QUERY FORM

JOURNAL: Machine Learning: Science and Technology

AUTHOR: K Agyei-Baah *et al*

TITLE: An interpretable convolutional neural network framework for fluid dynamics

ARTICLE ID: ae8072

Your article has been processed in line with the journal style. Your changes will be reviewed by the Production Editor, and any amendments that do not comply with journal style or grammatical correctness will not be applied and will not appear in the published article.

The layout of this article has not yet been finalized. Therefore this proof may contain columns that are not fully balanced/-matched or overlapping text in inline equations; these issues will be resolved once the final corrections have been incorporated.

Q1: Please check that the names, ORCID iDs, and contribution data for all authors are correct, and that all authors are linked to the correct affiliations. Please also confirm that the correct corresponding author has been indicated. This is your last opportunity to review this information before your article is published.

Q2: A standard data availability statement has been added to your article, based on the information you gave in your submission. This text cannot be changed unless there is an error. Please remove any other data statement if the information is now duplicated.

Q3: Please provide the significance of '(?)' here.

Q4: Please note that the Crossref and PubMed database have been used to validate the references and mismatches have been updated in the following references: Ambarzumian (1929), Green (1952), Kubo (1957), O'connell and Thompson (1995), Nie *et al* (2004), Petravic and Harrowell (2006), Raissi *et al* (2017, 2019), Rudy *et al* (2017), Raissi and Karniadakis (2018), Bar-Sinai *et al* (2019), Smith *et al* (2019), Zhang *et al* (2019), Agostini (2020), Cai *et al* (2021), Zobeiry and Humfeld (2021), Geneva and Zabaras (2022), Kim and Choi (2022), Yuan *et al* (2022), Lu (2023), Shan *et al* (2023), Sprittles *et al* (2023), Bonfanti *et al* (2024), Brunton *et al* (2024), Hassija *et al* (2024), Solera-Rico *et al* (2024), Gao *et al* (2026). Please check and confirm that these have been updated correctly.

Q5: Only minor, essential corrections to your article can be made at this stage. Minor changes are those that do not significantly affect either the scientific meaning of the article or its bibliographic identity. No changes can be made to authorship (other than those covered by our Name Change Policy) and references cannot be added or removed unless in response to a specific proof query. We reserve the right to deny correction requests from authors which we deem in breach of our ethical policy.

Q6: Please provide the page range or article number in reference Brunton *et al* (2024).

Q7: To improve the visibility of your data link we have added it as a new reference. Please provide the following details to complete the reference: Author(s), year, title of data, and repository name.

IOP Publishing Mach. Learn.: Sci. Technol. 0 (2026) xxxxxx https://doi.org/10.1088/2632-2153/ae8072

MACHINE LEARNING
Science and Technology

PAPER

OPEN ACCESS

RECEIVED
7 April 2026

REVISED
12 June 2026

ACCEPTED FOR PUBLICATION
22 June 2026

PUBLISHED
xx xx xxxx

An interpretable convolutional neural network framework for fluid dynamics

Kwame Agyei-Baah[1,*], **Muhammad Rizwanur Rahman**[2] **and Edward R Smith**[1] Q1

[1] Department of Mechanical and Aerospace Engineering, Brunel University of London, Uxbridge UB8 3PH, United Kingdom
[2] Department of Mechanical and Production Engineering, Islamic University of Technology, Dhaka, Bangladesh
[*] Author to whom any correspondence should be addressed.

E-mail: kwame.agyei-baah@brunel.ac.uk

Keywords: machine learning, numerical schemes, finite difference, convolutional neural networks, fluid dynamics

Abstract

Modelling fluid dynamics with machine learning (ML) has advanced rapidly, yet most data driven approaches remain opaque because they rely on complex architectures to capture nonlinear flow behaviour. This lack of interpretability limits their reliability and hinders understanding of when and why they succeed or fail. To address this, we present a transparent approach that provides insights into how data-driven fluids dynamics and ML work. This is achieved by training a convolutional neural network (CNN), on data from a simple laminar fluid flow, to behave as an operator that exactly matches the finite-difference numerics, providing a direct link between well-established theory and this new world of ML models. Importantly, the model demonstrates strong generalisation capability by reproducing the dynamics for a wide range of distinct and unseen flow conditions within the same flow category. The CNN learns the forward Euler three-point stencil weights, capturing physical principles such as consistency and symmetry despite having only three tuneable weights. This interpretable ML model goes beyond pure numerical training (numCNN), the approach is shown to work when trained on analytical (anCNN) and even molecular dynamics (mdCNN) data. In some cases, the physics is not captured, and thanks to the simple and interpretable form, these CNNs provide insight into the limits, pitfalls and best practice of data-driven fluid models. Because the approach is based on finite-difference operators, it naturally extends to many structured-grid computational fluid dynamics problems, including turbulent, multiphase and multiscale flows as well as systems beyond the continuum such as molecular dynamics. To support reproducibility and accelerate adoption, all simulation code, training pipelines, pretrained models, and processed datasets are available open source on GitHub under kwamea-b/CNN-numerical-schemes.

1. Introduction

At its core, fluid mechanics seeks to understand the motion of liquids and gases, most often through the solution of the Navier–Stokes equations. For decades, computational methods have been employed to solve these equations across a wide range of fluid mechanical problems. However, the application of traditional computational fluid dynamics (CFDs) techniques for tackling realistic flows, particularly those involving complex geometries, chemical agents e.g. surfactants and additives, and the resulting nonlinearities, turbulence, or multiphase interactions remains computationally expensive, and hence limited.

Machine learning (ML) offers a powerful complement to traditional fluid mechanical modelling, with algorithms capable of extracting structure from large datasets, automating feature detection, and revealing non-linear flow behaviour that may be inaccessible by classical techniques (Brunton *et al* 2024). As a result, ML has been rapidly adopted across many fields, including fluid mechanics, where it is being increasingly used to model complex flows that lack closed form governing equations. Existing operator learning approaches e.g. Fourier Neural Operators (FNOs) learn mappings between function spaces by combining neural network layers with Fourier-domain transformations (Li *et al* 2020); DeepONets

employ a branch network to encode input functions and a trunk network to encode output locations (Lu *et al* 2019); and PDE-Net constructs deep networks from stacked time-advancement blocks designed to approximate differential operators (Long *et al* 2018). These typically prioritise expressiveness through deep, multi-layer architecture whose internal representations do not correspond to identifiable numerical operators. This limits their interpretability and obscures how the learned mappings relate to the underlying physics. In contrast, the present work deliberately adopts a minimal convolutional neural network (CNN) architecture whose convolutional kernel is mathematically equivalent to a finite difference operator, enabling direct, coefficient level comparison with the governing PDE discretisation. This provides a transparent, physically grounded operator learning framework suited for analysing generalisability, stability, and data requirements in data-driven fluid dynamics.

Recently, a wide range of publications show the promise of ML for fluid dynamics. Raissi and Karniadakis (2018) proposed a framework that relies on Gaussian Process, a powerful tool that is used for probabilistic inference over functions that balances between model complexity and data fitting. This approach is applied to canonical problems including the Navier–Stokes equations, the Schrödinger equation and the time dependent linear fractional equations. Their work (Raissi *et al* 2017) facilitated the development of models capable of handling complex domains without requiring large quantities of data. A sparse regression framework for discovering the governing equations from spatiotemporal data (Rudy *et al* 2017) with associated code PySINDy. Thus enforcing sparsity on the coefficient space, it aims to produce compact and interpretable PDE models. However, because it relies on a global polynomial dictionary, PySINDy faces several practical challenges (Raissi and Karniadakis 2018). To address these issues, Raissi *et al* (2019) proposed modelling nonlinear PDEs using deep neural networks as black-box function approximators, avoiding explicit derivative computation and eliminating the need for large symbolic dictionaries. Building on this idea, Sasaki *et al* (2019) introduced a grey-box approach that injects prior physical knowledge into the neural network architecture. Agostini (2020) combined the concept of auto-encoders with other ML algorithms in studying the flow behind a cylinder providing a low-dimensional model (a probabilistic flow prediction). Li *et al* (2020) used neural operators via graph kernel networks to learn mappings between function spaces, enabling mesh-free generalisation across discretisations. Cai *et al* (2021) used Physics-Informed Neural Networks (PINNs) framework to tackle a heat transfer problem which cannot be handled by traditional computational methods. Vinuesa and Brunton (2022) described rapid advancements in CFD driven by ML, highlighting techniques such as proper orthogonal decomposition and autoencoders that help manage the complexity of fluid flow data. Deng *et al* (2023) employed a transformer-based encoder-decoder network for the prediction of transonic flow over supercritical airfoils, encoding the geometric input with diverse information points. Shan *et al* (2023) focused on turbulence modelling through data assimilation and ML for separated flows over airfoils. Several comprehensive reviews exist, for example Brunton *et al* (2024) discussed a variety of frameworks aimed at understanding, modelling, optimising, and controlling fluid flows. Very recently, Gao *et al* (2026) proposed a non intrusive reduced order model that combines POD with a hybrid CNN–LSTM–ECA network to achieve efficient long term prediction of unsteady flows. By integrating attention mechanisms to control frequency coupling errors, the method accurately reconstructs flow past single and tandem cylinders while significantly reducing computational cost.

Initial exploratory investigations in the present study employed artificial neural networks (ANNs) to solve the one-dimensional (1D) heat equation for a range of viscosities as a baseline to understand how neural networks capture flow dynamics. Feature engineering was applied to embed physical information into the model, an approach used successfully in Zobeiry and Humfeld (2021) together with PINNs enforcing both boundary conditions and the physics. However, the model's marked failure to predict beyond trained viscosity ranges reaffirmed the limitations of purely data-driven approaches. To address this, PINNs were explored by incorporating residuals of the 1D heat equation, along with boundary and initial conditions, directly into the loss function. While this improved physical consistency, the model still exhibited instability and failed to generalise reliably beyond the training range, an observation consistent with previous work (Fesser *et al* 2023) and Bonfanti *et al* (2024) who obtained accurate predictions only within the immediate proximity of the training domain, with the model performing poorly outside the domain. These underscore a broader gap in the field where most ML models for CFD remain tailored to specific flow conditions or geometrical settings, limiting their general applicability. Worse still, the black box nature means we do not know how general our model is.

Perhaps a more promising direction than learning fluid flow fields, is to learn the physical operators that can evolve these fields. This is the approach suggested by deep operator networks (Lu 2023) and more recently, using autoencoders to map to a latent space which is time evolved using transformers (Geneva and Zabaras 2022, Solera-Rico *et al* 2024) Operator learning has also seen significant advancement through FNOs. Recently, Kovachki *et al* (2024) reviewed the fundamental architectures of these

models providing the mathematical guarantee of the robustness of these approximations. Furthermore, Duruisseaux *et al* (2025) clarified how FNOs leverage spectral parametrisation to achieve resolution independence, i.e. allowing the model to learn operators efficiently regardless of the grid size. This work also addresses common technical nuances regarding Fourier modes and boundary condition handling, providing a clearer framework for practical implementation.

Hybrid approaches coupling ML with traditional numerical solvers, despite their potential to enhance accuracy and efficiency in solving complex flows, are only at their initial phase of development. This can be seen in Rackauckas *et al* (2020) where the authors proposed a tool for mixing the information of physical laws and scientific models with data-driven ML approaches for universal differential equations (UDEs). In addition, a major challenge lies in the interpretability of ML models, which are often treated as black boxes without sufficient clarity of the underlying mathematical processes. This issue is elaborated in a comprehensive review by Hassija *et al* (2024). To address these challenges, the present study investigates CNNs, an advanced version of ANNs primarily designed to extract features from grid-like matrix datasets (LeCun and Bengio 1998) and perform image recognition. Similar to ANNs, CNNs consist of neurons that learn and optimise over time. CNNs are specifically designed for recognizing patterns in images, thus, enabling the integration of image-related features directly into the architecture. This renders CNNs more effective for visual tasks while reducing the number of parameters required for the model (O'Shea and Nash 2015). Because CNNs require far fewer parameters than fully connected ANNs and their architecture is well suited to grid-structured data, CNNs are a natural choice for this work.

Similar ideas have already been covered in various works, Queiruga (2019) coupled CNNs with classical numerical schemes for solving spatio-temporal physics problems. Focusing on the 1D heat equation and the inviscid Burgers equation, Queiruga (2019) showed that a single-layer CNN trained on the heat-equation trajectories naturally converges to the traditional finite-difference stencil, whereas a GAN-style (generative adversarial networks) training fails to recover those weights. The idea of this work, however, was not expanded beyond an initial proof of concept. The use of CNNs for numerics represent a subset of more general approaches: Long *et al* (2018) introduced PDE-Net, a deep forward neural network that employs convolution kernels to predict complex system dynamics and to uncover the underlying PDE models, learning differential operators alongside the non-linear response functions. Rackauckas *et al* (2020) proposed UDEs as a unifying framework, demonstrating their use on the 1D Fisher–KPP equation with CNNs acting as learnable stencils. More recently, Kim and Choi (2022) showed that a CNN kernel based on the five-point finite difference stencil can learn numerical schemes with limited data while achieving low relative errors. The existing methods of preserving symmetries in fluid dynamics such as rotation invariance, translation invariance, or using equivariant CNNs where convolutional layers automatically encode the desired symmetry are discussed by Zhang *et al* (2025). Bar-Sinai *et al* (2019) employs deep and complex CNNs to implement data-driven discretisation, replacing standard numerical schemes with learned operators. By predicting adaptive coefficients that account for subgrid-scale physics, this approach achieves accurate simulations despite using grids $4\times$ to $8\times$ coarser than equivalent finite difference methods. The translation-invariant architecture ensures these learned models generalise effectively to systems much larger than their training domains.

Building on this foundation, the study introduces two key contributions that go beyond existing work. First, we demonstrate that a minimal CNN can learn interpretable operators that are exactly equivalent to classical finite difference stencils, recovering properties such as consistency and symmetry with only three trainable weights directly linking to years of finite difference theory development (Hirsch 2007); explicitly linking CNN architectures to the mathematical progression from $u^t \rightarrow u^{t+1}$. Second, we extend the approach beyond numerical data by training the same CNN architecture on analytical solutions and molecular-dynamics (MD) derived data, enabling the extraction of physically interpretable operators from fundamentally different sources. This multi-source capability allows the model not only to reproduce known numerical schemes but also to reveal when physical structure is or is not present in the data. Together, these contributions provide a technique which has elements of physics discovery similar to techniques like SINDy (Brunton *et al* 2024), with interpretability similar to grey-box models (Sasaki *et al* 2019), allowing solution to the inverse problem (Ambarzumian 1929) all while giving a ML trained model completely grounded in finite difference theory.

The remainder of this manuscript is organised as follows. Section 2 presents the methodology, including the governing equations, the CNN architecture, and the training procedure. Section 3 reports the results and offers a detailed discussion of models trained on numerical and analytical data, ending with an application of the CNN kernel learner to MDs data that provides additional insight. Finally, section 4 summarises the key findings.

2. Methodology

To evaluate the performance and interpretability of the proposed CNN kernel learner, three benchmark fluid mechanics problems were selected: wall-driven (Couette) flow, Stokes' second problem, and the non-linear Forced Burgers' equation. These cases were chosen not only to validate the model's accuracy, but also to demonstrate its ability to demystify the black box behaviour associated with deep learning. By applying the model to both linear (Couette and Stokes' second problem) and non-linear (Burgers equation) fluid dynamics, we illustrate how the learned kernels align with established physical operators, providing a clear pathway toward interpretable ML in diverse fluid-mechanical systems. In the methodology and results sections, the analysis primarily focuses on the 1D wall-driven flow and Stokes' second problem to establish a baseline for the model's interpretability.

