neural networks. In *Advances in neural information processing systems (NeurIPS)*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90

Holl, P., Koltun, V., & Thuerey, N. (2020). Learning to control PDEs with differentiable physics. arXiv preprint arXiv:2001.07457

Jones, D., Smith, J., & Brown, A. (2020). Digital twin technology for energy and climate management. *Renewable and Sustainable Energy Reviews, 135*, Article 110275.

Kashefi, A., Rempe, D., & Guibas, L. J. (2021). A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *Physics of Fluids, 33*, Article 027104.

Keenan, T. F., Migliavacca, M., Papale, D., Baldocchi, D., Reichstein, M., Torn, M., & Wutzler, T. (2019). Widespread inhibition of daytime ecosystem respiration. *Nature Ecology & Evolution, 3*, 407–415. https://doi.org/10.1038/s41559-019-0809-2

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks, 9*, 987–1000.

Lasslop, G., Reichstein, M., Papale, D., Richardson, A., Arneth, A., Barr, A., Stoy, P., & Wohlfahrt, G. (2010). Separation of net ecosystem exchange into assimilation and respiration using a light response curve approach: Critical issues and global evaluation. *Global Change Biology, 16*(1), 187–208.

Li, Z., Kovachki, N. B., & Azizzadenesheli, K. (2021). Fourier neural operator for parametric partial differential equations. In *International conference on learning representations (ICLR)*.

Li, Y., Wu, J., Zhu, J.-Y., Tenenbaum, J.B., Torralba, A., & Tedrake, R. (2019). Propagation networks for model-based control under partial observation. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Lloyd, J., & Taylor, J. A. (1994). On the temperature dependence of soil respiration. *Functional Ecology, 8*(3), 315–323.

Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence, 3*, 218–229.

Miles, C., Sam, G., Stephan, H., Peter, B., David, S., & Shirley, H. (2020). Lagrangian neural networks. In *Workshop on integration of deep neural models and differential equations at ICLR*.

Moffat, A. M., Papale, D., Reichstein, M., et al. (2007). Comprehensive comparison of gap-filling techniques for eddy covariance net carbon fluxes. *Agricultural and Forest Meteorology, 147*, 209–232.

Mohan, A., Chai, H., Khojasteh, M.J., Dey, D., & Joshi, K. (2021). Cdpn: Coordinated differentiable physics networks for interacting rigid bodies. In *Advances in neural information processing systems (NeurIPS)*.

Mrowca, D., Zhuang, C., Wang, E., Haber, N., Fei-Fei, L. F., Tenenbaum, J., & Yamins, D. (2018). Flexible neural representation for physics prediction. In *Advances in neural information processing systems (NeurIPS)*.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics, 378*, 686–707.

Rasheed, A., San, O., & Kvamsdal, T. (2020). Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access, 8*, 21980–22012. https://doi.org/10.1109/ACCESS.2020.2970143

Reichstein, M., Falge, E., Baldocchi, D., et al. (2005). On the separation of net ecosystem exchange into assimilation and ecosystem respiration: Review and improved algorithm. *Global Change Biology, 11*, 1424–1439.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., & Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control. In *Proceedings of the international conference on machine learning (ICML)*.

Seo, J., Smith, B. D., Greydanus, S., Saha, S., Paduraru, C., Wan, C., Bansal, A., Jain, G., Peng, A., Narvekar, S., Pezeshki, G., Brevdo, E., Moore, C. X., Patel, S., Wang, H., Merel, J., Tunyasuvunakool, K., Heess, N., Hadsell, R., & Varma, S. (2022). Learning to optimize complex real-world systems through automated differentiation & differentiable programming. In *Advances in neural information processing systems (NeurIPS)*.

Stewart, R., & Ermon, S. (2019). How to train your differentiable physical simulator. In *NeurIPS workshop on machine learning for physics and the physical sciences (ML4PS)*.

Tao, F., Qi, Q., Liu, A., & Kusiak, A. (2019). Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *IEEE Access, 7*, 127356–127367.

Thelen, A., Zhang, X., Fink, O., et al. (2022). A comprehensive review of digital twin – part 1: Modeling and twinning enabling technologies. *Structural and Multidisciplinary Optimization, 65*, 354. https://doi.org/10.1007/s00158-022-03425-4

Toussaint, M., Allen, K.R., Smith, K.A., & Tenenbaum, J. B. (2018). Differentiable physics and stable modes for tool-use and manipulation planning. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Wang, J., Li, Y., Gao, R., & Zhang, F. (2022). Hybrid physics-based and data-driven models for smart manufacturing: Modelling, simulation, and explainability. *Journal of Manufacturing Systems, 63*, 381–391. https://doi.org/10.1016/j.jmsy.2022.04.004

Wang, B., Zhang, W., & Cai, W. (2020). Multi-scale deep neural network (MscaleDNN) methods for oscillatory stokes flows in complex domains. *Communications in Computational Physics, 28*, 2139–2157.

White, L., & Luo, Y. (2008). Modeling and inversion of net ecological exchange data using an Ito stochastic differential equation approach. *Applied Mathematics and Computation, 196*(2), 686–704. https://doi.org/10.1016/j.amc.2007.07.004

Willcox, K. E., Ghattas, O., & Heimbach, P. (2021). The imperative of physics-based modeling and inverse theory in computational science. *Nature Computational Science, 1*, 166–168. https://doi.org/10.1038/s43588-021-00040-z

Wu, J. L., et al. (2020). Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *Journal of Computational Physics, 406*, Article 109209.

Wutzler, T., Lucas-Moffat, A., Migliavacca, M., Knauer, J., Sickel, K., Šigut, L., Menzer, O., & Reichstein, M. (2018). Basic and extensible post-processing of eddy covariance flux data with REddyProc. *Biogeosciences, 15*(16), 5015–5030. https://doi.org/10.5194/bg-15-5015-2018

Zhu, S., Clement, R., McCalmont, J., Davies, C., & Hill, T. (2022). Stable gap-filling for longer eddy covariance data gaps: A globally validated machine-learning approach for carbon dioxide, water, and energy fluxes. *Agricultural and Forest Meteorology, 314*, Article 108777.

Zhu, Y., Zabaras, N., Koutsourelakis, P. S., & Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics, 394*, 56–81.

Authors and Affiliations

Muhammad Saad Zia[1] · Corentin Houpert[1] · Ashiq Anjum[1] · Lu Liu[1] · Anthony Conway[1] · Anasol Peña-Rios[2]

✉ Muhammad Saad Zia
msz6@leicester.ac.uk

Corentin Houpert
ch586@leicester.ac.uk

Ashiq Anjum
a.anjum@leicester.ac.uk

Lu Liu
l.liu@leicester.ac.uk

Anthony Conway
asc45@leicester.ac.uk

Anasol Peña-Rios
anasol.penarios@bt.com

[1] School of Computing and Mathematical Sciences, University of Leicester, Leicester LE1 7RH, United Kingdom

[2] BT Research Labs, Ipswich IP5 3RE, United Kingdom