This study focuses on 1D diffusion type problems, as they provide the minimal setting in which the objective of the work, interpretability can be demonstrated without confounding factors. Diffusion equations offer a linear, well understood finite difference structure with known analytical solutions, making them an ideal testbed for isolating the behaviour of the learned operators. Restricting the domain to 1D ensures that the learned convolutional kernels can be interpreted unambiguously, allowing us to verify consistency, symmetry, and numerical equivalence with classical finite difference schemes. This controlled setting is essential for the central contribution of the study i.e. showing that a minimal CNN can recover physically meaningful operators from numerical, analytical and MDs derived data. Extending the analysis to higher dimensional or strongly nonlinear systems would obscure this interpretability and make it difficult to attribute model behaviour to the underlying operator learning mechanism. Data driven fluid dynamics often starts with very complex models and justifies validity by how they reproduce fluid flow. This work aims to start with the simplest model and explore where it fails to do this. Even in this simple 1D case, a number of convergence issues arise and the discovered weights can fail to reproduce fundamental physical requirements such as symmetry or consistency without balanced datasets or applied physical constraints like PINNs. The known finite difference operators in 2D and 3D give us certainty that the applied approach would be extendable in principle, but suggest that training would need more data, further constraints and more care in the training and validation process. However, this is worth mentioning that the framework developed in this work while successfully tackle 1D problems may not be straightforwardly extended for multidimensional problems. This is due to the complexities arising from the operators' requirement of wider convolutional stencils, increased computational cost, and importantly, the handling of cross derivative terms (e.g. $\partial^2 u/\partial x \partial y$). These complicate both training and interpretability of the outcome of the model. Additional constraints may also be needed to preserve physical invariants such as symmetry, isotropy, or conservation properties that are more easily maintained in 1D. These limitations, however, do not diminish the broader applicability of the approach, rather highlight that multidimensional extensions require careful architectural design and additional physical constraints.

By establishing a rigorous and transparent foundation in 1D, the work provides a validated framework that can be systematically extended to 2D and 3D structured grid problems in future studies. The restriction to 1D is therefore not a limitation of the method itself but a methodological choice to ensure clarity, interpretability, and theoretical grounding. The non-linear Burgers' equation is examined in detail in appendix A, where additional analysis illustrate the model's capacity to handle convective contributions. The appendix further demonstrates that the CNN kernel learner with a CNN term allowing for both convection and diffusion, applied to the diffusive cases in this work, consistently recovers the correct underlying physical operators. Finally, to assess the robustness and generalisability of the architecture, the model is tested against high-fidelity MD data, evaluating its ability to maintain physical consistency from continuum theory to discrete particle simulations. Using MD is a particularly interesting test of the CNN model, as the overall fluid flow is an emergent property from the average behaviour of thousands of interacting molecules. The CNN in this case functions as an equation discovery framework, with the simple 3-point kernel learning a numerical operator which can reproduce the dynamics of hundreds of thousand of molecules. This has interesting implications, from course-graining of MD in multi-scale modelling to data-driven physics discovery.

2.1. The 1D heat equation

This study uses two classical fluid flow problems, (i) Couette flow, and (ii) Stokes' second problem, to train and test a machine learned algorithm. Both of these wall driven fluid flows can be described by the one dimensional partial differential equation, most commonly known as the heat equation, or the

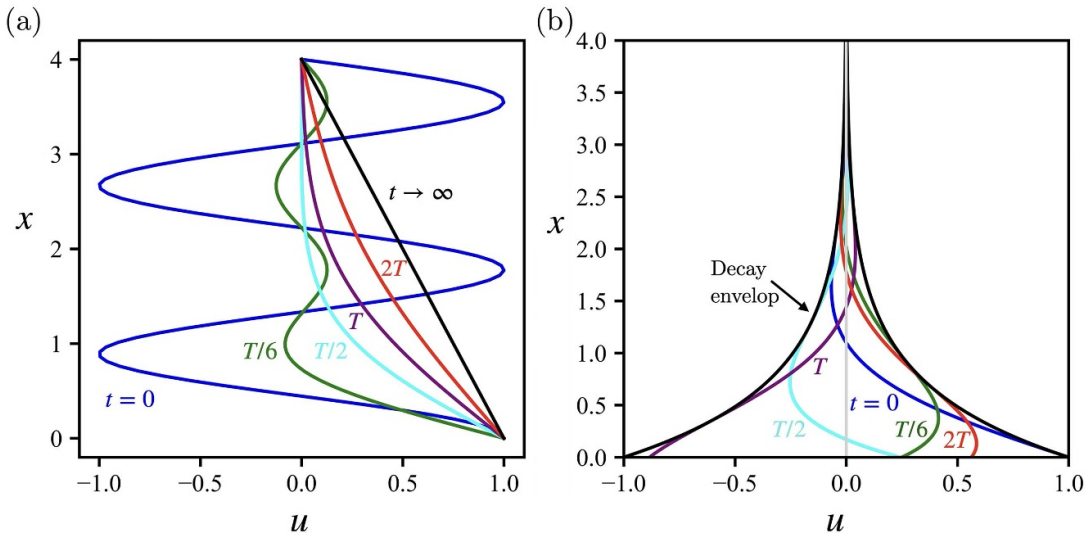

Figure 1. Illustrations of two cases studied: (a) Wall-driven flow (Couette flow) and (b) Stokes' second problem. Simulation parameters for cases (a) and (b): $\mu = 0.02$, $\Delta t = 0.1$, $L_x = 4.0$, $n_x = 80$, total time $T = 5000\,\Delta t$. Profiles at $t = \{0, T/6, T/2, T, 2T\}$ are shown for both cases. (a) Wall-driven flow: top wall moves in $+x$ with $U_0 = \sin((1.125x + 0.5)\pi)$; boundary values $u(0) = bBC = 1$, $u(L_x) = tBC = 0$. Curve labeled $t \to \infty$ is the steady Couette solution. For the Stokes' second problem in panel (b), $\omega = 1$ and $\kappa = 0.7$.

diffusion equation,

$$\frac{\partial u}{\partial t} = \mu \frac{\partial^2 u}{\partial x^2}. \tag{1}$$

The dependent variable u represents temperature for heat conduction problems, whereas, it denotes velocity field for wall-driven flows, μ is thermal diffusivity (heat conduction) or the dynamic viscosity (wall driven flow), and x is the spatial position. The present work considers wall driven flows, and as illustrated in figure 1(a), in the case of Couette flow, an initial condition u_0 with either the top tBC $= u_t$ or the bottom bBC $= u_b$ wall boundary slides with a constant velocity, whereas, for Stokes' second problem, the wall oscillates as a sinusoid with frequency, ω along the x direction.

2.2. Analytical solutions

As such, equation (1) gives either Couette flow or Stokes' second problem based on the boundary conditions. When the flow is driven by the sliding velocity of one of the walls (Couette flow), the analytical solution is given by

$$u(x,t) = u_{\text{steady}}(x) + \sum_{n=1}^{\infty} A_n \sin(\beta_n x)\, e^{-\mu\beta_n^2 t} \tag{2}$$

where $u_{steady}(x) = u_b + (u_t - u_b)x/L$, the coefficient $A_n = \frac{2}{L}\int_0^L \left[u_0(x) - u_{steady}(x)\right] \sin(\beta_n x)\, dx$ and wavelength is denoted as $\beta_n = (n\pi)/L$. Whereas, for the flow field resulting from the oscillation of the bottom wall $u(x = 0, t) = u_b(t) = U\sin(\omega t)$ (Stokes' second problem) and assuming an open top so $u(x \to \infty, t)$, the analytical solution takes the form,

$$u(x,t) = u_b \sin(\omega t - \kappa x)\, \mathrm{e}^{-\kappa x}, \tag{3}$$

where $\kappa = \sqrt{\omega/(2\mu)}$ and u_b is the wall oscillation magnitude. In practice, the top wall is actually a boundary at infinity but the domain is taken to be large enough to minimise the effect of this.

The analytical solutions, equations (2) and (3) provide controlled, closed form benchmarks that isolate the essential diffusion mechanisms relevant to operator learning. In the Couette case, momentum diffuses from the moving boundary toward the interior until a steady linear profile is reached, while in Stokes' second problem, the oscillatory forcing generates a decaying viscous wave whose penetration depth is governed by the parameter κ. These two solutions offer well characterised temporal and spatial structures that allow precise evaluation of whether the learned convolutional kernels reproduce the correct physical behaviour.

2.3. Numerical solution

The diffusion (or, heat) equation is discretised to mimic the setup of a convolution kernel. One of the simplest discretised form of (1) uses the forward Euler in time and a 3-point stencil in space,

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} = \mu \left[\frac{u_{i+1}^t - 2u_i^t + u_{i-1}^t}{(\Delta x)^2} \right], \tag{4}$$

Taking $C = \mu \Delta t/(\Delta x)^2$ this is rearranged to,

$$u_i^{t+1} = u_i^t + C\left(u_{i+1}^t - 2u_i^t + u_{i-1}^t\right). \tag{5}$$

The parameters used for the majority of simulations presented in this work are as follows, unless otherwise stated: dynamic viscosity, $\mu = 0.02$, timestep, $\Delta t = 0.10$ with a spatial grid, $n_x = 20$ for a domain length, $L_x = 4.0$ so a spatial resolution of, $\Delta x = L_x/(n_x - 1) = 0.2105$ over $N_t = 200$ timesteps which gives $C = 0.045125$.

In order to link these numerical models to CNN kernels and set out our notation, we provide some background theory on numerical methods (Hirsch 2007). Consider the general form of 3 point finite difference operator,

$$\frac{\partial^2 u}{\partial x^2} \approx b_1 u_{i+1}^t - b_0 u_i^t + b_{-1} u_{i-1}^t. \tag{6}$$

and the coefficients are required to satisfy,

$$\mathcal{A}_1 = \sum_{i \in \{-1,0,1\}} b_i \qquad = 0 \quad \text{(Consistency)} \tag{7a}$$

$$\mathcal{A}_2 = \sum_{i \in \{-1,0,1\}} i\, b_i \qquad = 0 \quad \text{(Symmetry)} \tag{7b}$$

$$\mathcal{A}_3 = \sum_{i \in \{-1,0,1\}} i^2\, b_i - 2C \qquad = 0 \quad \text{(Scaling)} \tag{7c}$$

Solving these three equations for 3 unknown weights yield values of $b_{-1} = C, b_0 = -2C, b_1 = C$ which is consistent with equation (5) above. These can be written as a vector $\boldsymbol{b} = b_i = [C, -2C, C]$ and written in C independent form by defining $\widetilde{\boldsymbol{b}} = \boldsymbol{b}/C = [1, -2, 1]$. These correspond to physical requirements for the kernel, with consistency equation (7a) ensuring that in the limit of zero volume, the kernel recovers the continuous differential operator. The symmetry condition of equation (7b) guarantees that no diffusion or numerical advection exists in a preferential direction, and the final condition equation (7c) is the scaling of the diffusion term in (5). None of these conditions enforce stability, the other major requirement of a numerical scheme.

Following the classical Von Neumann stability analysis on (4) gives G the amplification factor,

$$G = 1 - 4C\sin^2\left(\frac{k\Delta x}{2}\right) \tag{8}$$

where k is the wavenumber, which will include any modes that fit into the domain. To ensure that no Fourier modes are spuriously amplified, we require $|G| \leqslant 1$. The worst case mode is when $\sin^2(k\Delta x) = 1$ which leads to the stability condition $C \leqslant 1/2$. If C exceeds this limit, the highest-frequency modes become unstable, leading to oscillations and eventual blow-up of the numerical solution (Hirsch 2007). The condition equation (7c) therefore represents the edge of stability. In general the obtained weights during training will vary. The amplification factor can be expressed in terms of the 3-point stencil weights as,

$$G = \sum_{j \in \{-1,0,1\}} w_j e^{ij\theta} \tag{9}$$

where i denotes the imaginary number here and $\theta = k\Delta x$. The obtained weights from training can therefore be checked using equation (9) to ensure stability, with this condition enforced by a soft constraint or kernel construction if required. This section has introduced the forward Euler 3-point schemes (denoted as FE3). Higher order spatial and temporal numerical schemes were also studied and are outlined in appendix A. These include a spatially higher order schemes with 5 point stencil using the forward Euler (FE5) and the multiple time-step Adam-Bashforth for a 3-point stencil (AB3) and a 5 point stencil (AB5).

2.4. MDs

Alongside the analytical and finite difference schemes to solve the diffusion equation, we employ MD simulations to obtain an independent solution of the diffusion process described by equation (1) directly. Despite MD being a more fundamental model which depends only on the validity of Newton's law $\boldsymbol{F}_i = m\boldsymbol{a}_i$ for a system of N interacting molecules, this has been shown to reproduce the equations of fluid dynamics on average (Rapaport 2004). The chosen setup uses a simple Lennard Jones potential $U(r_{ij}) = \epsilon[(\sigma/r_{ij})^{12} - (\sigma/r_{ij})^6]$, with tethered thermostatted wall sliding to drive flow (Todd and Daivis 2017). This reproduces the continuum diffusion process as shown in figure 2. The velocity was obtained by an averaging process, binning the domain into 32 cells and averaging the velocity of the molecules. The viscosity is an output of the simulation, with the only empirical assumption in MD being the choice of intermolecular potential, as a result MD is an excellent way to test the validity of the CNN training approach as it provides an experimental-like setup where the fitted equations only approximate the true physics. The flow fields obtained from MD inherently include noise, to reduce this noise, we chose a relatively large system (with $N = 619,885$ molecules). The simulation was run with Flowmol, which has been extensively validated in previous work (Smith 2013). The setup is similar to Smith *et al* (2019), and readers are referred to this work for further details. Validation of the MD code, including Radial distribution functions compared to experimental data, phase diagrams, as well as diffusivity and viscosity compared to NIST data; interested readers are referred to Smith (2014). In reduced units, the walls were given a density of $\rho_{\text{wall}} = 1$, while the fluid was setup by randomly removing atoms to reach a density of $\rho_{\text{fluid}} = 0.8$ corresponding to the density of liquid argon at $\sim T = 0.8$ in coexistence with its vapor phase (NIST 2026). A Weeks–Chandler–Andersen cutoff was used for computational efficiency, with $r_c = 2^{1/6}$. Averaging cells are $\Delta y = 1.44$ with total domain $L_y = 46.03$ in the wall normal direction (*c.f.* the x axis used in continuum treatment which is 1D), and the size of the full domain in other directions, i.e. $\Delta x = L_x = 126.99$ and $\Delta z = L_z = 126.99$. The tethered walls have a width of $L_{\text{wall}} = 4$ at both the top and bottom boundaries, with the topmost and bottommost three cells of the domain treated as wall cells. This means the outer $3\Delta y = 4.3$ of the domain is a wall, which assumes a small region of stick-slip near the wall and gives a good comparison for the middle 26 cells compared to the analytical solution in figure 2. The outer portions of both the walls, each of width $L_{\text{therm}} = 2$ were thermostated, comprising approximately $N_{\text{therm}} \approx 65,000$ atoms in total. Temperature control was achieved using a Nosé Hoover thermostat with a heat bath coefficient of strength $Q = N_{\text{therm}}\Delta t_{\text{MD}}$ where $\Delta t_{\text{MD}} = 0.05$ is the MD timestep. Walls are tethered with an anharmonic potential from Petravic and Harrowell (2006) with $\eta_4 = 5,000$ and $\eta_6 = 5,000,000$ in $\phi_i(\boldsymbol{r}_i) = -\eta_4(\boldsymbol{r}_i - \boldsymbol{r}_i^{eq})^4 - \eta_6(\boldsymbol{r}_i - \boldsymbol{r}_i^{eq})^6$ where $\boldsymbol{r}_i^{eq}$ is the equilibrium position of the tethering site. The top wall was assigned a sliding velocity of $u_{\text{wall}} = 1$ applied to both the atoms and the tethering sites. The MD system was first equilibrated for 20,000 timesteps with wall thermostating. This equilibration phase preceded the velocity rescaling in the 26 fluid bins to match an initial sinusoidal solution, and the activation of the sliding motion of the top wall. Velocity rescaling was implemented by applying a small correction to all the molecules in the bin proportional to the difference between the instantaneous and the target velocities; consequently the system temperature remained unchanged. This rescaling, however, results in a minor jump between molecules in adjacent cells. The applied velocity correction per atom despite being negligible, their cumulative effect might explain the slightly larger effective viscosity seen in the $T/6$ case of figure 2. After sufficient equilibration of the system, the production run was commenced over 57,000 steps with $\Delta t_{\text{MD}} = 0.005$. During this phase, statistics were collected every timestep and average over a block of 10 steps giving 5700 MD snapshots in time, i.e. output times are each separated by $\Delta t = 0.05$. This spatial-temporal dataset of 32 spatial bins by 5700 times is then written to file to be used for CNN training. All data generated through the MD simulations, and the source codes to reproduce them are included along with other required scripts for the numerical modelling, as well as for the CNN model.

2.5. The CNN model

The operation of getting the next timestep from the previous one can be expressed using a convolution kernel of size 3, e.g. $\boldsymbol{k} = [k_{-1}, k_0, k_{+1}]$. Denoting the diffusion kernel as $\boldsymbol{k}_D = [C, -2C, C]$, and the identity kernel as $\boldsymbol{k}_I = [0, 1, 0]$, equation (5) can be rewritten in convolution form as:

$$u_i^{t+1} = \mathcal{F}_k(u_i^t) = (k_D * u^t)_i + (k_I * u^t)_i = (k * u^t)_i, \tag{10}$$

where '$*$' denotes the discrete convolution operator, as such $(k * u^t)_i = k_{-1}u_{i-1}^t + k_0 u_i^t + k_{+1}u_{i+1}^t$. To be consistent with the numerical operator in equation (5), the learned kernel, $\boldsymbol{k}$ is expected to converge to $[C, 1-2C, C]$. This corresponds to the diffusion kernel $\boldsymbol{k}_D$ learning the weights from the numerical schemes described in section 2.3, such that $\boldsymbol{k}_D \to \boldsymbol{b}$ which, when added to the identity kernel $\boldsymbol{k}_I$, yields

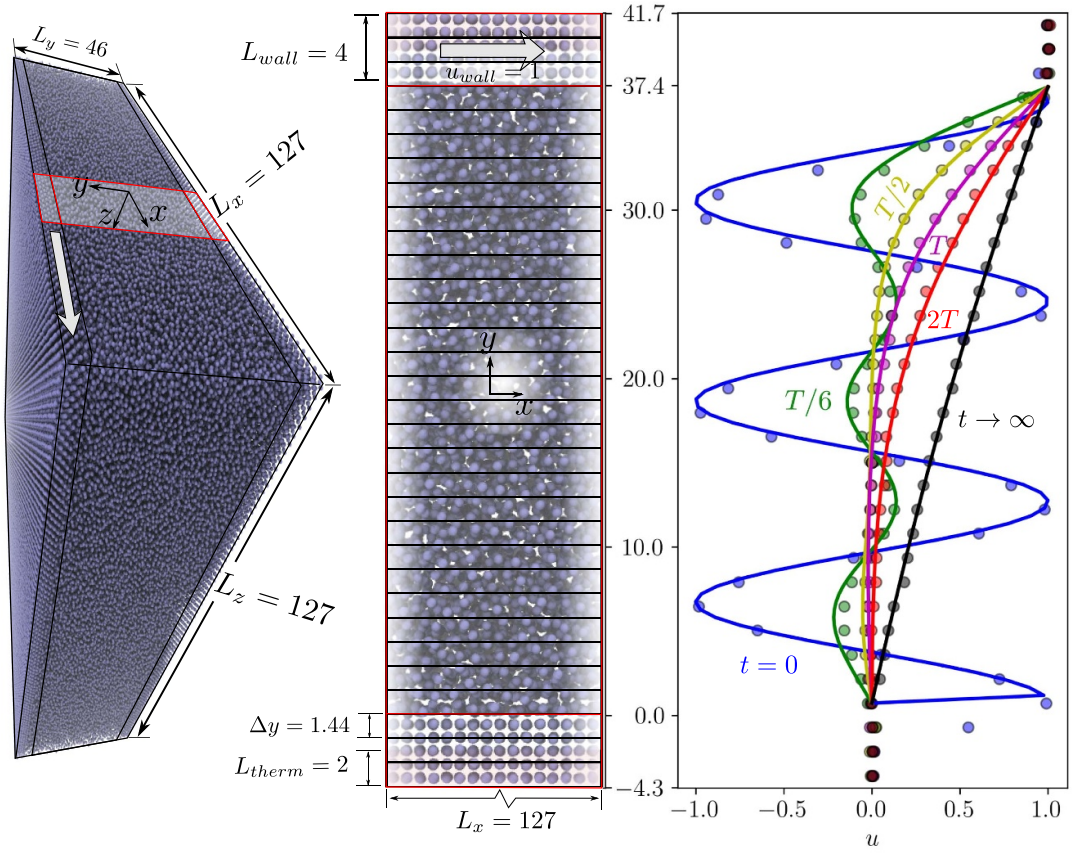

Figure 2. A perspective view of the full MD system is illustrated in the left panel, with the red box shown in the middle highlighting the region used in the schematic, where the corresponding MD cells line up with the velocities (symbols) on the plot on the right at a range of times as multiples of $T = 400\Delta t$ compared to the analytical solution (lines). The start of the fluid domain is shown as zero to match the analytical solution with 3 MD cells below and 3 above. The analytical solution is matched to the MD fluid part in y with $L_{y_{\text{fluid}}} = 37.4$, with half cell $\Delta y = 1.44$ extra at top and bottom and $\mu = 2.14$ with lines plotted at the same times in reduced units.

the full update kernel. In this way, the learnt kernel operator obtains the next timestep of a velocity field from the current one. We define the kernel normalised by C as $\widetilde{\boldsymbol{k}}_D = \boldsymbol{k}_D/C$.

To train the machine-learned CNN model for the flow fields of interest here, the same approach was followed for all cases, differing only in the dataset used for training the model. The first set of training data were obtained from the corresponding numerical solutions, resulting in the *numerically trained CNN* (abbreviated as numCNN). The second set of data is the exact analytical solutions of equations (2) and (3), yielding the *analytically trained CNN* (henceforth, abbreviated as anCNN). Since both the analytical and the numerical dataset must be generated by solving equation (1) either as an exact analytical solution of the PDE in terms of closed form functions, or as the numerical approximation; we employ a third dataset to examine the robustness of the CNN. This aims to have training dataset which is fundamentally different from the continuum approaches. This MD system has no link to the continuum equation (1) other than through the observation that the average resultant behaviour of unsteady Couette flow in an MD system broadly matches the continuum one (Smith 2013). The CNN trained on the dataset generated by these MD simulations is abbreviated as mdCNN.

The CNN architecture is identical in all the three cases, anCNN, numCNN, and mdCNN; the only distinction lies in the dataset on which it is trained. In this setup, the CNN effectively learns the numerical stencil weights of explicit time-integration schemes for the wall-driven flow. For numCNN, this is exactly reproducing the numerical scheme, as the CNN is mathematically identical to a finite difference operator. For the anCNN and mdCNN, the resulting CNN stencil is an effective numerical operator that reproduces that data. Using the Adam optimiser, the network iteratively adjusts its initially assigned random weights to minimise the loss. Through back propagation, the CNN progressively learns to reproduce the discrete evolution operator encoded in the training data, converging to kernel weights that are consistent with the target numerical stencil or solution.

The CNN is used as a numerical operator employing a 1D convolutional layer, with the kernel size determined by the chosen finite-difference scheme.

A single layer CNN is used in this study to learn a differential operator mathematically equivalent to a finite difference stencil. For a finite difference approximation to a PDE, the spatial update from u^t to u^{t+1} can be governed entirely by a fixed finite difference stencil, meaning that additional layers or non-linear activations would not improve this operation. Instead, a deeper architecture would obscure the direct correspondence between the learned kernel and the underlying numerical scheme, undermining the interpretability that is central to this work. Four schemes are considered for solving equation (2): forward Euler with (i) a 3-point stencil (FE3), and (ii) a 5-point stencil (FE5), and Adams–Bashforth with (iii) a 3-point stencil (AB3), and (iv) a 5-point stencil (AB5).

The kernel is of a length consistent with the finite-difference stencil. The bottom and top boundary conditions (henceforth, referred to as bBC and tBC respectively) are enforced at every timestep. This is done by setting the first ($x = 0$) and last ($x = l$) spatial points of the discrete flow field to the fixed values of the bBC and tBC, with higher order stencils needing special treatment, detailed in appendix A. This way, the boundary conditions are built into the convolutional operation, and do not need to be learnt from the data or require PINNs enforcement. The convolution operation applies the weighted sum to each point and its neighbouring ones, enforcing the finite difference update rule. The other numerical schemes (FE5, AB3, AB5) examined in this study are detailed in appendix A.

2.5.1. Training

In this section, a CNN kernel learner is trained based on the numerical-scheme operators. The weights of the CNN kernel learners are randomly initialised, with no clear convergence or stability advantage found in using common methods such as Xavier Initialisation (Glorot) method (Glorot and Bengio 2010). This random initialisation prevents the models from being biased toward the target values at the outset. Once initialised, training proceeds with the Adam optimiser (Kingma and Ba 2014), with a learning rate of 10^{-3} and a weight decay of zero. All codes are written in python, version 3.10 and Pytorch 2.8 with CUDA 12.6. Model accuracy is assessed by computing the mean squared error (MSE, denoted as ℓ) between the target value at the next timestep and the current prediction using weights k in the convolutional operator $\mathcal{F}_k$ defined in equation (10),

$$\ell\left(u_i^t, u_i^{t+1}\right) = \frac{1}{n_x N_t} \sum_{t=1}^{N_t} \sum_{i=1}^{n_x-1} \left[\mathcal{F}\left(u_i^t\right) - u_i^{t+1}\right]^2 + \sum_{n=1}^{3} \lambda_n \mathcal{A}_n. \tag{11}$$

This is minimised over the entire spatial domain ($n_x - 2$ as the outer two points enforce the boundaries) and for all time steps in the simulation to get the value of $\boldsymbol{k}$. This training can also be extended to multiple cases by simply defining the loss function over multiple cases (batches) where matched u_i^t and u_i^{t+1} pairs can be shuffled for a wide range of Couette and Stokes' flows with varying boundaries. Note that equation (11) and the convolution operator can be generalised to take multiple previous timesteps (e.g. u_i^{t-1}) in multi-step schemes like Adam-Bashforth or to predict multiple future steps by recursively applying the $\mathcal{F}_k$ operator. For presentational simplicity, these are not shown here.

The CNN model can be trained with physics constraints using the PINNs approach: the constraint A_n from equations 7a, 7b and 7c normalised by C, so $A_i = A_i/C$, which can be enforced during training via a Lagrange multiplier λ_n. These multipliers weight the constraint terms and are added to the loss in equation (9), so the total loss equals the data loss plus the three Lagrange-weighted constraint terms; tuning λ_n strengthens or relaxes each physical condition. The effect of the magnitude of the Lagrange multipliers is elaborated in the appendix, A.3 (figure A4), but are often chosen as zero so the constraints are not applied in general. Relevant sections discuss these in context. Thus, where PINNs is applied, these will be discussed in the various results. The learned kernels were also checked using the von neumann amplification factor. For stability, the maximum value should remain below one, $max|G| \leqslant 1$. This can be added as a fourth PINNs physics constraint, penalising any learned kernel where $max|G| > 1$.

3. Results and discussion

This section starts by training the CNN on a numerical reference case in section 3.1, with model weights shown to match the expected numerical stencil, giving results which generalise both to new boundary conditions and to Stokes' flow. This reference case is then used as the starting point to train the CNN on the analytical data in section 3.2. This solution is not the same as the numerical model, showing specialisation for the reference case it was train on. As a result, CNNs are trained on a range of

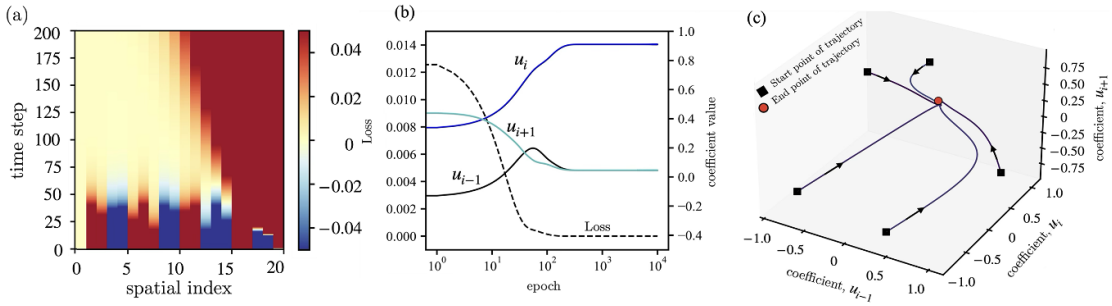

Figure 3. (a) The contour plot shows the reference case studied here with initial condition : $u_0 = \sin(2\pi x)$ with boundary conditions, bBC = 0 and tBC = 1.0 trained on numerical solution (b) Training loss (left y-axis) and coefficient values for u terms showing expecetd values $\boldsymbol{k} \rightarrow [0.0451, 1 - 0.0902, 0.0451]$ (right y-axis) for 3-point Forward Euler kernel, (c) Evolution of training trajectories for 5 randomly initialised weights. For all these cases, parameters used for the reference case simulations are: $\mu = 0.02$, $\Delta t = 0.1$, $n_x = 20$, $L_x = 4.0$, $N_t = 200$, bBC = 0.0, tBC = 1.0 & $u(x, 0) = \sin(2.0\pi x)$,.

other boundary and initial conditions analytical case, using energy maps to understand the convergence behaviour. Finally, multi-case training is shown which reproduces the same general numerical CNN values. A comparison between numCNN and anCNN is discussed in section 3.3 before exploring the process of training the CNN on MD data in section 3.4.

3.1. numCNN: CNN trained on numerical data

A wide range of boundary and initial conditions were run and analysed, training the CNN kernel learner on numerical solutions, with the resulting weights mostly converging to the same value. To exemplify this typical behaviour, we chose a reference case with $u(0, t) = 0, u(L, t) = 1.0, u(x, 0) = \sin(2.0\pi x)$, with numerical results shown in figure 3(a) using the standard simulation parameters with viscosity $\mu = 0.02$, spatial points $n_x = 20$, domain size $L_x = 4.0$ and timesteps $N_t = 200$ of size $\Delta t = 0.1$. The dataset for the training are generated through CFD simulations utilizing equation (5). Figure 3(a) is a contour plot representation, i.e. spatial and temporal plot of this reference case showing the velocity over the entire training data set.

Since the learned weights from the trained CNN model can be compared with the known numerical operators, the model becomes an interpretable numerical solver instead of a 'black box'. We show the CNN recovers the target stencil coefficients during training. Figure 3(b) shows the training loss and the coefficient values for the three-point Forward Euler (FE3) kernel. The loss rapidly decreases by nearly 10^{-15}, while the three learnt coefficients converge to the corresponding finite-difference weights. Figure 3(c) illustrates the trajectories of the coefficient vectors in the (u_{i-1}, u_i, u_{i+1}) space. Each trajectory begins with a randomly assigned weight vector (black square) but, irrespective of the initialisation, converges to the target FE3 coefficients. The target weights for the diffusion operator are: $\boldsymbol{k}_D = [0.045125, -0.09025, 0.045125]$, whereas the learned weights are $[0.04512625, -0.09024877, 0.04512625]$. These weights can be written in terms of $\widetilde{\boldsymbol{k}}_D$ which is scaled by $C = 0.045125$ giving $\widetilde{\boldsymbol{k}}_D = [1.00002768, -1.99997274, 1.00002768]$ with a MSE of $\mathcal{O}(10^{-12})$ so each weight has an error $\mathcal{O}(\boldsymbol{b} - \boldsymbol{k}_D) = [10^{-8}, 10^{-9}, 10^{-8}]$ compared to the exact numerical weights $b = [1, -2, 1]$.

This established that the CNN successfully identifies the correct stencil weights for a three point stencil, which is the simplest example that any form of explicit finite difference operators can be expressed as kernels and trained, with other stencils in space and time demonstrated in appendix A.

3.1.1. numCNN applied to unseen flow conditions

To demonstrate how the trained model performs on unseen flow conditions, we adopted two strategies. First, the model was tested on data from Couette flow, with the boundary conditions swapped/inverted as compared to the training data. The second test involved a time varying boundary condition, i.e. Stokes' second problem. Such condition is totally outside the training domain, and represents a stringent test of the generalisation of the numerical scheme. We elaborate the observations from these tests below. Although the agreement presented here is not unexpected, the exact reproduction of the numerical weights means we know the solver is already fully generalised; these tests emphasise this here and set out the methodology moving to the analytical case in section 3.2.

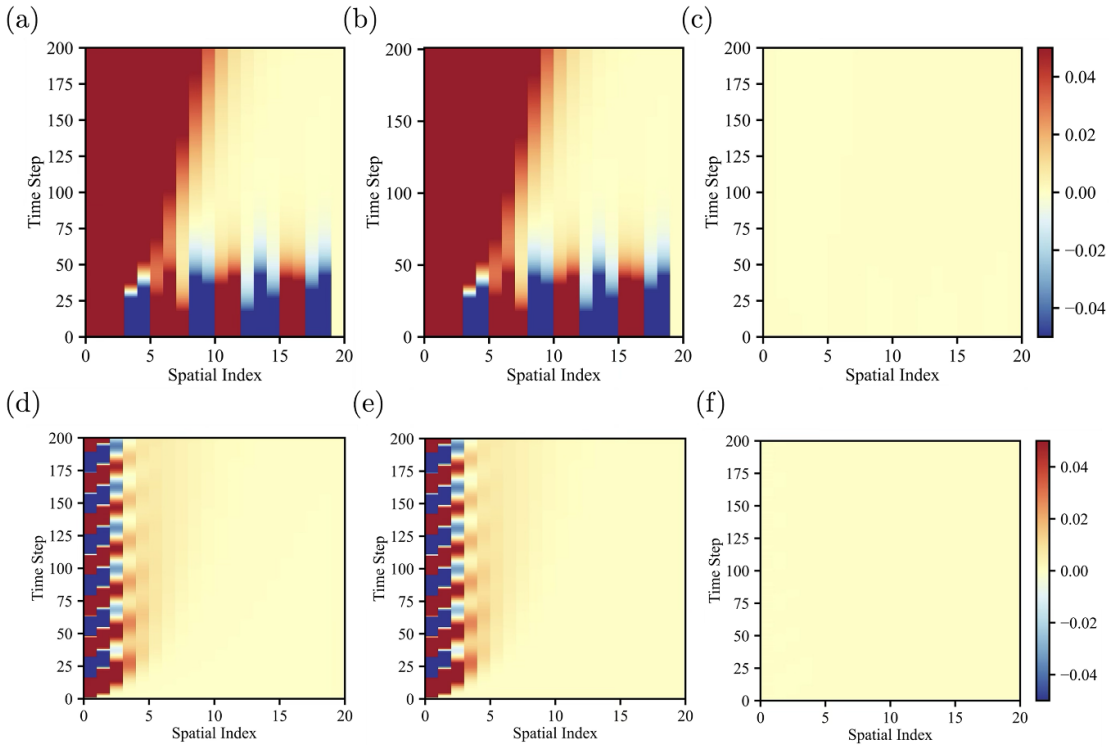

Figure 4. Performance of numerically trained kernel (numCNN) under swapped boundary conditions. Top panels represent Couette flow (a) CNN solution, (b) finite difference solution. Bottom panels show results for Stokes' second problem where (d) is obtained from the CNN model, and (e) from the finite difference scheme. Panels (c) and (f) illustrates the difference between the CNN solution and the numerical solution for each of the cases. The simulation parameters used here are: $\mu = 0.02$, $\Delta t = 0.1$, $n_x = 20$, $L_x = 4.0$, $N_t = 200$, $\Delta x = L_x/(n_x - 1)$. Couette flow: bBC = 1.0 & tBC = 0.0 and its initial condition is $u_0 = sin(2.0\pi x)$. Stokes' second problem: Boundary conditions $-$ $u(0,t) = \sin(\omega t)$ with $\omega = 2.0$, and $u(L_x,t) = 0$.

Couette flow with swapped boundary conditions: The model trained on the data of Couette flow with top wall sliding (tBC = 0, bBC = 1), is tested against similar Couette flow but with the bottom wall sliding and the top wall fixed (tBC = 1, bBC = 0). Recall boundaries are embedded in the training data, not the model, so any changes test the weights are not skewed. The solution produced by the numCNN is shown in figures 4(a)–(c), with the direct numerical solution shown in panel (b). A comparison of these two solutions can be seen from the error plot (panel c) which confirms a small difference between the direct numerical solution, and the solution from the CNN model. This indicates that the learned kernel reproduces the numerical scheme's behaviour without any considerable error, and is robust to this boundary-condition swap, reinforcing the numCNN's generality under the tested conditions. The advantage becomes particularly evident when the flow field spans many combinations of the parameter space. Once trained under a specific condition, numCNN generalises well to different combinations of the boundary-conditions. This highlights its potential as an efficient and computationally inexpensive alternative, without compromising the accuracy obtained from conventional numerical methods.

Stokes' second problem: To test the model's efficacy in solving related but unseen flow conditions, we test the CNN model (which is trained on Couette flow) with Stokes' second problem. Although both the Couette flow and the Stokes' second problem belong to the same class of viscous flows, their solutions are distinctly different, and involves different physical characteristics. The CNN's learned weights, as in figure 4(d) shows excellent agreement with those obtained from the direct numerical solution, as in panel (e). The difference between the two solutions is shown in panel (f) reaffirming the agreement. This demonstrates that even for unseen boundary conditions yielding distinct type of flow fields than the training data, the CNN model can serve as an efficient solver.

3.1.2. Beyond the black box

The robust agreement of figure 4, underscores the capability of the CNN framework to reproduce classical finite-difference coefficients directly from data, i.e. successfully learning the finite difference operators. This exercise demonstrates that CNN functions as a numerical operator capable of capturing the flow physics which are as fully general as the numerical methods they are trained on. In normal

ML operator training, the plots shown in figure 4 would be the only evidence the black box model behaves as expected and reproduces the fluid dynamics model they are trained on. Crucially, with these CNN stencils we retain a clear link from CNN to differential operator, where the coefficients k can be directly compared to the finite difference form they are trained on. This interpretability places the present approach in a unique position relative to other physics aware ML frameworks. Methods such as SINDy aim to discover governing equations by performing sparse regression on candidate nonlinear terms, yielding symbolic expressions that are highly interpretable but requiring explicit feature libraries and often struggling with noisy or high dimensional data. PINNs, on the other hand, embed the PDE residual into the loss function, enforcing physical constraints during training but relying on deep, multilayer architectures whose internal representations remain largely opaque. In contrast, the CNN stencil framework achieves interpretability not through symbolic regression or physics regularised training, but through architectural equivalence: the convolutional kernel is the numerical stencil. This provides a direct, parameter level correspondence between the learned operator and the underlying PDE discretisation, offering a form of interpretability that is both structural and quantitative. We now transition from working with numerical data with known weights, to see how CNNs behave when trained on the analytical solution, a sum of transcendental sine functions which cannot be expressed exactly in finite difference form. The resulting weights will therefore be an effective numerical approximation to the full analytical solution.

3.2. anCNN: CNN trained on analytical data

Section 3.1 discussed a CNN model trained on the dataset obtained from numerical solutions of wall driven flows. However, numerical modelling is an approximation of the true analytical solution, with each order of stencil getting closer to the exact solution. It is only in the limits of vanishingly small spatial and temporal step sizes or an infinite order stencil of points, that strict equivalence between the two is attained. Data obtained from the exact solution avoids the truncation and time integration errors associated with numerically solving the PDEs. Therefore, in this section, we train the CNN on analytical solutions, starting with the reference case used in the introduction of section 3 which allows us to compare to the numerical model. In practice, most fluid flow problems do not admit closed form solutions, so comparing the best numerical approximation to these true solutions provides an additional insight into the limits of numerical modelling. In practice for systems without such analytical solutions, a higher resolution numerical scheme could be used to create the reference and a low resolution stencil trained on it (Bar-Sinai *et al* 2019).

3.2.1. Comparison to Numerical case: $u(0,t) = 0, u(L,t) = 1.0, u(x,0) = \sin(2.0\pi x)$

The color map in figure 6(a) shows the analytical solution for fluid velocity (2) at different space-time coordinates. The anCNN is trained on this analytical data giving a set of weights,

$[1.09673750, -2.19871980, 1.09427700]$. These clearly differ from the finite difference weights $[1, -2, 1]$ as the training process aims to return an effective 3-point stencil operator which will optimally match the analytical solution, on average, over the whole spatio-temporal dataset. There is no correct solution, unlike the numerical case, so MSE in the training for the anCNN is $\mathcal{O}(10^{-7})$, much larger than training numCNN on numerical data (where error was $\mathcal{O}(10^{-12})$ and the resulting weights effectively match the numerical stencil $[1.00002768, -1.99997274, 1.00002768]$). Subtracting the anCNN and numCNN predicted flow fields from the analytical solution yields the error plots, these are illustrated in, figure 6(b) analytical minus anCNN, and (c) analytical minus numCNN. The time color bar in (b) and (c) ranges from blue at early times to yellow at the final timestep, with intermediate shades indicating the temporal progression. For anCNN (b) errors are initially much smaller but grow larger at longer times compared to those for numCNN which exhibits pronounced errors at the beginning. Because it is trained using the entire spatial-temporal analytical dataset, the anCNN operator must have a lower total error than numCNN when compared to the analytical solution. To quantify error norms over time for anCNN and numCNN models a further analysis is carried out where, we compared the time evolution of the CNN reconstruction error for the analytically trained model and the numerically trained model using the analytical solution as the common reference. The anCNN error was computed as the root mean square (RMS) L_2 error between the anCNN prediction and the analytical solution, while the numCNN error was computed as the corresponding difference between the numCNN prediction and the same analytical solution. The equation used is;

$$E(t_n) = \frac{1}{\sqrt{N_x}} \left[\sum_{i=1}^{N_x} \left(u_{\text{CNN}}(x_i, t_n) - u_{\text{analytical}}(x_i, t_n) \right)^2 \right]^{1/2} \tag{12}$$

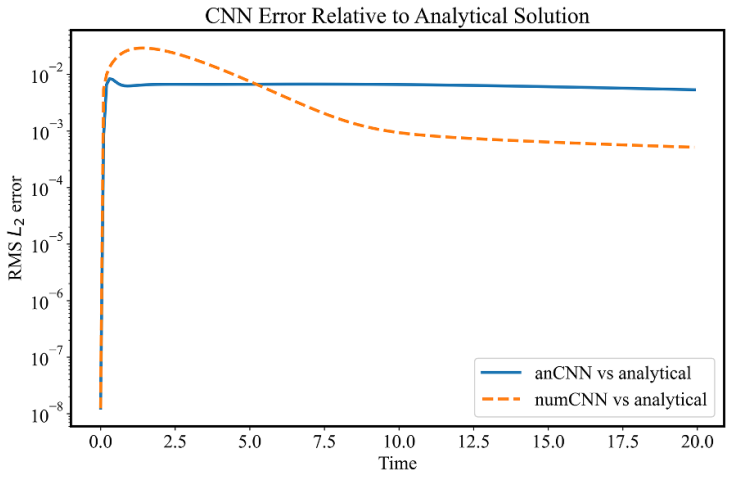

Figure 5. Time evolution of the RMS L_2 error for anCNN and numCNN, both evaluated relative to the analytical solution. The error is computed over the full spatial domain at each time step. The anCNN gives a smaller early-time error, while the numCNN error decreases over time and becomes lower at later times, indicating improved long-time rollout behaviour.

where, $E(t_n)$ is the error at timestep t_n, u_{CNN} is either anCNN or numCNN and $u_{\text{analytical}}$ is the analytical reference solution. Equation (12) is the time resolved RMS L_2 error relative to the analytical solution. In figure 5, the resulting time-resolved comparison shows that anCNN gives a smaller early-time error, whereas numCNN exhibits a larger initial transient but decreases over time and becomes lower than anCNN at later times. This indicates that numCNN provides improved long-time rollout accuracy when both models are evaluated against the analytical reference solution.

It is clear from figures 6(b) and (c) the trade off, with the weights of anCNN giving better short-time agreement at the cost of greater errors than numCNN when $t \to 20$. It is worth emphasising here that the training process of (11) learns the CNN weights to minimise error in predicting a single timestep forward, with the 200 step time history used as 200 samples to train on. However, the testing shown in figure 4 then uses the resulting anCNN and numCNN kernel weights to predict 200 timesteps forward from a single initial condition. As a result, differences in weights will result in an accumulated error over the entire simulation.

Multi-step training was also explored, i.e. minimising $[\mathcal{F}(u_i^t) - u_i^{t+1}] + [\mathcal{F}(\mathcal{F}(u_i^t)) - u_i^{t+2}] + \ldots$ from a single initial value. While this appeared to improve long time error, the recursive nature of this operation results in training time increases proportional to the number of steps. In addition, the number of steps chosen is arbitrary so will result in different weights based on this choice, with no clear guidelines for the appropriate number of multi-steps to use.

After comparing the numerical reference case to the analytical in section 3.2, we progressively raise the complexity of the dataset, i.e. by setting the boundary conditions in section 3.2.2 to zero, 3.2.3 finite values, 3.2.4 a parameter study of initial and boundary conditions, and finally 3.2.5 a single training case combining data from both Couette flow and Stokes' second problem.

3.2.2. Zero boundary conditions: $u(0,t) = 0, u(L,t) = 0, u(x,0) = \sin(0.5\pi x)$

The analytically trained CNN (anCNN) model is tested against a number of cases of homogeneous Dirichlet (zero) boundary conditions, each having randomly initialised weights. The initial condition is chosen as a single repeating sine function over the domain which has a value of zero at each boundary. Table 1 reports the final set of learned weights alongside the three fundamental criteria for classical numerical-schemes, i.e, consistency, symmetry, and scaling ($2\times$CFL). The most important observation is the model completely fails to train a consistent numerical operator, with all cases 1–4 training to different numerical values. Physically, this means that when given the case of a simply decaying initial condition, the CNN operator cannot find a unique set of weights which describes the physics of this problem. This is perhaps not surprising, as any choice of operator can act to simply diffuse a signal to zero, however the operators still appear to learn some aspects of the physics with symmetry almost respected by the learnt weights and consistency errors remaining relatively small. Following equations (7*a*)–(7*c*), consistency requires the sum of the three weights to be zero, symmetry requires the weights u_{i-1} and u_{i+1} to be equal, and scaling requires that the term is equal to $2C$. Please note, although the second column of table 1 shows the learned weights only to two decimal places for the sake of brevity, the model was

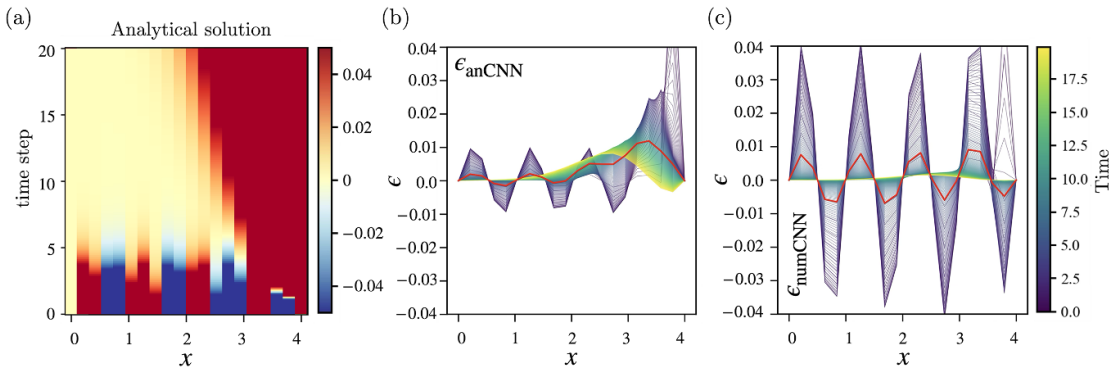

Figure 6. Comparison of CNN kernel training against the analytical solution for Couette flow bBC = 0.0, tBC = 1.0 and $u_0 = sin(2.0\pi x)$. (a) Analytical (exact) solution for Couette flow. The error curves (difference between the analytical solution and the solution obtained from the CNN models) at different timesteps for CNN trained on (b) analytical data, and (c) numerical data. The color bar at the far right denotes time and applies to both these panels. The red curve in both panels denotes the error when time-averaged over the entire simulation. The simulation parameters used are: $\mu = 0.02$, $\Delta t = 0.1$, $n_x = 20$, $L_x = 4.0$, $N_t = 200$, $\Delta x = L_x/(n_x - 1)$.

Table 1. Learned weights for zero boundary condition with random initial weights. A checkmark (✓) indicates the condition is satisfied, to 2 decimal places, and a cross (×) indicates it is not. The values in brackets next to the three conditions are to quantify the ✓ and × for all runs made. Criteria for these marks follow that the three conditions are satisfied per the equations (7*a*) Consistency: $\tilde{k}_{D-1} + \tilde{k}_{D0} + \tilde{k}_{D+1} = 0$, (7*b*) Symmetry: $\tilde{k}_{D+1} - \tilde{k}_{D-1} = 0$ and (7*c*) Scaling (2 × CFL): $2 - (\tilde{k}_{D-1} + \tilde{k}_{D+1}) = 0$. For Run 5, the (*) denostes that PINNs (equations (7*a*)–(7*c*)) is enforced with $\lambda_1, \lambda_2, \lambda_3 = 0.1$.

Run	Learned Weights ($\tilde{k}_D$)	Consistency	Symmetry	Scaling
1	[0.14, −0.38, 0.15]	× (−0.09)	× (0.01)	× (1.71)
2	[0.09, −0.28, 0.10]	×(−0.09)	× (0.01)	× (1.81)
3	[0.13, −0.36, 0.14]	× (−0.09)	× (0.01)	× (1.73)
4	[−0.04, −0.03, −0.0.4]	× (−0.11)	✓ (0.00)	× (2.08)
5*	[1.00, −2.00, 1.00]	✓ (0.00)	✓ (0.00)	✓ (0.00)

trained to full precision. Seen from the table, *Runs* 1 to 4, while failing to satisfy consistency and scaling, each of them successfully satisfies the symmetry criterion, with *runs* 1 to 3 having an approximate symmetry value almost zero (0.01). This indicates that even without any explicit constraint, the model captures the inherent symmetry of the data.

The systematic violation of consistency and CFL-scaling across all four runs can be attributed to two factors. First, zero Dirichlet boundaries provide no informative gradient at the edges. A diffusion process simply moves a signal to zero, a process that may not need a consistent stencil. Thus, leaving the CNN kernel learner with too weak a signal to train on near the domain limits. Second, random weight initialisation permits convergence to various symmetric stencils that fit the data numerically but do not capture physical consistency. Notably, failure to converge to $[1, -2, 1]$ was observed with these boundary conditions when trained on the numerical dataset. For run 5*, each of the constraints in equations (7*a*)–(7*c*) was enforced via a Lagrange multiplier, λ_n. A parametric study identified $\lambda_n = 0.1$ as the most effective and appropriately soft value to bias the CNN kernel learner toward the target weights (this is elaborated in the appendix, see figure A4 which summarises the results supporting this selection).

It is observed that the activation of all the constraints efficiently guide the model towards the target weights, $[1.00, -2.00, 1.00]$. Interestingly, the scaling constraint ($\lambda_3 = 0.1$) alone proves sufficient to guide the model towards the target weights. This implies that the consistency and symmetry constraints are already mostly learnt from the data, supported by near zero consistency and symmetry values in table 1 for runs 1–4.

3.2.3. Finite boundary conditions: $u(0,t) = 0, u(L,t) = 1, u(x,0) = \sin(0.625\pi x)$

The CNN model is now applied for Couette flow with top wall sliding (and fixed bottom wall) as contrasted to the fixed (both top and bottom) wall boundary conditions discussed in section 3.2.2 and with a different initial condition than the numerical comparison of section 3.2.1. The difference from the case in section 3.2.1 is the choice of initial condition, here a sine function chosen to give a smooth solution in both space and time, with $1\frac{1}{8}$ period to match the zero velocity at the bottom boundary, and go smoothly to the maximum velocity at the top boundary, i.e.[0,1]. Without any additional

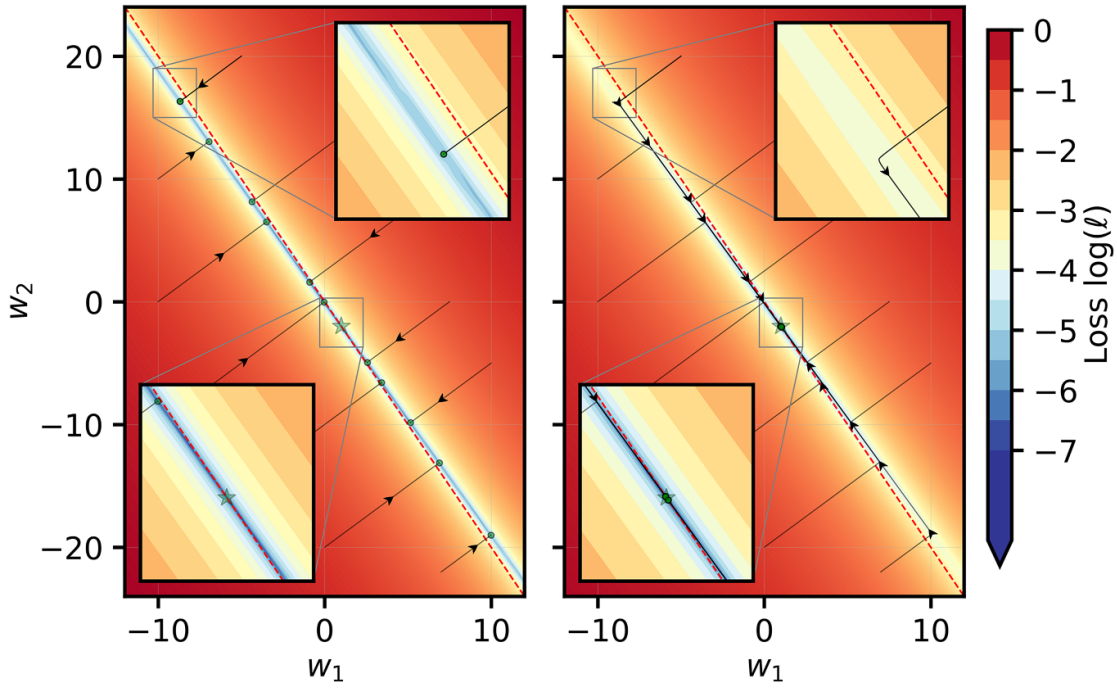

Figure 7. Contour plots showing energy landscape, generated by grid of sampled w_1 and w_2 values, assuming $w_3 = w_1$ ($w_1 = \tilde{k}_{D-1}, w_2 = \tilde{k}_{D0}$ and $w_3 = \tilde{k}_{D+1}$) and calculating cumulative loss over 200 single steps starting from analytical value at t and predicting $t+1$ numerically using these weights. Left : zero boundary conditions of section 3.2.2; right : finite boundary conditions of section 3.2.3.

constraints, the model consistently produces the learnt weights $[0.99, -1.98, 0.99]$ across six independent runs starting from random initial weights. When contrasted with the weights from section 3.2.1, $[1.097, -2.199, 1.094]$, which was also trained on an analytical solution, it is noteworthy the weights obtained here essentially match the finite difference solver. This is attributed to the use of a function which smoothly satisfies the boundaries at all times, not the impulse started Couette flow of section 3.2.1. An impulse started case has a discontinuity between boundary values $[0, 1]$ and initial condition $\sin(2\pi x)$ (as $u(x = L_x = 4, 0) = \sin(8\pi) = 0$ which is inconsistent with $u(x = L_x) = tBC = 1$). Such cases of impulse started (startup) Couette flow are well studied in the literature (Batchelor 2000) and relevant for many areas such as tribology or jet coating. Startup Couette flow inherently has a discontinuity $u(L_x, 0) = u_t H(t-0)$ with H the Heaviside function and $\partial u/\partial t|_{L_x,0} = \delta(t-0)$ representing an infinite impulse. The numerical approximation of this introduces errors and the training process appears to be attempting to this by skewing the weights of the anCNN in section 3.2.1. As a result, this explains the lower error in figure 6 at short times using the effective stencil, especially notable on the right hand side comparing figures 6(b) and (c) with a much larger error observed when numerical weights are used.

As the anCNN is learning weights very close to the numerical solution, these satisfy all three conditions: consistency (0.00), symmetry (0.00), and scaling (0.02) (equations (7*a*)–(7*c*)) so PINNs constraints are not needed. If PINNs is activated, with the Lagrange multiplier $\lambda_n = 0.1$ for all three constraints $(\lambda_1, \lambda_2, \lambda_3)$ across six runs, the resulting learnt weights $[1.00, -2.00, 1.00]$ satisfy all three constraints of consistency, symmetry and scaling exactly.

To better understand the optimisation process across the parameter space in sections 3.2.2 and 3.2.3, we adopt the idea of energy (or, MSE (ℓ)) landscape where a high energy (loss) means we are further away from the optimal solution, while the lowest energy corresponds to a stable solution. Figure 7 shows the energy (loss) landscape as a function of the parameters learned weights w_1 and w_2 with the assumption that $w_1 = w_3$, as expected for symmetry. The left panel of figure 7 shows loss landscape corresponding to the case with zero boundary condition as discussed in section 3.2.2, i.e. $u(0, t) = 0$, $u(L, t) = 0$, $u(x, 0) = \sin(0.50\pi x)$. Whereas, the right panel shows the case with finite boundary conditions discussed in this section, i.e. $u(0, t) = 0$, $u(L, t) = 1$, $u(x, 0) = \sin(0.625\pi x)$. Each point on the map represents a specific combination of these w_1 and w_2, and the colour indicates the logarithmic loss scale. Blue

Table 2. Learned weights and property evaluations under mixed boundary and initial conditions. A check mark (✓) indicates the condition is satisfied and a cross (×) indicates it is not. These three conditions are satisfied (✓) by the following : (1) Consistency: $\tilde{k}_{D-1} + \tilde{k}_{D0} + \tilde{k}_{D+1} = 0$, (2) Symmetry: $\tilde{k}_{D+1} - \tilde{k}_{D-1} = 0$ and (3) Scaling ($2 \times$ CFL): $2 - (\tilde{k}_{D-1} + \tilde{k}_{D+1}) = 0$. bBC = bottom boundary condition; tBC = top boundary condition.

Case	bBC	tBC	Initial condition	Learned weights ($\tilde{k}_D$)	Consistency	Symmetry	Scaling ($2 \times$ CFL)
1	0.0	1.0	$0.0\sin(0.0\pi x - 0.0)$	[0.39, −1.27, 0.77]	× (−0.11)	× (0.38)	× (0.84)
1*	1.0	0.0	$0.0\sin(0.0\pi x - 0.0)$	[0.77, −1.27, 0.39]	× (−0.11)	× (−0.38)	× (0.84)
2	0.5	1.0	$0.0\sin(0.0\pi x - 0.0)$	[0.85, −1.71, 0.87]	× (0.01)	× (0.02)	× (0.28)
3	0.5	1.0	$\sin(0.583\pi x + 0.524)$	[0.98, −1.97, 0.98]	× (−0.01)	✓ (0.00)	✓ (0.04)
4	−1.0	0.0	$\sin(1.125\pi x - 1.571)$	[1.03, −2.06, 1.03]	✓ (0.00)	✓(0.00)	× (−0.06)

regions (which would take the form of valleys in a 3D plot) correspond to low loss indicating most stable states, while red regions (would take the form of peaks in a 3D plot) indicate unstable states. A series of training trajectories are shown with initial values of $w_1 = w_3$ and w_1 vs w_2 values chosen to cover the space. Arrows show how a solution approaches a stable configurations, where figure 7(a) has a trench of possible solutions, with any starting point reaching its nearest point in this trench and stopping. This explains the range of possible solutions for the zero boundary case. Meanwhile, (b) sees a single global minimum where solutions move to the trench and then along until reaching the global minimum value.

3.2.4. Parameter study of initial and boundary conditions

Having studied zero and finite conditions with randomised initial weights training, the CNN model is now tested for analytical solutions with complete randomisation of the initial and boundary conditions. Specifically, all permutations of boundary values tBC and bBC, were from the range $\{-1, 0.5, 0, 0.5, 1\}$. The initial sine function is chosen to satisfy the boundary conditions, top and bottom boundary conditions (tBC, bBC) over the domain length L_x. The result is of the form $B\sin\left(a\pi x - b\right)$. The CNN kernel learner was trained using samples drawn from the specified ranges for both boundary conditions and initial conditions; for each training data, a random boundary condition and a random initial condition were selected from those ranges. 75 different cases were conducted for this section, hence having 75 different learnt weights shown in figure 8. Out of the 75 cases, 5 of the most interesting cases are highlighted in table 2.

In case 1, an asymmetric stencil emerges when the system is initialised with zero initial conditions. This is the extreme case of impulse started Couette flow. Under these settings, the CNN kernel does not learn a stencil near to the numerical weights $[1, -2, 1]$. Instead, the model repeatedly converges to the same one-sided stencil reported in table 2. This stencil is obtained for many independent initial weights, even trying to bias the solution by starting with weights $[1, -2, 1]$. This one-sided stencil is attributed to the impulse started nature of the flow from a zero initial condition, which results in a discontinuity at time zero next to the wall. There is a resulting error in the numerical scheme which cannot capture this discontinuity. As a result, the optimal stencil to reduce this error appears to be the one obtained which does not satisfy symmetry. Interestingly, when the boundary conditions are flipped (i.e. $\text{bBC} = 1, tBC = 0$) while keeping the initial condition unchanged, the CNN kernel learns weights of $[0.77, -1.27, 0.39]$, which represents a reversed form of the earlier stencil. This is represented as case 1* in table 2 and figure 8. This reversal demonstrates potential bias in ML model due to one-sided datasets. One way to fix this is to train on multiple datasets, here Cases 1 and 1* were jointly trained as a single scenario, referred to as case 1mix, to examine how the model integrates information from both. The resulting learned weights from case 1mix are $[0.8669979, -1.718784, 0.8669848]$ showing the asymmetry is removed on average, but the resulting solution is not simply the general one from the numerical solver. Another approach is to use PINNs. PINNs, when activated for all three constraints gives learnt weights $[0.99953544, -1.9975013, 0.99878305]$ close to the expected generalised form. This outcome demonstrates that although both original cases individually produced incorrect weights, their combination enabled the model to converge more closely toward the target weights and that adding PINNs gets a solution which matches the general numerical form. With symmetry (λ_2) enabled, case 1 matches case 1mix. Only the Scaling term (λ_3) drives convergence, as also seen in section 3.2.2.

For case 2, the learnt weights deviate significantly from the expected target weights. Similar to case 1, this scenario is also initialised with a zero initial condition, as shown in table 2. Repeated runs with randomised kernel weights consistently converge to the same set of weights. When the boundary conditions are flipped ($\text{bBC} = 1, \text{tBC} = 0.5$), the learnt weights ($w_1$ and w_3) also flip, mirroring the behaviour

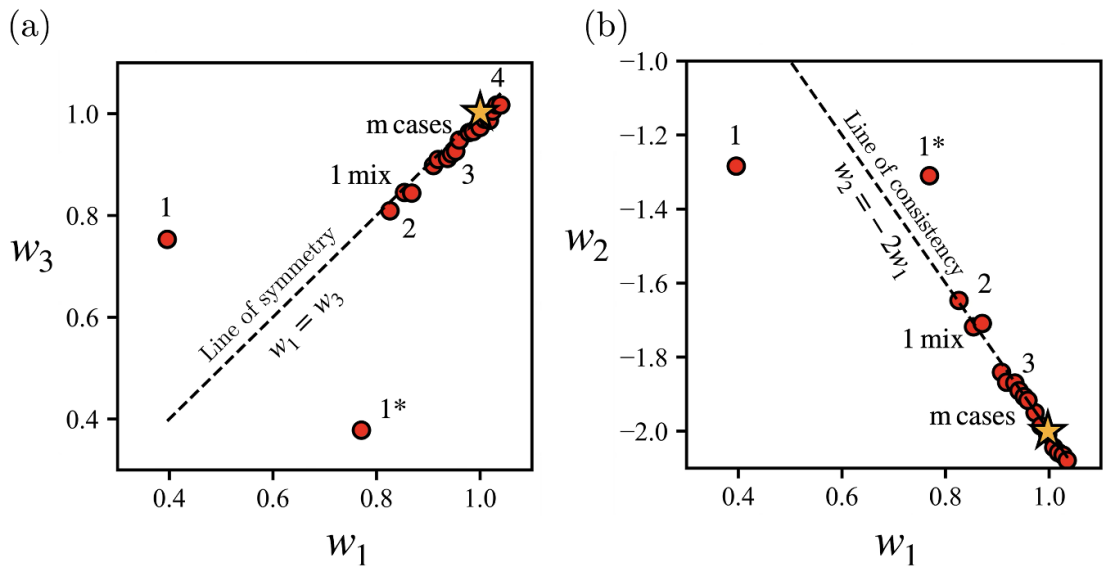

Figure 8. Scatter plot showing exploration of possible scenarios in the $w_1 - w_3$ and $w_1 - w_2$ ($w_1 = \tilde{k}_{D-1}, w_2 = \tilde{k}_{D0}$ and $w_3 = \tilde{k}_{D+1}$) parameter spaces, emphasising cases 1, 1*, 2, 3, 4 and 1mix which are discussed in section 3.2.4 (table 2), and 'm cases' (combined multi case) as in section 3.2.5. The star symbol denotes the target numerical weights $[1, -2, 1]$.

observed in case 1. This again underscores the strong role of boundary conditions in shaping the learning process.

In case 3, the CNN kernel consistently converges towards the prescribed set of weights, even after four repeated runs. This case demonstrates how continuous initial condition tend to provide a smooth learning process, guiding the kernel towards repeatable convergence. Case 4 shows a scenario where the learned weights are greater than the target weights for a defined boundary condition and initial condition.

However, for all five cases in table 2 trained on numerical solution converges towards the target weight $[1, -2, 1]$. This reinforces the discussion in section 3.1 which identifies the numCNN as the most general solution. Figure 8 highlights the weights from the 75 cases run made for section 3.2.4 and other *cases* defined in its caption.

3.2.5. Combining data from Couette flow and Stokes' second problem

A combined multi-case problem is analysed, unifying many of the studies so far, including cases which give asymmetric weights or fail to converge to the general $[1, -2, 1]$ case. We already observed that mixing cases, as shown in case 1mix $(1 + 1^*)$ in section 3.2.4, removes systematic bias in the training. This has deep analogies in both ML in general and fluid dynamics where datasets must be diverse and representative of the potential scenarios. For this case, the FE3 CNN kernel learner is trained on a combination of two Couette flow and four Stokes' second problem. These include Couette flow (a) with $u(0,t) = 1$ and $u(L,t) = 0$ with corresponding initial condition $u(x,0) = \sin(0.625\pi x)$ then (b) swapped top and bottom so $u(0,t) = 0$ and $u(L,t) = 1$ with $u(x,0) = \sin(0.625\pi x + \pi/2)$. For the four Stokes' second problem cases, the initial condition is given by $-A\exp(-\kappa x)\sin(\kappa x)$ where $\kappa = \sqrt{\omega/(2\mu)}$ with (c) and (d) having the top wall oscillating $u(L,t) = \sin\omega t$ with $\omega = 0.5$ and $\omega = 1.0$ respectively, while *e*) and *f*) have an oscillating bottom walls with $u(0,t) = \sin\omega t$ with $\omega = 0.5$ and $\omega = 1.0$ respectively.

Analytical settings similar to previous ones were used to train the model, with the exception that training was carried out in batches. This is due to the dataset being large (1194 samples) as compared to previous cases. The model was trained for 1000 epochs with Adam optimiser; learning rate was 10^{-3}, with no weight decay. Weights were initialised by employing Xavier (Glorot) algorithm, and all cases were shuffled together before training to avoid any order-dependent pattern. Batching improved memory, compute efficiency and averaged gradients per batch, which stabilised updates, while the added noise helped generalisation.

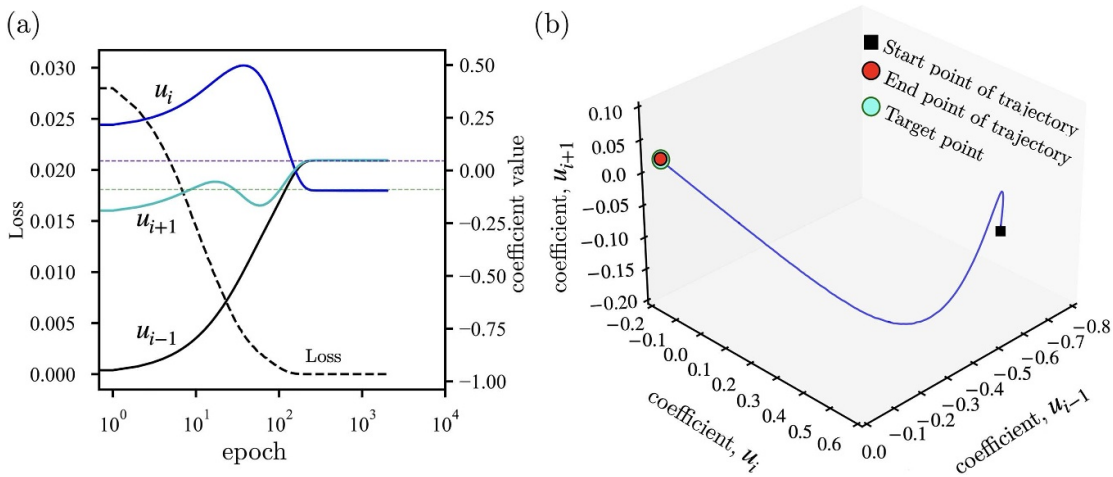

Figure 9. (a) Loss plot and convergence for complementary mixed flows (wall-driven flow, and Stokes' second problem). The coefficients are in the form of k_D. (b) 3D visualisation of training from start to end for a mixed (wall driven flow and Stokes' second problem) case.

We observed that the model learns the kernel weights as illustrated in figure 9(a). The corresponding weights are $[0.99507344, -1.9861549, 0.9918588]$. Consistency (0.00), symmetry (0.003), and CFL-Scaling (0.013) are simultaneously satisfied. This confirms that training on a dataset that covers a variety of cases, provides the model with sufficient information to recover the exact physics-preserving kernel. Overall, exposing the model to balanced flow cases and appropriate boundary conditions effectively constrains training, guiding the CNN to infer target weights that meet all numerical-scheme criteria.

3.3. Comparison between numCNN and anCNN

The preceding discussions on the performance of the two operator learning strategies, i.e. numCNN and anCNN, clearly demonstrate a major difference between the methods' generalisation ability. The numCNN is shown to give exact agreement to the numerical stencil, an expected result given the mathematical form of the CNN is identical to the finite difference stencil. Numerical data implicitly carry information concerning numerical discretisation, as such numCNN inheres this from the training data. For a 3 point stencil (as well as 5 point and multi-step Adam-Bashforth), the numCNN model can be viewed as the most *generalised* form a CNN can be, given it could be applied to any boundary condition or geometry yielding consistent results. The agreement between the numCNN and the numerical stencil reassures this point, and represents a major advantage of using simple NN to avoid black box behaviour.

When trained on the analytical form equation (2), an exact solution of the differential equation, the 3 point numerical stencil of the anCNN is only able to approximate this solution. However, this 3 point stencil trained in this way is potentially more *accurate*, as a result of being overtrained on a particular analytical case. The consequence of this overtraining is that the anCNN is indeed not expected to generalise to another case, since it is pertinent only to the original flow field. Consequently, when solving flows that span a wider range of parameter space, and therefore produce substantially different flow physics (see section 3.2.5 for the Couette flow and Stokes' second-problem cases), numCNN demonstrates greater generalisability than anCNN as seen in section 3.1.1. Accordingly, anCNN performs well on balanced flow cases but requires more exposure to varied flow regimes and carefully chosen boundary conditions to match numCNN's generality.

In some cases, such as a pure diffusion problem in the absence of boundary conditions, the anCNN fails to train the model to a consistent set of weights. This is attributed to the under-defined nature of the problem where any operator could take an initial condition and move it towards zero. Hence, enforcement of additional constraints, or PINNs is required to recover a consistent stencil. In particular, the scaling condition of equation (7*c*) which indirectly enforces a viscosity magnitude on the problem, results in the expected weights from the numerical scheme. This trade-off between the general numCNN and accurate anCNN has deep parallels to the field of ML in physics, where complex architectures struggle to generalise beyond the training dataset (Yuan *et al* 2022) with PINNs helping to expand generalisability by embedding physics and so requiring less data (Raissi *et al* 2019).

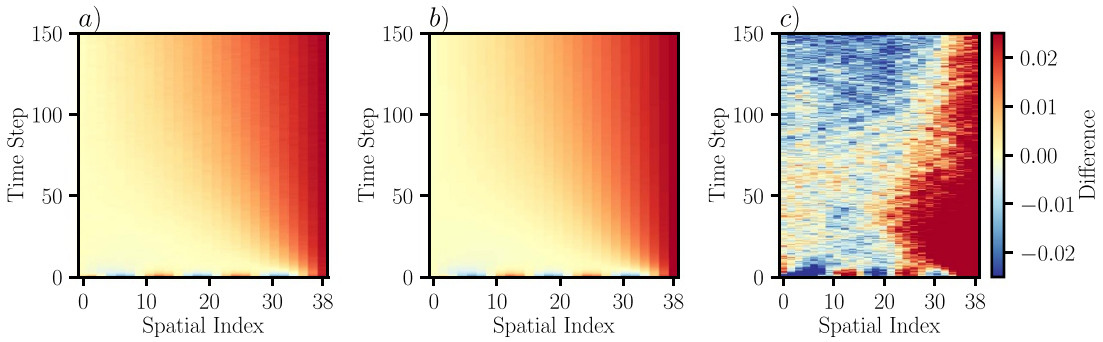

Figure 10. MD vs trained model with (a) the MD time space plot over 3000 timesteps of size 0.05, with model trained in (b) where weights are $[0.984, -1.96, 0.976]$ and difference between the model and MD in (c) showing the extra physics in the MD simulation not captured by the mdCNN, which includes both noise as well as an apparent signal, which may indicate local viscosity variation, near the top boundary.

Furthermore, appendix A.2.2 establishes the robustness of this interpretable framework; even with the inclusion of an auxiliary convection CNN kernel, the training dynamics reveal that reproducing both the numerical and analytical weights requires only the diffusion operator.

3.4. mdCNN: trained on MDs simulations

The training of CNNs on both analytical and numerical solutions demonstrate the effectiveness of CNNs in solving differential equations. As opposed to other forms of ANNs, the use of CNNs incorporates the spatial information of the physical problem. Hence, any variation in the local velocity field is automatically accounted for. This section examines whether a CNN can leverage spatial locality to predict flow fields with rich local variations described by a fundamentally different governing equation of MDs simulations.

The uniqueness of MD is that it does not require us to simulate the diffusion equation, nor does it require the assumptions of the transport coefficients (e.g. viscosity, surface tension, conductivity); instead these properties and the flow field itself, emerge naturally as average observations of the motion of thousands or millions of particles. The solving of fluid dynamics with molecules, known as Non Equilibrium MD (NEMD), has been shown to reproduce a wide range of fluid phenomena (Rapaport 2004, Todd and Daivis 2017). As a result, it is closer to running an experiment and taking observations of the collective behaviour of the particles. For this reason, MD provides an excellent test of the limits of employing CNN for ML driven flow modelling. The measured velocities include noise, with the average velocity known to broadly agree with the unsteady diffusion equation with what can be interpreted as additional fluctuations (Zhang *et al* 2019, Sprittles *et al* 2023). These noise terms can be seen in figure 10 with the difference between the predicted flowfield from the mdCNN trained on the MD data. The training is performed only for cells in the fluid region (the 26 bins not including tethered wall atoms), with the CNN trained on $n_x = 28$ cells with bottom and top set to boundary conditions ($u(0,t) = 0$ and $u(Ly,t) = 1$) respectively. The resulting mdCNN learns weights of $[0.984, -1.96, 0.976]$, with loss of order 10^{-6}, obtained assuming a viscosity $\mu = 2.14$ as in the matched analytical solution of figure 2, a value consistent with previous work (Smith 2013). Alternatively, the learnt CNN weights could be used to estimate the viscosity by solving the inverse problem, assuming the CNN should learn weights of $[1, -2, 1]$ (if the 3-point stencil approximation can be applied perfectly) a viscosity prediction of $\mu \approx 2.10$ is obtained (an error of $\sim$2%). The viscosity value of $\mu = 2.14$ is well established, taken from previous work (O'connell and Thompson 1995, Nie *et al* 2004) and checked using both Green–Kubo (Green 1952, Kubo 1957) as well as matching to analytical solutions in Smith (2014). Using an ensemble of MD runs or a wider MD domain to provide less noise in the averaging or mixing a range of starting sine waves and sliding conditions for a wider training set would be ways to potentially improve the viscosity estimation. The learnt stencil gives a scheme with consistency (0.00), symmetry (-0.008), and scaling $2C - \sum_{i=1}^{3} i^2 {k_i}^2 = 0.04$. However, starting the training from random initial weights can result in variation in final weights, some of which give an unstable numerical solvers, e.g. $[1.085, -1.860, 1.077]$. This was found to be fixed with careful choice of training algorithm, varying training rates or inclusion of a scheduler. The version of code provided with this work includes these improvements to ensure a consistent set of weights.

However, this approach could also be employed to solve the so-called inverse problem, where Δx and Δt are prescribed, and the deviation of the obtained weights from the numerical set $[1, -2, 1]$ allows an estimate of the effective viscosity in the MD system. This provides an alternative route to the various schemes of obtaining viscosity in MD, including Green Kubo and other NEMD methods (Evans and Morris 2007). Because the fit uses the entire time and space response of the NEMD system, and given the robust nature of the ML training algorithms, it would be expected to provide a fairly reliable way to judge viscosity. In addition, the process could be extended to more complex physics by modelling additional terms, for example the CNN approach is known to work with convective and diffusive terms (Queiruga 2019), identifying both individually. This also points to the tantalising possibility of training when the appropriate PDE is unknown, providing the numerical kernel of potential differential terms in CNN form and observing which terms are non-zero after training. This has similarities to pySindy (Brunton *et al* 2024) but replacing functional forms with CNN weights, driven by the ML backpropagation algorithm, may potentially allow more robust equation identification. Proof of the viability of this approach is shown in the appendix A.2.2, where the inclusion of a convection CNN kernel in additional to the diffusive kernel, learns that only the diffusion operator is needed to reproduce the MD data. It remains to be seen how resistant to noise the methodology will be, as the current case is extensively averaged to give a very low noise to signal ratio, as can be observed by the agreement between analytical and MD profiles in figure 2 as well as figures 10(a) and (b).

4. Conclusion

This study integrates ML with CFDs by utilising CNN kernels as numerical operators. The proposed framework is the limiting case of network simplicity in operator learning for data-driven fluid dynamics, requiring only three trainable parameters. The learnt CNN kernels are mathematically identical to the finite difference operators, which resolves the 'black box problem'. This provides a framework to understand generalisability, accuracy and reproducibility in training operators to match CFDs results. More complex CNNs, for example a 5 parameter kernel can be seen to reduce down to just the 3 parameters when trained on 3-point numerical datasets, demonstrating parsimony, but retaining all 5 points when the dataset used a 5-point numerical scheme. Complex CNN shapes could therefore be constructed to allow training to discover new numerical stencils in space and time, but retaining the link to finite-difference operators. Two benchmark problems, i.e. wall-driven (Couette) flow and Stokes' second problem illustrate the implementation and efficacy of these CNN-based schemes, supported by convective burgers equations test in the appendix. A dedicated 'CNN kernel learner' was trained on finite-difference solutions (numCNN) to recover the exact stencil coefficients, achieving consistent convergence regardless of random initialisation. The learned weights eventually come out as $[1.0000, -2.0001, 1.000]$ with a total MSE of $\mathcal{O}(10^{-12})$. The CNN kernel learner (numCNN) accurately inferred the expected flow fields when applied to unseen conditions, demonstrating robust generalisation. Similarly training the model on the same solution but analytically learns the weights $[1.0967, -2.1987, 1.0943]$ with the training error increasing to $\mathcal{O}(10^{-7})$. This reflects the fact that the analytical solution cannot be represented exactly by a three-point stencil. The performance of the CNN kernel learner trained on analytical solutions (anCNN) was evaluated across cases with increasing complexities to understand the ability of models to represent more complex cases. In some cases, such as zero boundary conditions, the kernel fails to repeatably find a unique numerical operator. In other cases, the learnt operator is different from the general numerical operator and for extreme cases, the training data can result in skewed and unphysical kernels. The simple nature of the CNN kernel means we retain clear insight into the limitation of the training approach, with symmetry or consistency violated in the operator despite the resulting flow fields still appearing to provide reasonable agreement with expected behaviour. An improvement in training is shown by mixing a wider range of training data from various problems or applying physics through PINNs like constraint terms. This provides deep insights into the limitation of data driven fluid dynamics, namely the requirement for balanced and diverse datasets and application of physical constraints when this is not possible. It is reasonable to expect such insights will carry into more complex network models, so these simple CNN networks are ideally suited as a test-bed for general ML training and dataset design.

These simple CNN operators also have the potential to learn physics, solving the inverse problem in both numerical and analytical settings. When trained on numerical solutions, the CNN kernel reliably recovers the target finite-difference weights; when trained on analytical data, effective kernels are

obtained with lower errors than a numerical solver for a given cases. As the outcome is sensitive to the composition of the training set, it is shown that well-designed and balanced datasets are required to guide the model toward the most general weights. Application of the CNN kernel learner to MDs data (mdCNN) shows the model effectively learns approximate target weights despite the inherent noise in MD datasets and the simulation having no underlying connection to the numerical operators. As a result, these interpretable operators can discover coefficients through the inverse problem and even determine the form of differential equations. Tests with MDs simulations show that this simple convolutional operator can extract meaningful operator weights from noisy MD data which have no underlying continuum equations, allowing effective differential operators to be determined from data. Training on MDs data (mdCNN) further highlights the robustness of the approach. Despite noise and the absence of an underlying continuum PDE, the CNN converges to $[0.984, -1.96, 0.976]$ with a loss of $\mathcal{O}(10^{-6})$, successfully extracting a stable diffusion-like operator from atomistic trajectories. This shows that the CNN kernel learner can perform inverse-problem inference even when the governing equations are not explicitly defined. These CNNs produce interpretable, engineering-compatible finite-difference operators which are fully transparent. The findings of this work serve as the ground work for future studies extending the proposed approach to equation discovery in experimental and higher-fidelity simulation data, exploring non-linear flow regimes and operator stability, and provide a simple baseline for evaluating alternative hybridisation strategies and ML architectures to broaden robustness and applicability.

A key caveat of the present study is its restriction to 1D diffusion problems that are devoid of higher-order interaction terms. The extension of the proposed framework to multi-dimensional and non-linear systems remains a non-trivial challenge, although a preliminary investigation of the nonlinear Burgers' equation is presented in appendix A.2. Nevertheless, this work provides a transparent framework for understanding how CNNs learn and emulate numerical operators. At the same time, this serves as an important first step toward the development of interpretable neural operators that are capable of solving more realistic and computationally demanding flow problems.

Funding

This work was supported by the EPSRC ref.EP/W524542/1

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/kwamea-b/CNN_numerical_schemes.

Q2

Author contributions

Kwame Agyei-Baah 0009-0007-1357-6470
Conceptualization (lead), Data curation (lead), Formal analysis (lead), Investigation (lead), Methodology (lead), Visualization (lead), Writing – original draft (lead), Writing – review & editing (equal)

Muhammad Rizwanur Rahman 0000-0002-1867-0737
Formal analysis (supporting), Investigation (supporting), Methodology (supporting), Supervision (equal), Visualization (equal), Writing – review & editing (equal)

Edward R Smith
Data curation (supporting), Formal analysis (supporting), Investigation (supporting), Methodology (supporting), Supervision (lead), Visualization (supporting), Writing – review & editing (supporting)

Appendix

A.1. Other numerical schemes

The weights for the numerical schemes considered are shown in figure 11.

A.1.1. Forward Euler 5-point stencil

A similar approach for discretisation is taken for the forward Euler 5-point as done in equations (4) and (5) above in section 2.3. However, since a 5-point stencil is discussed here the weightings and terms of k_1 changes now. Thus equations (3) and (4) changes to:

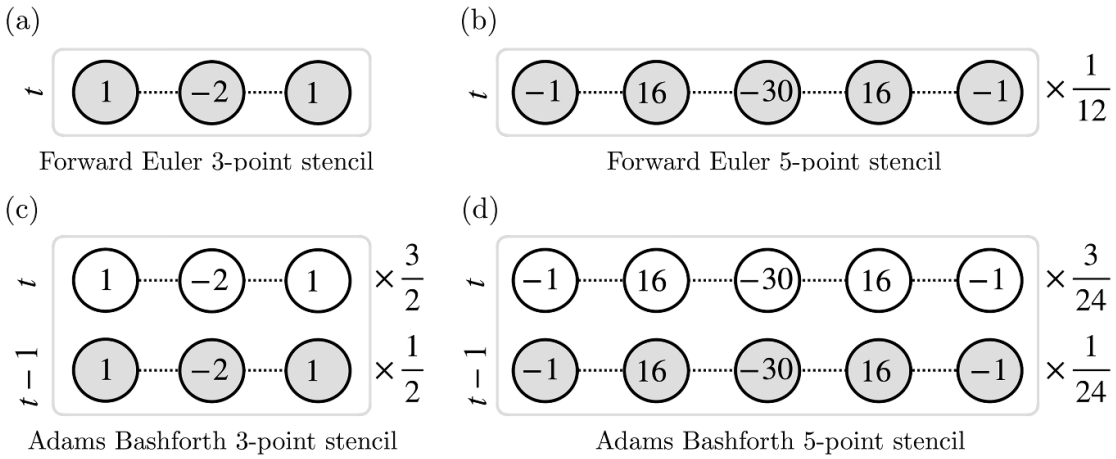

Figure 11. Time-integration stencils and their coefficients: (a) Forward Euler uses a 3-point stencil, $[1, -2, 1]$ and (b) a 5-point stencil, $[-1, 16, -30, 16, -1]$ finite-difference scheme. The 3-point versions are second-order accurate in space, whereas, the 5-point versions achieve fourth-order spatial accuracy. The multi-timestep Adams–Bashforth employs analogous (c) 3-point, and (d) 5-point stencils, which offers second order temporal accuracy.

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} = \mu \left(\frac{-u_{i-2}^t + 16u_{i-1}^t - 30u_i^t + 16u_{i+1}^t - u_{i+2}^t}{12\,(\Delta x)^2} \right) \tag{13}$$

$$u_i^{t+1} = C\left(-u_{i-2}^t + 16u_{i-1}^t - 30u_i^t + 16u_{i+1}^t - u_{i+2}^t\right) + u_i^t \tag{14}$$

where,

$$C = \frac{1}{12}\frac{\mu \Delta t}{(\Delta x)^2} = \text{constant} \tag{15}$$

A.1.2. Boundary conditions handling: mirroring approach

For this numerical-scheme operator, a mirroring approach is applied to handle the boundary conditions instead of using a one-sided stencil. This choice avoids any modification to the convolution kernel. In practice, ghost cells are generated at each boundary by reflecting the adjacent interior values across the boundary: the right-hand ghost cell is obtained by mirroring its neighbouring interior point about the right boundary, and the left-hand ghost cell is generated similarly at the left boundary. The mirroring procedure is expounded below. As defined earlier the boundary condition used here are Dirichlet boundary conditions:

$\text{bBC} = u(0) = 0$ & $\text{tBC} = u(L) = 1$

Two ghost cells are needed on the left side of the boundary domain, denoted u_{l1} and u_{l2} to compute a 5-point stencil at the first interior point. The mirroring approach sets out these ghost cells by reflecting interior values u_1 and u_2 about the boundary location. The boundary value bBC is to lie halfway between the ghost cell and its corresponding interior cell.

$$\text{For ghost cell 1 } (u_{l1}): \quad \text{bBC} = \frac{u_{l1} + u_1}{2} \tag{16}$$

$$\text{For ghost cell 2 } (u_{l2}): \quad \text{bBC} = \frac{u_{l2} + u_2}{2} \tag{17}$$

which can be rearranged to compute the ghost cells. A similar approach can be employed for the top boundary condition (tBC) reflecting the last and second to last to interior values, u_N and u_{N-1} respectively.

For both boundaries, the mirroring is exactly implemented in the code as well. The ghost cells are concatenated with the interior values to form an extended array (padded tensor) over which convolution i.e. finite difference update is performed. In this way, the approach is valid for CNN, and can be extended to arbitrary order stencils/kernel sizes. The mirroring approach allows for the finite difference stencil (5-point) to remain centred hence maintaining the accuracy of the scheme near the boundaries as well as satisfying the boundary conditions.

A.1.3. Adams Bashforth

The Adams Bashforth 2nd order is a multistep method taking into account discretising in both temporal and spatial space. Below is the equation for Adams Bashforth 2nd order.

$$u_i^{t+1} = u_i^t + \frac{3}{2}H_i^t - \frac{1}{2}H_i^{t-1} \tag{18}$$

For the 3 point stencil, the operator is,

$$H^n(u) = C\left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right), \tag{19}$$

which can give the current (n = t) and previous n = t-1) timesteps. For equation (18), the model uses two convolution layers. Thus, one for the current time step and another for the previous timestep. A forward Euler is performed initially to "bootstrap" the system. Bootstrapping is used here as Adams–Bashforth 2nd order is a multi-step method where the scheme requires two past states u^t and u^{t-1} to obtain the next state u^{t+1}. Both current (n) and previous timesteps $(n-1)$ are represented by two kernels in the code. These are taken in and passed in as 2 channel input where the next state is obtained. However, functionally this could also be treated as a 2×3 spatio-temporal stencil. The Adams Bashforth model is iterated over time where the two convolutional layers (previous and current) are updated. After, the boundary conditions are explicitly re-imposed again so the boundary conditions are satisfied.

When the stencil is a 5-point stencil,

$$H^n(u) = C\left[\frac{-u_{i-2} + 16u_{i-1} - 30u_i + 16u_{i+1} - u_{i+2}}{12}\right] \tag{20}$$

with overall time step as in (18).

The model uses two convolution layers, one for current timestep and another for previous timestep. Bootstrapping is used here as well as it is a multistep method. The mirroring approach is also needed as it is a 5-point stencil. Ghost cells are created to ensure that the results stay consistent and to ensure the boundary conditions are handled properly.

A.1.4. Comparing errors in learnt weights

Figure A1 presents the mean-squared error (MSE) for a variety of finite-difference schemes trained on the reference case, including a larger spatial stencil with the 5 point scheme, and the multi-timestep Adam-Bashforth technique. These are compared with the CNN-learned diffusion kernel (k_D), where the kernel structure is adapted to the corresponding numerical scheme: e.g. a five-point spatial stencil k_i with $i \in \{-2,-1,0,1,2\}$ for the higher-order finite difference scheme, and a three-point spatial and two-point temporal stencil with k_i^t with $i \in \{-1,0,1\}$ and $t \in \{-1,0\}$ for the third-order Adams–Bashforth scheme. It is clear from the error magnitudes that each of the numerical schemes (CNN) model yields results that closely matches the original finite difference solution in figure A1. Notably, the FE3 numerical scheme consistently exhibits the lowest error across all weight configurations. The asymmetry of the schemes is due to the training data being driven by one wall.

The training results demonstrate that the CNN-based kernel learner successfully recovers the stencil coefficients for all four numerical schemes when trained on their finite-difference solutions. For all the schemes tested in this work including Forward euler 5-point (FE5) and Adams–Bashforth (AB3 and AB5) the CNN accurately recovers the weights. It is worth noting that when the 5 point Euler learner (FE5) is trained on numerical data generated from a lower order scheme, for example data generated from a 3 point forward Euler (FE3) numerical method, it identifies only the three weights reproducing

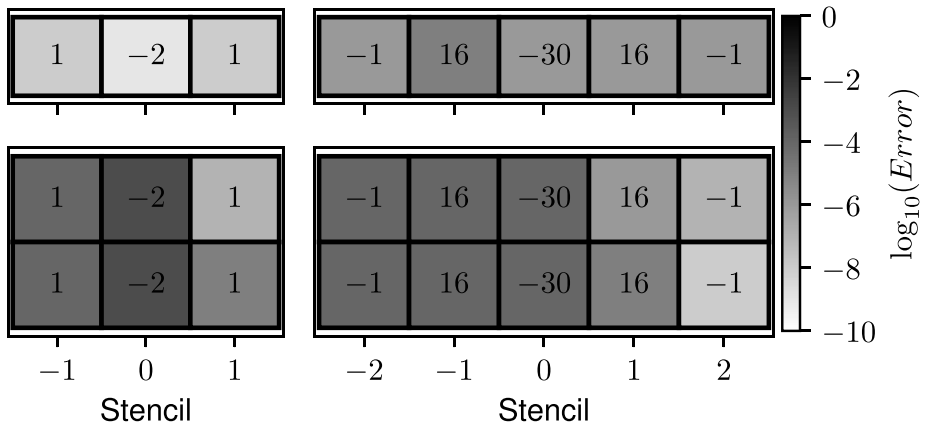

Figure A1. Stencils coloured by the difference between the target and the learned weights for different finite difference schemes, where error is from $\mathcal{O}(w_{\text{target}} - w_{\text{learned}})$ with weights shown from figure 11 to aid identification (without multiplying factors).

the stencil, i.e. $\widetilde{\boldsymbol{k}}_D \approx [0, 1, -2, 1, 0]$. This appears to be a demonstration of the lottery ticket hypothesis even in a small model (da Cunha *et al* 2022), in that for a model given more weights than required, the relevant subset of the weights eventually train as required to capture the physics. The matching to various numerical schemes and even identification of simpler solutions highlights the robustness of these kernel learners.

A.2. Burgers equation (BEQ)

This section extends the evaluation to Burgers' equation to assess the CNN kernel operator's performance on non-linear dynamics. While higher-order schemes such as FE5, AB3, and AB5 (detailed in the appendix) are possible stencils to be used, this study focuses on the FE3 Euler scheme. This choice prioritises interpretability, providing the clearest baseline for demonstrating how the CNN learner captures non-linear effects within the framework of equation (21).

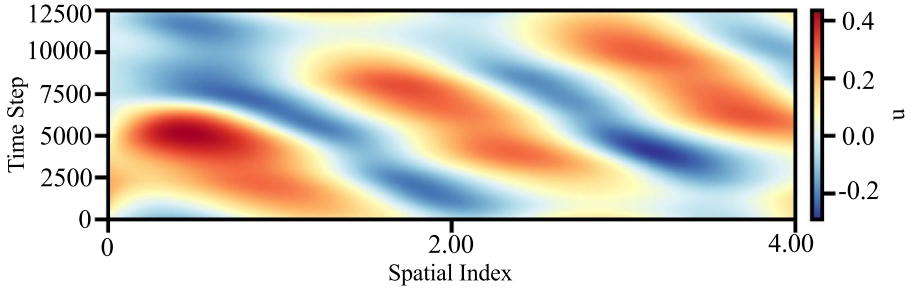

Figure A2. Illustration of Forced Burgers' equation studied based on the simulation parameters: dynamic viscosity $\mu = 0.05$, timestep $\Delta t = 0.002$,spatial grid of $n_x = 250$ points over a domain of length $L_x = 4.0$. $\Delta x = \frac{L_x}{n_x - 1} = 0.0161$ and $N_t = 12,500$ time steps.

The Forced Burgers' equation extends the diffusion model by introducing a nonlinear convective term and a source term (force term), yielding:

$$\frac{\partial u}{\partial t} = \mu \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x} + f. \tag{21}$$

Now in its discretised form becomes:

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} = \mu \frac{u_{i+1}^t - 2u_i^t + u_{i-1}^t}{\Delta x^2} - u_i \frac{u_i^t - u_{i-1}^t}{\Delta x} + f. \tag{22}$$

Taking $C_{\text{diffusion}} = \mu \Delta t / (\Delta x)^2$ and $C_{\text{convective}} = \Delta t / \Delta x$ this is rearranged to

$$u_i^{t+1} = u_i^t + C_{\text{diffusion}} \left(u_{i+1}^t - 2u_i^t + u_{i-1}^t\right) + C_{\text{convective}}\, u_i \left(u_i^t - u_{i-1}^t\right) + f. \tag{23}$$

Q3 Training is performed using numerical simulations under periodic boundary conditions. The convective term is discretised using an upwind scheme (?). The simulation parameters used here are: dynamic viscosity $\mu = 0.05$, timestep $\Delta t = 0.002$, and a spatial grid of $n_x = 250$ points over a domain of length $L_x = 4.0$. This gives a spatial resolution of $\Delta x = \frac{L_x}{n_x - 1} = 0.0161$. Each run spans $N_t = 12,500$ timesteps, resulting in a diffusion Courant number $C_{\text{diff}} = \mu \Delta t / \Delta x^2 = 0.387\,506$ and a convective Courant number $C_{\text{conv}} = \Delta t / \Delta x = 0.124\,500$.

A.2.1. Learning the forced BEQ

The forcing term f is constructed as a superposition of several sine modes, forming a multi-mode forcing of the form

$$f(x,t) = \sum_{i=1}^{N} A_i \sin\left(\omega_i t + \frac{2\pi l_i}{L} x + \phi_i\right) \tag{24}$$

where N, L, A_i, ω_i, and l_i denote the number of forcing modes, the domain length, the amplitude, the temporal frequency, and the spatial mode number of the i^{th} forcing component, respectively, and $\phi_i \sim \mathcal{U}(0, 2\pi)$ is a random phase.

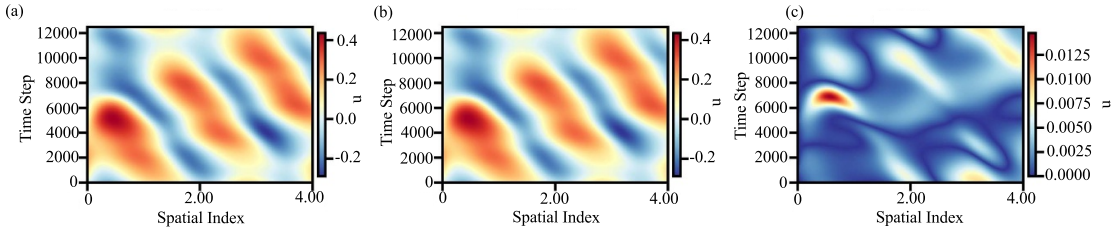

Figure A3. (a) Contour plot of the numerical solution of the forced Burgers equation. (b) Contour plot of the CNN–predicted solution. (c) Contour plot of the error between the numerical and CNN solutions. The simulation uses a dynamic viscosity of $\mu = 0.05$, a timestep of $\Delta t = 0.002$, and a spatial grid of $n_x = 250$ points over a domain of length $L_x = 4.0$, giving $\Delta x = \frac{L_x}{n_x - 1} = 0.0161$. The total number of timesteps is $N_t = 12,500$.

The CNN kernel learner is configured to recover the discrete operators associated with each term of the governing equation. It employs a three–point convolutional kernel with two input channels, corresponding to the diffusive and convective terms. Each channel therefore learns a three–point stencil representing the associated finite–difference weights. The expected weights for the diffusive and convective parts are $[1, -2, 1]$ for the FE3 Euler discretisation and $[-1, 1, 0]$ for a positive–velocity upwind scheme with a backward difference, respectively. When the model is trained on the numerical solution of the forced BEQ, the learned weights are $[0.9533, -1.9072, 0.9539]$ for the diffusive component and $[-0.9544, 0.9317, 0.0224]$ for the convective component. This demonstrates that the approach remains effective even when multiple terms are added to the main diffusion equation and nonlinearity is introduced through the learned weights. Thus, despite the simplicity of the model, the inclusion of additional terms enables it to extend naturally to nonlinear behaviour.

A.2.2. Diffusion-only kernel learning across numerical, analytical, and MD data

Removing the forcing term, the BEQ CNN kernel learner was applied to the three categories of solution data examined in the majority of this study: numerical solutions, analytical solutions, and MDs data of Couette flow (no convective part). Table 3 summarises the learned convolutional kernels obtained from each dataset. The training datasets correspond directly to the solution sets introduced in sections 3.1, 3.2 and 3.4. When trained on numerical solutions i.e. including the 1D diffusion problem and the Stokes' second problem, the model reliably recovered the expected diffusive stencil [1,-2,1], while the convective component converged toward zero, consistent with the underlying physics. For the analytical solutions, the model exhibited two distinct behaviours. In the zero-initial-condition cases (d-i and d-ii), the learned kernels resembled those previously observed in section 3.2.4. In contrast, for the remaining analytical cases, the CNN consistently converged toward the canonical diffusive stencil, demonstrating its ability to infer the correct operator even when the solution structure varies. When applied to MD data, the BEQ CNN kernel learner again recovered a physically meaningful diffusive stencil, as shown in table 3. This result highlights the model's capacity to extract continuum-level operators from inherently noisy, particle-based data. Across all datasets, the BEQ CNN kernel learner consistently identified

Table 3. Diffusion and convection residuals for numerical, analytical, and MD datasets used earlier in studies trained with BEQ CNN kernel learner.

Case	Diffusive Weights	Convective Weights
1) Numerical Solution		
a) Diffusion, $u(0,t)=0$, $u(L,t)=1$, $u(x,0)=\sin(2\pi x)$	$[0.9999908, -1.9999914, 1.0000004]$	$[3.59\times10^{-7}, 1.05\times10^{-6}, -2.53\times10^{-6}]$
b) Stokes' Second Problem, $u(0,t)=0$, $u(L,t)=0$, freq $=0.5$	$[1.0000001, -1.9999996, 1.0000001]$	$[-2.37\times10^{-8}, 5.62\times10^{-10}, 4.63\times10^{-8}]$
2) Analytical Solution		
a) Diffusion, $u(0,t)=0$, $u(L,t)=1$, $u(x,0)=\sin(2\pi x)$	$[1.0896944, -2.1671793, 1.0781881]$	$[3.01\times10^{-5}, 8.54\times10^{-3}, -3.30\times10^{-3}]$
b) Zero BCs, $u(0,t)=0$, $u(L,t)=0$, $u(x,0)=\sin(0.5\pi x)$	$[0.8061033, -1.6343783, 0.806552]$	$[-2.64\times10^{-4}, 5.20\times10^{-4}, -2.86\times10^{-4}]$
c) Finite BCs, $u(0,t)=0$, $u(L,t)=1$, $u(x,0)=\sin(0.625\pi x)$	$[1.015961, -2.0302017, 1.0150297]$	$[3.09\times10^{-4}, -5.84\times10^{-4}, 3.88\times10^{-4}]$
d-i) Random IC/BC: $u(0,t)=0$, $u(L,t)=1$, $u(x,0)=0$	$[-0.47087845, -0.31043014, 0.5051787]$	$[-0.28317, 0.20126948, -0.03350251]$
d-ii) Random IC/BC: $u(0,t)=0.5$, $u(L,t)=1$, $u(x,0)=0$	$[0.65238667, -1.2853659, 0.67983276]$	$[-0.03653744, 0.08104009, -0.02866757]$
d-iii) Random IC/BC: $u(0,t)=0.5$, $u(L,t)=1$, $u(x,0)=\sin(0.583\pi x+0.524)$	$[1.0284132, -2.0533178, 1.0270225]$	$[3.91\times10^{-4}, -7.72\times10^{-4}, 6.48\times10^{-4}]$
d-iv) Random IC/BC: $u(0,t)=-1$, $u(L,t)=0$, $u(x,0)=\sin(1.125\pi x-1.571)$	$[1.0339645, -2.0704024, 1.0371671]$	$[-6.86\times10^{-4}, 5.52\times10^{-4}, -1.81\times10^{-4}]$
3) MD Data		
MD dataset	$[0.9812755, -1.9643935, 0.979582]$	$[0.07003298, -0.1637399, 0.07922383]$

the underlying diffusive operator with high accuracy. The method demonstrated robustness to differing flow regimes, boundary conditions, and data sources, without requiring hyperparameter tuning or additional training augmentation. These findings underscore the capability of the BEQ framework to infer governing physical operators directly from solution fields, reinforcing its potential as a general tool for data-driven operator discovery in fluid and transport systems.

A.3. PINNs constraints study

A parametric study using PINNs identified $\lambda_n = 0.1$ as the optimal Lagrange multiplier to guide the CNN kernel toward the target weights. Error magnitude, ϕ was measured between learned (L) and target (T) weights. This is computed as:

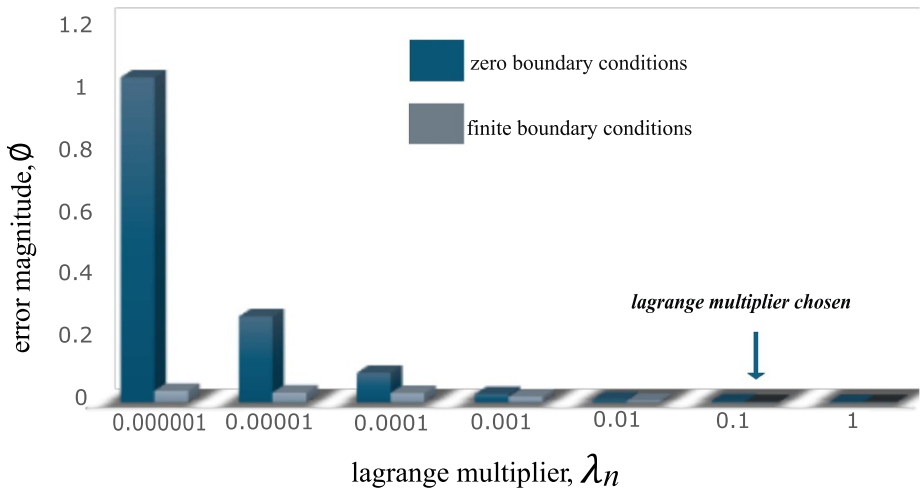

Figure A4. Parametric studies plot done for the selection of lagrange multiplier(λ_n) using cases (i)zero boundary conditions and (ii)finite boundary conditions on a range of values.

$$\phi = \sqrt{(L_1 - T_1)^2 + (L_2 - T_2)^2 + (L_3 - T_3)^2}$$

and results showed that for $\lambda_n \geqslant 0.1$, both boundary conditions (ZBC and FBC) converged with minimal error. Thus, $\lambda_n = 0.1$ was chosen to ensure effective PINNs activation. This studies is seen in figure A4 indicating the error analysis conducted.

References

Q4
Q5
Agostini L 2020 Exploration and prediction of fluid dynamical systems using auto-encoder technology *Phys. Fluids* **32** 067103
Ambarzumian V 1929 Über eine frage der eigenwerttheorie *Z. Phys.* **53** 690–5
Bar-Sinai Y, Hoyer S, Hickey J and Brenner M P 2019 Learning data-driven discretizations for partial differential equations *Proc. Natl Acad. Sci.* **116** 15344–9
Batchelor G K 2000 *An Introduction to Fluid Dynamics* (*Cambridge Mathematical Library*) (Cambridge University Press)
Bonfanti A, Santana R, Ellero M and Gholami B 2024 On the generalization of pinns outside the training domain and the hyperparameters influencing it *Neural Comput. Appl.* **36** 22677–96
Q6
Brunton S L, Noack B R and Koumoutsakos P 2024 Machine learning for fluid mechanics *Annu. Rev. Fluid Mech.* **7**
Cai S, Wang Z, Wang S, Perdikaris P and Karniadakis G E 2021 Physics-informed neural networks for heat transfer problems *J. Heat Transfer* **143** 060801
da Cunha A, Natale E and Viennot L 2022 Proving the strong lottery ticket hypothesis for convolutional neural networks *ICLR 2022 - 10th Int. Conf. Learning Representations(Virtual, France)* (available at: https://hal.science/hal-03548226)
Deng Z, Wang J, Liu H, Xie H, Li B, Zhang M, Jia T, Zhang Y, Wang Z and Dong B 2023 Prediction of transonic flow over supercritical airfoils using geometric-encoding and deep-learning strategies (arXiv:2300.00000)
Duruisseaux V, Kossaifi J and Anandkumar A 2025 Fourier neural operators explained: a practical perspective (arXiv:2512.01421)
Evans D J and Morris G P 2007 *Statistical Mechanics of Non-Equilibrium Liquids* 2nd edn (Australian National University Press)
Fesser L, D'Amico-Wong L and Qiu R 2023 Understanding and mitigating extrapolation failures in physics-informed neural networks (arXiv:2306.09478)
Gao C, Cai J, Brazhenko V, Mochalin I, Wang X and Cao J 2026 Efficient long-term prediction of unsteady flow using hybrid neural network with non-intrusive reduced-order modeling *Meas. Sci. Technol.* **37** 055302
Geneva N and Zabaras N 2022 Transformers for modeling physical systems *Neural Netw.* **146** 272–89
Glorot X and Bengio Y 2010 Understanding the difficulty of training deep feedforward neural networks *Proc. 13th Int. Conf. on Artificial Intelligence and Statistics* (JMLR Workshop and Conference Proceedings) pp 249–56
Green M S 1952 Markoff random processes and the statistical mechanics of time-dependent phenomena *J. Chem. Phys.* **20** 1281–95
Hassija V, Chamola V, Mahapatra A, Singal A, Goel D, Huang K, Scardapane S, Spinelli I, Mahmud M and Hussain A 2024 Interpreting black-box models: a review on explainable artificial intelligence *Cogn. Comput.* **16** 45–74
Hirsch C 2007 *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics* (Elsevier)
Kim Y and Choi Y 2022 Learning finite difference methods for reaction-diffusion type equations with fcnn [formula presented] *Comput. Math. Appl.* **123** 115–22
Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)
Kovachki N B, Lanthaler S and Stuart A M 2024 Operator learning: algorithms and analysis *Handbook Numer. Anal.* **25** 419–67
Kubo R 1957 Statistical-mechanical theory of irreversible processes. I. General theory and simple applications to magnetic and conduction problems *J. Phys. Soc. Japan* **12** 570–86
LeCun Y and Bengio Y 1998 Convolutional networks for images, speech and time series *The Handbook of Brain Theory and Neural Networks*
Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A and Anandkumar A 2020 Fourier neural operator for parametric partial differential equations (arXiv:2010.08895)
Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A and Anandkumar A 2020 Neural operator: graph kernel network for partial differential equations (arXiv:2003.03485)
Long Z, Lu Y, Ma X and Dong B 2018 PDE-Net: learning pdes from data *Technical Report* (arXiv:1710.09668 [math.NA])
Lu L, Jin P, Pang G, Zhang Z and Karniadakis G E 2023 Learning nonlinear operators via deeponet based on the universal approximation theorem of operators *Nat. Mach. Intell.* **3** 218–29
Lu L, Jin P and Karniadakis G E 2019 Deeponet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators (arXiv:1910.03193)
Nie X B *et al* 2004 A continuum and molecular dynamics hybrid method for micro-and nano-fluid flow *J. Fluid Mech.* **500** 55–64
NIST 2026 SAT-TMMC: liquid-vapor coexistence properties -linear-force shifted potential at 4.0 σ (National Institute of Standards and Technology) (Accessed 05 January 2026)
O'connell S T and Thompson P A 1995 Molecular dynamics–continuum hybrid computations: a tool for studying complex fluid flows *Phys. Rev. E* **52** R5792
O'Shea K and Nash R 2015 An introduction to convolutional neural networks (arXiv:1511.08458)
Petravic J and Harrowell P 2006 The boundary fluctuation theory of transport coefficients in the linear-response limit *J. Chem. Phys.* **124** 014103
Queiruga A F 2019 Studying shallow and deep convolutional neural networks as learned numerical schemes on the 1D heat equation and Burgers' equation (arXiv:1909.08142)
Rackauckas C *et al* 2020 Universal differential equations for scientific machine learning (arXiv:2001.04385)
Rackauckas C, Ma Y, Martensen J, Warner C, Zubov K, Supekar R, Skinner D, Ramadhan A and Edelman A 2020 Universal differential equations for scientific machine learning (arXiv:2001.04385)
Raissi M and Karniadakis G E 2018 Hidden physics models: machine learning of nonlinear partial differential equations *J. Comput. Phys.* **357** 125–41
Raissi M, Perdikaris P and Karniadakis G E 2017 Inferring solutions of differential equations using noisy multi-fidelity data *J. Comput. Phys.* **335** 736–46

Raissi M, Perdikaris P and Karniadakis G E 2017 Machine learning of linear differential equations using gaussian processes *J. Comput. Phys.* **348** 683–93
Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
Rapaport D C 2004 *The Art of Molecular Dynamics Simulation* (Cambridge University Press)
Rudy S H, Brunton S L, Proctor J L and Kutz J N 2017 Data-driven discovery of partial differential equations *Sci. Adv.* **3** e1602614
Sasaki R, Takeishi N, Yairi T and Hori K 2019 Neural gray-box identification of nonlinear partial differential equations *Pacific Rim Int. Conf. Artificial Intelligence* (Springer) pp 309–21
Shan X, Liu Y, Cao W, Sun X and Zhang W 2023 Turbulence modelling via data assimilation and machine learning for separated flows over airfoils *AIAA J.* **61** 3883–99
Smith E 2013 On the coupling of molecular dynamics to continuum computational fluid dynamics *Mech. Eng.*
Smith E 2014 On the coupling of molecular dynamics to continuum computational fluid dynamics *PhD Thesis* (Imperial College London)
Smith E R, Daivis P J and Todd B D 2019 Measuring heat flux beyond Fourier's law *J. Chem. Phys.* **150** 064103
Solera-Rico A, Vila C S, Gómez-López M, Wang Y, Almashjary A, Dawson S T M and Vinuesa R 2024 β-variational autoencoders and transformers for reduced-order modelling of fluid flows *Nat. Commun.* **15** 1361
Sprittles J E, Liu J, Lockerby D A and Grafke T 2023 Rogue nanowaves: a route to film rupture *Phys. Rev. Fluids* **8** L092001
Todd B D and Daivis P J 2017 *Nonequilibrium Molecular Dynamics: Theory, Algorithms and Applications* (Cambridge University Press)
Vinuesa R and Brunton S 2022 Emerging trends in machine learning for computational fluid dynamics (arXiv:2211.15145)
Yuan L, Park H S and Lejeune E 2022 Towards out of distribution generalization for problems in mechanics *Comput. Methods Appl. Mech. Eng.* **400** 115569
Zhang X *et al* 2025 Artificial intelligence for science in quantum, atomistic and continuum systems *Found. Trends Mach. Learn.* **18** 385–912
Zhang Y, Sprittles J E and Lockerby D A 2019 Molecular simulation of thin liquid films: thermal fluctuations and instability *Phys. Rev. E* **100** 023108
Zobeiry N and Humfeld K D 2021 A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications *Eng. Appl. Artif. Intell.* **101** 104232
Q7 Available at: https://github.com/kwamea-b/CNN_numerical_schemes